



**Class - XI**

**Part - I**



**Government of Kerala  
DEPARTMENT OF EDUCATION**

**State Council of Educational Research and Training (SCERT); Kerala  
2016**

## THE NATIONAL ANTHEM

Jana-gana-mana adhinayaka, jaya he  
Bharatha-bhagya-vidhata.  
Punjab-Sindh-Gujarat-Maratha  
Dravida-Utkala-Banga  
Vindhya-Himachala-Yamuna-Ganga  
Uchchala-Jaladhi-taranga  
Tava subha name jage,  
Tava subha asisa mage,  
Gahe tava jaya gatha.  
Jana-gana-mangala-dayaka jaya he  
Bharatha-bhagya-vidhata.  
Jaya he, jaya he, jaya he,  
Jaya jaya jaya, jaya he!

## PLEDGE

India is my country. All Indians are my brothers and sisters.

I love my country, and I am proud of its rich and varied heritage. I shall always strive to be worthy of it.

I shall give respect to my parents, teachers and all elders and treat everyone with courtesy.

I pledge my devotion to my country and my people. In their well-being and prosperity alone lies my happiness.

*Prepared by :*

**State Council of Educational Research and Training (SCERT)**

Poojappura, Thiruvananthapuram 695012, Kerala

Website : [www.scertkerala.gov.in](http://www.scertkerala.gov.in) e-mail : [scertkerala@gmail.com](mailto:scertkerala@gmail.com)

Phone : 0471 - 2341883, Fax : 0471 - 2341869

Typesetting and Layout : SCERT

© Department of Education, Government of Kerala

Dear students,

Computer Science, a subject belonging to the discipline of Science and of utmost contemporary relevance, needs continuous updating. The Higher Secondary Computer Science syllabus has been revised with a view to bringing out its real spirit and dimension. The constant and remarkable developments in the field of computing as well as the endless opportunities of research in the field of Computer Science and Technology have been included.

This textbook is prepared strictly in accordance with the revised syllabus for the academic year 2014 - 15. It begins with the historical developments in computing and familiarises the learner with the latest technological advancements in the field of computer hardware, software and network. The advancement in computer network, Internet technology, wireless and mobile communication are also dealt with extensively in the content. In addition to familiarising various services over the Internet, the need to be concerned about the factors that harness morality and the awareness to refrain from cyber security threats are also highlighted.

The major part of the textbook as well as the syllabus establishes a strong foundation to construct and enhance the problem solving and programming skills of the learner. The multi-paradigm programming language C++ is presented to develop programs which enable computers to manage different real life applications effectively. The concepts and constructs of the principles of programming are introduced in such a way that the learner can grasp the logic and implementation methods easily.

I hope this book will meet all the requirements for stepping to levels of higher education in Computer Science and pave your way to the peak of success.

Wish you all success.

**Dr P. A. Fathima**  
Director, SCERT, Kerala

## Textbook Development Team

### COMPUTER SCIENCE

**Joy John**

HSST, St. Joseph's HSS  
Thiruvananthapuram

**Asees V.**

HSST, GHSS Velliyode, Kozhikode

**Roy John**

HSST, St. Aloysius HSS  
Elthuruth, Thrissur

**Aboobacker P.**

HSST, Govt. GHSS Chalappuram,  
Kozhikode

**Shajan Jos N.**

HSST, St. Joseph's HSS, Pavaratty,  
Thrissur

**Afsal K. A.**

HSST, GHSS Sivapuram,  
Kariyathankare P.O., Kozhikode

**Prasanth P. M.**

HSST, St. Joseph's Boys' HSS,  
Kozhikode

**Vinod V.**

HSST, NSS HSS, Prakkulam, Kollam

**Rajamohan C.**

HSST, Nava Mukunda HSS,  
Thirunavaya, Malappuram

**A. S. Ismael**

HSST, Govt. HSS Palapetty,  
Malappuram

**Sunil Kariyatan**

HSST, Govt. Brennen HSS,  
Thalassery

**Sai Prakash S.**

HSST, St. Thomas HSS,  
Poonthura, Thiruvananthapuram

### Experts

**Dr Lajish V. L.**

Assistant Professor, Dept. of Computer Science, University of Calicut

**Dr Madhu S. Nair**

Assistant Professor, Dept. of Computer Science, University of Kerala

**Madhu V. T.**

Director, Computer Centre, University of Calicut

**Dr Binu P. Chacko**

Associate Professor, Dept. of Computer Science,  
Prajyoti Niketan College, Pudukad

**Dr Sushil Kumar R.**

Associate Professor, Dept. of English, D.B. College, Sasthamcotta

**Dr Vineeth K. Paleri**

Professor, Dept. of Computer Science and Engineering, NIT, Kozhikode

**Maheswaran Nair V.**

Sub Divisional Engineer, Regional Telecom Training Centre, Thiruvananthapuram

### Artist

Sudheer Y.

Vineeth V.

### Academic Co-ordinator

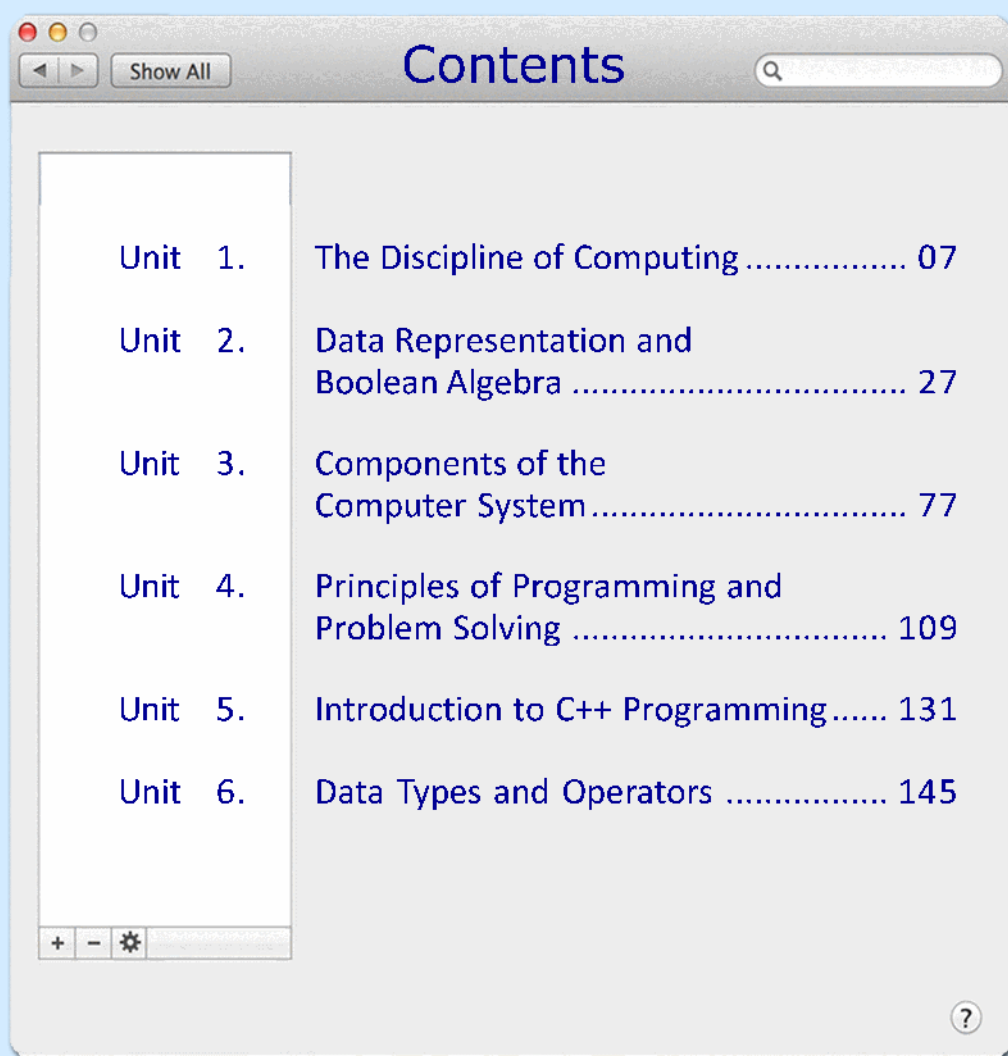
**Dr Meena S.**

Research Officer, SCERT

**Jancy Rani A. K.**

Research Officer, SCERT





Unit 1.	The Discipline of Computing .....	07
Unit 2.	Data Representation and Boolean Algebra .....	27
Unit 3.	Components of the Computer System .....	77
Unit 4.	Principles of Programming and Problem Solving .....	109
Unit 5.	Introduction to C++ Programming.....	131
Unit 6.	Data Types and Operators .....	145

### Icons used in this textbook



Let us do



Check yourself



Information box



Lab activities



Learning outcomes

## Key concepts

- **Computing milestones and machine evolution**
  - Counting and the evolution of the positional number system
  - Evolution of the computing machine
- **Generations of computers**
  - First generation computers
  - Second generation computers
  - Third generation computers
  - Fourth generation computers
  - Fifth generation computers
- **Evolution of computing**
  - Programming languages
  - Algorithm and computer programs
  - Theory of computing

## The Discipline of Computing

Computers now play a major role in almost every aspect of life and influence our lives in one way or the other. Today, almost everyone is a computer user and many are computer programmers. Getting computers to do what you want them to do requires intensive hands-on experience. But computer science can be seen on a higher level, as a science of problem-solving. Computer scientists must be able to analyse problems and design solutions for real world problems. The Computer Science discipline covers a wide range of topics from theoretical aspects like design of algorithms to more practical aspects like application development and its implementation. The discipline of Computer Science involves the systematic study of algorithmic processes – their theory, analysis, design, efficiency, implementation and application – that describe and transform information.

The concept of computing has evolved from the abacus to super computers of today. This chapter discusses the evolution of computing devices and provides an overview of the different generations of computers. The evolution of programming languages and the contributions of Alan Turing to computer science are also discussed.





## 1.1 Computing milestones and machine evolution

In ancient times people used stones for counting. They made scratches on walls or tied knots in ropes to record information. Progressively many attempts had been made to replace these manual computing techniques with faster computing machines. In this section, we will have a look at the ancient methods of counting and the evolution of the positional number system.

### 1.1.1 Counting and the evolution of the positional number system

The idea of number and the process of counting goes back far before history began to be recorded. It is believed that even the earliest humans had some sense of 'more' or 'less'. As human beings differentiated into tribes and groups, it became necessary to be able to know the number of members in the group and in the enemy's camp. And it was important for them to know if the flock of sheep or other animals was increasing or decreasing in size. In order to count items, such as animals, 'sticks' were used, each stick representing one animal or object.

Let us now see how the number system evolved. It is important to note that the system that we use everyday is a product of thousands of years of progress and development. It represents contributions of many civilisations and cultures. The number system is a method in which we represent numbers. The chronological development of the number system throughout the history is discussed below.

To begin with let us see the Egyptian number system that emerged around 3000BC. It used 10 as a radix (base). They had unique symbols for 1 to 9, 10 to 90, 100 to 900 and 1000 to 9000. As the Egyptians write from right to left, the largest power of ten appears to the right of the other numerals.

Later on, the era of Sumerian/Babylonian number system began. It used 60 as its number base, known as the sexagesimal system. Numerals were written from left to right. It was the largest base that people ever used in number systems. They did not use any symbol for zero, but they used the idea of zero. When they wanted to express zero, they just left a blank space within the number they were writing.

The Chinese number system emerged around in 2500 BC. It was the simplest and the most efficient number system. The Chinese had numbers from 1 to 9. It had the base 10, very similar to the one we use today. They used small bamboo rods to represent the numbers 1 to 9.

Approximately in 500 BC, the Greek number system known as Ionian number system evolved. It was a decimal number system and the Greeks also did not have any symbol for zero.



The Romans started using mathematics for more practical purposes, such as in the construction of roads, bridges, etc. They used 7 letters (I, V, X, L, C, D and M) of the alphabet for representing numbers.

The Mayans used a number system with base 20. There is a simple logic behind this base 20. It is the sum of the number of fingers and toes. This number system could produce very accurate astronomical observations and make measurements with greater accuracy.

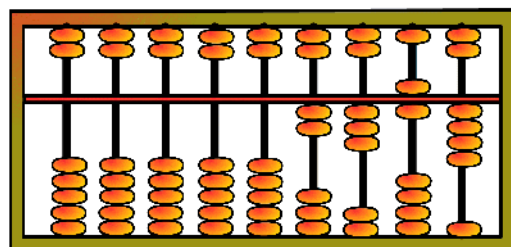
The Hindu-Arabic numeral system actually originated in India, around 1500 years ago. It was a positional decimal numeral system and had a symbol for zero. This invention can indeed be termed as one of India's greatest contributions to the world. Later on many of the countries adopted this numeral system. Now let us discuss the evolution of computing machines.

### 1.1.2 Evolution of the computing machine

During the period from 3000 BC to 1450 AD, human beings started communicating and sharing information with the aid of simple drawings and later through writings. The introduction of numbers led to the invention of Abacus, the first computing machine. In the following section, we will examine some important milestones in the evolution of computing machines.

#### a. Abacus

Abacus was discovered by the Mesopotamians around 3000 BC. The word 'abacus' means calculating board. An abacus consisted of beads on movable rods divided into two parts. The abacus may be considered the first computer for basic arithmetical calculations. An abacus is shown in Figure 1.1.



*Fig. 1.1 : Abacus*

The abacus is also called a counting frame, a calculating tool for performing arithmetic operations. The Chinese improved abacus as a frame holding vertical wires, with seven beads on each wire. A horizontal divider separates the top two beads from the bottom five. Addition and multiplication of numbers was done using the place value of digits and position of beads in an abacus.

The abacus works on the basis of the place value system. Reading it is almost like reading a written numeral. Each of the five beads below the bar has a value of 1. Each of the two beads above the bar has a value of 5. The beads which are pushed

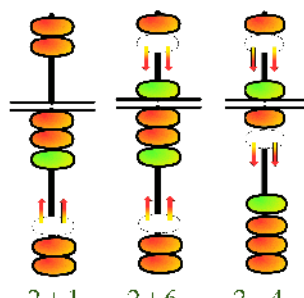


Fig. 1.2(a) : Addition using Abacus

against the bar represent the number. The number on the abacus given in Figure 1.1 is 2364.

Abacus is used even today by children to learn counting. A skilled abacus operation can be as fast as a hand held calculator. Figure 1.2(a) shows the addition of two single digit numbers. Figure 1.2(b) shows how two numbers (54 and 46) are added.

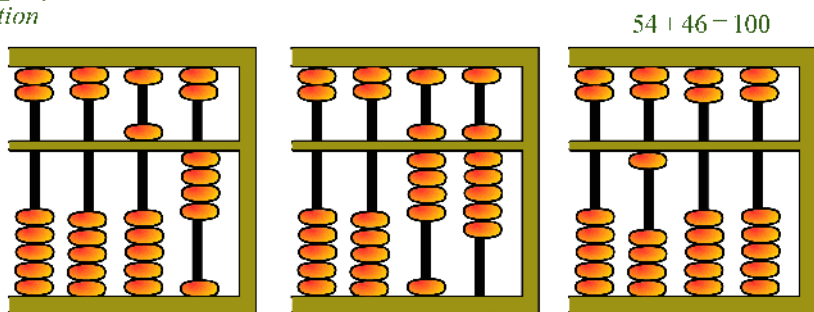


Fig. 1.2(b) : Addition using Abacus

By shifting carry to left and adding we get 100

## b. Napier's bones

John Napier was a mathematician who devised a set of numbering rods known as Napier's bones in 1617 AD, by which a multiplication problem could be easily performed. These numbered rods could perform multiplication of any number by a number in the range of 2-9. There are 10 bones corresponding to the digits 0-9 and a special eleventh bone that is used to represent the multiplier. This device was known as Napier's bones. John Napier also invented logarithm in 1614, that reduced tedious multi-digit multiplications to addition problems. A representation of Napier's bones is given in Figure 1.3.

1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

Fig. 1.3 : John Napier (1550 - 1617) and Napier's Bones



The strips of Napier's bones are the times tables. Each square gives  $2 \times$  number,  $3 \times$  number and so on, but the tens and units are written above and below a slanting line respectively. Napier's bones is good for multiplying a long number by a single digit number. Let us multiply 425928 by 7. First take the strips for 4, 2, 5, 9, 2 and 8, and fit them into the frame. Look at the squares next to the 7 on the side. It is coloured green in Figure 1.4. Now read the digits – any number within slanting lines must be added. So the answer is  $2(8+1)(4+3)(5+6)(3+1)(4+5)6$  or 297(11)496. All the digits except 11 are in their position. 11 needs to have 10 carried to the left. This makes  $29(7+1)496$  or 2981496, which is the correct answer.

1	4	2	5	9	2	8	1	4	2	5	9	2	8
2	8	4	1	1	8	4	1	8	4	1	1	8	4
3	1	2	6	1	5	2	7	6	2	4	2	7	6
4	1	6	8	2	0	3	6	8	3	2	3	6	8
5	2	0	0	2	5	4	5	1	0	4	0	5	1
6	2	4	1	2	3	5	4	1	2	4	8	5	4
7	2	8	1	3	6	4	5	1	4	5	6	6	4
8	3	2	1	4	0	7	2	1	6	6	4	7	2
9	3	6	1	4	5	8	1	1	8	7	2	8	1

$7 \times 425928 =$   
 $= 2(8+1)(4+3)(5+6)(3+1)(4+5)6$   
 $= 297(11)496 = 2981496$

Fig. 1.4 : Multiplication using Napier's Bones

### c. Pascaline

Blaise Pascal was a French mathematician and one of the first modern scientists to develop a calculator. In 1642, at the age of 19, he developed a computing machine that was capable of adding and subtracting two numbers directly and that could multiply and divide by repetition. Pascal invented this arithmetic calculator to assist his father in his work as a tax collection supervisor. This machine was operated by dialing a series of wheels, gears and cylinders. He called it 'Pascaline'. Figure 1.5 shows a Pascaline.

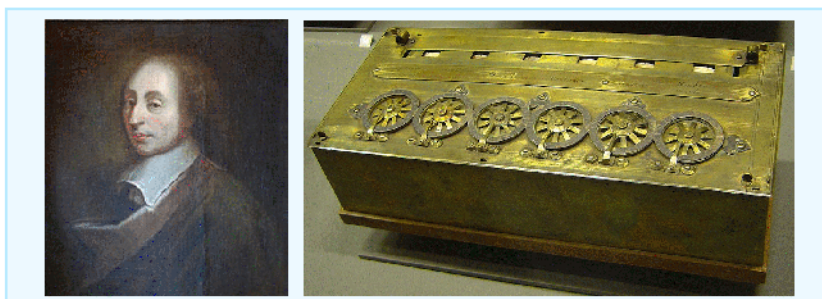
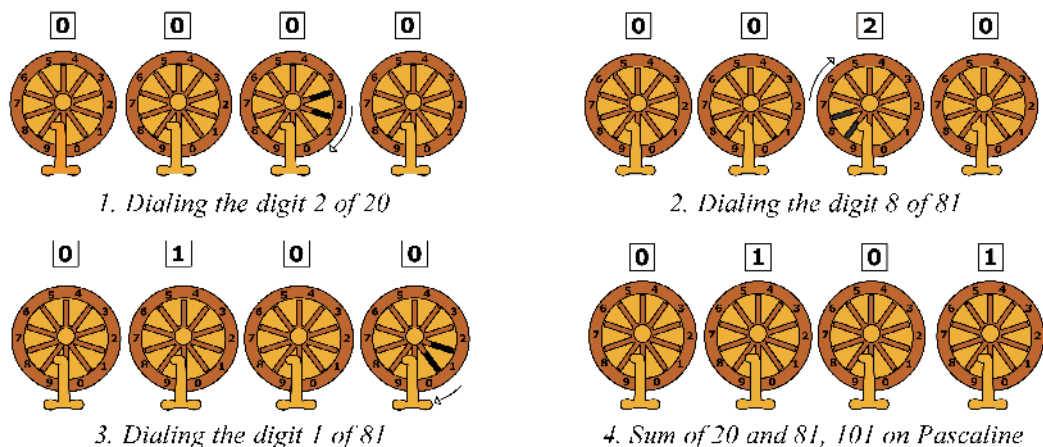


Fig. 1.5 : Blaise Pascal (1623 - 1662) and Pascaline

Consider adding the numbers 20 and 81 using Pascaline. Initially, the Pascaline will be set to 0 for all the six digits. To dial 20, you just have to put your finger into the space between the spokes next to digit 2 of the second wheel and rotate the wheel in clockwise direction until your finger strikes against the fixed stop on the bottom of the wheel. This rotation transmits the value of two into the second window from the right. Now the machines will display number 0020.



*Pascaline wheel*



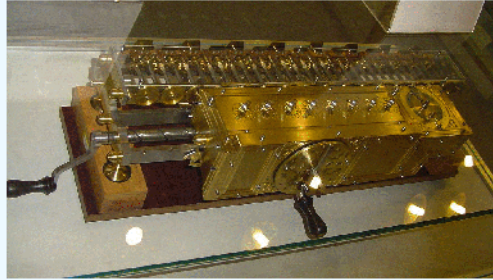
*Fig. 1.6 : Adding using Pascaline*

To dial 81, put your finger into the space between the spokes next to digit 8 of the second wheel and rotate it. After the second drum reaches number 9, the gears inside Pascaline will carry to the next drum one unit and the third drum of the machine will rotate by one tenth. So after the end of dialing number 8, the machine will display the number 100. Now put your finger into the space between the spokes next to digit 1 and rotate it in the same way you did before. Now the machine will display the number 0101, which is the final result of addition.

#### **d. Leibniz's calculator**

In 1673 the German mathematician-philosopher Gottfried Wilhelm von Leibniz designed a calculating machine called the Step Reckoner. The Step Reckoner expanded on Pascal's ideas and extended the capabilities to perform multiplication and division as well. Leibniz successfully introduced this calculator onto the market. His unique, drum-shaped gears formed the basis of many successful calculator designs in later years.

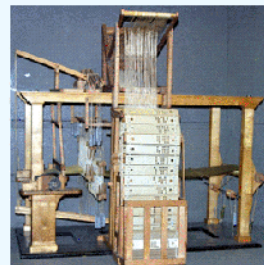




*Fig. 1.7 : Gottfried Wilhelm von Leibniz (1646 - 1716) and Leibniz Calculator*

### e. Jacquard's loom

The Jacquard loom is a mechanical loom, invented by Joseph Marie Jacquard in 1801, that simplifies the process of manufacturing textiles with complex patterns. The loom is controlled by punched cards with punched holes, each row of which corresponds to one row of the design. Multiple rows of holes are punched on each card and the many cards that compose the design of the textile are joined together in order. The Jacquard loom not only reduced the amount of human labour, but also allowed to store patterns on cards to be utilised again to create the same product. These punched cards were innovative because the cards had the capability to store information on them. This ability to store information triggered the computer revolution. The punched card concept was adopted by Charles Babbage to control his Analytical Engine and later by Herman Hollerith.



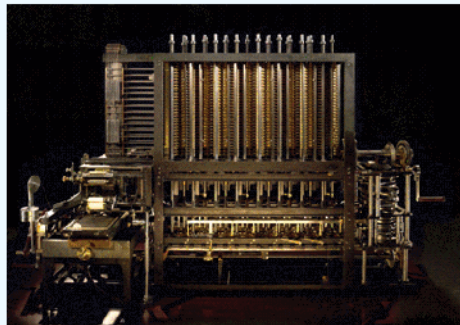
*Fig. 1.8 : Joseph Marie Jacquard (1752 - 1834) and Jacquard's Loom*

### f. Difference engine

The first step towards the creation of computers was made by a mathematics professor, Charles Babbage. He dreamed of removing the human element from the calculations. He realised that all mathematical calculations can be broken up into simple operations which are constantly repeated and these operations could be carried out by an automatic machine. Charles Babbage started working on a



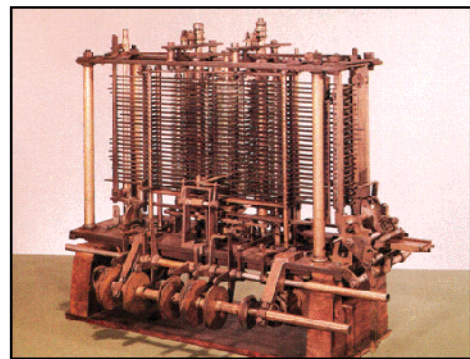
Difference Engine that could perform arithmetic calculations and print results automatically. In 1822, Babbage invented the Difference Engine to compile mathematical tables. On completing it, he conceived the idea of a better machine that could perform not just one mathematical task but any kind of calculation.



*Fig. 1.9 : Charles Babbage (1791 - 1871) and Difference Engine*

### **g. Analytical engine**

In 1833, Charles Babbage started designing the Analytical Engine – the real predecessor of the modern day computer. Analytical Engine marks the development from arithmetic calculation to general-purpose computation. The Analytical Engine has many features found in the modern digital computer. The Engine had a ‘Store’ (memory) where numbers and intermediate results could be stored, and a separate ‘Mill’ (processor) where arithmetic processing could be performed. Its input/output devices were in the form of punched cards containing instructions. These instructions were written by Babbage’s assistant, Augusta Ada King, the first programmer in the world. Owing to the lack of technology at that time, the Analytical Engine was never built, but Babbage established the basic principles on which today’s modern computers work. Charles Babbage’s great inventions – the Difference Engine and the Analytical Engine earned Charles Babbage the title ‘Father of Computer’.



*Fig. 1.10 : A model of Analytical Engine*

### **h. Hollerith’s machine**

In 1887, an American named Herman Hollerith fabricated the first electromechanical punched card tabulator that used punched cards for input, output and instructions.



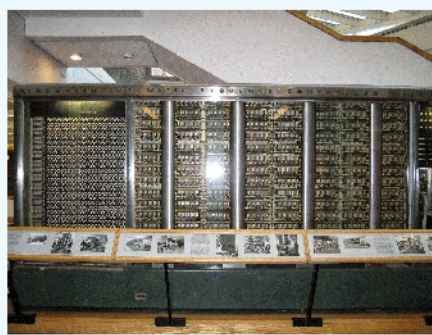
The card had holes on them in a particular pattern, having special meaning for each kind of data. In 1880's, the US Census Bureau had huge amounts of data to tabulate. It would take at least ten years to analyse population statistics manually. Herman Hollerith's greatest breakthrough was his use of electricity to read, count and sort punched cards whose holes represented data. His machines were able to accomplish the task in one year. In 1896, Hollerith started the Tabulating Machine Corporation which after a series of mergers, became International Business Machines (IBM) Corporation in 1924.



*Fig. 1.11 : Herman Hollerith (1860 - 1929) and Hollerith Census Machine*

### **i. Mark - I**

In 1944, Howard Aiken, in collaboration with engineers at IBM, constructed a large automatic electromechanical computer. Aiken's machine, called the Harvard Mark I, based on Babbage's Analytical Engine, handled 23-decimal-place numbers and could perform all four arithmetic operations. It was preprogrammed to handle logarithms and trigonometric functions. Using Mark I, two numbers could be added in three to six seconds. For input and output, it used paper-tape readers, card readers, card punch and typewriters.



*Fig. 1.12 : Howard Aiken (1900 - 1973) and Mark I computer*



### Check yourself



1. The Sumerian Number System is also known as \_\_\_\_\_.
2. What are the features of Hindu Arabic Number system?
3. How is the zero represented in the Babylonian Number System?
4. Who is the first programmer in the world?
5. The computing machine developed by Blaise Pascal is known as \_\_\_\_\_.

## 1.2 Generations of computers

The evolution of computer started from the 16<sup>th</sup> century, resulting in today's modern machines. It is distinguished into five generations of computers from the first programmable computer to the ones based on artificial intelligence. Each generation of computers is characterised by a major technological development that fundamentally changed the way computers operate, resulting smaller, cheaper, more powerful, more efficient and reliable computing devices. Based on various stages of development, computers can be divided into different generations. They are:

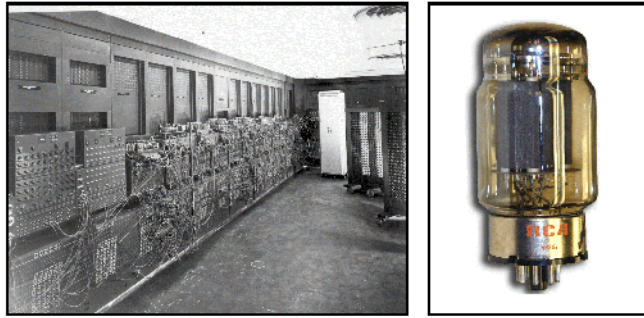
- First Generation Computers (1940 – 1956)
- Second Generation Computers (1956 – 1963)
- Third Generation Computers (1964 – 1971)
- Fourth Generation Computers (1971 – Present)
- Fifth Generation Computers (Present and beyond)

### 1.2.1 First generation computers (1940 – 1956)

The first generation computers were built using vacuum tubes. This generation implemented the stored program concept. A vacuum tube is a device controlling electric current through a vacuum in a sealed container. This cylindrical shaped container is made of thin transparent glass. The input was based on punched cards and paper tapes and output was displayed on printouts.

The first general purpose programmable electronic computer, the Electronic Numerical Integrator and Calculator (ENIAC) belongs to this generation. ENIAC was built by J. Presper Eckert and John V. Mauchly. The ENIAC was 30-50 feet long, weighed 30 tons, contained 18,000 vacuum tubes, 70,000 registers, 10,000 capacitors and required 1,50,000 watts of electricity. First generation computers





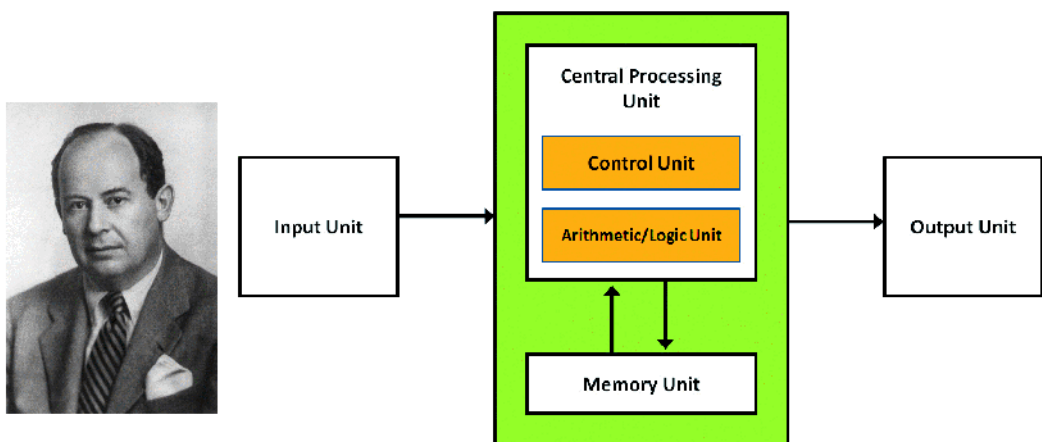
*Fig. 1.13 : ENIAC and Vacuum tube*

were too bulky in size, required a large room for installation and used to emit large amount of heat. Consequently, air-conditioner was a must for the proper working of computers.

Before ENIAC was completed, Von Neumann designed the Electronic Discrete Variable Automatic Computer (EDVAC) with a memory to hold both stored program as well as data. Eckert and Mauchly later developed the first commercially successful computer, the Universal Automatic Computer (UNIVAC), in 1952.

### **Von Neumann architecture**

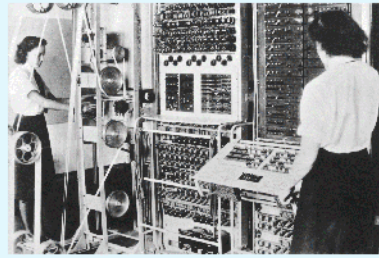
The mathematician John Von Neumann conceived a computer architecture which forms the core of nearly every computer system in use today. This architecture known as Von Neumann architecture consists of a central processing unit (CPU) containing arithmetic logic unit (ALU) and control unit (CU), input-output unit and a memory for storing data and instructions. This model implements the 'Stored Program Concept' in which the data and the instructions are stored in the memory.



*Fig. 1.14 : John Von Neumann (1903 - 1957) and Von Neumann architecture*

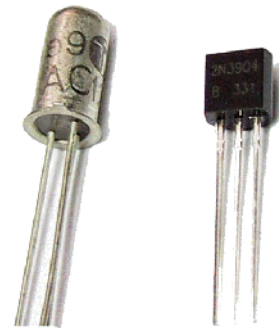
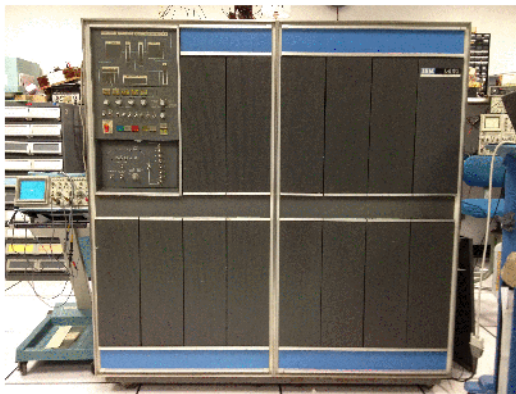


In 1943, the British developed a secret code-breaking computer called Colossus to decode German messages. It was designed using vacuum tubes by the engineer Tommy Flowers. The Colossus's impact on the development of the computer industry was rather limited for two important reasons. First, Colossus was not a general-purpose computer; it was only designed to decode secret messages. Second, the existence of the machine was kept secret until 1970s. This deprived most of those involved with Colossus of credit for their pioneering advancements in electronic digital computing.



### 1.2.2 Second generation computers (1956 – 1963)

In second generation computers, vacuum tubes were replaced by transistors. It was developed at Bell Laboratories by John Bardeen, Walter Brattain and William Shockley in 1947. Replacing vacuum tubes with transistors, allowed computers to become smaller and more powerful and faster. They also became less expensive, required less electricity and emitted less heat. The manufacturing cost was also less.



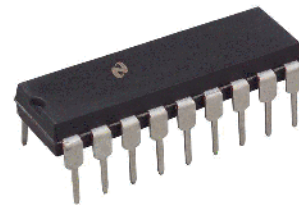
*Fig. 1.15 : IBM 1401 and Transistors*

It is in the second generation that the concept of programming language was developed. This generation used magnetic core memory and magnetic disk memory for primary and secondary storage respectively. Second-generation computers moved from cryptic binary machine language to symbolic or assembly languages. During second generation, many high level programming languages like FORTRAN and COBOL were introduced that allowed programmers to specify instructions in English like words. IBM 1401 and IBM 1620 are popular computers in this generation.



### 1.2.3 Third generation computers (1964 – 1971)

Third generation computers are smaller in size due to the use of integrated circuits (IC's). IC's or silicon chips that contained miniaturised transistors were developed by Jack Kilby, an engineer with Texas Instruments. IC drastically reduced the size and increased the speed and efficiency of computing. Multilayered printed circuits were developed and core memory was replaced by faster, solid state memories with large capacity.

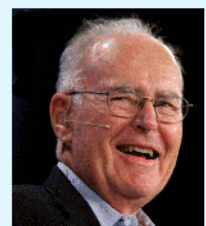


*Fig. 1.16 : IBM 360 and Integrated Circuit*

This generation of computers had better processing speed, consumed less power and was less costly. Integrated circuits, improved secondary storage devices and new input/output devices like keyboards and monitors were introduced. Arithmetic and logical operations were performed in microseconds or even nanoseconds. These computers could run many different programs at one time with a central program that monitored the memory. The high level language BASIC which made programming easy was developed during this period. Computers for the first time became accessible to a mass audience because they were smaller and cheaper than their predecessors. Some computers in this generation are IBM 360 and IBM 370.



Moore's Law states that the number of transistors on integrated circuits doubles approximately every two years. The law is named after Gordon E Moore, who described the trend in 1965. He noted that the number of components in IC had doubled every year from 1958 to 1965. He predicted that the trend would continue 'for at least ten years'. His prediction has proven to be accurate. Although this trend continued for more than half a century, Moore's Law is considered only as an observation and not a physical or natural law.





### 1.2.4 Fourth generation computers (1971 onwards)

The computers that we use today belong to this generation. These computers use microprocessors and are called microcomputers. Microprocessor is a single chip which contains Large Scale of Integration (LSI) of electronic components like transistors, capacitors, resistors, etc. Due to the development of microprocessor, it is possible to place computer's Central Processing Unit (CPU) on a single chip. Because of microprocessors, the fourth generation includes more data processing capacity than third generation computers. Later LSI circuits were replaced by Very Large Scale Integrated (VLSI) circuits which further increased the scale of integration. The fourth generation computers are smaller in size and have faster accessing and processing speeds.

The computer which occupied a very large room in earlier days could now fit in a palm. These computers were interconnected to form computer networks, which eventually led to the development of the Internet. As computers became less costly and more user-friendly, large number of people began buying them for personal use. Some computers in this generation are IBM PC and Apple II.



Fig. 1.17 : VLSI Chip

### 1.2.5 Fifth generation computers (future)

Fifth generation computers are based on Artificial Intelligence (AI). AI is the ability to simulate human intelligence. Such intelligent systems are still in the development stage, though there are some applications, such as speech recognition, face recognition and robotic vision and movement that are already available.

AI is the branch of computer science concerned with developing computer programs (intelligent systems) for solving complex problems (which are normally done by human beings without any effort) by the application of process that are analogues to human reasoning process. The two most common AI programming languages are LISP and Prolog. The fifth-generation computing also aims at developing computing machines that respond to natural language input and are capable of learning and self-organisation.

Table 1.1 shows comparative features of five generations of computers.





The first processor Intel 4004 was developed in 1971 by Intel Corporation and consisted of 2,300 transistors integrated into a single IC. Some popular microprocessors and the number of transistors integrated in them are given below.

Processor	Transistor Count
Intel 8086	29,000
Motorola 68000 (used in Apple)	68,000
Intel Pentium	31,00,000
AMD K7	2,20,00,000
Core i7	73,10,00,000

Criteria	Generation				
	First	Second	Third	Fourth	Fifth
<b>Technology</b>	Vacuum Tube	Transistor	Integrated Circuit	Microprocessor	Artificial Intelligence
<b>Operating System</b>	None	None	Yes	Yes	Yes
<b>Language</b>	Machine	Assembly	High Level	High Level	High Level
<b>Period</b>	1940-1956	1956-1963	1964-1971	1971-Present	Present and Yet to come

*Table 1.1 : Comparative features of five generations of computers*

### Check yourself



1. UNIVAC belongs to \_\_\_\_\_ generation..
2. What is meant by stored program concept?
3. Say True or False "Computers can understand only machine languages".
4. First generation computers are characterised by the use of \_\_\_\_\_.
5. What is the major technological advancement in the fourth generation computers?



### 1.3 Evolution of computing

Computing machines are used for processing, storing, and displaying information. The processing is done according to the instructions given to it. Early computers built during 1940's were only capable of performing series of single tasks, like a calculator. They were special-purpose systems programmed by rows of mechanical switches or by jumper wires on plug. The implementation of branching/looping statements, subroutine calls, etc. was not possible or was difficult. Later computers solved this problem by implementing John Von Neumann's revolutionary innovation the 'Stored Program Concept', which suggested storing data and programs in memory. The set of detailed instructions given to computer for executing a specific task is called a program.

#### Agusta Ada Lowelace

Augusta Ada King, Countess of Lowelace commonly known as Ada Lowelace, was an English mathematician and writer known for her work on Charles Babbage's early mechanical general-purpose computer, the Analytical Engine. Her notes on the engine included the first algorithm intended to be carried out by a machine. Because of this, she is often described as the world's first computer programmer.



*Fig.1.18 : Agusta Ada Lowelace (1815 - 1852)*

#### 1.3.1 Programming languages

A programming language is an artificial language designed to communicate instructions to a computer. Programming languages can be used to create programs that control the behavior of a machine and/or to express algorithms.

The first programming language developed for use in computers was called machine language. Machine language consisted of strings of the binary digits 0 and 1. Introduction of this language improved the overall speed and efficiency of the programming process. This language had many drawbacks like difficulty in finding and rectifying programming errors and its machine dependency. The programmer also needed to have a good knowledge of the computer architecture.

To make programming easier, a new language with instructions consisting of English-like words instead of 0's and 1's, was developed. This language was called assembly language. Electronic Delay Storage Automatic Calculator (EDSAC) built during 1949 was the first to use assembly language. Although this made writing programs easier, it had limitations. It is specific to a given machine and the programs written in this language are not transferable from one machine to another.



This led to the development of new languages called high level languages which are machine independent and which used simple English-like words and statements. It allowed people having less knowledge of computer architecture to develop easy-to-understand programs. A-0 programming language developed by Rear Admiral Dr. Grace Hopper, in 1952, for UNIVAC-I computer is the first language of this type. FORTRAN developed by the team led by John Backus at IBM for IBM 704 machine and 'Lisp' developed by Tim Hart and Mike Levin at MIT are other examples.



*Fig. 1.19 : Dr. Grace Hopper (1906-1992)*

### 1.3.2 Algorithm and computer programs

A programmer cannot write the instruction to be followed by a computer, unless he/she knows how to solve the problem manually. In order to ensure that the program instructions are appropriate for solving the given problem, and are in the correct sequence, program instructions are to be planned before they are written. An effective tool for planning a computer program is an algorithm. An algorithm provides a step by step solution for a given problem. These steps can then be converted to machine instructions using a programming language.

### 1.3.3 Theory of computing

The theory of computation is the branch that deals with how efficiently problems can be solved based on computation models and related algorithms. In order to perform a rigorous study of computation, computer scientists work with a mathematical abstraction of computers called a model of computation. There are several models in use, but the most commonly examined is the Turing Machine named after the famous computer scientist Alan Turing.

#### a. Contribution of Alan Turing

Alan M. Turing (1912 - 1954) was a British mathematician, logician, cryptographer and computer scientist. He made significant contributions to the development of computer science, by presenting the concepts of algorithm and computing with the help of his invention the Turing Machine, which is considered as a theoretical model of a general purpose computer. In 1950's, Alan Turing proposed to consider the question, 'Can machines think?' and later it turns out to be the foundation for the studies related to the computing machinery and intelligence. Turing proposed an imitation game which is later modified to Turing test and it is considered to be the test for determining a machine's intelligence. Considering these contributions he is regarded as the Father of Modern Computer Science as well as Artificial Intelligence.



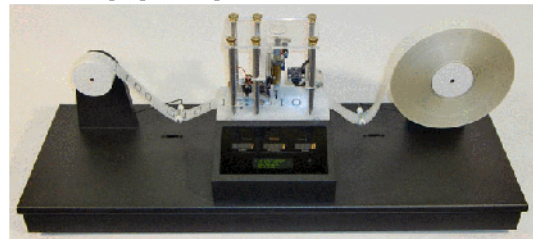
## b. Turing machine

Turing machine is a model of a computer proposed by Alan Turing in 1936. It is conceived as an ideal model of 'computing'. Alan Turing reasoned that any computation that could be performed by a human involved writing down intermediate results, reading them back and carrying out actions that depend only on what has been read and the current state of things. Turing machine is a theoretical computing device that could print symbols on a paper tape in a manner that emulate a person following a series of logical instructions.

A Turing machine consists of an infinitely-long tape which acts like the memory in a computer. The cells on the tape are usually blank at the start and can be written with symbols. In this case, each cell can contain the symbols '0', '1' and ' ' (blank), and is thus said to be a 3-symbol Turing machine (refer Figure 1.22). At each step, the machine can read the symbol on the cell under the head, edit the symbol and move the tape left or right by one square so that the machine can read and edit the symbol in the neighbouring cells.



*Fig. 1.20 : Alan Turing (1912 - 1954)*



*Fig. 1.21 : Turing machine*



*Fig. 1.22 : The head movement over tape*

Any particular Turing machine is defined by a rule which specifies what the head should do at each step. The action of a Turing machine is determined by (a) the current state of the machine (b) the symbol in the cell currently being scanned by the head and (c) a table of transition rules, which serve as the 'program' for the machine.

In modern terms, the tape serves as the memory of the machine, while the read-write head is the memory bus through which data is accessed and updated by the machine. The initial arrangement of symbols of cells on the tape corresponds to the input given to the computer. The steps of the Turing machine correspond to the running of the computer. For a given input, each part of the rule specifies what 'operation' the machine should perform. Even though Turing machines are equivalent to modern electronic computers at a certain theoretical level, they differ in many details.



### The Turing test

Alan Turing introduced the 'Turing test' in his paper titled 'Computing Machinery and Intelligence'. The 'Turing Test' involves a human interrogator and two contestants - a computer and a human. The interrogator converses with these contestants via computer terminals, without knowing the identity of the contestants. After a sufficiently long period of conversation, if the interrogator is unable to identify the computer, then the computer is said to have passed 'Turing test' and must be considered intelligent. Turing predicted that by 2000, computers would pass the test. There have been various programs that have demonstrated some amount of human like behaviour, but no computer has this far passed the Turing test.



### Let us sum up

In this chapter, we briefly sketched the evolution of the number system and counting. We went through the development of computing machines and described the structure of the modern computing system. The evolution of computing was also discussed along with the different types of programming languages. The concepts of algorithms and computer programs were also discussed. The detailed description of generations of computers was also seen. Finally we discussed the theory of computation, in which the contributions of Alan Turing and the concept of Turing Machine were outlined.



### Learning outcomes

After the completion of this chapter the learner will be able to

- categorize the basic concept of computing milestones in history.
- sketch the modern computing machine.





- recognise the impact of John Von Neumann's Architecture in today's world.
- identify the pioneers of Computer Science.
- list the characteristics of computers in each generation.
- explain the contribution of Alan Turing and the concept of Turing Machine.

### Sample questions

#### Very short answer type

1. Which is the base of Mayan's Number System?
2. Greek Number System is known as \_\_\_\_\_.
3. Which was the first computer for basic arithmetic calculations?
4. Who invented logarithms?
5. What is the name of the machine developed by Blaise Pascal?
6. Who was the first programmer in the world?
7. Computing machines recognizes and operates in \_\_\_\_\_ language.
8. What does EDVAC stand for?
9. Give the name for a simple kind of theoretical computing machine.

#### Short answer type

1. Discuss the developments of the number system from the Egyptian to the Chinese Era.
2. Discuss the impact of Hindu-Arabic numeral system in the world.
3. Compare the Roman Number system and Mayan's Number System.
4. Discuss the features of Abacus.
5. Compare the Analytical Engine and Difference Engine of Charles Babbage.
6. Bring out the significance of Hollerith's machine.
7. What are the developments in computing machines that took place during the Second World War?
8. Discuss the evolution of computer languages.
9. Discuss the working of Turing Machine.

#### Long answer type

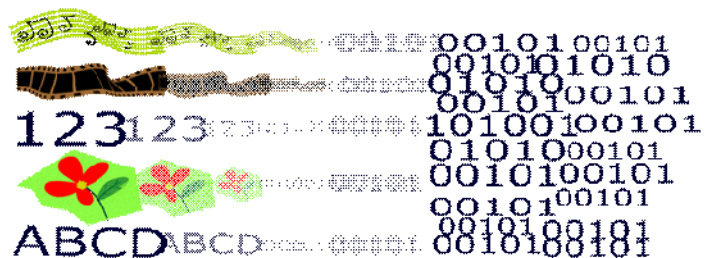
1. List out and explain the various generations of Computers.
2. Prepare a seminar report on evolution of positional number system.
3. Discuss the various computing machines emerged till 1900's.

## Key concepts

- **Number systems**
  - Decimal, binary, octal, hexadecimal
- **Number conversions**
- **Binary arithmetic**
  - Addition, subtraction
- **Data representation**
  - Representation of integers and floating point numbers
  - Character representation: ASCII, EBCDIC, ISCII, UNICODE
  - Representation of audio, image and video data
- **Introduction to Boolean algebra**
  - Logic operators and logic gates
- **Basic postulates**
- **Basic theorems**
- **Circuit designing for simple Boolean expressions**
- **Universal gates**

## Data Representation and Boolean Algebra

Computer is a machine that can handle different types of data items. We feed data such as numbers, characters, images, videos and sounds to a computer for processing. We know that computer is an electronic device functioning on the basis of two electric states - ON and OFF. All electronic circuits have two states - open and closed. The two-state operation is called binary operation. Hence, the data given to computer should also be in binary form. In this chapter we will discuss various methods for representing data such as numbers, characters, images, videos and sounds.



*Fig. 2.1: External and internal form of data*

**Data representation** is the method used internally to represent data in a computer.

Before discussing data representation of numbers, let us see what a number system is.





## 2.1 Number systems

A number is a mathematical object used to count, label and measure. A number system is a systematic way to represent numbers. The number system we use in our day to day life is the decimal number system that uses 10 symbols or digits. The number 289 is pronounced as two hundred and eighty nine and it consists of the symbols 2, 8 and 9. Similarly there are other number systems. Each has its own symbols and method for constructing a number. A number system has a unique base, which depends upon the number of symbols. The number of symbols used in a number system is called **base** or **radix** of a number system.

Let us discuss some of the number systems.

### 2.1.1 Decimal number system

The decimal number system involves ten symbols 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 to form a number. Since there are ten symbols in this number system, its base is 10. Therefore, the decimal number system is also known as base-10 number system.

Consider two decimal numbers 743 and 347

743  $\rightarrow$  seven hundred + four tens + three ones ( $7 \times 10^2 + 4 \times 10^1 + 3 \times 10^0$ )

347  $\rightarrow$  three hundreds + four tens + seven ones ( $3 \times 10^2 + 4 \times 10^1 + 7 \times 10^0$ )

Here, place value (weight) of 7 in first number 743 is  $10^2=100$ . But weight of 7 in second number 347 is  $10^0=1$ . The weight of a digit depends on its relative position. Such a number system is known as **positional number system**. All positional number systems have a base and the place value of a digit is some power of this base.

Place value of each decimal digit is power of 10 ( $10^0, 10^1, 10^2, \dots$ ). Consider a decimal number 5876.

This number can be written in expanded form as

Weight	$10^3$	$10^2$	$10^1$	$10^0$
Decimal Number	5	8	7	6

$$= 5 \times 10^3 + 8 \times 10^2 + 7 \times 10^1 + 6 \times 10^0$$

$$= 5 \times 1000 + 8 \times 100 + 7 \times 10 + 6 \times 1$$

$$= 5000 + 800 + 70 + 6$$

$$= 5876$$

In the above example, the digit 5 has the maximum place value,  $10^3=1000$  and 6 has the minimum place value,  $10^0=1$ . The digit with most weight is called Most Significant





Digit (MSD) and the digit with least weight is called Least Significant Digit (LSD). So in the above number MSD is 5 and LSD is 6.

**Left most digit of a number is MSD and right most digit of a number is LSD**

For fractional numbers weights are negative powers of 10 ( $10^{-1}$ ,  $10^{-2}$ ,  $10^{-3}$ , ...) for the digits to the right of decimal point. Consider another example 249.367

Weight	$10^2$	$10^1$	$10^0$	$10^{-1}$	$10^{-2}$	$10^{-3}$
Decimal Number	2	4	9	3	6	7

MSD

()

LSD

$$\begin{aligned}
 &= 2 \times 10^2 + 4 \times 10^1 + 9 \times 10^0 + 3 \times 10^{-1} + 6 \times 10^{-2} + 7 \times 10^{-3} \\
 &= 2 \times 100 + 4 \times 10 + 9 \times 1 + 3 \times 0.1 + 6 \times 0.01 + 7 \times 0.001 \\
 &= 200 + 40 + 9 + 0.3 + 0.06 + 0.007 \\
 &= 249.367
 \end{aligned}$$

So far we have discussed a number system which uses 10 symbols. Now let us see the construction of other number systems with different bases.

### 2.1.2 Binary number system

A number system which uses only two symbols 0 and 1 to form a number is called binary number system. Bi means two. Base of this number system is 2. So it is also called base-2 number system. We use the subscript 2 to indicate that the number is in binary.

e.g.  $(1101)_2$ ,  $(101010)_2$ ,  $(1101.11)_2$

Each digit of a binary number is called bit. A **bit** stands for **binary digit**.

The binary number system is also a positional number system where place value of each binary digit is power of 2. Consider an example  $(1101)_2$ . This binary number can be written in expanded form as shown below:

Weight	$2^3$	$2^2$	$2^1$	$2^0$
Binary Number	1	1	0	1

MSB

LSB

$$\begin{aligned}
 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\
 &= 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 \\
 &= 8 + 4 + 0 + 1 \\
 &= 13
 \end{aligned}$$

The right most bit in a binary number is called Least significant Bit (**LSB**). The leftmost bit in a binary number is called Most significant Bit (**MSB**).

The binary number 1101 is equivalent to the decimal number 13. The number 1101 also exists in the decimal number system. But it is interpreted as one thousand one hundred and one. To avoid this confusion, base must be specified in all number systems other than decimal number system. The general format is

$$(\text{Number})_{\text{base}}$$

This notation helps to differentiate numbers of different bases. So a binary number must be represented with base 2 as  $(1101)_2$  and it is read as “one one zero one to the base two”.

If no base is given in a number, it will be considered as decimal. In other words, specifying the base is not compulsory in decimal number.

For fractional numbers, weights are negative powers of 2 ( $2^1, 2^2, 2^3, \dots$ ) for the digits to the right of the binary point. Consider an example  $(111.011)_2$

Weight	$2^2$	$2^1$	$2^0$	$2^{-1}$	$2^{-2}$	$2^{-3}$
Binary Number	1	1	1	0	1	1
	MSB		(.)			LSB

$$= 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$$

$$= 1 \times 4 + 1 \times 2 + 1 \times 1 + 0 \times \frac{1}{2} + 1 \times \frac{1}{4} + 1 \times \frac{1}{8}$$

$$= 4 + 2 + 1 + 0 + 0.25 + 0.125$$

$$= 7.375$$

### Importance of binary numbers in computers

We have seen that binary number system is based on two digits 1 and 0. The electric state ON can be represented by 1 and the OFF state by 0 as in Figure 2.2. Because of this, computer uses binary number system as the basic number system for data representation.



Fig. 2.2 : Digital representation of ON and OFF

#### 2.1.3 Octal number system

A number system which uses eight symbols 0, 1, 2, 3, 4, 5, 6 and 7 to form a number is called octal number system. Octa means eight, hence this number system is called



octal. Base of this number system is 8 and hence it is also called base-8 number system. Consider an example  $(236)_8$ . Weight of each digit is power of 8 ( $8^0, 8^1, 8^2, 8^3, \dots$ ). The number  $(236)_8$  can be written in expanded form as

Weight	$8^2$	$8^1$	$8^0$
Octal Number	2	3	6

$$\begin{aligned}
 &= 2 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 \\
 &= 2 \times 64 + 3 \times 8 + 6 \times 1 \\
 &= 128 + 24 + 6 \\
 &= 158
 \end{aligned}$$

For fractional numbers weights are negative powers of 8, i.e. ( $8^{-1}, 8^{-2}, 8^{-3}, \dots$ ) for the digits to the right of the octal point. Consider an example  $(172.4)_8$

Weight	$8^2$	$8^1$	$8^0$	$8^{-1}$
Octal Number	1	7	2	4

$$\begin{aligned}
 &= 1 \times 8^2 + 7 \times 8^1 + 2 \times 8^0 + 4 \times 8^{-1} \\
 &= 64 + 56 + 2 + 4 \times \frac{1}{8} \\
 &= 122 + 0.5 \\
 &= 122.5
 \end{aligned}$$

### 2.1.4 Hexadecimal number system

A number system which uses 16 symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F) to form a number is called hexadecimal number system. Base of this number system is 16 as there are sixteen symbols in this number system. Hence this number system is also called base-16 number system.

In this system, the symbols A, B, C, D, E and F are used to represent the decimal numbers 10, 11, 12, 13, 14 and 15 respectively. The hexadecimal digit and their equivalent decimal numbers are shown below.

<b>Hexadecimal</b>	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
<b>Decimal</b>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Consider a hexadecimal number  $(12AB)_{16}$ . Weights of each digit is power of 16 ( $16^0, 16^1, 16^2, \dots$ ).



This number can be written in expanded form as shown below:

Weight	$16^3$	$16^2$	$16^1$	$16^0$
Hexadecimal Number	1	2	A	F

$$\begin{aligned}
 &= 1 \times 16^3 + 2 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 \\
 &= 1 \times 4096 + 2 \times 256 + 10 \times 16 + 15 \times 1 \\
 &= 4096 + 512 + 160 + 15 \\
 &= 4783
 \end{aligned}$$

For fractional numbers, weights are some negative power of 16 ( $16^{-1}$ ,  $16^{-2}$ ,  $16^{-3}$ , ...) for the digits to the right of the hexadecimal point. Consider an example  $(2D.4)_{16}$

Weight	$16^1$	$16^0$	$16^{-1}$
Hexadecimal	2	D	4

$$\begin{aligned}
 &= 2 \times 16^1 + 13 \times 16^0 + 4 \times \frac{1}{16} \\
 &= 32 + 13 + 0.25 \\
 &= 45.25
 \end{aligned}$$

Table 2.1 shows the base and symbols used in different number systems:

Number System	Base	Symbols used
Binary	2	0, 1
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

*Table 2.1 : Number systems with base and symbols*

### Importance of octal and hexadecimal number systems

As we have discussed, digital hardware uses the binary number system for its operations and data. Representing numbers and operations in binary form requires too many bits and needs lot of effort. With octal, the bits are grouped in threes (because  $2^3 = 8$ ) and with hexadecimal, the bits are grouped in four (because  $2^4 = 16$ ) and these groups are replaced with the respective octal or hexadecimal symbol. This conversion processes of binary numbers to octal and hexadecimal number systems and vice versa are very easy. This short-hand notation is widely used in the design and operations of electronic circuits.

**Check yourself**

1. Number of symbols used in a number system is called \_\_\_\_.
2. Pick invalid numbers from the following  
i)  $(10101)_8$     ii)  $(123)_4$     iii)  $(768)_8$     iv)  $(ABC)_{16}$
3. Define the term 'bit'.
4. Find MSD in the decimal number 7854.25.
5. The base of hexadecimal number system is \_\_\_\_\_.

**2.2 Number conversions**

After having learnt the various number systems, let us now discuss how to convert the numbers of one base to the equivalent numbers in other bases. There are different types of number conversions like decimal to binary, binary to decimal, decimal to octal etc. This section discusses how to convert one number system to another.

**2.2.1 Decimal to binary conversion**

The method of converting decimal number to binary number is by repeated division. In this method the decimal number is successively divided by 2 and the remainders are recorded. The binary equivalent is obtained by grouping all the remainders, with the last remainder being the Most Significant Bit (MSB) and first remainder being the Least Significant Bit (LSB). In all these cases the remainders will be either 0 or 1 (binary digit).

**Examples:**

Find binary equivalent of decimal number 25.

		Remainders	
2	25		
2	12	1	↑ LSB
2	6	0	
2	3	0	
2	1	1	
	0	1	MSB

$$(25)_{10} = (11001)_2$$

Find binary equivalent of  $(80)_{10}$ .

		Remainders	
2	80		
2	40	0	↑ LSB
2	20	0	
2	10	0	
2	5	0	
2	2	1	
2	1	0	
	0	1	MSB

$$(80)_{10} = (1010000)_2$$

**Hint:** Binary equivalent of an odd decimal number ends with 1 and binary of even decimal number ends with zero.

### Converting decimal fraction to binary

To convert a fractional decimal number to binary, we use the method of repeated multiplication by 2. At first the decimal fraction is multiplied by 2. The integer part of the answer will be the MSB of binary fraction. Again the fractional part of the answer is multiplied by 2 to obtain the next significant bit of binary fraction. The procedure is continued till the fractional part of product is zero or a desired precision is obtained.

**Example:** Convert 0.75 to binary.

	0.75 × 2 = 1.50
1	.50 × 2 = 1.00
1	.00

$$(0.75)_{10} = (0.11)_2$$

**Example:** Convert 0.625 to binary.

	0.625 × 2 = 1.25
1	.25 × 2 = 0.50
0	.50 × 2 = 1.00
1	.00

$$(0.625)_{10} = (0.101)_2$$

**Example:** Convert 15.25 to binary.

Convert 15 to binary

2	15	Remainders
2	7	1
2	3	1
2	1	1
	0	1

Convert 0.25 to binary

	0.25 × 2 = 0.50
0	.50 × 2 = 1.00
1	.00

$$(15.25)_{10} = (1111.01)_2$$

### 2.2.2 Decimal to octal conversion

The method of converting decimal number to octal number is also by repeated division. In this method the number is successively divided by 8 and the remainders

are recorded. The octal equivalent is obtained by grouping all the remainders, with the last remainder being the MSD and first remainder being the LSD. Remainders will be 0, 1, 2, 3, 4, 5, 6 or 7.

**Example:** Find octal equivalent of decimal number 125.

8	125	Remainders	
8	15	5	↑ LSD
8	1	7	
	0	1	MSD

$$(125)_{10} = (175)_8$$

**Example:** Find octal equivalent of  $(400)_{10}$ .

8	400	Remainders	
8	50	0	↑
8	6	2	
	0	6	

$$(400)_{10} = (620)_8$$

### 2.2.3 Decimal to hexadecimal conversion

The method of converting decimal number to hexadecimal number is also by repeated division. In this method, the number is successively divided by 16 and the remainders are recorded. The hexadecimal equivalent is obtained by grouping all the remainders, with the last remainder being the Most Significant Digit (MSD) and first remainder being the Least Significant Digit (LSD). Remainders will be 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E or F.

**Example:** Find hexadecimal equivalent of decimal number 155.

16	155	Remainders	
16	9	11 (B)	↑ LSD
	0	9	MSD

$$(155)_{10} = (9B)_{16}$$

**Example:** Find hexadecimal equivalent of 380.

16	380	Remainders	
16	23	12 (C)	↑
16	1	7	
	0	1	

$$(380)_{10} = (17C)_{16}$$



### 2.2.4 Binary to decimal conversion

A binary number can be converted into its decimal equivalent by summing up the product of each bit and its weight. Weights are some power of 2 ( $2^0, 2^1, 2^2, 2^3, \dots$ ).

**Example:** Convert  $(11011)_2$  to decimal.

$$\begin{aligned}(11011)_2 &= 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 16 + 8 + 2 + 1 \\ &= 27\end{aligned}$$

Weight	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
Bit	1	1	0	1	1

$$(11011)_2 = (27)_{10}$$

**Example:** Convert  $(1100010)_2$  to decimal.

Weight	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
Bit	1	1	0	0	0	1	0

$$\begin{aligned}(1100010)_2 &= 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &= 64 + 32 + 2 \\ &= 98\end{aligned}$$

$$(1100010)_2 = (98)_{10}$$

Table 2.2 may help us to find powers of 2.

$2^{10}$	$2^9$	$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
1024	512	256	128	64	32	16	8	4	2	1

Table 2.2 : Powers of 2

### Converting binary fraction to decimal

A binary fraction number can be converted into its decimal equivalent by summing up the product of each bit and its weight. Weights of binary fractions are negative powers of 2 ( $2^{-1}, 2^{-2}, 2^{-3}, \dots$ ) for the digits after the binary point.

**Example:** Convert  $(0.101)_2$  to decimal.

$$\begin{aligned}(0.101)_2 &= 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 0.5 + 0 + 0.125 \\ &= 0.625\end{aligned}$$

Weight	$2^{-1}$	$2^{-2}$	$2^{-3}$
Bit	1	0	1

$$(0.101)_2 = (0.625)_{10}$$

**Example:** Convert  $(1010.11)_2$  to decimal.

$$\begin{aligned}(1010)_2 &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &= 8 + 0 + 2 + 0 \\ &= 10\end{aligned}$$

$$(1010)_2 = (10)_{10}$$

Weight	$2^3$	$2^2$	$2^1$	$2^0$
Bit	1	0	1	0



$$\begin{aligned}
 (0.11)_2 &= 1 \times 2^{-1} + 1 \times 2^{-2} \\
 &= 0.5 + 0.25 \\
 &= 0.75
 \end{aligned}$$

$$(0.11)_2 = (0.75)_{10}$$

Weight	$2^{-1}$	$2^{-2}$
Bit	1	1

$$(1010.11)_2 = (10.75)_{10}$$

Table 2.3 shows some negative powers of 2.

$2^{-1}$	$2^{-2}$	$2^{-3}$	$2^{-4}$	$2^{-5}$
0.5	0.25	0.125	0.0625	0.03125

Table 2.3 : Negative powers of 2

### 2.2.5 Octal to decimal conversion

An octal number can be converted into its decimal equivalent by summing up the product of each octal digit and its weight. Weights are some powers of 8 ( $8^0$ ,  $8^1$ ,  $8^2$ ,  $8^3$ , ...).

**Example:** Convert  $(157)_8$  to decimal.

$$\begin{aligned}
 (157)_8 &= 1 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 \\
 &= 64 + 40 + 7 \\
 &= 111
 \end{aligned}$$

Weight	$8^2$	$8^1$	$8^0$
Octal digit	1	5	7

$$(157)_8 = (111)_{10}$$

**Example:** Convert  $(1005)_8$  to decimal.

$$\begin{aligned}
 (1005)_8 &= 1 \times 8^3 + 0 \times 8^2 + 0 \times 8^1 + 5 \times 8^0 \\
 &= 512 + 5 \\
 &= 517
 \end{aligned}$$

Weight	$8^3$	$8^2$	$8^1$	$8^0$
Octal digit	1	0	0	5

$$(1005)_8 = (517)_{10}$$

### 2.2.6 Hexadecimal to decimal conversion

An hexadecimal number can be converted into its decimal equivalent by summing up the product of each hexadecimal digit and its weight. Weights are powers of 16 ( $16^0$ ,  $16^1$ ,  $16^2$ , ...).

**Example:** Convert  $(AB)_{16}$  to decimal.

$$\begin{aligned}
 (AB)_{16} &= 10 \times 16^1 + 11 \times 16^0 \\
 &= 160 + 11 \\
 &= 171
 \end{aligned}$$

Weight	$16^1$	$16^0$
Hexadecimal digit	A	B

$$A = 10 \quad B = 11$$

$$(AB)_{16} = (171)_{10}$$

**Example:** Convert  $(2D5)_{16}$  to decimal.

$$\begin{aligned}(2D5)_{16} &= 2 \times 16^2 + 13 \times 16^1 + 5 \times 16^0 \\ &= 512 + 208 + 5 \\ &= 725\end{aligned}$$

Weight	$16^2$	$16^1$	$16^0$
Hexadecimal digit	2	D	5

$$D = 13$$

$$(2D5)_{16} = (725)_{10}$$

## 2.2.7 Octal to binary conversion

An octal number can be converted into binary by converting each octal digit to its 3 bit binary equivalent. Eight possible octal digits and their binary equivalents are listed in Table 2.4.

Octal Digit	0	1	2	3	4	5	6	7
Binary Equivalent	000	001	010	011	100	101	110	111

Table 2.4 : Binary equivalent of octal digit

**Example:** Convert  $(437)_8$  to binary.

3-bit binary equivalents of each octal digit are

$$\begin{array}{ccc} 4 & 3 & 7 \\ \downarrow & \downarrow & \downarrow \\ 100 & 011 & 111 \end{array}$$

$$(437)_8 = (100011111)_2$$

**Example:** Convert  $(7201)_8$  to binary.

3-bit binary equivalents of each octal digits are

$$\begin{array}{cccc} 7 & 2 & 0 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 111 & 010 & 000 & 001 \end{array}$$

$$(7201)_8 = (111010000001)_2$$

## 2.2.8 Hexadecimal to binary conversion

A hexadecimal number can be converted into binary by converting each hexadecimal digit to its 4 bit binary equivalent. Sixteen possible hexadecimal digits and their binary equivalents are listed in Table 2.5.

**Example:** Convert  $(AB)_{16}$  to binary.

4-bit binary equivalents of each hexadecimal digit are

$$\begin{array}{cc}
 A & B \\
 \downarrow & \downarrow \\
 1010 & 1011 \\
 (AB)_{16} = (10101011)_2
 \end{array}$$

**Example:** Convert  $(2F15)_{16}$  to binary.

4-bit binary equivalents of each hexadecimal digit are

$$\begin{array}{cccc}
 2 & F & 1 & 5 \\
 \downarrow & \downarrow & \downarrow & \downarrow \\
 0010 & 1111 & 0001 & 0101 \\
 (2F15)_{16} = (10111100010101)_2
 \end{array}$$

### 2.2.9 Binary to octal conversion

A binary number can be converted into its octal equivalent by grouping binary digits to group of 3 bits and then each group is converted to its octal equivalent. Start grouping from right to left.

**Example:** Convert  $(101100111)_2$  to octal.

We can group above binary number 101100111 from right as shown below.

$$\begin{array}{ccc}
 101 & 100 & 111 \\
 \downarrow & \downarrow & \downarrow \\
 5 & 4 & 7 \\
 (101100111)_2 = (547)_8
 \end{array}$$

**Example:** Convert  $(10011000011)_2$  to octal.

We can group above binary number 10011000011 from right as shown below.

$$\begin{array}{cccc}
 010 & 011 & 000 & 011 \\
 \downarrow & \downarrow & \downarrow & \downarrow \\
 2 & 3 & 0 & 3 \\
 (10011000011)_2 = (2303)_8
 \end{array}$$

After grouping, if the left most group has no 3 bits, then add leading zeros to form 3 bit binary.

Hexa decimal	Binary equivalent
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Table 2.5 :  
Binary equivalent of  
hexadecimal digits



### 2.2.10 Binary to hexadecimal conversion

A binary number can be converted into its hexadecimal equivalent by grouping binary digits to group of 4 bits and then each group is converted to its hexadecimal equivalent. Start grouping from right to left.

**Example:** Convert  $(101100111010)_2$  to hexadecimal.

We can group the given binary number 101100111010 from right as shown below:

$$\begin{array}{ccc} 1011 & 0011 & 1010 \\ \downarrow & \downarrow & \downarrow \\ B & 3 & A \end{array}$$

$$(101100111010)_2 = (B3A)_{16}$$

**Example:** Convert  $(110111100001100)_2$  to hexadecimal.

We can group the given binary number 110111100001100 from right as shown below:

After grouping, if the left most group has no 4 bits, then add leading zeros to form 4 bit binary.

$$\begin{array}{cccc} 0110 & 1111 & 0000 & 1100 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 6 & F & 0 & C \end{array}$$

$$(110111100001100)_2 = (6F0C)_{16}$$

### 2.2.11 Octal to hexadecimal conversion

Conversion of an octal number to hexadecimal number is a two step process. Octal number is first converted into binary. This binary equivalent is then converted into hexadecimal.

**Example:** Convert  $(457)_8$  to hexadecimal equivalent.

First convert  $(457)_8$  into binary.

$$\begin{array}{ccc} (457)_8 = & 4 & 5 & 7 \\ & \downarrow & \downarrow & \downarrow \\ & 100 & 101 & 111 \\ & & & \\ & = (100101111)_2 \end{array}$$

Then convert  $(100101111)_2$  into hexadecimal as follows:

$$\begin{array}{ccc} (100101111)_2 = & 0001 & 0010 & 1111 \\ & \downarrow & \downarrow & \downarrow \\ & 1 & 2 & F \\ & = (12F)_{16} \end{array}$$

$$(457)_8 = (12F)_{16}$$



### 2.2.12 Hexadecimal to octal conversion

Conversion of an hexadecimal to octal number is also a two step process. Hexadecimal number is first converted into binary. This binary equivalent is then converted into octal.

**Example:** Convert  $(A2D)_{16}$  into octal equivalent.

First convert  $(A2D)_{16}$  into binary.

$$\begin{aligned}
 (A2D)_{16} &= \begin{array}{ccc} A & 2 & D \\ \downarrow & \downarrow & \downarrow \\ 1010 & 0010 & 1101 \end{array} \\
 &= (101000101101)_2
 \end{aligned}$$

Then convert  $(101000101101)_2$  into octal as follows:

$$\begin{aligned}
 (101000101101)_2 &= \begin{array}{cccc} 101 & 000 & 101 & 101 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 5 & 0 & 5 & 5 \end{array} \\
 &= (5055)_8
 \end{aligned}$$

$$(A2D)_{16} = (5055)_8$$

Table 2.6 shows procedures for various number conversions.

Conversion	Procedure
Decimal to Binary	Repeated division by 2 and grouping the remainders
Decimal to Octal	Repeated division by 8 and grouping the remainders
Decimal to Hexadecimal	Repeated division by 16 and grouping the remainders
Binary to Decimal	Multiply binary digit by place value (power of 2) and find their sum
Octal to Decimal	Multiply octal digit by place value (power of 8) and find their sum
Hexadecimal to Decimal	Multiply hexadecimal digit by place value (power of 16) and find their sum
Octal to Binary	Converting each octal digit to its 3 bit binary equivalent
Hexadecimal to Binary	Converting each hexadecimal digit to its 4 bit binary equivalent
Binary to Octal	Grouping binary digits to group of 3 bits from right to left
Binary to Hexadecimal	Grouping binary digits to group of 4 bits from right to left
Octal to Hexadecimal	Convert octal to binary and then binary to hexadecimal
Hexadecimal to Octal	Convert hexadecimal to binary and then binary to octal

Table 2.6 : Procedure for number conversions

**Check yourself**

- 1 Convert the decimal number 31 to binary.
- 2 Find decimal equivalent of  $(10001)_2$
- 3 If  $(x)_8 = (101011)_2$ , then find  $x$ .
- 4 Fill in the blanks:
  - a)  $(\quad)_2 = (AB)_{16}$
  - b)  $(\text{___D___})_{16} = (1010\text{___}1000)_2$
  - c)  $0.25_{10} = (\quad)_2$
- 5 Find the largest number in the list
  - (i)  $(1001)_2$  (ii)  $(A)_{16}$  (iii)  $(10)_8$  (iv)  $(11)_{10}$

**2.3 Binary arithmetic**

As in the case of decimal number system, arithmetic operations are performed in binary number system. When we give instruction to add two decimal numbers, the computer actually adds their binary equivalents. Let us see how binary addition and subtraction are carried out.

**2.3.1 Binary addition**

The rules for adding two bits are as follows:

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Note that a carry bit 1 is created only when two ones are added. If three ones are added (i.e.  $1+1+1$ ), then the sum bit is 1 with a carry bit 1.

**Example:** Find sum of binary numbers 1011 and 1001.

$$\begin{array}{r} 1011 + \\ 1001 \\ \hline 10100 \end{array}$$

**Example:** Find sum of binary numbers 110111 and 10011.

$$\begin{array}{r} 110111 + \\ 100110 \\ \hline 1011101 \end{array}$$





### 2.3.2 Binary subtraction

The rules for subtracting a binary digit from another digit are as follows.

A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Note that when 1 is subtracted from 0 the difference is 1, but 1 is borrowed from immediate left bit of first number. The above rules can be used only when a small binary number is subtracted from a large binary number.

**Example:** Subtract  $(10101)_2$  from  $(11111)_2$ .

$$\begin{array}{r} 11111 \\ - 10101 \\ \hline 01010 \end{array}$$

**Example:** Subtract  $(10111)_2$  from  $(101000)_2$ .

$$\begin{array}{r} 101000 \\ - 10111 \\ \hline 10001 \end{array}$$

## 2.4 Data representation

Computer uses a fixed number of bits to represent a piece of data which could be a number, a character, image, sound, video etc. Data representation is the method used internally to represent data in a computer. Let us see how various types of data can be represented in computer memory.

### 2.4.1 Representation of numbers

Numbers can be classified into integer numbers and floating point numbers. Integers are whole numbers or numbers without any fractional part. A floating point number or a real number is a number with fractional part. These two numbers are treated differently in computer memory. Let us see how integers are represented.

#### a. Representation of integers

There are three methods for representing an integer number in computer memory. They are

- Sign and magnitude representation
- 1's complement representation
- 2's complement representation

The following data representation methods are based on 8 bit word length.



A **word** is basically a fixed-sized group of bits that are handled as a unit by a processor. Number of bits in a word is called **word length**. The word length is the choice of computer designer and some popular word lengths are 8, 16, 32 and 64.

### i. Sign and magnitude representation

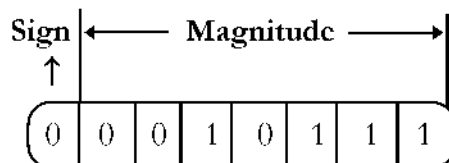
In this method, first bit from left (MSB) is used for representing sign of integer and remaining 7-bits are used for representing magnitude of integer. For negative integers sign bit is 1 and for positive integers sign bit is 0. Magnitude is represented as 7-bit binary equivalent of the integer.

**Example:** Represent + 23 in sign and magnitude form.

Number is positive, so first bit (MSB) is 0.

7 bit binary equivalent of  $23 = (0010111)_2$

So + 23 can be represented as  $(00010111)_2$

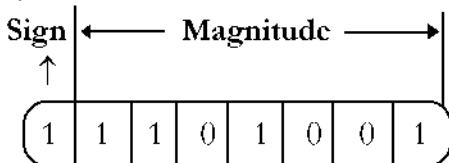


**Example:** Represent -105 in sign and magnitude form.

Number is negative, so first bit(MSB) is 1

7 bit binary equivalent of  $105 = (1101001)_2$

So -105 can be represented as  $(11101001)_2$



**Note:** In this method an 8 bit word can represent  $2^8-1=255$  numbers (i.e. -127 to +127). Similarly, a 16 bit word can represent  $2^{16}-1 = 65535$  numbers (i.e. -32767 to +32767). So, an  $n$ -bit word can represent  $2^n-1$  numbers i.e.,  $-(2^{n-1}-1)$  to  $+(2^{n-1}-1)$ . The integer 0 can be represented in two ways:  $+0 = 00000000$  and  $-0 = 10000000$ .

### ii. 1's complement representation

In this method, first find binary equivalent of absolute value of integer. If number of digits in binary equivalent is less than 8, provide zero(s) at the left to make it 8-bit form. 1's complement of a binary number is obtained by replacing every 0 with 1 and every 1 with 0. Some binary numbers and the corresponding 1's compliments are given below:

Binary Number	1's Complement
11001	00110
10101	01010

If the number is negative it is represented as 1's complement of 8-bit form binary. If the number is positive, the 8-bit form binary equivalent itself is the 1's complement representation.



**Example:** Represent -119 in 1's complement form.

$$\text{Binary of } 119 \text{ in 8-bit form} = (01110111)_2$$

$$-119 \text{ in 1's complement form} = (10001000)_2$$

**Example:** Represent +119 in 1's complement form.

$$\text{Binary of } 119 \text{ in 8-bit form} = (01110111)_2$$

$$+119 \text{ in 1's complement form} = (01110111)_2$$

*(No need to find 1's complement, since the number is positive)*

**Note:** In this representation if first bit (MSB) is 0 then number is positive and if MSB is 1 then number is negative. So 8 bit word can represent integers from -127 (represented as 10000000) to +127 (represented as 01111111). Here also integer 0 can be represented in two ways:  $+0 = 00000000$  and  $-0 = 11111111$ . An  $n$ -bit word can represent  $2^n - 1$  numbers i.e.  $-(2^{n-1} - 1)$  to  $+(2^{n-1} - 1)$ .

### iii. 2's complement representation

In this method, first find binary equivalent of absolute value of integer and write it in 8-bit form. If the number is negative, it is represented as 2's complement of 8-bit form binary. If the number is positive, 8-bit form binary itself is the representation. 2's complement of a binary number is calculated by adding 1 to its 1's complement.

For example, let us find the 2's complement of  $(10101)_2$ .

$$1's \text{ complement of } (10101)_2 = (01010)_2$$

$$\text{So } 2's \text{ complement of } (10101)_2 = 01010 +$$

$$1$$

$$= (01011)_2$$

**Example:** Represent -38 in 2's complement form.

$$\text{Binary of } 38 \text{ in 8-bit form} = (00100110)_2$$

$$-38 \text{ in 2's complement form} = 11011001 +$$

$$1$$

$$= (11011010)_2$$

**Example:** Represent +38 in 2's complement form.

$$\text{Binary of } 38 \text{ in 8-bit form} = (00100110)_2$$

$$+38 \text{ in 2's complement form} = (00100110)_2 \text{ (No need to find 2's complement)}$$

**Note:** In this representation if first bit (MSB) is 0 then number is positive and if MSB is 1 then number is negative. Here integer 0 has only one way of representation and is 00000000. So an 8 bit word can represent integers from -128 (represented as 10000000) to +127 (represented as 01111111). It is the most common integer representation. An  $n$ -bit word can represent  $2^n$  numbers  $-(2^{n-1})$  to  $+(2^{n-1} - 1)$ . Table 2.7 shows the comparison of different representation methods of integers in 8-bit word length.



Features	Sign & Magnitude	1's Complement	2's Complement	Remarks
<b>Range</b>	-127 to +127	-127 to +127	-128 to +127	Range is more in 2's complement
<b>Total Numbers</b>	255	255	256	
<b>Representation of integer 0</b>	Two ways of representation	Two ways of representation	Only one way of representation	In 2's complement there is no ambiguity in 0 representation
<b>Representation of positive integers</b>	Binary equivalent of integer in 8 bit form	Binary equivalent of integer in 8 bit form	Binary equivalent of integer in 8 bit form	All three forms are same.
<b>Representation of negative integers</b>	Sign bit 1 and magnitude is represented in 7 bit binary form	Find 1's complement of 8 bit form binary	Find 2's complement of 8 bit form binary	For all negative numbers MSB is 1

*Table 2.7 : Comparison for representation of integers in 8-bit word length*

## Subtraction using complements

We have discussed how to subtract a binary number from another binary number. But to design and implement an electronic circuit for this method of subtraction is really complex and difficult. Circuitry for binary addition is simpler. So it is better if we can subtract through addition. For that we use the concept of complements. There are two methods of subtraction using complements.

### Subtraction using 1's complement

The steps for subtracting a smaller binary number from a larger binary number are:

- Step 1:** Add 0s to the left of smaller number, if necessary, to make two numbers with same number of bits.
- Step 2:** Find 1's complement of subtrahend (Number to be subtracted, here small number)
- Step 3:** Add the complement with minuend (Number from which subtracting, here larger number)
- Step 4:** Add the carry 1 to the sum to get the answer.

**Example:** Subtract  $100_2$  from  $1010_2$  using 1's complement.

At first find 1's complement of  $0100$  and it is  $1011$



To compare the three types of representations let us consider the following table. For clarity and easy illustration, 4-bits are used to represent the numbers in this table .

Number	Sign & Magnitude	1's Complement	2's Complement
-8	Not possible	Not possible	1000
-7	1111	1000	1001
-6	1110	1001	1010
-5	1101	1010	1011
-4	1100	1011	1100
-3	1011	1100	1101
-2	1010	1101	1110
-1	1001	1110	1111
0	1000 or 0000	0000 or 1111	0000
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0011
4	0100	0100	0100
5	0101	0101	0101
6	0110	0110	0110
7	0111	0111	0111

From this table, it is clear that the MSB of a binary number indicates the sign of the corresponding decimal number irrespective of the representation. That is, if the MSB is 1, the number is negative and if it is 0, the number is positive. The table also shows that only 2's complement method can represent the maximum numbers for a given number of bits. This fact reveals that, a number below -7 and above +7 cannot be represented using 4-bits in sign & magnitude form and 1's complement form. So we go for 8-bit representation. Similarly in 2's complement method, if we want to handle numbers outside the range -8 to +7, eight bits are required.

In 8-bits implementation, the numbers from -128 to +127 can be represented in 2's complement method. The range will be -127 to +127 for the other two methods. For the numbers outside this range, we use 16 bits and so on for all the representations.



Add it with larger number, i.e.

$$\begin{array}{r}
 1010 \quad + \\
 \underline{1011} \\
 1 \ 0101 \\
 0101 \quad + \\
 \underline{1} \\
 0110
 \end{array}$$

MSB

The result is

MSB is removed and added with the result

### Subtraction using 2's complement

To subtract a smaller binary number from a larger binary number the following are the steps.

- Step 1:** Add 0s to the left of smaller number, if necessary, to make the two numbers have the same number of bits.
- Step 2:** Find 2's complement of subtrahend (Number to be subtracted, here the smaller number).
- Step 3:** Add the 2's complement with minuend (Number from which subtracting, here the larger number).
- Step 3:** Ignore the carry.

**Example:** Subtract  $(100)_2$  from  $(1010)_2$  using 2's complement.

$$\begin{array}{r}
 \text{2's complement of } 0100 \quad 1100 \\
 \text{Add it with larger number, i.e.} \quad 1010 \quad + \\
 \underline{1100} \\
 1 \ 0110 \\
 \text{Ignore the carry to get the result} \\
 \text{The result is} \quad 110
 \end{array}$$

### b. Representation of floating point numbers

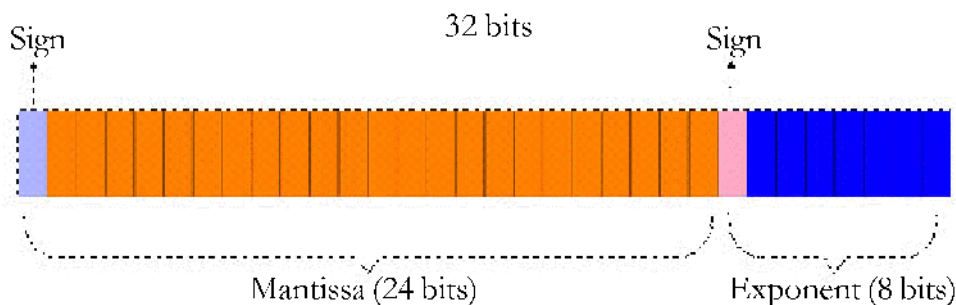
A floating point number / real number consists of an integer part and a fractional part. A real number can be written in a special notation called the floating point notation. Any number in this notation contains two parts, *mantissa* and *exponent*.

For example, 25.45 can be written as  $0.2545 \times 10^2$ , where 0.2545 is the mantissa and the power 2 is the exponent. (In normalised floating point notation mantissa is between 0.1 and 1). Similarly -0.0035 can be written as  $-0.35 \times 10^{-2}$ , where -0.35 is mantissa and -2 is exponent.

Let us see how a real number is represented in 32 bit word length computer. Here 24 bits are used for storing mantissa (among these the first bit is for sign) and 8 bits are used for storing exponent (first bit for sign) as in Figure 2.3. Assume that decimal



point is to the right of the sign bit of mantissa. No separate space is reserved for storing decimal point.



*Fig 2.3: Representation of floating point numbers*

Consider the real number 25.45 mentioned earlier, that can be written as  $0.2545 \times 10^2$ , where 0.2545 is the mantissa and 2 is the exponent. These numbers are converted into binary and stored in respective locations. Various standards are followed for representing mantissa and exponent. When word length changes, bits used for storing mantissa and exponents will change accordingly.



In real numbers, binary point keeps track of mantissa part and exponent part. Since the value of mantissa and exponent varies from number to number the binary point is not fixed. In other words it floats and hence such a representation is called floating point representation.

### 2.4.2 Representation of characters

We have discussed methods for representing numbers in computer memory. Similarly there are different methods to represent characters. Some of them are discussed below.

#### a. ASCII

The code called ASCII (pronounced “AS-key”), which stands for American Standard Code for Information Interchange, uses 7 bits to represent each character in computer memory. The ASCII representation has been adopted as a standard by the U.S. government and is widely accepted. A unique integer number is assigned to each character. This number called ASCII code of that character is converted into binary for storing in memory. For example, ASCII code of A is 65, its binary equivalent in 7-bit is 1000001. Since there are exactly 128 unique combinations of 7 bits, this 7-bit code can represent only 128 characters.

Another version is ASCII-8, also called extended ASCII, which uses 8 bits for each character, can represent 256 different characters. For example, the letter A is represented by 01000001, B by 01000010 and so on. ASCII code is enough to represent all of the standard keyboard characters.



### **b. EBCDIC**

It stands for Extended Binary Coded Decimal Interchange Code. This is similar to ASCII and is an 8 bit code used in computers manufactured by International Business Machine (IBM). It is capable of encoding 256 characters. If ASCII coded data is to be used in a computer which uses EBCDIC representation, it is necessary to transform ASCII code to EBCDIC code. Similarly if EBCDIC coded data is to be used in a ASCII computer, EBCDIC code has to be transformed to ASCII.

### **c. ISCII**

ISCII stands for Indian Standard Code for Information Interchange or Indian Script Code for Information Interchange. It is an encoding scheme for representing various writing systems of India. ISCII uses 8-bits for data representation. It was evolved by a standardisation committee under the Department of Electronics during 1986-88, and adopted by the Bureau of Indian Standards (BIS). Nowadays ISCII has been replaced by Unicode.

### **d. Unicode**

Using 8-bit ASCII we can represent only 256 characters. This cannot represent all characters of written languages of the world and other symbols. Unicode is developed to resolve this problem. It aims to provide a standard character encoding scheme, which is universal and efficient. It provides a unique number for every character, no matter what the language and platform be.

Unicode originally used 16 bits which can represent up to 65,536 characters. It is maintained by a non-profit organisation called the Unicode Consortium. The Consortium first published the version 1.0.0 in 1991 and continues to develop standards based on that original work. Nowadays Unicode uses more than 16 bits and hence it can represent more characters. Unicode can represent characters in almost all written languages of the world.

## **2.4.3 Representation of audio, image and video**

In the previous sections we have discussed different data representation techniques and standards used for the computer representation of numbers and characters. While we attempt to solve real life problems with the aid of a digital computer, in most cases we may have to represent and process data other than numbers and characters. This may include audio data, images and videos. We can see that like numbers and characters, the audio, image and video data also carry information. In this section we will see different file formats for storing sound, image and video.

### Digital audio, image and video file formats

Multimedia data such as audio, image and video are stored in different types of files. The variety of file formats is due to the fact that there are quite a few approaches to compressing the data and a number of different ways of packaging the data. For example an image is most popularly stored in Joint Picture Experts Group (JPEG) file format. An image file consists of two parts - header information and image data. Information such as name of the file, size, modified data, file format, etc. are stored in the header part. The intensity value of all pixels is stored in the data part of the file.

The data can be stored uncompressed or compressed to reduce the file size. Normally, the image data is stored in compressed form. Let us understand what compression is. Take a simple example of a pure black image of size  $400 \times 400$  pixels. We can repeat the information black, black, ..., black in all 16,0000 ( $400 \times 400$ ) pixels. This is the uncompressed form, while in the compressed form black is stored only once and information to repeat it 1,60,000 times is also stored. Numerous such techniques are used to achieve compression. Depending on the application, images are stored in various file formats such as bitmap file format (BMP), Tagged Image File Format (TIFF), Graphics Interchange Format (GIF), Portable (Public) Network Graphic (PNG).

What we said about the header file information and compression is also applicable for audio and video files. Digital audio data can be stored in different file formats like WAV, MP3, MIDI, AIFF, etc. An audio file describes a format, sometimes referred to as the 'container format', for storing digital audio data. For example WAV file format typically contains uncompressed sound and MP3 files typically contain compressed audio data. The synthesised music data is stored in MIDI (Musical Instrument Digital Interface) files. Similarly video is also stored in different files such as AVI (Audio Video Interleave) - a file format designed to store both audio and video data in a standard package that allows synchronous audio with video playback, MP3, JPEG-2, WMV, etc.

#### Check yourself



1. Which is the MSB of representation of -80 in the sign and magnitude method?
2. Write 28.756 in mantissa exponent form.
3. ASCII stands for \_\_\_\_\_.
4. Represent -60 in 1's complement form.
5. Define Unicode.
6. List any two image file formats.

## 2.5 Introduction to Boolean algebra

In many situations in our life we face questions that require 'Yes' or 'No' answers. Similarly much of our thinking process involves answering questions with 'Yes' or 'No'. The way of finding truth by answering such two-valued questions is known as human reasoning or logical reasoning. These values can be expressed as 'True' or 'False' and numerically 1 or 0. These values are known as binary values or Boolean values. Boolean algebra is the algebra of logic which is a part of mathematical algebra that deals with the operations on variables that represent the values 1 and 0. The name Boolean algebra is given to honour the British mathematician George Boole, as he was the person who established the link between logic and mathematics. His revolutionary paper 'An Investigation of the laws of thought' led to the development of Boolean algebra.



Fig. 2.4: George Boole (1815 - 1864)

### 2.5.1 Binary valued quantities

Let us consider the following:

1. Should I take an umbrella?
2. Will you give me your pen?
3. George Boole was a British mathematician.
4. Kerala is the biggest state in India.
5. Why were you absent yesterday?
6. What is your opinion about Boolean algebra?

1<sup>st</sup> and 2<sup>nd</sup> sentences are questions which can be answered as YES or NO. These cases are called binary decisions and the results are called binary values. The 3<sup>rd</sup> statement is TRUE and 4<sup>th</sup> statement is FALSE. But 5<sup>th</sup> and 6<sup>th</sup> sentences cannot be answered like the cases above. The sentences which can be determined to be TRUE or FALSE are called **logical statements** or truth functions and the results TRUE or FALSE are called binary values or logical constants. The **logical constants** are represented by 1 and 0, where 1 stands for TRUE and 0 for FALSE. The variables which can store (hold) logical constants 1 and 0 are called logical variables or **Boolean variables**.

### 2.5.2 Boolean operators and logic gates

We have already seen that data fed to a computer must be converted into a combination of 1s and 0s. All data, information and operations are represented inside the computer



using 0s and 1s. The operations performed on these Boolean values are called **Boolean operations**. As we know, operators are required to perform these operations. These operators are called Boolean operators or logical operators. There are three basic logical operators in Boolean algebra. These operators and their operations are as follows:

- OR → Logical Addition
- AND → Logical Multiplication
- NOT → Logical Negation

The first two operators require two operands and the third requires only one operand. Here the operands are always Boolean variable or constants and the result will always be either True (1) or False (0).

Computers perform these operations with some electronic circuits, called logic circuits. A logic circuit is made up of individual units called gates, where a gate represents a Boolean operation. A **Logic gate** is a physical device that can perform logical operations on one or more logical inputs and produce a single logical output. Logic gates are primarily implemented using diodes or transistors acting as electronic switches. There are three basic logic gates and they represent the three basic Boolean operations. These gates are OR, AND and NOT.

### a. The OR operator and OR gate

Let us consider a real life situation. When do you use an umbrella? When it rains, isn't it? And of course, if it is too sunny. We can combine these two situations using a compound statement like "If it is raining or if it is sunny, we use an umbrella". Note down the use of **or** in this statement. The interpretation of this statement can be shown as in Table 2.8. The logical reasoning of the use of umbrella in our example very much resembles the Boolean OR operation.

Raining	Sunny	Need Umbrella
No	No	No
No	Yes	Yes
Yes	No	Yes
Yes	Yes	Yes

Table 2.8: Logical OR operation

The OR operator performs logical addition and the symbol used for this operation is + (plus). The expression  $A + B$  is read as A OR B. Table 2.9 is the truth table that represents the OR operation. Assume that the variables A and B are the inputs (operands) and  $A + B$  is the output (result). It is clear from the truth table that, if any one of the inputs is 1 (True), the output will be 1 (True).

**Truth Table** is a table that shows Boolean operations and their results. It lists all possible inputs for the given operation and their corresponding output. Usually these operations consist of operand variables and operators. The operands and the operators together are called Boolean expression. Truth Table represents all possible values of the operands and the corresponding results (values) of the operation. A Boolean expression with  $m$  operands (variables) and  $n$  operators require  $2^m$  rows and  $m + n$  columns.

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

Table 2.9 : Truth table of OR operation

While designing logic circuits, the logic gate used to implement logical OR operation is called logical **OR gate**. Figure 2.5 shows the OR gate symbol in Boolean algebra.

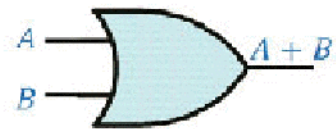


Fig. 2.5 : Logical OR gate

The working of this gate can be illustrated with an electronic circuit. Figure 2.6 illustrates the schematic circuit of parallel switches which shows the idea of an OR gate. Here A and B are two switches and Y is a bulb. Each switch and the bulb can take either close (ON) or open (OFF) state. Now let us relate the operation of the above circuit with the functioning of OR gate. Assume that OFF represents the logical LOW state (say 0) and ON represents the logical HIGH state (say 1). If we consider the state of switches A and B as input to the OR gate and state of bulb as output of OR gate, then the truth table shown in Table 2.9 will describe the operation of an OR gate. Thus the Boolean expression for OR gate can be written as:  $Y = A + B$

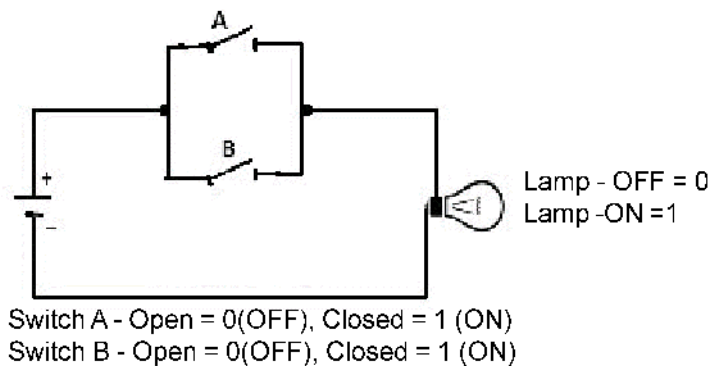


Fig. 2.6 : Circuit with two switches and a bulb for parallel connection

An OR gate can take more than two inputs. Let us see what will be the truth table, Boolean expression and logical symbol for the three input OR gate.



The truth table and the gate symbol shows that, the Boolean expression for the OR gate with three inputs is  $Y = A + B + C$ . Figure 2.7 shows the representation of OR gate with three inputs. From the truth tables 2.9 and 2.10, we can see that the output of OR gate is 1 if any input is 1; and output is 0 if and only if all inputs are 0.

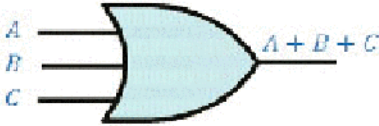


Fig 2.7 : OR gate  
with three inputs

A	B	C	$A + B + C$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Table 2.10 : Truth table for OR gate with 3 inputs

### b. The AND operator and AND gate

We will discuss another situation to understand the concept of AND Boolean operation. Suppose you are away from home and it is lunch time. You can have your food only if two conditions are satisfied – (i) there should be a hotel and (ii) you should have enough money. Here also, we can make a compound statement like “If there is a hotel and if we have money, we can have food”. Note the use of **and** in this statement. Table 2.11 shows the logical reasoning of getting food and it very much resembles the Boolean AND operation.

Hotel	Money	Take Food
No	No	No
No	Yes	No
Yes	No	No
Yes	Yes	Yes

Table 2.11 : Logical AND operation

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.12 : Truth table of AND operation

The AND operator performs logical multiplication and the symbol used for this operation is  $\cdot$  (dot). The expression  $A \cdot B$  is read as A AND B. Table 2.12 is the truth table that represents the AND operation. Assume that the variables A and B are the inputs (operands) and  $A \cdot B$  is the output (result).

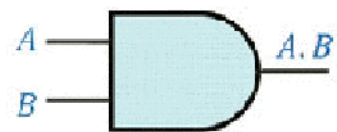


Fig. 2.8 : Logical AND gate

While designing logic circuits, the logic gate used to implement logical AND operation is called logical **AND gate**. Figure 2.8 shows the AND gate symbol in Boolean algebra.

The working of this gate can be illustrated with an electronic circuit shown in Figure 2.9. This schematic circuit has two serial switches which illustrates the idea of an AND gate. Here A and B are two switches and Y is a bulb. Each switch and the bulb can take either close (ON) or open (OFF) state. Now let us relate the operation of the above circuit with the functioning of AND gate. Assume that OFF represents the logical LOW state (say 0) and ON represents the logical HIGH state (say 1). If we consider the state of switches A and B as input to the AND gate and state of bulb as output of AND gate, then the Boolean expression for AND gate can be written as:  $Y = A \cdot B$

An AND gate can take more than two inputs. Let us see what will be the truth table, Boolean expression and logical symbol for three input AND gate. The truth table and the gate symbol shows that the Boolean expression for the AND gate with three inputs is  $Y = A \cdot B \cdot C$

Figure 2.10 shows the representation of AND gate with three inputs. From Truth Tables 2.12 and 2.13, we can see that the output of AND gate is 0 if any input is 0; and output is 1 if and only if all inputs are 1.

### c. The NOT operator and NOT gate

Let us discuss another case to familiarise the Boolean NOT operation. Suppose you jog everyday in the morning. Can you do it every day? If it rains, can you jog in the morning? Table 2.14 shows all the possibilities of this situation. It is quite similar to Boolean NOT operation.

It is a unary operator and hence it requires only one operand. The NOT operator performs logical negation and the symbol used for this operation is - (over-bar).

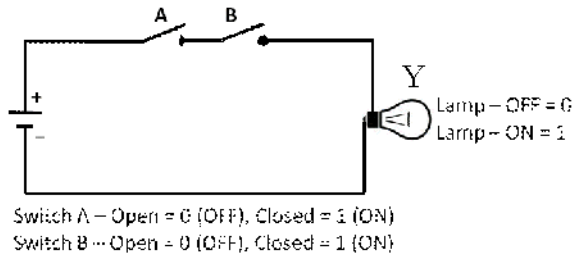


Fig. 2.9 : Circuit with two switches and a bulb for serial connection

A	B	C	A.B.C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 2.13 : Truth table for AND gate with 3 inputs



Fig. 2.10 : AND gate with three inputs

Raining	Jogging
No	Yes
Yes	No

Table 2.14: Logical NOT

The expression  $\bar{A}$  is read as A bar. It is also expressed as  $A'$  and read as A dash. Table 2.15 is the truth table that represents the NOT operation. Assume that the variable A is the input (operand) and  $\bar{A}$  is the output (result). It is clear from the truth table that, the output will be the opposite value of the input. The logic gate used to implement NOT operation is NOT gate. Figure 2.11 shows the NOT gate symbol.

A	$\bar{A}$
0	1
1	0

Table 2.15 : Truth table of NOT operation

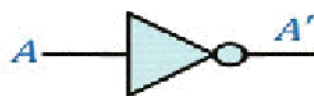


Fig. 2.11 : NOT gate

A NOT gate is also called **inverter**. It has only one input and one output. The input is always changed into its opposite state. If input is 0, the NOT gate will give its complement or opposite which is 1. If the input is 1, then the NOT gate will complement it to 0.

### Check yourself



1. Define the term Boolean variable.
2. A logic circuit is made up of individual units called \_\_\_\_\_.
3. Name the logical operator/gate which gives high output if and only if all the inputs are high.
4. Define the term truth table.
5. An AND operation performs logical \_\_\_\_\_ and an OR operation performs logical \_\_\_\_\_.
6. Draw the logic symbol of OR gate.

## 2.6 Basic postulates of Boolean algebra

Boolean algebra being a system of mathematics, consists of certain fundamental laws. These fundamental laws are called postulates. They do not have proof, but are made to build solid framework for scientific principles. On the other hand, there are some theorems in Boolean algebra which can be proved based on these postulates and laws.

### Postulate 1: Principles of 0 and 1

If  $A \neq 0$ , then  $A = 1$  and if  $A \neq 1$ , then  $A = 0$

### Postulate 2: OR Operation (Logical Addition)

$$0 + 0 = 0 \quad 0 + 1 = 1 \quad 1 + 0 = 1 \quad 1 + 1 = 1$$

### Postulate 3: AND Operation (Logical Multiplication)

$$0 \cdot 0 = 0 \quad 0 \cdot 1 = 0 \quad 1 \cdot 0 = 0 \quad 1 \cdot 1 = 1$$

**Postulate 4: NOT Operation (Logical Negation or Compliment Rule)**

$$\bar{0} = 1 \quad \bar{1} = 0$$

**Principle of Duality**

When Boolean variables and/or values are combined with Boolean operators, Boolean expressions are formed.  $X + Y$  and  $\bar{A} + 1$  are examples of Boolean expressions. The postulates 2, 3 and 4 are all Boolean statements. Consider the statements in postulate 2. If we change the value 0 by 1 and 1 by 0, and the operator OR (+) by AND (.), we will get the statements in postulate 3. Similarly, if we change the value 0 by 1 and 1 by 0, and the operator AND (.) by OR (+) in statements of postulate 3, we will get the statements of postulate 2. This concept is known as principle of duality.

The principle of duality states that for a Boolean statement, there exists its dual form, which can be derived by

- (i) changing each OR sign (+) to AND sign (.)
- (ii) changing each AND sign (.) to OR sign (+)
- (iii) replacing each 0 by 1 and each 1 by 0

**2.7 Basic theorems of Boolean algebra**

There are some standard and accepted rules in every theory. The set of rules are known as **axioms** of the theory. A conclusion can be derived from a set of presumptions by using these axioms or postulates. This conclusion is called law or theorem. Theorems of Boolean algebra provide tools for simplification and manipulation of Boolean expressions. Let us discuss some of these laws or theorems. These laws or theorems can be proved using truth tables and Boolean laws that are already proved.

**2.7.1 Identity law**

If  $X$  is a Boolean variable, the law states that:

- (i)  $0 + X = X$
- (ii)  $1 + X = 1$
- (iii)  $0 \cdot X = 0$
- (iv)  $1 \cdot X = X$

Statements (i) and (ii) are known as additive identity law; and statements (iii) and (iv) are called multiplicative identity. Also note that, statement (iv) is the dual of (i) and vice versa. Similarly, statements (ii) and (iii) are dual forms. The truth tables shown in Tables 2.16(a), 2.16(b), 2.17(a) and 2.17 (b) prove these laws.

0	X	$0 + X$
0	0	0
0	1	1

Table 2.16 (a) : Additive Identity law

1	X	$1 + X$
1	0	1
1	1	1

Table 2.16 (b) : Additive Identity law

Table 2.16 (a) shows that columns 2 and 3 are the same and it is proved that  $0 + X = X$ . Similarly, columns 1 and 3 of table 2.16 (b) are the same and hence the statement  $1 + X = 1$  is true.

0	X	$0 \cdot X$
0	0	0
0	1	0

Table 2.17(a) : Multiplicative Identity law

1	X	$1 \cdot X$
1	0	0
1	1	1

Table 2.17(b) : Multiplicative Identity law

Table 2.17 (a) shows that columns 1 and 3 are the same and it is proved that  $0 \cdot X = 0$ . Similarly, columns 2 and 3 of Table 2.17 (b) are the same and hence the statement  $1 \cdot X = X$  is true.

### 2.7.2 Idempotent law

The idempotent law states that: (i)  $X + X = X$

and (ii)  $X \cdot X = X$

If the value of X is 0, the statements are true, because  $0 + 0 = 0$  (Postulate 2) and  $0 \cdot 0 = 0$  (Postulate 3). Similarly the statements will be true

X	X	$X + X$
0	0	0
1	1	1

Table 2.18 (a) : Idempotent law

X	X	$X \cdot X$
0	0	0
1	1	1

Table 2.18 (b) : Idempotent law

when the value of X is 1. Truth Tables 2.18 (a) and 2.18 (b) shows the proof of these laws. Also note that the statements are dual to each other.

### 2.7.3 Involution law

This law states that:  $\overline{\overline{X}} = X$

Let  $X = 0$ , then  $\overline{X} = 1$  (Postulate 4); and if we take its complement,  $\overline{\overline{X}} = \overline{1} = 0$ , which is same as X. The statement will also be true, when the value of X is 1.

Columns 1 and 3 of Table 2.19 show that  $\overline{\overline{X}} = X$ .

X	$\overline{X}$	$\overline{\overline{X}}$
0	1	0
1	0	1

Table 2.19 : Involution law



### 2.7.4 Complimentary law

The complimentary law states that: (i)  $X + \bar{X} = 1$

and (ii)  $X \cdot \bar{X} = 0$

If the value of  $X$  is 0, then  $\bar{X}$  becomes 1. Hence,  $X + \bar{X}$  becomes  $0 + 1$ , which results into 1 (*Postulate 2*). Similarly when  $X$  is 1,  $\bar{X}$  will be 0. The truth tables 2.20 (a) and 2.20 (b) show the proof of these laws taking all the possibilities. Also note that the statements are dual to each other.

$X$	$\bar{X}$	$X + \bar{X}$
0	1	1
1	0	1

Table 2.20 (a) : Complimentary law

$X$	$\bar{X}$	$X \cdot \bar{X}$
0	1	0
1	0	0

Table 2.20 (b) : Complimentary law

### 2.7.5 Commutative law

Commutative law allows to change the position of variables in OR and AND operations. If  $X$  and  $Y$  are two variables, the law states that:

$$(i) X + Y = Y + X$$

$$\text{and } (ii) X \cdot Y = Y \cdot X$$

The truth table shown in Tables 2.21 (a) and 2.21 (b) prove these statements.

$X$	$Y$	$X + Y$	$Y + X$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Table 2.21 (a) : Commutative law

$X$	$Y$	$X \cdot Y$	$Y \cdot X$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Table 2.21 (b) : Commutative law

The law ensures that order of the operands for OR and AND operations does not affect the output in each case.

### 2.7.6 Associative law

In the case of three operands for OR and AND operations, associative law allows grouping of operands differently. If  $X$ ,  $Y$  and  $Z$  are three variables, the law states that:

$$(i) X + (Y + Z) = (X + Y) + Z$$

$$\text{and } (ii) X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$$





The truth tables shown in Tables 2.22(a) and 2.22(b) prove these statements.

X	Y	Z	$X + Y$	$Y + X$	$X + (Y + Z)$	$(X + Y) + Z$
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Table 2.22(a) : Associative law 1

In Table 2.22(a), columns 6 and 7 show that  $X + (Y + Z) = (X + Y) + Z$ . Columns 6 and 7 of Table 2.22(b) show the validity of the experience  $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$ .

X	Y	Z	$X \cdot Y$	$Y \cdot Z$	$X \cdot (Y \cdot Z)$	$(X \cdot Y) \cdot Z$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	1	0	0	0
1	1	1	1	1	1	1

Table 2.22(b) : Associative law 2

Associative law ensures that the order and combination of variables in OR (logical addition) or AND (logical multiplication) operations do not affect the final output.

### 2.7.7 Distributive law

Distributive law states that Boolean expression can be expanded by multiplying terms as in ordinary algebra. It also supports expansion of addition operation over multiplication. If X, Y and Z are variables, the law states that:

$$(i) X \cdot (Y + Z) = X \cdot Y + X \cdot Z$$

and  $(ii) X + Y \cdot Z = (X + Y) \cdot (X + Z)$

The following truth tables prove these statements:

X	Y	Z	$Y + Z$	$X.(Y+Z)$	$X.Y$	$X.Z$	$X.Y + X.Z$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Table 2.23(a) : Distribution of Multiplication over Addition

Columns 5 and 8 of Table 2.23(a) show that  $X . (Y + Z) = X . Y + X . Z$

X	Y	Z	$Y . Z$	$X + Y . Z$	$X+Y$	$X+Z$	$(X+Y) . (X+Z)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

Table 2.23(b) : Distribution of Addition over Multiplication

Columns 5 and 8 of Table 2.23(b) show that  $X + Y . Z = (X + Y) . (X + Z)$

We are familiar with the first statement in ordinary algebra. To remember the second statement of this law, find the dual form of the first.

### 2.7.8 Absorption law

Absorption law is a kind of distributive law in which two variables are used and the result will be one of them. If X and Y are variables, the absorption law states that:

$$(i) X + (X . Y) = X$$

and

$$(ii) X . (X + Y) = X$$



The truth tables shown Tables 2.24(a) and 2.24(b) prove the validity of the statements of absorption law.

X	Y	$X \cdot Y$	$X + (X \cdot Y)$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Table 2.24 (a) : Absorption law

X	Y	$X + Y$	$X \cdot (X + Y)$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

Table 2.24 (b) : Absorption law

Columns 1 and 4 of Table 2.24(a) and columns 1 and 4 of Table 2.24(b) show that the laws are true.

Table 2.25 depicts all the Boolean laws we have discussed so far.

No.	Boolean Law	Statement 1	Statement 2
1	Additive Identity	$0 + X = X$	$1 + X = 1$
2	Multiplicative Identity	$0 \cdot X = 0$	$1 \cdot X = X$
3	Idempotent Law	$X + X = X$	$X \cdot X = X$
4	Involution Law	$\overline{\overline{X}} = X$	
5	Complimentary Law	$X + \overline{X} = 1$	$X \cdot \overline{X} = 0$
6	Commutative Law	$X + Y = Y + X$	$X \cdot Y = Y \cdot X$
7	Associative Law	$X + (Y + Z) = (X + Y) + Z$	$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$
8	Distributive Law	$X \cdot (Y + Z) = X \cdot Y + X \cdot Z$	$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$
9	Absorption Law	$X + (X \cdot Y) = X$	$X \cdot (X + Y) = X$

Table 2.25 : Boolean laws

The laws we discussed have been proved using truth tables. Some of them can be proved by applying some other laws. This method of proof is called algebraic proof. Let us see some of them.

#### i. To prove that $X \cdot (X + Y) = X$ – Absorption law

$$\begin{aligned}
 \text{LHS} &= X \cdot (X + Y) \\
 &= X \cdot X + X \cdot Y && (\text{Distribution of multiplication over addition}) \\
 &= X + X \cdot Y && (\text{Idempotent law}) \\
 &= X \cdot (1 + Y) && (\text{Distribution of multiplication over addition})
 \end{aligned}$$



$$\begin{aligned}
 &= X \cdot 1 && (\text{Additive identity}) \\
 &= X && (\text{Multiplicative identity}) \\
 &= \text{RHS}
 \end{aligned}$$

**ii. To prove that  $X + (X \cdot Y) = X$  – Absorption law**

$$\begin{aligned}
 \text{LHS} &= X + (X \cdot Y) \\
 &= X \cdot 1 + X \cdot Y && (\text{Multiplicative identity}) \\
 &= X \cdot (1 + Y) && (\text{Distribution of multiplication over addition}) \\
 &= X \cdot 1 && (\text{Additive identity}) \\
 &= X && (\text{Multiplicative identity}) \\
 &= \text{RHS}
 \end{aligned}$$

**iii. To prove that  $X + (Y \cdot Z) = (X+Y) \cdot (X+Z)$  – Distributive Law**

Let us take the expression on the RHS of this statement.

$$\begin{aligned}
 &(X+Y) \cdot (X+Z) \\
 &= (X+Y) \cdot X + (X+Y) \cdot Z && (\text{Distribution of multiplication over addition}) \\
 &= X \cdot (X+Y) + Z \cdot (X+Y) && (\text{Commutative law}) \\
 &= X \cdot X + X \cdot Y + Z \cdot X + Z \cdot Y && (\text{Distribution of multiplication over addition}) \\
 &= X + X \cdot Y + Z \cdot X + Z \cdot Y && (\text{Idempotent law}) \\
 &= X \cdot 1 + X \cdot Y + Z \cdot X + Z \cdot Y && (\text{Multiplicative identity}) \\
 &= X \cdot (1 + Y) + Z \cdot X + Z \cdot Y && (\text{Distribution of multiplication over addition}) \\
 &= X \cdot 1 + Z \cdot X + Z \cdot Y && (\text{Additive identity}) \\
 &= X \cdot (1 + Z) + Z \cdot Y && (\text{Distribution of multiplication over addition}) \\
 &= X \cdot 1 + Z \cdot Y && (\text{Additive identity}) \\
 &= X + Y \cdot Z && (\text{Multiplicative identity and Commutative law}) \\
 &= \text{LHS}
 \end{aligned}$$

'The expression obtained is the LHS of the given statement. Thus the theorem is proved.



## 2.8 De Morgan's theorems

Augustus De Morgan (1806 – 1871), a famous logician and mathematician of University College, London proposed two theorems to simplify complicated Boolean expressions. These theorems are known as De Morgan's theorems. The two theorems are:

$$(i) \quad \overline{X+Y} = \bar{X} \cdot \bar{Y}$$

$$(ii) \quad \overline{X \cdot Y} = \bar{X} + \bar{Y}$$

Literally these theorems can be stated as

- (i) “the complement of sum of Boolean variables is equal to product of their individual complements” and
- (ii) “the complement of product of Boolean variables is equal to sum of their individual complements”.

### Algebraic proof of the first theorem

We have to prove that,  $\overline{X+Y} = \bar{X} \cdot \bar{Y}$

Let us assume that,  $Z = X + Y$  \_\_\_\_\_ (1)

$$\text{Then, } \bar{Z} = \overline{X+Y} \text{ _____ (2)}$$

We know that, by complementary law, the equations (3) and (4) are true.

$$Z + \bar{Z} = 1 \text{ _____ (3)}$$

$$Z \cdot \bar{Z} = 0 \text{ _____ (4)}$$

Substituting expressions (1) in (3) and (2) in (4), we will get equations (5) and (6).

$$(X + Y) + (\overline{X+Y}) = 1 \text{ _____ (5)}$$

$$(X + Y) \cdot (\overline{X+Y}) = 0 \text{ _____ (6)}$$

For the time being let us assume that De Morgan's first theorem is true. If so,  $(\overline{X+Y})$  in equations (5) and (6) can be substituted with  $(\bar{X} \cdot \bar{Y})$ . Thus equations (5) and (6) can be modified as follows:

$$(X + Y) + (\bar{X} \cdot \bar{Y}) = 1 \text{ _____ (7)}$$

$$(X + Y) \cdot (\bar{X} \cdot \bar{Y}) = 0 \text{ _____ (8)}$$

Now we will prove equations (7) and (8) separately. If they are correct, we can conclude that the assumptions we made to form those equations are also correct. That is, if equations (7) and (8) are true, De Morgan's theorem is also true.

Consider the LHS of equation (7),

$$\begin{aligned} (X + Y) + (\bar{X} \cdot \bar{Y}) &= (X + Y + \bar{X}) \cdot (X + Y + \bar{Y}) && \text{(Distributive Law)} \\ &= (X + \bar{X} + Y) \cdot (X + Y + \bar{Y}) && \text{(Associative Law)} \\ &= (1 + Y) \cdot (X + 1) && \text{(Complimentary Law)} \end{aligned}$$



$$= 1 \cdot 1 \quad (\text{Additive Identity})$$

$$= 1$$

$$= \text{RHS}$$

Now, let us consider the LHS of equation (8),

$$(X + Y) \cdot (\bar{X} \cdot \bar{Y}) = (X \cdot \bar{X} \cdot \bar{Y}) + (Y \cdot \bar{X} \cdot \bar{Y}) \quad (\text{Distributive Law})$$

$$= (X \cdot \bar{X} \cdot \bar{Y}) + (Y \cdot \bar{Y} \cdot \bar{X}) \quad (\text{Associative Law})$$

$$= (0 \cdot \bar{Y}) + (0 \cdot \bar{X}) \quad (\text{Complimentary Law})$$

$$= 0 + 0 \quad (\text{Multiplicative Identity})$$

$$= 0$$

$$= \text{RHS}$$

We have algebraically proved equations (7) and (8), which mean that De Morgan's first theorem is proved. The theorem can also be proved using truth table, but it is left to you as an exercise.

### Algebraic proof of the second theorem

We have to prove that,  $\overline{X \cdot Y} = \bar{X} + \bar{Y}$

Let us assume that,  $Z = X \cdot Y$  \_\_\_\_\_ (1)

Then,  $\bar{Z} = \overline{X \cdot Y}$  \_\_\_\_\_ (2)

We know that, by complimentary laws the equations (3) and (4) are true.

$$Z + \bar{Z} = 1 \quad \text{_____ (3)}$$

$$Z \cdot \bar{Z} = 0 \quad \text{_____ (4)}$$

Substituting expressions (1) in (3) and (2) in (4), we will get the expressions (5) and (6).

$$(X \cdot Y) + (\overline{X \cdot Y}) = 1 \quad \text{_____ (5)}$$

$$(X \cdot Y) \cdot (\overline{X \cdot Y}) = 0 \quad \text{_____ (6)}$$

For the time being let us assume that De Morgan's second theorem is true. If so,  $(\overline{X \cdot Y})$  in equations 5 and 6 can be substituted with  $(\bar{X} + \bar{Y})$ . Thus equations (5) and (6) can be modified as follows:

$$(X \cdot Y) + (\bar{X} + \bar{Y}) = 1 \quad \text{_____ (7)}$$

$$(X \cdot Y) \cdot (\bar{X} + \bar{Y}) = 0 \quad \text{_____ (8)}$$

Now we will prove equations (7) and (8) separately. If they are correct, we can conclude that the assumptions we made to form those equations are also correct. That is, if equations (7) and (8) are true, De Morgan's theorem is also true.



Consider the LHS of equation (7),

$$\begin{aligned}
 (X \cdot Y) + (\bar{X} + \bar{Y}) &= (\bar{X} + \bar{Y}) + (X \cdot Y) && \text{(Commutative Law)} \\
 &= (\bar{X} + \bar{Y} + X) \cdot (\bar{X} + \bar{Y} + Y) && \text{(Distributive Law)} \\
 &= (\bar{X} + X + \bar{Y}) \cdot (\bar{X} + \bar{Y} + Y) && \text{(Associative Law)} \\
 &= (1 + \bar{Y}) \cdot (\bar{X} + 1) && \text{(Complimentary Law)} \\
 &= 1 \cdot 1 && \text{(Additive Identity)} \\
 &= 1 \\
 &= \text{RHS}
 \end{aligned}$$

Now, let us consider the LHS of equation (8),

$$\begin{aligned}
 (X \cdot Y) \cdot (\bar{X} + \bar{Y}) &= (X \cdot Y \cdot \bar{X}) + (X \cdot Y \cdot \bar{Y}) && \text{(Distributive Law)} \\
 &= (X \cdot \bar{X} \cdot Y) + (X \cdot Y \cdot \bar{Y}) && \text{(Associative Law)} \\
 &= (0 \cdot Y) + (X \cdot 0) && \text{(Complimentary Law)} \\
 &= 0 + 0 && \text{(Multiplicative Identity)} \\
 &= 0 \\
 &= \text{RHS}
 \end{aligned}$$

We have algebraically proved equations (7) and (8), which mean that De Morgan's second theorem is proved. The theorem can also be proved using truth table, but it is left to you as an exercise.



We can extend Demorgan's theorem for any number of variables as shown below:

$$\begin{aligned}
 \overline{A + B + C + D + \dots} &= \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \bar{D} \cdot \dots \\
 \overline{A \cdot B \cdot C \cdot D \cdot \dots} &= \bar{A} + \bar{B} + \bar{C} + \bar{D} + \dots
 \end{aligned}$$

Although the identities above represent De Morgan's theorem, the transformation is more easily performed by following the steps given below:

- (i) Complement the entire function
- (ii) Change all the ANDs (.) to ORs (+) and all the ORs (+) to ANDs (.)
- (iii) Complement each of the individual variables.

This process is called *demorganisation* and simply demorganisation is 'Break the line, change the sign'.



### Check yourself



- Find the dual of Boolean expression  $A.B+B.C=1$
- Name the law which states that  $A+A=A$ .  
(a) Commutative law (b) Idempotent Law (c) Absorption Law
- State De Morgan's theorems.

## 2.9 Circuit designing for simple Boolean expressions

By using basic gates, circuit diagrams can be designed for Boolean expressions. We have seen that the Boolean expressions  $A.B$  is represented using an AND gate,  $A+B$  is represented using an OR gate and  $\bar{A}$  is represented using a NOT gate. Let us see how a circuit is designed for other Boolean expressions.

Consider a boolean expression  $\bar{A} + B$ , which is an OR operation with two input and first input is inverted. So circuit diagram can be drawn as shown in Figure 2.12.

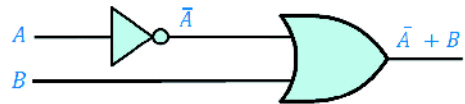


Fig 2.12 :  $f(A, B) = \bar{A} + B$

**Example:** Construct a logical circuit for Boolean expression  $f(X, Y) = X.Y + \bar{Y}$

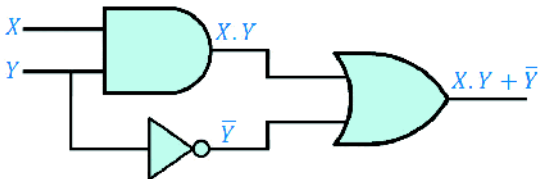


Fig 2.13 :  $f(X, Y) = X.Y + \bar{Y}$

**Example:** Construct a logical expression for  $f(a, b) = (a + b) . (\bar{a} + \bar{b})$

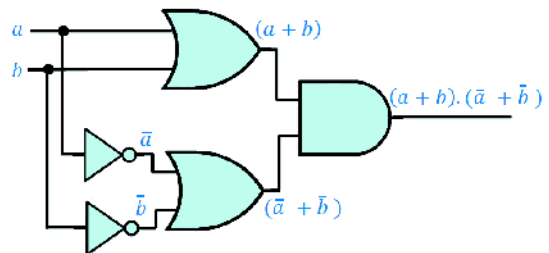


Fig 2.14 :  $f(a, b) = (a + b) . (\bar{a} + \bar{b})$

**Example:** Construct logical circuit for the Boolean expression  $\bar{a} . b + a . \bar{b}$

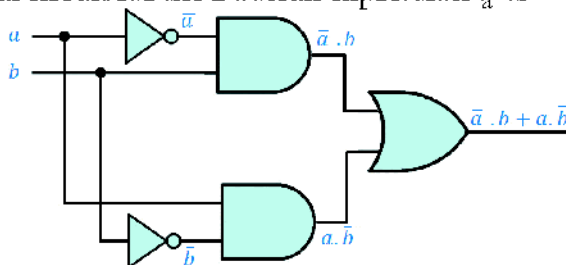


Fig 2.15 :  $f(a, b) = \bar{a} . b + a . \bar{b}$

## 2.10 Universal gates

The NAND and NOR gates are called universal gates. A universal gate is a gate which can implement any Boolean function without using any other gate type. In practice, this is advantageous, since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in most of IC digital logic families.

### 2.10.1 NAND gate

This is an AND gate with its output inverted by a NOT gate. The logical circuit arrangement is shown in Figure 2.16.

Note that A and B are the inputs of AND gate and its output is  $(A.B)$ . The output of AND gate is inverted by an inverter (NOT gate) to get the resultant output Y as  $(\overline{A.B})$ .

So the logical expression for a NAND gate is  $(\overline{A.B})$ . From the truth table shown as Table 2.26, we can see that output of a NAND gate is 1 if any one of the input is 0. It produces output 0 if and only if all inputs are 1. This is the inverse operation of an AND gate. So we can say that **a NAND gate is an inverted AND gate.**

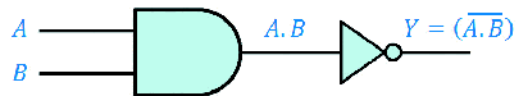


Fig. 2.16 : Circuit realisation of NAND gate

A	B	$Y = (\overline{A.B})$
0	0	1
0	1	1
1	0	1
1	1	0

Table 2.26 : NAND truth table

The logical symbol of NAND gate is shown in Figure 2.17. Note that the NAND symbol is an AND symbol with a small bubble at the output. The bubble is sometimes called an invert bubble.

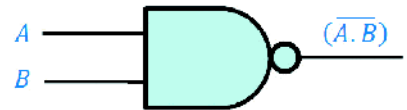


Fig. 2.17 : NAND gate

### 2.10.2 NOR gate

This is an OR gate with its output inverted by a NOT gate. The logical circuit arrangement is shown in Figure 2.18.

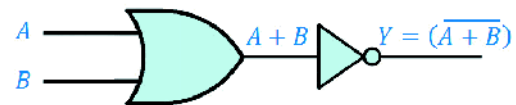


Fig. 2.18 : Circuit realisation of NOR gate

Note that A and B are the input of OR gate and its output is  $(A+B)$ . The output of OR gate is inverted by an inverter (NOT gate) to get resultant output as  $(\overline{A+B})$ . So the logical expression for a NOR gate is  $(\overline{A+B})$ . Let us see the truth table of the two input NOR gate.

From the truth table shown on as 'Table 2.27, we can see that output of a NOR gate is 1 if and only if all inputs are 0. If any one of the inputs is 1 it produces an output 0. This is the inverse operation of an OR gate. So we can say that **a NOR gate is an inverted OR gate.**

A	B	$Y = \overline{(A + B)}$
0	0	1
0	1	0
1	0	0
1	1	0

Table 2.27 : NOR truth table

The logical symbol of NOR gate is shown in Figure 2.19. Note that the NOR symbol is an OR symbol with a small bubble at the output.

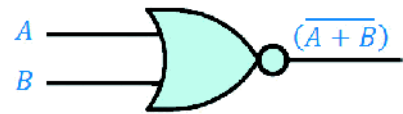


Fig. 2.19 : NOR gate

### 2.10.3 Implementation of basic gates using NAND and NOR

We can design all basic gates (AND, OR and NOT) using NAND or NOR gate alone. Let us see the implementation of basic gates using NAND gate.

#### NOT gate using NAND gate

We can implement a NOT gate (inverter) using a NAND by applying the same signal to both inputs of a NAND gate as shown in Figure 2.20.

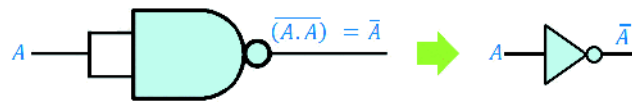


Fig. 2.20 : NOT gate using NAND gate

**Proof:**

$$A \text{ NAND } A = \overline{(A.A)}$$

$$= \overline{A} \quad \text{Since } A.A = A$$

The truth table shown as 'Table 2.28 is the proof for obtaining NOT gate using NAND gate.

A	AA	$\overline{(A.A)}$	$\overline{A}$
0	0	1	1
1	1	0	0

Table 2.28 : Proof using truth table

#### AND gate using NAND gate

We can implement an AND gate by using a NAND gate followed by another NAND gate to invert the output as shown in Figure 2.21.

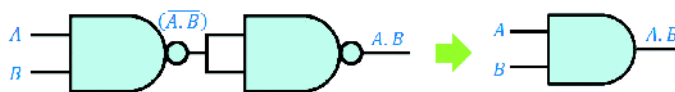


Fig. 2.21 : AND gate using NAND gate

**Proof**

$$\begin{aligned}
 \text{We know that } A \text{ NAND } B &= \overline{(A \cdot B)} \\
 (A \text{ NAND } B) \text{ NAND } (A \text{ NAND } B) &= \overline{(\overline{(A \cdot B)}) \text{ NAND } (\overline{(A \cdot B)})} \\
 &= \overline{((\overline{A \cdot B}) \cdot (\overline{A \cdot B}))} \\
 &= \overline{(\overline{A \cdot B})} \quad \text{Since } A \cdot A = A \\
 &= A \cdot B \quad \text{Since } \overline{(\overline{A})} = A
 \end{aligned}$$

Table 2.29 shows the proof for obtaining AND gate using NAND gate with the help of the truth table.

A	B	A.B	$\overline{(A \cdot B)}$	$\overline{(A \cdot B)} \cdot \overline{(A \cdot B)}$	$\overline{((\overline{A \cdot B}) \cdot (\overline{A \cdot B}))}$
0	0	0	1	1	0
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	0	0	1

Table 2.29 : Proof using truth table

**OR gate using NAND gate**

The OR gate is replaced by a NAND gate with all its inputs complemented by NAND gate inverters as shown in Figure 2.22.

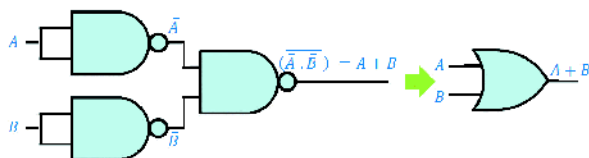


Fig. 2.22 : OR gate using NAND gate

**Proof:**

$$\begin{aligned}
 A \text{ NAND } A &= \overline{(A \cdot A)} \\
 &= \overline{A}
 \end{aligned}$$

$$\text{Similarly, } B \text{ NAND } B = \overline{B}$$

$$\begin{aligned}
 \text{Therefore, } (A \text{ NAND } A) \text{ NAND } (B \text{ NAND } B) &= \overline{\overline{A}} \text{ NAND } \overline{\overline{B}} \\
 &= \overline{(\overline{\overline{A}} \cdot \overline{\overline{B}})} \\
 &= \overline{\overline{A}} + \overline{\overline{B}} \quad \text{Since } \overline{(\overline{A \cdot B})} = \overline{\overline{A}} + \overline{\overline{B}} \\
 &= A + B \quad \text{Since } \overline{(\overline{A})} = A
 \end{aligned}$$

Table 2.30 shows the proof for obtaining OR gate using NAND gate with the help of truth table.

A	B	$\bar{A}$	$\bar{B}$	$\bar{A} \cdot \bar{B}$	$\overline{(\bar{A} \cdot \bar{B})}$	$A + B$
0	0	1	1	1	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	0	1	1

Table 2.30 : Proof using truth table

Thus, the NAND gate is a universal gate since it can implement AND, OR and NOT operations. Now let us see implementation of basic gates by using another universal gate, the NOR gate.

### NOT gate using NOR gate

We can implement a NOT gate (inverter) using a NOR by applying the same signal to both inputs of a NOR gate as shown in Figure 2.23.

**Proof:**

$$\begin{aligned} A \text{ NOR } A &= \overline{(A + A)} \\ &= \bar{A} \text{ Since } A + A = A \end{aligned}$$

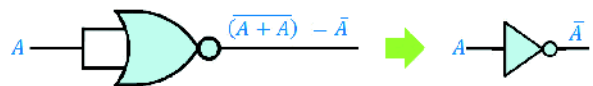


Fig 2.23 : NOT gate using NOR gate

Table 2.31 shows the proof for obtaining NOT gate using NOR gate with the help of truth table.

A	A+A	$\overline{(A + A)}$	$\bar{A}$
0	0	1	1
1	1	0	0

Table 2.31 : Proof using truth table

### OR gate using NOR gate

We can implement an OR gate by using a NOR gate followed by another NOR gate to invert the output as shown in Figure 2.24.

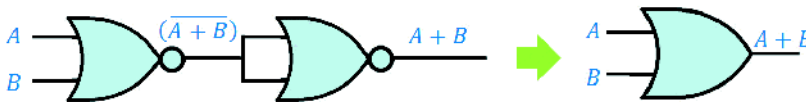


Fig 2.24: OR gate using NOR gate

**Proof:**

$$\text{We know that } A \text{ NOR } B = \overline{(A + B)}$$

$$\begin{aligned} (A \text{ NOR } B) \text{ NOR } (A \text{ NOR } B) &= \overline{(\overline{(A + B)})} \text{ NOR } \overline{(\overline{(A + B)})} \\ &= \overline{((\overline{(A + B)}) + (\overline{(A + B)}))} \end{aligned}$$





$$= ((\overline{\overline{A+B}})) \quad \text{Since } A+A=A$$

$$= A+B \quad \text{Since } (\overline{\overline{A}})=A$$

Table 2.32 shows the proof of obtaining OR gate using NOR gate with the help of the truth table.

A	B	A+B	$(\overline{A+B})$	$(\overline{A+B}) + (\overline{A.B})$	$((\overline{A+B}) . (\overline{A.B}))$
0	0	0	1	1	0
0	1	1	0	0	1
1	0	1	0	0	1
1	1	1	0	0	1

Table 2.32 : Proof using truth table

### AND gate using NOR gate

The AND gate is replaced by a NOR gate with all its inputs complemented by NOR gate inverters as shown in Figure 2.25.

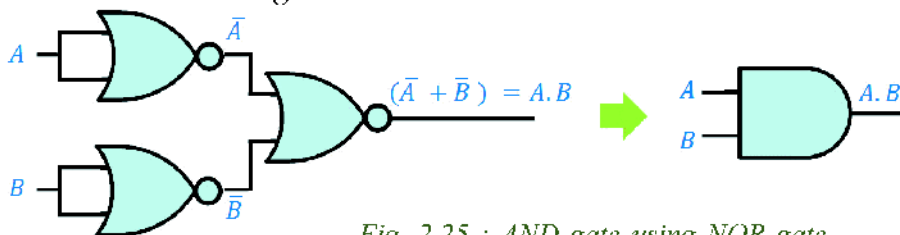


Fig. 2.25 : AND gate using NOR gate

#### Proof

$$A \text{ NOR } A = (\overline{A+A}) = \overline{A}$$

$$\text{Similarly, } B \text{ NOR } B = (\overline{B+B}) = \overline{B}$$

$$\begin{aligned} \text{Therefore, } (A \text{ NOR } A) \text{ NOR } (B \text{ NOR } B) &= \overline{A} \text{ NOR } \overline{B} \\ &= \overline{(\overline{A} + \overline{B})} \\ &= (\overline{\overline{A}}) . (\overline{\overline{B}}) \quad \text{Since } (\overline{A+B}) = \overline{A} . \overline{B} \\ &= A.B \quad \text{Since } \overline{\overline{A}} = A \end{aligned}$$

Thus, the NOR gate is also a universal gate since it can implement the AND, OR and NOT operations. Table 2.33 represents the proof for obtaining AND gate using NOR gate with the help of truth table.

A	B	$\overline{A}$	$\overline{B}$	$\overline{A+B}$	$(\overline{\overline{A+B}})$	A.B
0	0	1	1	1	0	0
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	0	1	1

Table 2.33 : Proof using truth table

**Check yourself**

1. Draw logic circuits for the Boolean expression  $X + \bar{Y}$ .
2. Which gates are called universal gates?
3. \_\_\_\_\_ gate produces low (0) output if any one of the input is high(1)  
 (a) OR                      (b) AND                      (c) NAND                      (d) NOR
4.  $A \text{ NAND } B = \underline{\hspace{2cm}}$ .  
 (a)  $A+B$                       (b)  $A.B$                       (c)  $(\overline{A+B})$                       (d)  $(\overline{A.B})$

**Let us sum up**

Different methods of data representation were discussed in this chapter. Before discussing data representation of numbers, we introduced different number systems and their conversions. After the discussion of integer and floating point number representation we have mentioned different methods for character, sound, image and sound data representation. We have also discussed the concept of Boolean algebra in detail including logical operators, logic gates and laws of Boolean algebra. We concluded the chapter by introducing methods for designing basic logic circuits and by discussing the importance of universal gates in circuit designing.

**Learning outcomes**

After the completion of this chapter the learner will be able to

- explain the characteristics of different number systems.
- convert one number system to another.
- perform binary arithmetic.
- represent numbers and characters in computer memory.
- list the formats of sound, image and video file formats.
- identify the concept of Boolean algebra.
- explain the working of logical operators and logic gates with the help of examples.
- state and prove basic postulates and laws of Boolean algebra.
- design circuits for simple Boolean expressions.
- implement basic gates using universal gates.

**Sample questions****Very short answer type**

- What is the place value of 9 in  $(296)_{10}$ ?
- Find octal equivalent of the decimal number 55.
- Find missing terms in the following series
  - $101_2, 1010_2, 1111_2, \underline{\hspace{2cm}}, \underline{\hspace{2cm}}$
  - $15_8, 16_8, 17_8, \underline{\hspace{2cm}}, \underline{\hspace{2cm}}$
  - $18_{16}, 1A_{16}, 1C_{16}, \underline{\hspace{2cm}}, \underline{\hspace{2cm}}$
- If  $(X)_2 - (1010)_2 = (1000)_2$  then find X.
- Name the coding system that can represent almost all the characters used in the human languages in the world.
- Find out the logical statement(s) from the following .
  - Why are you late?
  - Will you come with me to market?
  - India is my country.
  - Go to class room.
- List three basic logic gates.
- Which gate is called inverter?
- List two complementarity Laws.
- The Boolean expression  $\overline{(A+B)}$  represents \_\_\_\_\_ gate.
  - AND
  - NOR
  - OR
  - NAND

**Short answers type**

- Define the term data representation.
- What do you mean by a number system? List any four number systems.
- Convert the following numbers into the other three number systems:
  - $(125)_8$
  - 98
  - $(101110)_2$
  - $(A2B)_{16}$
- Find the equivalents of the given numbers in the other three number systems.
  - $(71.1)_{16}$
  - $(207.13)_8$
  - 93.25
  - $(10111011.1101)_2$
- If  $(X)_2 = (Y)_8 = (Z)_{16} = (28)_{10}$  Then find X, Y and Z.
- Arrange the following numbers in descending order
  - $(101)_{16}$
  - $(110)_{10}$
  - $(111000)_2$
  - $(251)_8$
- Find X, if  $(X)_2 = (10111)_2 + (11011)_2 - (11100)_2$



8. What are the methods of representing integers in computer memory?
9. Represent the following numbers in sign and magnitude method, 1's complement method and 2's complement method  
a) -19                      b) +49                      c) -97                      d) -127
10. Find out the integer which is represented as  $(10011001)_2$  in sign and magnitude method.
11. Explain the method of representing a floating point number in 32 bit computer.
12. What are the methods of representing characters in computer memory?
13. Briefly explain the significance of Unicode in character representation.
14. Match the following:

A	B
i) If any input is 1 output is 1	a) NAND
ii) If an input is 0 output is 0	b) OR
iii) If any input is 0 output is 1	c) NOR
iv) If any input is 1 output is 0	d) AND

15. Find dual of following Boolean expressions  
a)  $X.Y+Z$                       b)  $A.C+A.1+A.C$                       c)  $(A+0).(A.1.\bar{A})$
16. Find complement of following Boolean expressions  
a)  $\bar{A} \bar{B}$                       b)  $\overline{A.B} + \overline{C.D}$
17. Construct logic circuit for the following Boolean expression.  
(i)  $\bar{a}\bar{b} + c$                       (ii)  $ab + \bar{a}b + \bar{a}\bar{b}$                       (iii)  $(a + \bar{b}).(\bar{a} + \bar{b})$
18. Why are NAND and NOR gates called universal gates? Justify with an example.

### Long answer type

1. Briefly explain different methods for representing numbers in computer memory.
2. Briefly explain different methods for representing characters in computer memory.
3. What are the file formats for storing image, sound and video data?
4. Give logic symbol, Boolean expression and truth table for three input AND gate.
5. Prove that NOR gate is a universal gate by implementing all the basic gates.

## Components of the Computer System

### Key concepts

- **Data processing**
- **Functional units of a computer**
- **Hardware**
  - Processors
  - Motherboard
  - Memory - primary storage, secondary storage
  - Use of memory in computer
  - Input output devices
- **e-Waste**
  - Disposal methods
  - Green computing
- **Software**
  - System software (operating system, language processors, utility software)
  - Application software (general purpose, specific purpose)
  - Free software and open source software
  - Freeware and shareware
  - Proprietary software
- **Humanware / Liveware**

We are familiar with computers and their uses in today's world. Computer can be defined as a fast electronic device that accepts data, processes it as per stored instructions and produces information as output. This chapter presents an overview of the basic design of a computer system: how the different parts of a computer system are organised and various operations are performed to do a specific task. We know that a computer has two major components - hardware and software. Hardware refers to all physical components associated with a computer system while software is a set of instructions for the hardware to perform a specific task. When we use computers to solve any problem in real life situations, we define the tasks required to process data for generating information. This chapter presents the concepts of data processing at first and discusses how the functional units of a computer help data processing. Then various hardware components are presented followed by electronic waste, its disposal methods and the concept of green-computing. A detailed classification of software is listed along with different types of computer languages. We will also discuss the concepts of open source, freeware, shareware and proprietary software.



### 3.1 Data processing

Data processing refers to the operations or activities performed on data to generate information. Data denotes raw facts and figures such as numbers, words, amount, quantity, etc. that can be processed or manipulated. Information is a meaningful and processed form of data. It adds to our knowledge and helps in making decisions. Data processing proceeds through six stages as in figure 3.1.

- Capturing data
- Input of data
- Storage of data
- Processing / manipulating data
- Output of information
- Distribution of information

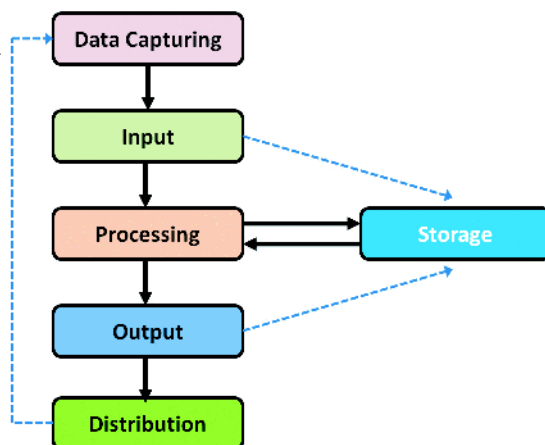


Fig. 3.1 : Data processing stages

Let us take a close look at these stages.

**Capturing Data:** This is the first stage in data processing. Here a proforma, known as the source document, is designed to collect data in proper order and format. This document is used for data collection.

**Input:** In this stage, the data collected through the source documents is fed to the computer for processing. But now a days, in many cases, the data are directly fed into the computer without using source documents.

**Storage:** In data processing, the input data are stored before processing. The information obtained after processing may also be stored.

**Process:** The data stored in computers is retrieved for processing. Various operations like calculation, classification, comparison, sorting, filtering, summarising, etc. may be carried out as part of processing.

**Output:** The processed data is obtained in this stage in the form of information. The output may be stored for future reference as it may be used for generating some other information in another context.

**Distribution of information:** The information obtained from the output stage is distributed to the beneficiaries. They take decisions or solve problems according to the information.

We have seen the activities involved in data processing. Computers are designed in such a way that it can be involved in these activities. Let us see how the functional units of a computer are organised.



### 3.2 Functional units of a computer

Even though computers differ in size, shape, performance and cost over the years, the basic organisation of a computer is the same. As we discussed in Chapter 1, it is based on a model proposed by John Von Neumann, a mathematician and computer scientist. It consists of a few functional units namely, Input Unit, Central Processing Unit, Storage Unit and Output Unit as shown in Figure 3.2. Each of these units is assigned to perform a particular task. Let us discuss the functions of these units.

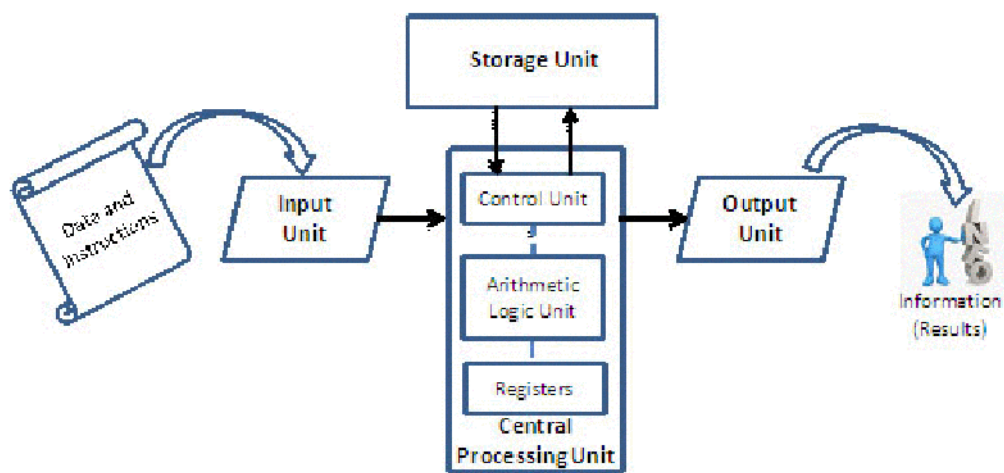


Fig. 3.2 : Basic organisation of computer

#### a. Input unit

The collected data and the instructions for their processing are entered the computer through the input unit. They are stored in the memory (storage unit). The data may be in different forms like number, text, image, audio, video, etc. A variety of devices are available to input the data depending on its nature. Keyboard, mouse, scanner, mic, digital camera, etc. are some commonly used input devices. In short, the functions performed by input unit are as follows:

1. Accepts instructions and data from the outside world.
2. Converts these instructions and data into a form acceptable to the computer.
3. Supplies the converted instructions and data to the computer for processing.

#### b. Central Processing Unit (CPU)

The CPU is the brain of the computer. In a human body, all major decisions are taken by the brain, and other parts of the body function as directed by the brain. Similarly, in a computer system, all major calculations and comparisons are made inside the CPU. It is also responsible for activating and controlling the operations of other units of the computer. The functions of CPU are performed by three components - Arithmetic Logic Unit (ALU), Control Unit (CU) and Registers.



### i. Arithmetic Logic Unit (ALU)

The actual operations specified in the instructions are carried out in the Arithmetic Logic Unit (ALU). It performs calculations and logical operations such as comparisons and decision making. The data and instructions stored in the storage unit are transferred to the ALU and the processing takes place in it. Intermediate results produced by the ALU are temporarily transferred back to the storage and are retrieved later when needed for further processing. Thus there is a data flow between the storage and the ALU many times before the entire processing is completed.

### ii. Control Unit (CU)

Each of the functional units has its own function, but none of these will perform the function until it is asked to. This task is assigned to the control unit. It invokes the other units to take charge of the operation they are associated with. It is the central nervous system that manages and coordinates all other units of the computer. It obtains instructions from the program stored in the memory, interprets the operation and issues signals to the unit concerned in the system to execute them.

### iii. Registers

These are temporary storage elements that facilitate the functions of CPU. There are variety of registers; each is designated to store unique items like data, instruction, memory address, results, etc.

## c. Storage unit

The data and instructions entered in the computer through input unit are stored inside the computer before actual processing starts. Similarly, the information or results produced after processing are also stored inside the computer, before transferring to the output unit. Moreover, the intermediate results, if any, must also be stored for further processing. The storage unit of a computer serves all these purposes. In short, the specific functions of storage unit are to hold or store:

1. Data and instructions required for processing.
2. Intermediate results for ongoing processing.
3. Final results of processing, before releasing to the output unit.

The storage unit comprises of two types of storages as detailed below:

- i. **Primary storage:** It is also known as main memory. It is again divided into two – Random Access Memory (RAM) and Read Only Memory (ROM). RAM holds instructions, data and intermediate results of processing. It also holds the recently produced results of the job done by the computer. ROM contains instructions for the start up procedure of the computer. The Central Processing



Unit can directly access the main memory at a very high speed. But it has limited storage capacity.

- ii. **Secondary storage:** It is also known as auxiliary storage and it takes care of the limitations of primary storage. It has huge storage capacity and the storage is permanent. Usually we store data, programs and information in the secondary storage, but we have to give instruction explicitly for this. Hard disk, CDs, DVDs, memory sticks, etc. are some examples.

#### d. Output unit

The information obtained after data processing is supplied to the outside world through the output unit in a human-readable form. Monitor and printer are the commonly used output devices. The functions performed by output unit can be concluded as follows:

1. Receives the results produced by the CPU in coded form.
2. Converts these coded results to human-readable form.
3. Supplies the results to the outside world.

### 3.3 Hardware

We know that a computer system consists of hardware and software. The term hardware represents the tangible and visible parts of a computer, which consists of some electro mechanical components. These hardware components are associated with the functional units of a computer. Let us discuss some of these components.

#### 3.3.1 Processors

In the previous section, we learned that CPU (Central Processing Unit) is responsible for all computing and decision making operations and coordinates the working of a computer. The performance of a CPU determines the overall performance of the computer. Since CPU is an Integrated Circuit (IC) package which contains millions of transistors and other components fabricated into a single silicon chip, it is also referred as microprocessor. Figure 3.3 shows the processors developed by some manufacturers. A CPU is usually plugged into a large socket on the main circuit board (the motherboard) of the computer. Since heat is generated when the CPU works, a proper cooling system is provided with a heat sink and fan. The commonly used processors are Intel core i3, core i5, core i7, AMD Quadcore, etc.

Registers are storage locations inside the CPU, whose contents can be accessed more quickly by the CPU than



Fig. 3.3 : Different Processors



Every computer contains an internal clock that regulates the rate at which instructions are executed. The CPU requires a fixed number of clock ticks (or clock cycles) to execute each instruction. The faster the clock, the more the instructions the CPU can execute per second. Another factor is the architecture of the chip. The number of bits the processor can process at one time is called word size. Processors with many different word sizes exist: 8-bit, 16-bit, 32-bit, 64-bit, etc.

other memory. Registers are temporary storage areas for instructions or data. They are not a part of memory; rather they are special additional storage locations that offer the advantage of speed. Registers work under the direction of the control unit to accept, hold and transfer instructions or data and perform arithmetic or logical operations at high speed. It speeds up the execution of programs. Important registers inside a CPU are:

- i. **Accumulator:** The accumulator is a part of the arithmetic/logic unit (ALU). This register is used to store intermediate result while performing arithmetic and logical operations. It is also called register A.
- ii. **Memory Address Register (MAR):** It stores the address of a memory location to which data is either to be read or written by the processor.
- iii. **Memory Buffer Register (MBR):** It holds the data, either to be written to or read from the memory by the processor.
- iv. **Instruction Register (IR):** The instructions to be executed by the processor are stored in the Instruction Register.
- v. **Program Counter (PC):** It holds the address of the next instruction to be executed by the processor.

### 3.3.2 Motherboard

A motherboard is a large Printed Circuit Board (PCB) to which all the major components including the processor are integrated. It also provides expansion slots for adding additional circuit boards like memory, graphics card, sound card, etc. (refer Figure 3.4). The motherboard must be compatible with the processor chosen.

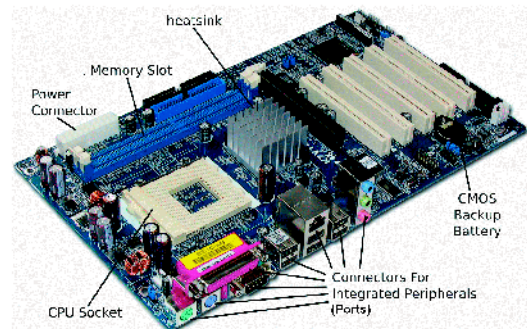


Fig. 3.4 : Motherboard

### 3.3.4 Peripherals and ports

Peripherals are devices that are attached to a computer system to enhance its capabilities. Ports on the motherboard are used to connect external devices. Peripherals include input and output devices, external storage and communication devices. These devices communicate with the motherboard through the ports like VGA, PS/2, USB, Ethernet, HDMI, etc. that are available on the motherboard. Figure 3.5 shows some ports used in personal computers.

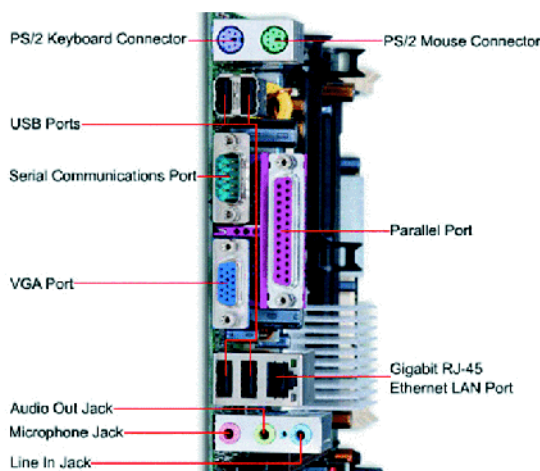


Fig. 3.5 : Ports

### 3.3.5 Memory

Memory is a place where we can store data, instructions and results temporarily or permanently. Memory can be classified into two - primary memory and secondary memory. Primary memory holds data, intermediate results and results of ongoing jobs temporarily. Secondary memory, on the other hand, holds data and information permanently. Before learning more about memory, let us discuss the different memory measuring units. The measuring units are:

Binary Digit = 1 Bit	1 MB (Mega Byte) = 1024 KB
1 Nibble = 4 Bits	1 GB (Giga Byte) = 1024 MB
1 Byte = 8 Bits	1 TB (Tera Byte) = 1024 GB
1 KB (Kilo Byte) = 1024 Bytes	1 PB (Peta Byte) = 1024 TB

#### a. Primary storage

Primary memory is a semiconductor memory that is accessed directly by the CPU. It is capable of sending and receiving data at high speed. This includes mainly three types of memory such as RAM, ROM and cache memory.

##### i. Random Access Memory (RAM)

RAM, shown in Figure 3.6 refers to the main memory that microprocessor can read from and write into. Data can be stored and retrieved at random from anywhere within the RAM, no matter where the data is. Data or instructions to be processed by the CPU must be placed in the RAM. The contents of RAM are lost when power is switched off. Therefore, RAM is a volatile memory. Storage capacity of RAM is 2 GB and above.

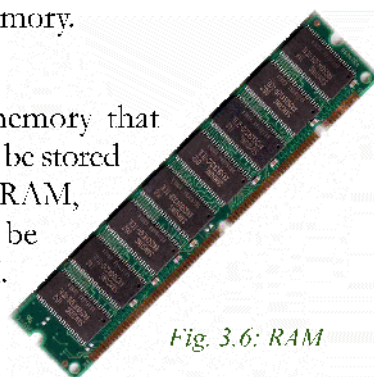


Fig. 3.6: RAM





The speed of a RAM refers to how fast the data in memory is accessed. It is measured in Mega Hertz (MHz). When a computer is in use, its RAM contains the following:

1. The operating system software.
2. The application software currently being used.
3. Any data that is being processed.

## ii. Read Only Memory (ROM)

ROM is a permanent memory that can perform only read operations and its contents cannot be easily altered. ROM is non-volatile; the contents are retained even after the power is switched off. ROM, shown in Figure 3.7, is used in most computers to hold a small, special piece of 'boot up' program known as Basic Input Output System (BIOS). This software runs when the computer is switched on or 'boots up'. It checks the computer's hardware and then loads the operating system. There are some modified types of ROM that include:



Fig.: 3.7 : ROM Chip

1. PROM - Programmable ROM which can be programmed only once.
2. EPROM - Erasable programmable ROM that can be rewritten using ultra violet radiation.
3. EEPROM - Electrically Erasable Programmable ROM which can be rewritten electrically.

Table 3.1 shows the comparison between RAM and ROM.

RAM	ROM
<ul style="list-style-type: none"> <li>• It is faster than ROM.</li> <li>• It stores the operating system, application programs and data when the computer is functioning.</li> <li>• It allows reading and writing.</li> <li>• It is volatile, i.e. its contents are lost when the device is powered off.</li> </ul>	<ul style="list-style-type: none"> <li>• It is a slower memory.</li> <li>• It stores the program required to boot the computer initially.</li> <li>• Usually allows reading only.</li> <li>• It is non-volatile, i.e. its contents are retained even when the device is powered off.</li> </ul>

Table 3.1 : RAM - ROM comparison

## iii. Cache memory

Cache memory is a small and fast memory between the processor and RAM (main memory). Frequently accessed data, instructions, intermediate results, etc. are stored in cache memory for quick access. When the processor needs to read from or write





to a location in RAM, it first checks whether a copy of that data is in the cache. If so, the processor immediately reads the cache, which is much faster than reading from the RAM. Cache is more expensive than RAM, but it is worth getting a CPU and motherboard with built-in cache in order to maximise system performance.

### b. Secondary or Auxiliary storage

Secondary memory is permanent in nature. Unlike the contents of RAM, the data stored in these devices does not vanish when power is turned off. Secondary memory is much larger in size than RAM, but is slower. It stores programs and data but the processor cannot access them directly. Secondary memory is also used for transferring data or programs from one computer to another. It can also act as a backup. The major categories of storage devices are magnetic, optical and semiconductor memory.

#### i. Magnetic storage devices

Magnetic storage devices use plastic tape or metal/plastic disks coated with magnetic materials. Data is recorded magnetically in these devices. Read/write heads are used to access data from these devices. Some of the popular magnetic storage devices are magnetic tapes, hard disks, etc.

#### ii. Optical storage devices

Optical disk is a data storage medium which uses low-powered laser beam to read and write data into it. It consists of an aluminum foil sandwiched between two circular plastic disks. Data is written on a single continuous spiral in the form of pits and lands. The laser beam reads this pits and lands as 0s and 1s. Optical disks are very cheap to produce in large quantities and are popular secondary storage media. The main types of optical disks are CD, DVD and Blu-Ray.

#### iii. Semi-conductor storage (Flash memory)

Flash drives use EEPROM chips for data storage. They do not contain any moving parts and hence they are shockproof. Flash memory is faster and durable when compared to other types of secondary memory. The drawback is that they are limited to a certain number of write cycles. The different variants of flash memories are USB flash drives and flash memory cards. Figure 3.8 shows different types of flash memories.

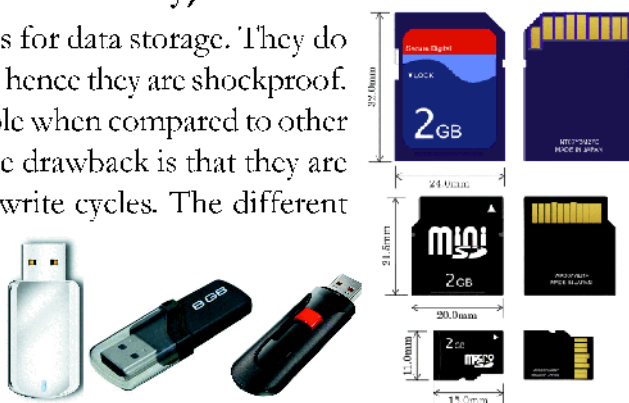


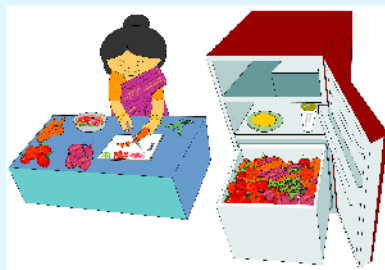
Fig. 3.8: Flash drive and memory cards



To see how registers, primary memory and secondary storage work together in a computer, let us use the analogy of making a salad in our kitchen. Suppose we have:

- A refrigerator where we store vegetables for the salad
- A counter where we place all vegetables before putting them on the cutting board for chopping.
- A cutting board on the counter where we chop vegetables.
- A recipe that details what vegetables to chop.
- The corners of the cutting board are kept free for partially chopped piles of vegetables that we intend to chop more or to mix with other partially chopped vegetables.
- A bowl on the counter where we mix and store the salad.
- Space in the refrigerator to put the mixed salad after it is made.

The process of making the salad is then: bring the vegetables from the fridge to the counter top; place some vegetables on the chopping board according to the recipe; chop the vegetables, possibly storing some partially chopped vegetables temporarily on the corners of the cutting board; place all the chopped vegetables in the bowl and keep it back in the fridge if not served on the dinner table.



In this context the refrigerator serves as secondary (hard disk) storage. It can store high volumes of vegetables for long periods of time. The counter top functions like the computer's motherboard - everything is done on the counter (inside the computer). The cutting board is the ALU - the work gets done there. The recipe is the control unit - it tells you what to do on the cutting board (ALU). Space on the counter top is the equivalent of RAM - all required vegetables must be brought from the fridge and placed on the counter top for fast access. Note that the counter top (RAM) is faster to access than the fridge (disk), but cannot hold as much, and cannot hold it for long periods of time. The corners of the cutting board where we temporarily store partially chopped vegetables are equivalent to the registers. The corners of the cutting board are very fast to access for chopping, but cannot hold much. The salad bowl is like a cache memory, it is for storing chopped vegetables to be temporarily removed from the corners of the cutting board (as there is too much) or the salad waiting to be taken back to the fridge (putting data back on a disk) or to the dinner table (outputting the data to an output device).

### 3.3.6 Role of memories in computers

Let us consider the case of a payroll program to calculate the salary of an employee. The data for all the employees is available in the hard disk. All the data about a particular employee is taken to the RAM and from there data related to salary calculation - bonuses, deductions, etc. is taken to the cache. The data representing the hours worked and the rate of pay is moved to their respective registers. Using data on the hours worked and the rate of pay, ALU makes calculations based on instructions from control unit. For further calculations, it moves the overtime hours, bonuses, etc. from cache to registers. As the CPU finishes calculations about one employee, the data about the next employee is brought from secondary storage into RAM, then cache and eventually into the registers.

Figure 3.9 depicts the hierarchy of different memories according to the storage capacity and access speed.

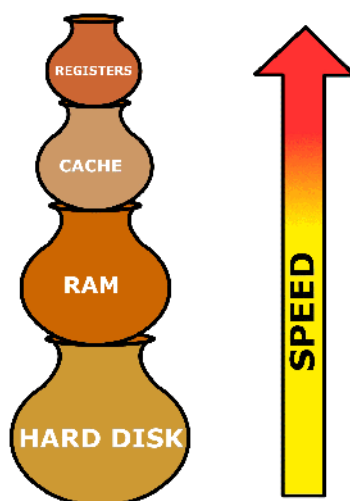


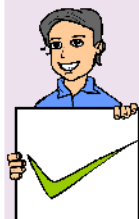
Fig. 3.9 : Memory hierarchy

Table 3.2 summarises the characteristics of various kinds of data storage in the storage hierarchy.

Storage	Speed	Capacity	Relative Cost	Volatile
<b>Registers</b>	Fastest	Lowest	Highest	Yes
<b>Cache</b>	More Fast	Low	Very High	Yes
<b>RAM</b>	Very Fast	Low/Moderate	High	Yes
<b>Hard Disk</b>	Moderate	Very High	Very Low	No

Table 3.2 : Comparison of different characteristics of various types of memories

### Check yourself





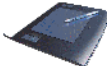




1. The fastest memory in a computer is \_\_\_\_\_.
2. Define data processing.
3. What is cache memory?
4. What is the use of program counter register?
5. What is the use of ALU?



### 3.3.7 Input/Output devices

The computer will be of no use unless it is able to communicate with the outside world. Input/output devices are required for users to communicate with the computer. In simple terms, input devices feed data and instructions into the computer and output devices presents information from a computer system. These input/output devices are connected to the CPU through various ports or with the help of wireless technology. Since they reside outside the CPU, they are called peripherals. The following table shows various **input devices** and their uses.

Device	Features / Uses
<b>Keyboard</b> 	Allows the user to input text data consisting of alphabets, numbers and other characters. Detects the key pressed and generates the corresponding ASCII code which can be recognised by the computer. Wired and wireless keyboards are available.
<b>Mouse</b> 	A small handheld device used to position the cursor or move the pointer on the computer screen by rolling it over a mouse pad / flat surface. Different types of mouse are ball, optical and laser mouse. Wireless mouse is also available.
<b>Light pen</b> 	A pointing device shaped like a pen. It has the advantage of 'drawing' directly onto the screen. Used by engineers, artists, fashion designers for Computer Aided Designing (CAD) and drawing purposes.
<b>Touch screen</b> 	Allows the user to operate/make selections by simply touching on the display screen. It can also be operated using a stylus which gives more precision.
<b>Graphic tablet</b> 	Consists of an electronic writing area and a special 'pen' that works with it. Allows artists to create graphical images with actions similar to traditional drawing tools.
<b>Joystick</b> 	Used to playing video games, control training simulators and robots. Has a vertical stick which can move in any direction and a button on top that is used to select the option pointed by the cursor.
<b>Microphone</b> 	Accepts sound which is analogue in nature as input and converts it to digital format. The digitised sound can be stored in the computer for later processing or playback.

**Scanner**

Allows capturing of information, like pictures or text and converting it into a digital format that can be edited using a computer. Quality of the image depends on the resolution of the scanner. Different variants of scanners are flat bed, sheet feed and hand held scanner. Optical Character Recognition (OCR) software is used to recognise the text in an image scanned and convert it into text, which can be edited by a text editor.

**Optical Mark Reader (OMR)**

Another scanning device that reads predefined positions and records where marks are made on the printed form. Useful for applications in which large numbers of hand-filled forms need to be processed quickly with great accuracy, such as objective type tests and questionnaires.

**Barcode/Quick Response (QR) code reader**

A bar code is a set of vertical lines of different thicknesses and spacing that represent a number. Barcode readers are used to input data from such set of barcodes. Hand-held scanners, mobile phones with camera and special software are used as barcode readers. QR (Quick Response) code is similar to barcodes. Barcodes are single dimensional where as QR codes are two dimensional. The two dimensional way of storing data allows QR code to store more data than a standard barcode. This code can store website URIs, plain text, phone numbers, email addresses and any other alphanumeric data. The QR code can be read using a barcode reader or a mobile with a camera and special software installed.

**Magnetic Ink Character Recognition (MICR) Reader**

MICR readers are used in banks for faster electronic clearing of cheques. The lower portion of a cheque contains cheque number, branch code, bank code, etc. printed in a special font using an ink containing iron oxide particles. Iron oxide has magnetic properties. MICR reader can easily recognise these characters by magnetically charging them while scanning. This MICR data along with the image of the cheque is sent to the cheque drawer's (the person who issues the cheque) branch to transfer the amount. This reduces errors in data entry and speeds up money transfer.

**Biometric sensor**

Identifies unique human physical features with high accuracy. It is an essential component of a biometric system which uses physical features like fingerprints, retina, iris patterns, etc. to identify, verify and authenticate the identity of the user.









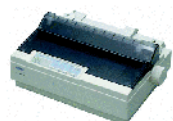





<b>Smart card reader</b> 	Smart card is a plastic card that stores and transacts data. It may contain a memory or a micro processor. Used in banking, healthcare, telephone calling, electronic cash payments and other applications. These are used to access data in a smart card.
<b>Digital camera</b> 	Takes pictures and videos and converts it to the digital format. The images are stored in the memory and can be transferred to computer. Web camera is a compact and less expensive version of a digital camera. It is used in computers for video calling, video chatting, etc. It does not have an internal memory.

Table 3.3 : Input devices and their uses

Now let us see some **output devices** and their features. Table 3.4 shows various output devices and their uses.

Device	Features / Uses
<b>Visual Display Unit (VDU)</b> 	Display devices include CRT monitors, LCD monitors, TFT monitors, LED monitors, gas plasma monitors, Organic Light Emitting Diode (OLED) Monitors, etc. Information shown on a display device is called soft copy. The size of a monitor is measured diagonally across the screen, in inches.
<b>LCD projector</b> 	An LCD projector is a type of video projector for displaying video, images or computer data on a large screen or other flat surface. A beam of high-intensity light which travels through thousands of shifting pixels in an LCD is focused by a lens on the surface.
<b>Printer</b>   	Used to produce hardcopy output. The output printed on paper is known as hardcopy. Classified as Impact or Non-impact printers. Dot-matrix uses impact mechanism. It can print carbon copies with less printing cost. Speed is measured in number of characters printed in a unit of time and is represented as characters per second (cps), lines per minute (lpm) or pages per minute (ppm). These printers are slow and noisy. Inkjet printers are non-impact printers that form the image on the page by spraying tiny droplets of ink from the print head. Ink jet printers are inexpensive, but the cost of ink cartridges makes them costly to operate in the long run. Laser printers are non-impact printers that

	<p>produce good quality images. Monochrome and color laser printers are available. Color laser printers use multiple color toner cartridges to produce color output and are expensive. Laser printers are faster and their speed is rated in pages per minute (ppm). Thermal printer is a non-impact printer that produces a printed image by selectively heating heat sensitive thermal paper when it passes over the thermal print head. The coating turns black in the areas where it is heated, producing an image. It is popular as a portable printer.</p>
<p><b>Plotter</b></p> 	<p>A plotter is an output device used to produce hardcopies of graphs and designs on the paper. A plotter is typically used to print large-format graphs or maps such as construction maps, engineering drawings and big posters. Plotters are of two types: Drum plotters and Flatbed plotters. A drum plotter is also known as Roller plotter. A flatbed plotter is also known as Table plotter.</p>
<p><b>3D printer</b></p> 	<p>A 3D printer is an output device used to print 3D objects. It can produce different kinds of objects, in different materials, using the same printer. The 3D printing process turns the object to be printed, into thousands of tiny little slices. It then prints it from the bottom to top, slice by slice. Those tiny layers stick together to form a solid object.</p>
<p><b>Audio output device</b></p> 	<p>The audio output is the ability of the computer to produce sound. Speakers are the output devices that produces sound. It is connected to the computer through audio ports.</p>

*Table 3.4 : Output devices and their uses*

We have seen different types of printers. Table 3.5 shows a comparison on various characteristics of these printers.

Features	Laser Printers	Inkjet Printers	Thermal Printers	Dot Matrix Printers
<b>Printing material used</b>	Ink powder	Liquid ink	Heat sensitive paper	Ink soaked ribbon
<b>How it prints</b>	It fuses the powder on the paper through heating.	It sprays liquid ink on paper through microscopic nozzles.	Thermal paper is passed over the thermal print head.	Pins are pushed against ribbon on paper.





<b>Printing speed</b>	20 pages per minute	6 pages per minute	150 mm per second	30-550 characters per second
<b>Quality</b>	Printing quality is good. Best for black and white.	Printing quality is good, especially for smaller fonts.	Poor quality printing of images. Good quality text printing.	Poor printing quality for images. In terms of text, printing is good.
<b>Advantages</b>	Quiet, prints faster, high print quality.	Quiet, high print quality, no warm up time, device cost is less.	Quiet, fast, smaller, lighter, consume less power and portable.	Cheaper to print as ribbon is cheap. Carbon copy possible.
<b>Disadvantages</b>	More susceptible to paper jams. Toner is expensive. Device itself is expensive.	Ink is expensive, ink is not waterproof and nozzle is prone to clogging.	Requires special thermal quality paper. Poor quality printing.	Initial purchase is expensive, prints are not fast, makes noise.

Table 3.5 : Comparison of printers

### 3.4 e-Waste

e-Waste refers to electronic products nearing the end of their 'useful life'. Electronic waste may be defined as discarded computers, office electronic equipment, entertainment devices, mobile phones, television sets and refrigerators. The used electronics which are destined for reuse, resale, salvage, recycling or disposal are also considered as e-waste.

Nowadays electronics is part of modern life – desktops, laptops, cell phones, refrigerators, TVs and a growing number of other gadgets. Every year we buy new, updated equipments to satisfy our needs. More than 300 million computers and one billion cell phones are produced every year. All of these electronics goods become obsolete or unwanted, often, within two or three years of purchase. This global mountain of waste is expected to continue growing at 8% per year.

Rapid changes in technology, changes in media, falling prices and planned obsolescence have resulted in a fast-growing surplus of electronic waste around the globe. It is estimated that 50 million tons of e-Waste are produced each year. Only 15-20% of e-Waste is recycled, the rest of these materials go directly into landfills and incinerators. Sale of electronic products in countries such as India and China and across continents such as Africa and Latin America are set to rise sharply over the next 10 years.

### 3.4.1 Why should we be concerned about e-Waste?

Electronic waste is not just waste. It contains some very toxic substances, such as mercury, lead, cadmium, brominated flame retardants, etc. The toxic materials can cause cancer, reproductive disorders and many other health problems, if not properly managed. It has been estimated that e-Waste may be responsible for up to 40% of the lead found in landfills. Important hazardous chemicals, their sources and consequences are listed in Table 3.6.

Chemical	Source	Consequence
Lead	Found as solder on printed circuit boards and in computer monitor glass.	Lead can cause damage to the central and peripheral nervous systems, blood systems and kidneys in humans.
Mercury	Found in printed circuit boards, LCD screen backlights.	Affect a baby's growing brain and nervous system. Adults can suffer organ damage, mental impairment and a variety of other symptoms.
Cadmium	Found in chip resistors and semiconductors.	Cause various types of cancer. Cadmium can also accumulate in the kidney and harm it.
BFRs-Brominated Flame Retardants	Found in printed circuit boards and some plastics.	These toxins may increase the risk of cancer.

Table 3.6 : Hazardous chemicals, its source and consequence

### 3.4.2 What happens to the e-Waste?

Unfortunately, an incredibly small percentage of e-waste is recycled. Even when we take it to a recycling center it is often not actually recycled – in the way most of us expect. CRTs have a relatively high concentration of lead and phosphors both of which are necessary for the display. The United States Environmental Protection Agency (EPA) includes discarded CRT monitors in its category of 'hazardous household waste'.

The majority of e-Waste is most often dumped or burned – either in formal landfills and incinerators or informally dumped or burned. These inappropriate disposal methods for electronic waste fail to reclaim valuable materials or manage the toxic materials safely. In effect, our soil, water and air are easily contaminated.

e-Wastes should never be disposed with garbage and other household wastes. This should be segregated at



Fig. 3.10 : Defective and obsolete electronic items



the site and sold or donated to various organisations. Considering the severity of the e-Waste problem, it is necessary that certain management options be adopted by government, industries and the public to handle the bulk e-Wastes.

Realising the growing concern over e-Waste, Central Pollution Control Board (CPCB) of Government of India has formulated 'The e-Waste (Management & Handling) Rules, 2011' and are effective from 01-05-2012. These rules shall apply to every producer, consumer, collection centre, dismantler and recycler of e-Waste involved in the manufacture, sale and processing of electrical and electronic equipment or components. The implementation and monitoring of these guidelines shall be done by the State Pollution Control Boards concerned.

Government of Kerala has introduced strict measures for safe collection and disposal of e-Waste through a government order. The government has defined the role of manufacturers, local bodies and the Pollution Control Board (PCB) in safe disposal of e-Waste. Under the Extended Producer Responsibility, manufacturers of electrical and electronic goods will be required to take back used products from consumers directly or through agents or introduce buyback arrangement. They will also have to supply the e-Waste to authorised recycling units. Consumers have been directed to return used products of known brands to the manufacturers or deposit them at the collection centres set up by local bodies. The PCB will be required to identify agencies for recycling or disposal of e-Waste and organise awareness programmes on e-Waste disposal.

### 3.4.3 e-Waste disposal methods

The following methods can be used for disposing e-Waste.

- a. **Reuse:** It refers to second-hand use or usage after the equipment has been upgraded or modified. Most of the old computers are passed on to relatives/friends or returned to retailers for exchange or for money. Some computers are also passed on to charitable institutions, educational institutions, etc. Inkjet cartridges and laser toners are also used after refilling. This method reduces the volume of e-Waste generation.
- b. **Incineration:** It is a controlled and complete combustion process in which the waste is burned in specially designed incinerators at a high temperature in the range of 900 to 1000 degree Celsius.
- c. **Recycling of e-Waste:** Recycling is the process of making or manufacturing new products from a product that has originally served its purpose. Monitors, keyboards, laptops, modems, telephone boards, hard drives, compact disks, mobiles, fax machines, printers, CPUs, memory chips, connecting wires and cables can be recycled.
- d. **Land filling:** It is one of the widely used but not recommended method for the disposal of e-Waste.



### Role of students in e-Waste disposal

- Stop buying unnecessary electronic equipments.
- When electronic equipments get faulty try to repair it instead of buying a new one.
- Try to recycle electronic equipments by selling them or donating them to others extending their useful life and keeping them out of the waste stream.
- If you really need to buy new electronics, choose items with less hazardous substances, greater recycled content, higher energy efficiency, longer life span, and those that will produce less waste.
- Visit the manufacturer's website or call the dealer to find out if they have a take back programme or scheme for your discarded electronics.
- If the device is battery-operated, buy rechargeable instead of disposable batteries.
- Buy products with good warranty and take back policies.

### 3.4.4 Green computing or Green IT

Green computing is the study and practice of environmentally sustainable computing or IT. Green computing is the designing, manufacturing, using and disposing of computers and associated components such as monitors, printers, storage devices, etc., efficiently and effectively with minimal or no impact on the environment.

One of the earliest initiatives towards green computing was the voluntary labelling program known as 'Energy Star'. It was conceived by the Environmental Protection Agency (EPA) in 1992 to promote energy efficiency in hardware of all kinds. The Energy Star label has become a common sight, especially in notebook computers and displays. Similar programmes have been adopted in Europe and Asia. The commonly accepted Energy Star symbol is shown in Figure 3.11.

Government regulation is only a part of an overall green computing idea. The work habits of computer users and business firms have to be modified to minimise adverse impact on the global environment. Here are some steps that can be taken:



Fig. 3.11 : Energy Star label

- Turn off computer when not in use.
- Power-on the peripherals such as laser printers only when needed.
- Use power saver mode.
- Use laptop computers rather than desktop computers whenever possible.
- Take printouts only if necessary.
- Use liquid crystal display (LCD) monitors rather than cathode ray tube (CRT) monitors.
- Use hardware/software with Energy Star label.
- Dispose e-Waste according to central, state and local regulations.
- Employ alternative energy sources like solar energy.

The environmentally responsible and eco-friendly use of computers and their resources is known as green computing.

### How to make computers green?

The features that are important in making a computer greener include size, efficiency and materials. Smaller computers are greener because they use fewer materials and require less electricity to run. Efficient use of energy is also an important component of a green computer. Smaller computers such as laptops are more energy-efficient than bigger models and LCD screens use much less energy than the older CRT models. The use of hazardous materials such as lead and mercury should be minimised.

To promote green computing the following four complementary approaches are employed:

**Green design:** Designing energy-efficient and eco-friendly computers, servers, printers, projectors and other digital devices.



**Green manufacturing:** Minimising waste during the manufacturing of computers and other components to reduce the environmental impact of these activities.

**Green use:** Minimising the electricity consumption of computers and peripheral devices and using them in an eco-friendly manner.

**Green disposal:** Reconstructing used computers or appropriately disposing off or recycling unwanted electronic equipment.

### Check yourself



1. The environmentally responsible and eco-friendly use of computers and their resources is known as \_\_\_\_\_.
2. The process of making or manufacturing new products from the product that has originally served its purpose is called \_\_\_\_\_.
3. The labelling programme to promote energy efficiency in computers and their resources is called \_\_\_\_\_.
4. List any two input and output devices each.



Let us do

1. Conduct a survey in your locality to study the impact of e-Waste on the environment and health of the people and write a report.
2. Discuss the importance of green computing.





### 3.5 Software

Software is a general term used to denote a set of programs that help us to use the computer system and other electronic devices efficiently and effectively. If hardware is said to form the body of a computer system, software is its mind or soul. There are two types of software:

- System software
- Application software

#### 3.5.1 System software

It is a set of one or more programs designed to control the operations of a computer. They are general programs designed to assist humans in the use of computer system by performing tasks such as controlling the operations, move data into and out of a computer system and to do all the steps in executing application programs. In short, system software supports the running of other software, its communication with other peripheral devices. It helps users to use computer in an effective manner. It implies that system software helps to manage resources of the computer. Figure 3.12 depicts how system software interfaces between user and hardware.

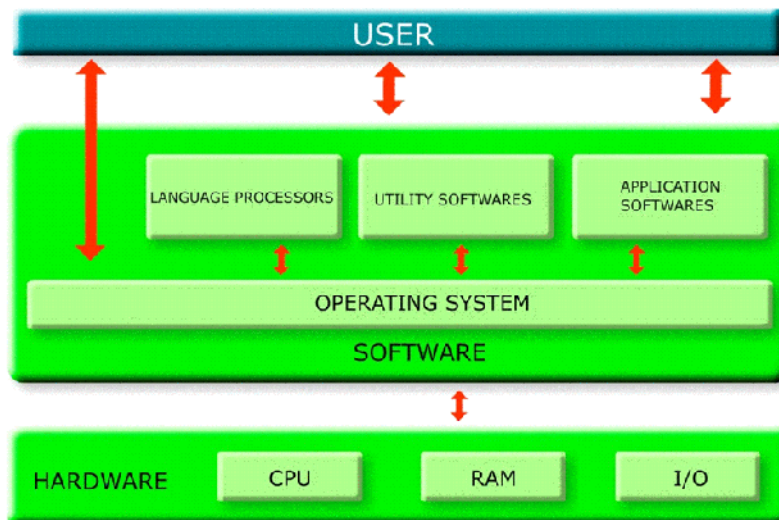


Fig. 3.12: Software with user and hardware interface

System software is a set of system programs which aids in the execution of a general user's computational requirements on a computer system. The following are the components of system software:

- a. Operating system
- b. Language processors
- c. Utility software

### a. Operating system

Operating system is a set of programs that acts as an interface between the user and computer hardware. The primary objective of an operating system is to make the computer system convenient to use. Operating system provides an environment for user to execute programs. It also helps to use the computer hardware in an efficient manner.

Operating system controls and co-ordinates the operations of a computer. It acts as the resource manager of the computer system as shown in Figure 3.13. Operating system is the most important system software. It is the first program to be loaded from hard disk in the computer and it resides in the memory till the system is shut down. It tries to prevent errors and the improper use of computer.

The major functions of an operating system are process management, memory management, file management, security management and command interpretation.

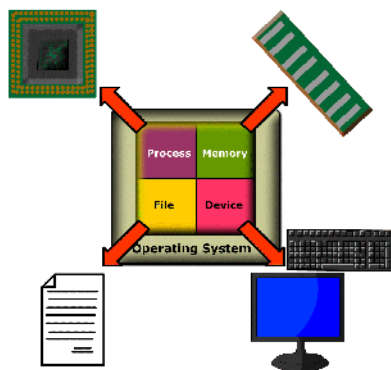


Fig. 3.13 : Operating System as a resource manager

#### i. Process management

By the term process we mean a program in execution. The process management module of an operating system takes care of the allocation and deallocation of processes and scheduling of various system resources to the different requesting processes.

#### ii. Memory management

Memory management is the functionality of an operating system which handles or manages primary memory. It keeps track of each and every memory location whether it is allocated to some process or it is free. It calculates how much memory is to be allocated to each process and allocates it. It de-allocates memory if it is not needed further.

#### iii. File management

The file management module of an operating system takes care of file related activities such as organising, naming, storing, retrieving, sharing, protection and recovery.

#### iv. Device management

Device management module of an operating system performs the management of devices attached to the computer. It handles the devices by combining both hardware





and software techniques. The OS communicates with the hardware device via the device driver software. Examples of various operating systems are DOS, Windows, Unix, Linux, Mac OS X, etc.

### **b. Language processors**

We know that natural languages are the medium of communication among human beings. Similarly, in order to communicate with the computer, the user also needs to have a language that should be understood by the computer. Computer languages may be broadly classified into low level languages and high level languages.

Low-level languages are described as machine-oriented languages. In these languages, programs are written using the memory and registers available on the computer. Since the architecture of computer differs from one machine to another, there is separate low level programming language for each type of computer. Machine language and assembly language are the different low level languages.

**Machine language:** We know that a computer can understand only special signals, which are represented by 1s and 0s. These two digits are called binary digits. The language, which uses binary digits, is called machine language. Writing a program in machine language is definitely very difficult. It is not possible to memorise a long string of 0s and 1s for every instruction.

**Assembly language:** Assembly language is an intermediate-level programming language. Assembly languages use mnemonics. Mnemonic is a symbolic name given to an operation. For example ADD for addition operation, SUB for subtraction operation, etc. It is easier to write computer programs in assembly language as compared to machine language. It is machine dependent and programmer requires knowledge of computer architecture.

**High Level Languages (HLL):** These are like English languages and are simpler to understand than the assembly language or machine language. High level language is not understandable to the computer. A computer program written in a high level language is to be converted into its equivalent machine language program. So these languages require a language translator (compilers or interpreters) for conversion. Examples of high-level programming languages are BASIC, C, C++, Java, etc.

### **Need for language processor**

The programs consisting of instructions to the computer, written in assembly language or high level language are not understood by the computer. We need language processors to convert such programs into low level language, as computer can only understand machine language. Language processors are the system programs that translate programs written in high level language or assembly language into its equivalent machine language.



## Types of language processors

- **Assembler:** Assembly languages require a translator known as assembler for translating the program code written in assembly language to machine language. Because computer can interpret only the machine code instruction, the program can be executed only after translating. An assembler is highly machine dependent.
- **Interpreter:** Interpreter is another kind of language processor that converts a HLL program into machine language line by line. If there is an error in one line, it reports and the execution of the program is terminated. It will continue the translation only after correcting the error. BASIC is an interpreted language.
- **Compiler:** Compiler is also a language processor that translates a program written in high level language into machine language. It scans the entire program in a single run. If there is any error in the program, the compiler provides a list of error messages along with the line number at the end of the compilation. If there are no syntax errors, the compiler will generate an object file. Translation using compiler is called compilation. After translation compilers are not required in memory to run the program. The programming languages that have a compiler are C, C++, Pascal, etc.

Figure 3.14 shows process involved in the translation of assembly language and high level language programs into machine language programs

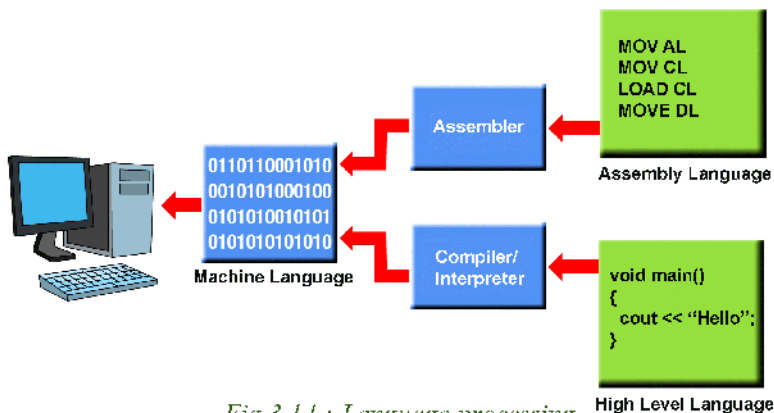


Fig.3.14 : Language processing

## c. Utility software

Utility software is a set of programs which help users in system maintenance tasks and in performing tasks of routine nature. Some of the utility programs with their functions are listed below:

- **Compression tools:** Large files can be compressed so that they take less storage area. These compressed files can be decompressed into its original form when needed. Compression of files is known as zipping and decompression is called unzipping. WinZip, WinRAR, etc. are examples.



- **Disk defragmenter:** Disk defragmenter is a program that rearranges files on a computer hard disk. The files are arranged in such a way that they are no longer fragmented. This enables the computer to work faster and more efficiently.
- **Backup software:** Backup means duplicating the disk information so that in an event of disk failure or in an event of accidental deletion, this backup may be used. Backup utility programs facilitates the backing up of disk.
- **Antivirus software:** A computer virus is a program that causes abnormality in the functioning of a computer. Antivirus software is a utility program that scans the computer system for viruses and removes them. As new viruses are released frequently, we have to make sure that latest antivirus versions are installed on the computer. Most of the antivirus programs provide an auto-update feature which enables the user to download profiles of new viruses so as to identify and inactivate them. Norton Antivirus, Kaspersky, etc. are examples of antivirus programs.

### 3.5.2 Application software

Software developed for specific application is called application software. It includes general purpose software packages and specific purpose software. GIMP, Payroll System, Air line reservation System, Tally, etc. are examples of application software.

#### a. General purpose software packages

General purpose software are used to perform tasks in a particular application area. Such software is developed keeping in mind the various requirements of its users. They provide a vast number of features for its users. General purpose software is classified as word processors, spreadsheet software, presentation software, database software and multimedia software.

- **Word processing software:** Word Processing software is designed for creating and modifying documents. It helps to create, edit, format and print textual matters easily. Formatting features include different font settings, paragraph settings, bullets and numbering, alignments and more. In addition to this it can check spelling and grammar in the document, insertion of pictures, charts and tables. We can specify headers and footers for every page in the document. The most popular examples of this type of software are MS Word, Open Office Writer, Apple iWork Pages, etc.
- **Spreadsheet software:** Spreadsheet software allows users to perform calculations using spreadsheets. They simulate paper worksheets by displaying multiple cells that make up a grid. It also allows us to insert drawing objects in the worksheet and create different types of charts for graphical representation



of numerical data. Microsoft Excel, Open Office Calc, Lotus 1-2-3 and Apple iWork Numbers are some examples of spreadsheet software.

- **Presentation software:** The software that is used to display information in the form of a slide show is known as presentation software. Presentation software allows preparing slides containing pictures, text, animation, video and sound effects. Microsoft PowerPoint, Apple iWork Keynote and Open Office Impress are examples for presentation software.
- **Database software:** Database is an organised collection of data arranged in tabular form. Database Management System (DBMS) consists of a collection of interrelated data and a set of programs to access those data. The primary goal of a DBMS is to provide an environment that is both convenient and efficient to use in retrieving and storing database information. They provide privacy and security to data and enforce standards for data. Examples of DBMS software are Microsoft Access, Oracle, Postgres SQL, My SQL, etc.
- **Multimedia software:** Multimedia is the integration of multiple forms of media. This includes text, graphics, audio, video, etc. Multimedia software can process information in a number of media formats. It is capable of playing media files. Some multimedia software allows users to create and edit audio and video files. Audio converters, audio players and video editing software are some forms of multimedia software. Examples are VLC Player, Adobe Flash, Real Player, Media Player, etc.

### b. Specific purpose software

Specific purpose software is a highly specialised software designed to handle particular tasks. These are tailor-made software to satisfy the needs of an organisation or institution. It is also known as customised software. Since custom software is developed for a single customer, it can accommodate that customer's particular preferences and expectations.

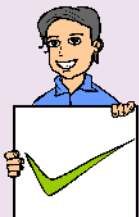
Some examples of specific purpose application software are listed in Table 3.7.

Application Software	Purpose
Payroll System	Payroll system maintains the details of employees of an organisation and keeps track of their salary details.
Inventory Management System	It is used for tracking inventory levels, orders, sales and deliveries in a business firm.
Human Resource Management System	It is used for managing human resource in an organisation.

Table 3.7 : Examples of application software

**Let us do**

- Discuss the classification of software.
- Compare and contrast the features of Linux and Windows operating systems with the help of your teacher and prepare short notes (Lab Demonstration). Discuss the role of utility software.
- Write short notes on the following:  
     *Language processors*  
     *General purpose software packages*

**Check yourself**

1. Define operating system.
2. Give two examples for OS.
3. A program in execution is called \_\_\_\_\_.
4. Mention any two functions of OS
5. Name the software that translates assembly language program into machine language program.
6. Name the two different language processors which translate high level language programs into machine language programs.
7. Differentiate between compiler and interpreter.
8. DBMS stands for \_\_\_\_\_.
9. Give two examples for customized software.
10. Duplicating disk information is called \_\_\_\_\_.

**3.5.3 Free and open source software**

Free and open source software gives the user the freedom to use, copy, distribute, examine, change and improve the software. Nowadays free and open source software is widely used throughout the world because of adaptable functionality, less overall costs, vendor independency, adherence to open standards, interoperability and security.

The Free Software Foundation (FSF) defines the four freedoms for free and open source software:

**Freedom 0** : The freedom to run program for any purpose.

**Freedom 1** : The freedom to study how the program works and adapt it to your needs. Access to source code should be provided.

**Freedom 2** : The freedom to distribute copies of the software.



**Freedom 3** : The freedom to improve the program and release your improvements to the public, so that the whole community benefits.

The following are some of the examples of free and open source software:

**GNU/Linux:** GNU/Linux is a computer operating system assembled under the model of free and open source software development and distribution. It was organised in the GNU project introduced in 1983 by Richard Stallman in the FSH.

**GIMP:** It stands for GNU Image Manipulation Program. It is an image editing software. It can be used for retouching photographs, creating and editing images. It supports graphic files of different formats and allows converting from one format to another.

**Mozilla Firefox:** It is one of the most popular web browsers created by the Mozilla Corporation. It provides added security features for safe browsing.

**OpenOffice.org:** It is a complete office suite that contains word processor (Writer) to prepare and format documents, spreadsheets (Calc) and presentations (Impress). It works on both Linux and Windows platforms.

### 3.5.4 Freeware and Shareware

Freeware refers to copyrighted computer software which is made available for use, free of charge, for an unlimited period.

The term shareware refers to commercial software that is distributed on a trial basis. It is distributed without payment and with limited functionality. Shareware is commonly offered in a downloadable format on the Internet. The distribution of this kind of software aims at giving users a chance to analyse the software before purchasing it. Some shareware works for a limited period of time only. Table 3.8 highlights a comparison between freeware and shareware:

Freeware	Shareware
<ul style="list-style-type: none"> <li>Freeware refers to software that anyone can download from the Internet and use for free.</li> <li>All the features are free.</li> <li>Freeware programs can be distributed free of cost.</li> </ul>	<ul style="list-style-type: none"> <li>Shareware gives users a chance to try the software before buying it.</li> <li>All features are not available. To use all the features of the software, user has to purchase it.</li> <li>Shareware may or may not be distributed freely. In many cases, author's permission is needed to distribute the shareware.</li> </ul>

Table 3.8 : Comparison of Freeware and Shareware



### 3.5.5 Proprietary software

Proprietary software is a computer program that is an exclusive property of its developer or publisher and cannot be copied or distributed without licensing agreements. It is sold without any access to source code and is therefore cannot be changed for improved by the user. Some examples of proprietary software are Microsoft Windows operating system, MS Office, Mac OS, etc.

### 3.6 Humanware or Liveware

Humanware or liveware refers to humans who use computer. It was used in computer industry as early as 1966 to refer to computer users, often in humorous contexts by analogy with software and hardware. It refers to programmers, systems analysts, operating staff and other personnel working in a computer system. Table 3.9 shows various categories of humanware and their job description.

Humanware	Job Description
System Administrators	Upkeep, configuration and reliable operation of computer systems; especially multi-user computers such as servers.
System Managers	Ensure optimal level of customer services and maintain expertise in all business unit systems and develop professional relationships with all vendors and contractors.
System Analysts	Design new IT solutions to improve business efficiency and productivity.
Database Administrators	Create, monitor, analyse and implement database solutions.
Computer Engineers	Design either the hardware or software of a computer system.
Computer Programmers	Write the code that computers read in order to operate properly.
Computer Operators	Oversee the running of computer systems, ensuring that the machines are running, physically secured and free of any bugs.

Table 3.9 : Categories of humanware with job description

#### Check yourself



1. An example of free and open source software is \_\_\_\_\_.
2. The software that give users a chance to try it before buying is \_\_\_\_\_.
3. What do you mean by free and open source software?
4. Give an example for proprietary software.
5. Give two examples of humanware.





## Let us sum up

Data processing is a series of activities by which data is converted into information. The limitations of manual data processing are overcome by electronic data processing and computer is the best electronic data processing machine. A computer has five functional units such as input unit, storage unit, arithmetic and logic unit, control unit and output unit. This chapter provided a general overall introduction to computer organisation. Input and output devices, e-waste and its disposal methods and the importance of green computing were introduced. The classification of software and the need of operating system in a computer with its major functions were discussed. Following this, the categories of computer languages were presented. The concepts of open source, freeware, shareware, free software and proprietary software were also discussed in detail. The chapter concluded outlining the concept of humanware.



## Learning outcomes

After the completion of this chapter the learner will be able to

- distinguish between data and information.
- identify various stages in data processing.
- explain basic organisation of computer system.
- recognise the different types of input and output devices.
- distinguish between system software and application software.
- identify the importance of e-Waste disposal.
- identify the importance of green computing concept.
- classify the different types of software.
- recognise the functions of operating system.
- use word processor, electronic spreadsheets and presentation software.
- classify the different types of computer languages.
- list the different types of utility software.
- promote open source software.
- explain the term humanware or liveware.

**Sample questions****Very short answer type**

1. What is data?
2. Processed data is known as \_\_\_\_\_.
3. What are the components of a digital computer?
4. Write the main functions of central processing unit.
5. What are the different types of main memory?
6. What is the advantage of EEPROM over EPROM?
7. When do we use ROM?
8. What is an input device? List few commonly used input devices.
9. What do you mean by an output device? List few commonly used output devices.
10. What is a storage device? List few commonly used storage devices.
11. What is the role of ALU?
12. What is a control unit?
13. What are registers? Write and explain any two of them.
14. Differentiate hard copy and soft copy.
15. What is e-Waste?
16. What is operating system?
17. What is a language processor?
18. Mention the categories of computer languages.
19. What is disk defragmenter?
20. Why is OS considered as a 'resource manager'?
21. What is proprietary software?
24. What do you mean by open source software?

**Short answer type**

1. Distinguish between data and information.
2. The application form for Plus One admission contains your personal details and your choice of groups and schools.
  - (a) Identify the data and information in the admission process.
  - (b) Explain how the information helps the applicants and school authorities.
  - (c) Write down the activities involved in the processing of the data.



3. Briefly explain any three input devices.
4. Compare CRT with LED monitor
5. Differentiate between RAM and ROM
6. List and explain e-waste disposal methods.
7. Enumerate the steps that can be taken for the implementation of green computing philosophy.
8. What do you mean by customised software? Give examples.
9. Distinguish between low level and high level languages.
10. Differentiate compiler and interpreter.
11. Describe the use of electronic spreadsheets.
12. What is utility software? Give two examples.
13. Categorise the software given below into operating system, application packages and utility programs. Linux, Tally, WinZip, MS-Word, Windows, MS-Excel
14. Differentiate between freeware and shareware.
15. What are the four freedoms which make up free and open source software?
16. What do you mean by human-ware? Give any two examples.

### Long answer type

1. Taking the case of a real life example, briefly describe the activities involved in each stage of data processing.
2. With the help of a block diagram, explain the functional units of a computer.
3. Describe in detail the various units of the Central Processing Unit.
4. Briefly explain various types of memory.
5. Explain classification of printers.
6. "e-Waste is hazardous to our health and environment." Justify the statement. List and explain the methods commonly used for e-Waste disposal.
7. Define the term green computing. List and explain the approaches that you can adopt to promote green computing concepts at all possible levels.
8. List and explain various categories of software.
9. Describe the use of various utility software.
10. Define the term 'operating system'. List and explain the major functions of operating system.
11. List and explain general purpose application software with examples.
12. Compare freeware and shareware.

## Key concepts

- **Problem solving using computers**
- **Approaches in problem solving**
  - Top down design
  - Bottom up design
- **Phases in programming**
  - Problem identification
  - Algorithms and Flowcharts
  - Coding the program
  - Translation
  - Debugging
  - Execution and testing
  - Documentation
- **Performance evaluation of algorithms**



# Principles of Programming and Problem Solving

We have learnt the concept of data processing and the role of computers in data processing. We have also discussed the computer as a system with components such as hardware, software and users. We had a detailed discussion on these components in the previous chapter. Let us recall the definition of software. In its simplest form, we can say that software means a collection of programs to solve problems using computers. As we know, a computer cannot do anything on its own. It must be instructed to perform the desired job. Hence it is necessary to specify a sequence of instructions that the computer must perform to solve a problem. Such a sequence of instructions written in a language that is understood by a computer is called a “computer program”. Writing a computer program is a challenging task. However, we can attempt it by procuring the concepts of problem solving techniques and different stages of programming.

## 4.1 Problem solving using computers

A computer can solve problems only when we give instructions to it. If it understands the tasks contained in the instructions, it will work accordingly. An instruction is an action oriented statement. It tells the computer what operation it should perform. A computer can execute (carry out the task contained in) an instruction only if the task is specified precisely and accurately. As we learned in the previous chapter, there are programmers who develop sequence of instructions for solving problems. Once the program is developed and stored permanently in a computer, we can ask the computer to execute it as and when required.



We should be cautious about the clarity of the logic of the solution and the format of instructions while designing a program, because computer does not possess common sense or intuition. As human beings, we use judgments based on experience, often on subjective and emotional considerations. Such value oriented judgments often depend on what is called "common sense". As opposed to this, a computer exhibits no emotion and has no common sense. That is why we say that computer has no intelligence of its own.

In a way, computer may be viewed as an 'obedient servant'. Being obedient without exercising 'common sense' can be very annoying and unproductive. Take the instance of a master who sent his obedient servant to a post office with the instruction "Go to the post office and buy ten 5 rupees stamps". The servant goes to the post office with the money and does not return even after a long time. The master gets worried and goes in search of him to the post office and finds the servant standing there with the stamps in his hand. When the angry master asks the servant for an explanation, the servant replies that he was ordered to buy ten 5 rupees stamps but not to return with them!

## 4.2 Approaches in problem solving

A problem may be solved in different ways. Even the approach may be different. In our life, we may seek medical treatment for some diseases. We can consult an allopathic, ayurvedic or homoeopathic doctor. Each of their approaches may be different, though all of them are solving the same problem. Similarly in problem solving also different approaches are followed. Let us discuss the two popular designing styles of problem solving – Top down design and Bottom up design.

### 4.2.1 Top down design

Look at Figure 4.1. If you are asked to draw this picture, how will you proceed? It may be as follows:

1. Draw the outline of the house.
2. Draw the chimney
3. Draw the door
4. Draw the windows

The procedure described above may be summarised as follows:

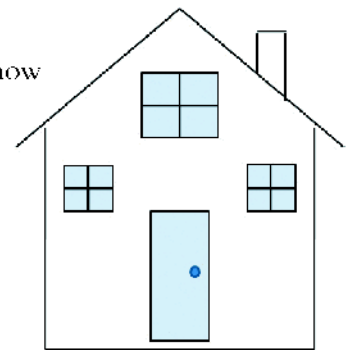


Fig. 4.1 : Lay-out of a House

While drawing the door in Step 3, the procedure may be as follows:

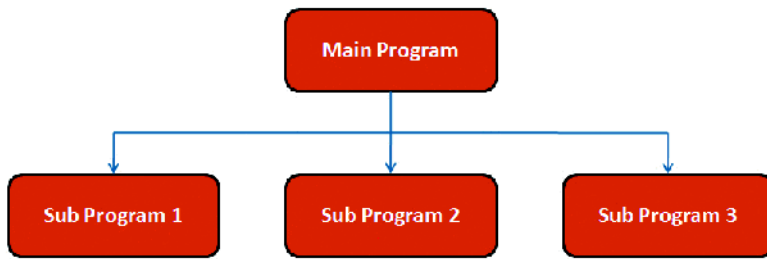
- 3.1 Outline of the door
- 3.2 Shading
- 3.3 Handle

Similarly the windows may be drawn as follows:

- 4.1 Outline of the window
- 4.2 Shading
- 4.3 Horizontal and Vertical lines

The whole problem (here drawing the picture) is broken down into smaller tasks. Thus four tasks are identified to solve the problem. Some of these tasks (here steps 3 and 4 for drawing the door and windows) are further subdivided. Thus any complex problem can be solved by breaking it down into different tasks and solving each task by performing simpler activities. This concept is known as **top down design** in problem solving.

It is one of the programming approaches that has been proven the most productive. As shown in Figure 4.2, top down design is the process of breaking the overall procedure or task into component parts (modules) and then subdividing each component module until the lowest level of detail is reached. It is also called top down decomposition since we start "at the top" with a general problem and design specific solutions to its sub problems. In order to obtain an effective solution for the main problem, it is desirable that the sub problems (sub programs) should be independent of each other. So, each sub problem can be solved and tested independently.



*Fig. 4.2 : Decomposition of a problem*

The following are the advantages of problem solving by decomposition:

- Breaking the problem into parts helps us to clarify what is to be done in each part.
- At each step of refinement, the new parts become less complicated and therefore, easier to figure out.
- Parts of the solution may turn out to be reusable.
- Breaking the problem into parts allows more than one person to work for the solution.

### 4.2.2 Bottom up design

Consider the case of constructing a house. We do not follow the top down design, but the bottom up design. The foundation will be the first task and roofing will be the last task. Breaking down of tasks is carried out here too. It is true that some tasks can be carried out only after the completion of some other tasks. However roofing which is the main task will be carried out only after the completion of bottom level tasks.

Similarly in programming, once the overall procedure or task is broken down into component parts (modules) and each component module is further sub divided until the lowest level of detail has been reached, we start solving from the lowest module





onwards. The solution for the main module will be developed only after designing specific solutions to its sub modules. This style of approach is known as **bottom-up design** for problem solving. Here again, it is desirable that the sub problems (subprograms) should be independent of each other.



**Let us do**

*Youth festivals are conducted every year in our schools. Usually the duties and responsibilities are decomposed. Discuss how the tasks are divided and executed to organise the youth festival successfully.*

### 4.3 Phases in programming

As we have seen, problem solving using computer is a challenging task. A systematic approach is essential for this. The programs required can be developed only by going through different stages. Though we have in-born problem solving skills, it can be applied effectively only by proper thinking, planning and developing the logical reasoning to solve the problem. We can achieve this by proceeding through the following stages, in succession:

1. Problem identification
2. Preparing algorithms and flowcharts
3. Coding the program using programming language
4. Translation
5. Debugging
6. Execution and Testing
7. Documentation

Figure 4.3 shows the order of performing the tasks in various stages of programming. Note that the debugging stage is associated with both translation and execution. The activities involved in the seven stages mentioned above are detailed in the following sections.

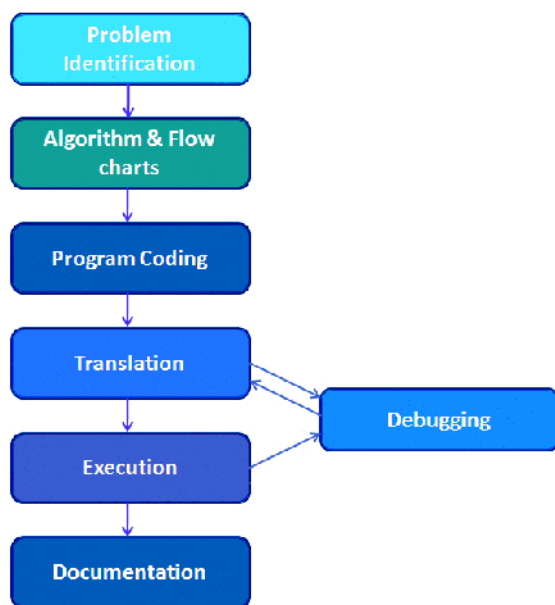


Fig. 4.3 : Phases of Programming

#### 4.3.1 Problem identification

Let us discuss a real life situation; suppose you are suffering from a stomach ache. As you know this problem can be solved by a doctor. Your doctor may ask you some questions regarding the duration of pain, previous occurrence, your diet etc., and examine some parts of your body using the stethoscope, X-ray or scan. All these are a



part of the problem study. After these procedures, your doctor may be able to identify the problem and state it using some medical term. Now the second stage begins with the derivation of some steps for solution known as prescriptions.

It is clear that before deriving the steps for solution, the problem must be analysed. During this phase you will be able to identify the data involved in processing, its type and quantity, formula to be used, activities involved, and the output to be obtained. Once you have studied the problem clearly, and are convinced about the sequence of tasks required for the solution, you can go to the next phase. This is the challenging phase as it exploits the efficiency of the programmer (problem solver).

### 4.3.2 Algorithms and Flowcharts

Once the problem is identified, it is necessary to develop a precise step-by-step procedure to solve the problem. This procedure is not new or confined to computers. It has been in use for a very long time, and in almost all walks of life. One such procedure taken from real life is described below. It is a cooking recipe of an omlette taken from a magazine.

#### Ingredients

Eggs - 2 Nos, Onion - 1 (small sized, chopped); Green chili - 2 (finely chopped); Oil - 2 tea spoon, Salt - a pinch.

#### Method

Step 1 : Break the eggs and pour the contents in a vessel and stir.

Step 2 : Mix chopped onion, green chilies and salt with the stirred egg.

Step 3 : Place a pan on the stove and light the stove.

Step 4 : Pour the oil in the pan and wait till it gets heated.

Step 5 : Pour the mixture prepared in step 2 into the pan and wait till the side is fried.

Step 6 : Turn over to get the other side fried well.

Step 7 : Take it out after some seconds.

#### Result

An omlette is ready to be served with pepper powder.

The recipe given above has the following properties:

1. It begins with a list of ingredients required for making the omlette. These may be called the inputs.
2. A sequence of instructions is given to process the inputs.





3. As a result of carrying out the instructions, some outputs (here, omlette) are obtained.

The instructions given to process the inputs are, however, not precise. They are ambiguous. For example, the interpretation of "till the side is fried" in step 5 and "fried well" in step 6 can vary from person to person. Due to such imprecise instructions, different persons following the same recipe with the same inputs can produce omlettes which differ in size, shape and taste.

The above ambiguities should be avoided while writing steps to solve the problems using the computer.

### a. Algorithm

Mathematicians trace the origin of the word algorithm to a famous Arab mathematician, Abu Jafar Mohammed Ibn Musaa Al-Khowarizmi. The word 'algorithm' is derived from the last part of his name Al-Khowarizmi. In computer terminology an **algorithm** may be defined as a finite sequence of instructions to solve a problem. It is a step-by-step procedure to solve a problem, where each step represents a specific task to be carried out. However, in order to qualify an algorithm, a sequence of instructions must possess the following characteristics:



*Fig. 4.4 : Abu Jafar Mohammed  
Ibn Musaa Al-Khowarizmi  
(780 - 850)*

- (i) It should begin with instruction(s) to accept inputs. These inputs are processed by subsequent instructions. Sometimes, the data to be processed will be given along with the problem. In such situations, there will be no input instruction at the beginning of the algorithm.
- (ii) Use variables to refer the data, where variables are user-defined words consisting of alphabets and numerals that are similar to those used in mathematics. Variables must be used for inputting data and assigning values or results.
- (iii) Each and every instruction should be precise and unambiguous. In other words, the instructions must not be vague. It must be possible to carry them out. For example, the instruction "Catch the day" is precise, but cannot be carried out.
- (iv) Each instruction must be sufficiently basic such that it can, in principle, be carried out in finite time by a person with paper and pencil.
- (v) The total time to carry out all the steps in the algorithm must be finite. As algorithm may contain instructions to repetitively carry out a group of instructions, this requirement implies that the number of repetitions must be finite.
- (vi) After performing the instructions given in the algorithm, the desired results (outputs) must be obtained.



To gain insight into algorithms, let us consider a simple example. We have to find the sum and average of any three given numbers. Let us write the procedure for solving this problem. It is given below:

- Step 1: Input three numbers.
- Step 2: Add these numbers to get the sum
- Step 3: Divide the sum by 3 to get the average
- Step 4: Print sum and average

Though the procedure is correct, while preparing an algorithm, we have to follow any of the standard formats. Let us see how the procedure listed above can be written in an algorithm style.

#### Example 4.1: Algorithm to find the sum and average of three numbers

Let A, B, C be variables for the input numbers; and S, Avg be variables for sum and average.

- Step 1: Start
- Step 2: Input A, B, C
- Step 3:  $S = A + B + C$
- Step 4:  $Avg = S / 3$
- Step 5: Print S, Avg
- Step 6: Stop

The above set of instructions qualifies as an algorithm for the following reasons:

- It has input (The variables A, B and C are used to hold the input data).
- The processing steps are precisely specified (Using proper operators in Steps 3 and 4) and can be carried out by a person using pen and paper.
- Each instruction is basic (Input, Print, Add, Divide) and meaningful.
- It produces two outputs such as sum (S) and average (Avg).
- The beginning and termination points are specified using Start and Stop.

#### Types of instructions

As we know, a computer can perform only limited types of operations. So we can use only that many instructions to solve problems. Before developing more algorithms, let us identify the types of instructions constituting the algorithm.

- Computer can accept data that we input. So, we can use input instructions. The words **Input**, **Accept** or **Read** may be used for this purpose.
- Computer gives the results as output. So we can use output instructions. The words **Print**, **Display** or **Write** may be used for this purpose.



- Data can directly be stored in a memory location or data may be copied from one location to another. Similarly, results of arithmetic operations on data can also be stored in memory locations. We use assignment (or storing) instruction for this, similar to that used in mathematics. Variables followed by the equal symbol ( = ) can be used for storing value, where variables refer to memory locations.
- A computer can compare data values (known as logical operation) and make decisions based on the result. Usually, the decision will be in the form of selecting or skipping a set of one or more statements or executing a set of instructions repeatedly.

## b. Flowcharts

An idea expressed in picture or diagram is preferred to text form by people. In certain situations, algorithm may be difficult to follow, as it may contain complex tasks and repetitions of steps. Hence, it will be better if it could be expressed in pictorial form. The pictorial representation of an algorithm with specific symbols for instructions and arrows showing the sequence of operations is known as **flowchart**. It is primarily used as an aid in formulating and understanding algorithms. Flowcharts commonly use some basic geometric shapes to denote different types of instructions. The actual instructions are written within these boxes using clear and concise statements. These boxes are connected by solid lines with arrow marks to indicate the flow of operation; that is, the exact sequence in which the instructions are to be executed.

Normally, an algorithm is converted into a flowchart and then the instructions are expressed in some programming language. The main advantage of this two-step approach in program writing is that while drawing a flowchart one is not concerned with the details of the elements of programming language. Hence he/she can fully concentrate on the logic (step-by-step method) of the procedure. Moreover, since a flowchart shows the flow of operations in pictorial form, any error in the logic of the procedure can be detected more easily than in a program. The algorithm and flow chart are always a reference to the programmer. Once these are ready and found correct as far as the logic of the solution is concerned, the programmer can concentrate on coding the operations following the constructs of programming language. This will normally ensure an error-free program.

### Flowchart symbols

The communication of program logic through flowcharts is made easier through the use of symbols that have standardised meanings. We will only discuss a few symbols that are needed to indicate the necessary operations. These symbols are standardised by the American National Standards Institute (ANSI).

#### 1. Terminal

As the name implies, it is used to indicate the beginning (START) and ending (STOP) in the program logic flow. It is the first symbol and the last symbol in the flowchart.

It has the shape of an ellipse. When it is used as START, an exit flow will be attached. But when it is used as a STOP symbol, an entry flow will be attached.



## 2. Input / Output

The parallelogram is used as the input/output symbol. It denotes the function of an input/output device in the program. All the input/output instructions are expressed using this symbol. It will be attached with one entry flow and one exit flow.



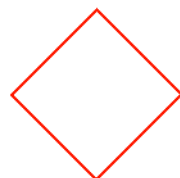
## 3. Process

A rectangle is used to represent the processing step. Arithmetic operations such as addition, subtraction, multiplication, division as well as assigning a value to a variable are expressed using this symbol. Assigning a value to a variable means moving data from one memory location to another (e.g.  $a=b$ ) or moving the result from ALU to memory location (e.g.  $a=b+5$ ) or even storing a value to a memory location (e.g.  $a=2$ ). Process symbol also has one entry flow and one exit flow.



## 4. Decision

The rhombus is used as decision symbol and is used to indicate a point at which a decision has to be made. A branch to one of two or more alternative points is possible here. All the possible exit paths will be specified, but only one of these will be selected based on the result of the decision. Usually this symbol has one entry flow and two exit flows - one towards the action based on the truth of the condition and the other towards the alternative action.



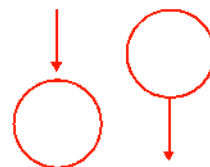
## 5. Flow lines

Flow lines with arrow heads are used to indicate the flow of operation, that is, the exact sequence in which the instructions are to be executed. The normal flow is from top to bottom and left to right. But in some cases, it can be from right to left and bottom to top. Good practice also suggests that flow lines should not cross each other and such intersections should be avoided wherever possible.



## 6. Connector

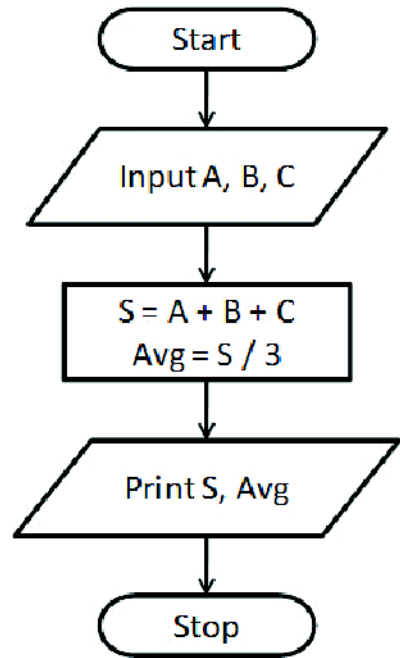
When a flowchart becomes bigger, the flow lines start criss-crossing at many places causing confusion and reducing comprehension of the flowchart. Moreover, when the flowchart becomes too long to fit





into a single page the use of flow lines becomes impossible. Whenever a flowchart becomes complex and the number and direction of flow lines is confusing or it spreads over more than one page, a pair of connector symbols can be used to join the flow lines that are broken. This symbol represents an "entry from", or an "exit to" another part of the flowchart. A connector symbol is represented by a circle and a letter or digit is placed within the circle to indicate the link. A pair of identically labelled connector symbols is commonly used to indicate a continuous flow. So two connectors with identical labels serve the same function as a long flow line. That is, in a pair of identically labelled connectors, one shows an exit to some other chart section and the other indicates an entry from another part of the chart.

Figure 4.5 shows that flowchart of the problem discussed in Example 4.1.



*Fig. 4.5 : Flow chart for Sum and Average*

The instruction given in each step in the algorithm is represented using the concerned symbol. Each symbol is labelled properly with the respective instruction. The flow of operations is clearly shown using the flow lines.

### Advantages of flowcharts

Flowcharts are beneficial in many ways in program planning.

- **Better communication:** Since a flowchart is a pictorial representation of a program, it is easier for a programmer to explain the logic of the program to some other programmer through a flowchart rather than the program itself.
- **Effective analysis:** The whole program can be analysed effectively through the flowchart as it clearly specifies the flow of the steps constituting the program.
- **Effective synthesis:** If a problem is divided into different modules and the solution for each module is represented in flowcharts separately, they can finally be placed together to visualize the overall system design.
- **Efficient coding:** Once a flowchart is ready, programmers find it very easy to write the concerned program because the flowchart acts as a road map for them. It guides them to go from the starting point of the program to the final point ensuring that no steps are omitted.





## Limitations of flowcharts

In spite of their many obvious advantages, flowcharts have some limitations:

- Flowcharts are very time consuming and laborious to draw with proper symbols and spacing, especially for large complex algorithms.
- Owing to the symbol-string nature of flowcharting, any change or modification in the logic of the algorithm usually requires a completely new flowchart.
- There are no standards determining the amount of detail that should be included in a flowchart.

Now let us develop algorithms and draw flowcharts for solving various problems.

### Example 4.2: To find the area and perimeter of a rectangle

We know that this problem can be solved, if the length and breadth of the rectangle are given. The result can be obtained by using the following formula:

Perimeter =  $2 (\text{Length} + \text{Breadth})$ , Area =  $\text{Length} \times \text{Breadth}$

Let L and B be variables for length and breadth; and P, A be variables for perimeter and area.

- Step 1: Start  
 Step 2: Input L, B  
 Step 3:  $P = 2 * (L + B)$   
 Step 4:  $A = L * B$   
 Step 5: Print P, A  
 Step 6: Stop

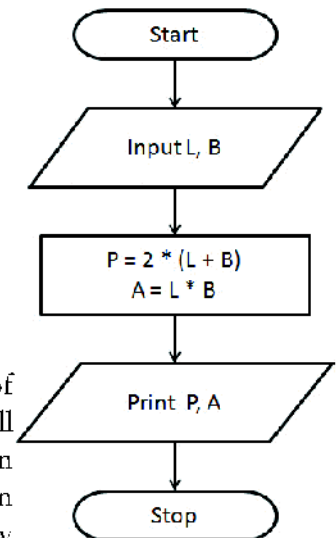


Fig. 4.6 : Flow chart for Area and Perimeter

The flowchart is given in Figure 4.6.

The algorithms developed in Examples 4.1 and 4.2 consist of six instructions each. In both the cases, the instructions will be executed one by one in a sequential fashion as shown in Figure 4.7. The order of execution of instructions is known as flow of control. We can say that the two algorithms follow in a sequential flow of control.

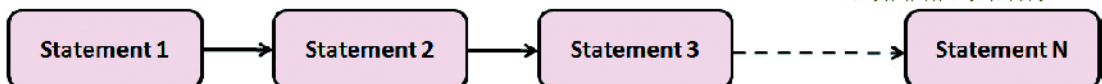


Fig. 4.7 : Sequential flow of control



Let us do

Develop an algorithm and draw the flow chart to input a time in seconds and convert it into the Hr: Min: Sec format. (For example, if 3700 is given as input, the output should be 1 Hr: 1 Min: 40 Sec).


**Example 4.3: Find the height of the taller one among two students**

Here, two numbers representing the height of two students are to be input. The larger number is to be identified as the result. We know that a comparison between these numbers is to be made for this. The algorithm is given below:

- Step 1: Start
- Step 2: Input H1, H2
- Step 3: If  $H1 > H2$  Then
- Step 4:       Print H1
- Step 5: Else
- Step 6:       Print H2
- Step 7: End of If
- Step 8: Stop

The flowchart of this algorithm is shown in Figure 4.8. This algorithm uses the decision making aspect. In step 3, a condition is checked. Obviously the result may be True or False based on the values of variables H1 and H2. The decision is made on the basis of this result. If the result is True, step 4 will be selected for execution, otherwise step 6 will be executed. Here one of the two statements (either step 4 or step 6) is selected for execution based on the condition. A branching is done in step 3. That is, this algorithm uses the selection structure to solve the problem. As shown in Figure 4.9, the condition branches the flow to one of the two sets based on the result of condition.

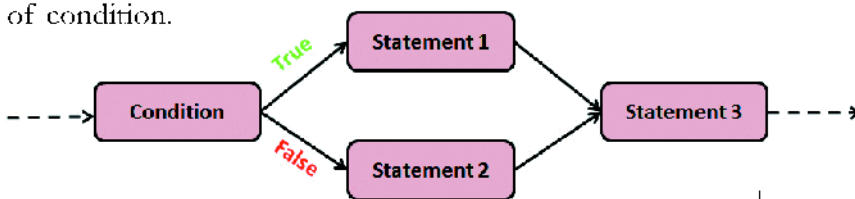


Fig. 4.9 : Selection structure

The working of selection construct is as shown in Figure 4.10. The flow of control comes to the condition; it will be evaluated to True or False. If the condition is True, the instructions given in the true block will be executed and false block will be skipped. But if the condition is False, the true block will be skipped and the false block will be executed. Now let us solve another problem.

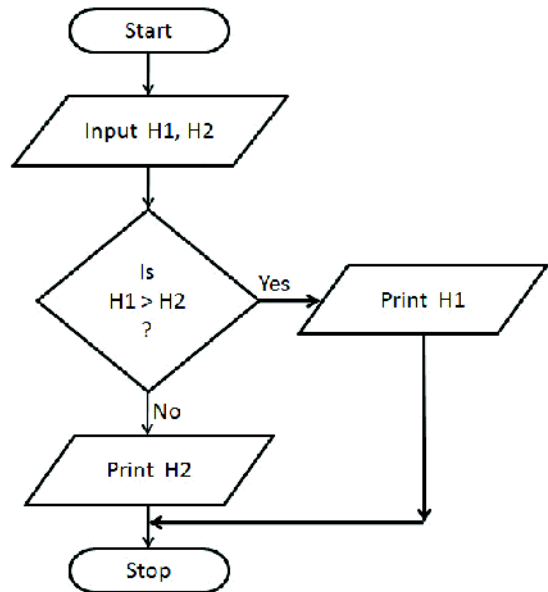


Fig. 4.8 : Flowchart to find larger value

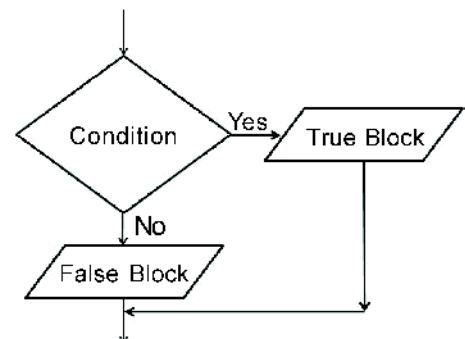


Fig. 4.10 : Flowchart of selection

**Example 4.4: To input the scores obtained in 3 unit-tests and find the highest score**

Here we have to input three numbers representing the scores and find the largest number among them. The algorithm is given below and flowchart is shown in Figure 4.11.

- Step 1: Start  
 Step 2: Input M1, M2, M3  
 Step 3: If  $M1 > M2$  And  $M1 > M3$  Then  
 Step 4: Print M1  
 Step 5: Else If  $M2 > M3$  Then  
 Step 6: Print M2  
 Step 7: Else  
 Step 8: Print M3  
 Step 9: End of If  
 Step 10: Stop

This algorithm uses multiple branching based on different conditions. Three different actions are provided, but only one of them is executed. Another point we have to notice is that the first condition consists of two comparisons. This kind of condition is known as compound condition.

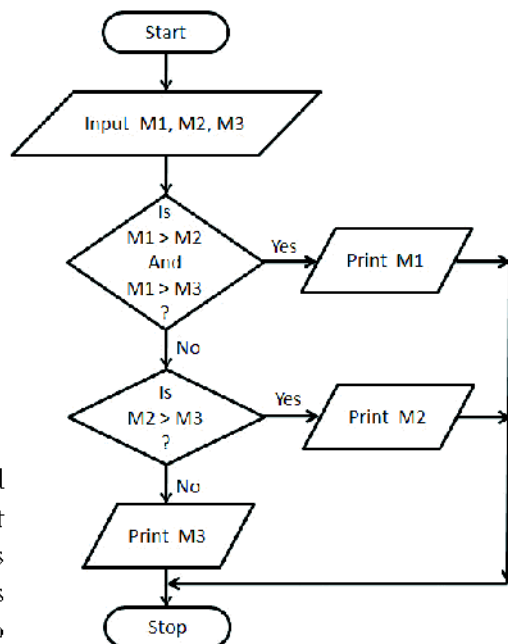


Fig. 4.11 : Flowchart to find the largest of three numbers

**Let us do**

1. Develop an algorithm and draw the flowchart to check whether a given number is even or odd.
2. Design an algorithm and flow chart to input a day number and display the name of the day. (For example, if 1 is the input, the output should be Sunday. If it is 2, the output should be Monday. If the number is other than 1 to 7, the output should be "Invalid data").
3. Based on the evaluation system for standard X, develop an algorithm to accept a score out of 100 and find the grade.

Now, consider a case in which some task is to be performed in a repeated fashion. Suppose we want to print the first 100 natural numbers. How do we do it? We know that the first number is 1. It should be printed. The next number is obtained by adding 1 to the first number. Again it should be printed. It is clear that the two tasks - printing a number and adding 1 to it - are to be executed repeatedly. The execution should be terminated when the last number is printed. Let us develop the algorithm for this.

**Example 4.5: To print the numbers from 1 to 100**

- Step 1: Start  
 Step 2:  $N = 1$   
 Step 3: Print N  
 Step 4:  $N = N + 1$   
 Step 5: If  $N \leq 100$  Then Go To Step 3  
 Step 7: Stop

In the algorithm given as Example 4.5, a condition is checked at step 5. If the condition is found true, the flow of control is transferred back to step 3. So the steps 3, 4 and 5 are executed repeatedly as long as the condition is true. We will say that a loop is formed here. Steps 3, 4 and 5 constitute a loop. The control comes out of the loop only when the condition becomes false. The flowchart of this algorithm is shown in Figure 4.12.

The above algorithm can be simplified as follows:

- Step 1: Start  
 Step 2:  $N = 1$   
 Step 3: Repeat Steps 4 and 5 While ( $N \leq 100$ )  
 Step 4: Print N  
 Step 5:  $N = N + 1$   
 Step 6: Stop

Note that in step 3, the words "**Repeat**" and "**While**" are used to construct a loop. The statements (step numbers) that need repeated execution is specified with the word "**Repeat**" and the condition is given with "**While**". As the algorithm looks slightly different, the flow chart also differs slightly as in Figure 4.13. The execution style of a loop is shown in Figure 4.14.

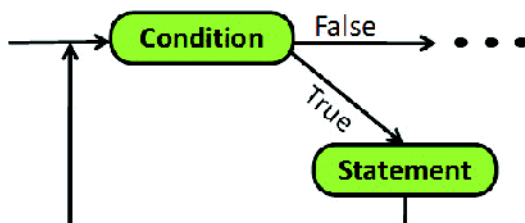


Fig. 4.14 : Looping construct

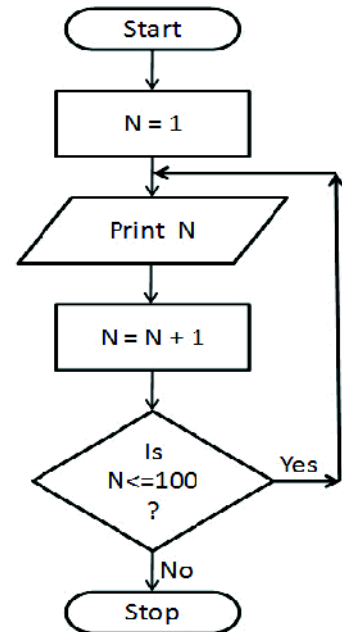


Fig. 4.12 : Flowchart to print numbers from 1 to 100

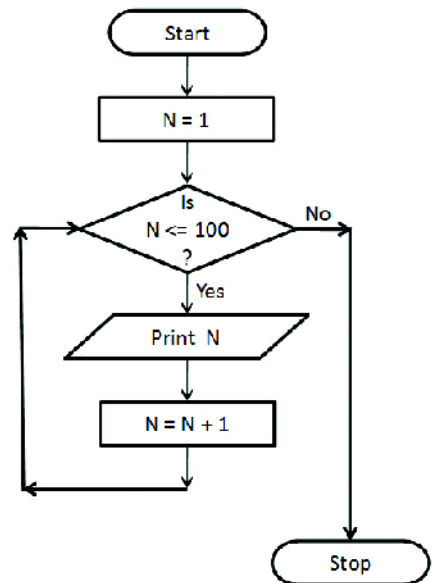


Fig. 4.13 : Flowchart to print numbers from 1 to 100

A loop has four elements. Obviously, one of them is the **condition**. We know that at least one variable will be used to put up a condition and let us call it loop control variable. Before the condition being checked, the loop control variable should get a value. It is possible through input or assignment. Such an instruction is called **initialisation instruction** for the loop. The third element, called **update instruction**, changes the value of the loop control variable. It is essential; otherwise the execution of the loop will never be terminated. The fourth element is the **loop body**, which is the set of instructions to be executed repeatedly. The flowchart shown in Figure 4.15 depicts the working of looping structure.

The initialisation instruction will be executed first and then the condition will be checked. If the condition is true, the body of the loop will be executed followed by the update instruction. After the execution of the update instruction, the condition will be checked again. This process will be continued until the condition becomes false. The loop that checks the condition before executing the body is called entry-controlled loop. There is another style of looping construct. In this case, the condition will be checked only after the execution of loop-body and update instruction. Such a loop is called exit-controlled loop.

#### Example 4.6: To print the sum of the first $N$ natural numbers

Here we have to input the value of  $N$ . The sum of numbers from 1 to the input number  $N$  is to be found out. Let  $S$  be the variable to store the sum. Figure 4.16 shows the flowchart of this algorithm

- Step 1: Start
- Step 2: Input  $N$
- Step 3:  $A = 1, S = 0$
- Step 4: Repeat Steps 5 and 6 While  $(A \leq N)$
- Step 5:  $S = S + A$
- Step 6:  $A = A + 1$
- Step 7: Print  $S$
- Step 8: Stop

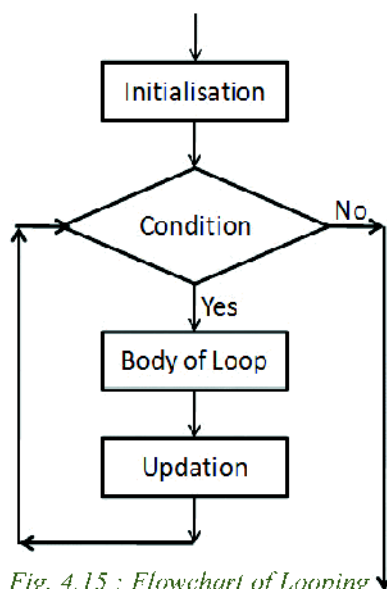


Fig. 4.15 : Flowchart of Looping

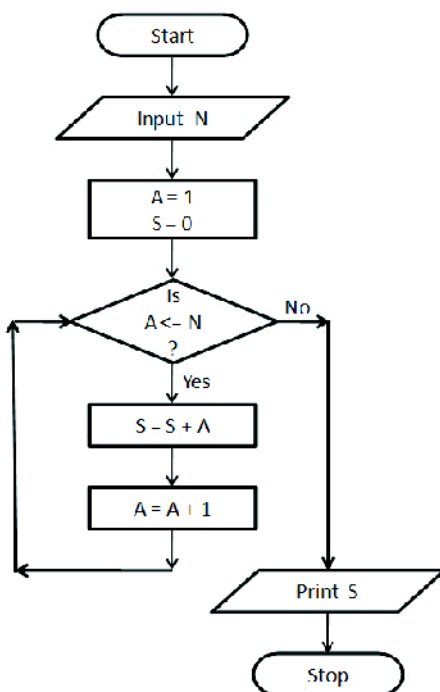


Fig. 4.16 : Flowchart for the sum of first  $N$  natural numbers



This algorithm uses an entry-controlled loop. In the next example we can see an algorithm that uses exit-controlled loop for problem solving.

**Example 4.7: To print the first 10 multiples of a given number**

- Step 1: Start  
 Step 2: Input N  
 Step 3:  $A = 1$   
 Step 4:  $M = A \times N$   
 Step 5: Print M  
 Step 6:  $A = A + 1$   
 Step 7: Repeat Steps 4 to 6 While ( $A \leq 10$ )  
 Step 8: Stop

This algorithm and the corresponding flowchart shown in Figure 4.17 contain a loop in which the condition is checked only after executing the body. Table 4.1 shows comparison between entry controlled loops and exit controlled loops.

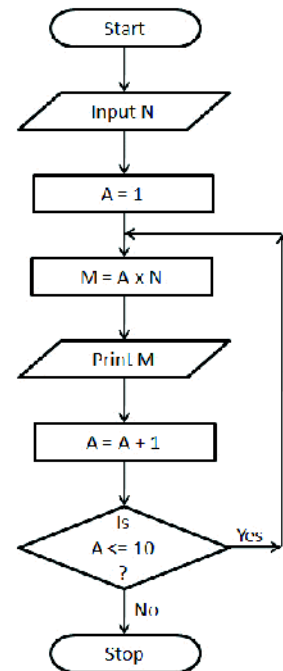


Fig. 4.17 : Flowchart for the first 10 multiples of a number

Entry Controlled Loop	Exit Controlled Loop
<ul style="list-style-type: none"> <li>Condition is checked before the execution of the body</li> <li>Body may never be executed.</li> <li>Suitable when skipping of the body from being executed is required</li> </ul>	<ul style="list-style-type: none"> <li>Condition is checked after the execution of the body</li> <li>Body will surely be executed at least once.</li> <li>Suitable when normal execution of the body is to be ensured.</li> </ul>

Table 4.1 : Comparison of Loops

Let us practice with more algorithms and flowcharts for solving the problems that are given as learning activities.



**Let us do**

Develop an algorithm and draw the flowcharts for the following problems:

- To print all even numbers below 100 in the descending order.
- To find the sum of odd numbers between 100 and 200.
- To print the multiplication table of a given number.
- To find the factorial of a number.
- To input a number and check whether it is prime or not.



### 4.3.3 Coding the program

Once we have developed the skill to design algorithms and flowcharts, the next step in programming is to express the instructions in a more precise and concise notation. That is, the instructions are to be expressed in a programming language. The process of writing such program instructions to solve a problem is called **coding**. Text editor programs are available to write the code in computer.



A language is a system of communication. We communicate our ideas and emotions to one another through natural languages such as English, Malayalam, etc. Similarly, a computer language is used to communicate between user and computer. A human being who writes a computer program is to be familiar with a language that is understandable to the computer also. We have already seen that computer knows only the binary language which is very difficult for human beings to understand and use.

As we saw in Chapter 3, we can use a human friendly language, known as High Level Language (HLL) that looks similar to English. Again there is a facility of using language processors to convert or translate the program written in HLL into machine language. The program written in any HLL is known as **source code**.

Hence, to be a programmer we have to be well-versed in any HLL such as BASIC, COBOL, Pascal, C++ etc. to express the instructions in a program. Each language has its own character set, vocabulary, grammar (we call it syntax) to write programs. Once the program is written using a language, it should be saved in a file (called source file), and then proceed to the next phase of programming.

### 4.3.4 Translation

While selecting a language for developing source code, certain criteria such as volume of data, complexity in process, usage of files, etc. are to be considered. Once a language is selected and the source code is prepared, it should be translated using the concerned language processor. **Translation** is the process of converting a program written in high level language into its equivalent version in machine language. The compiler or interpreter is used for this purpose. During this step, the syntax errors of the program will be displayed. These errors are to be corrected by opening the file that contains the source code. The source code is again given for compilation (translation). This process will be continued till we get a message such as "No errors or warnings" or "Successful compilation". Now we have a program fully constituted by machine language instructions. This version of the source code is known as **object code** and it will be usually stored in a file by the compiler itself.



Fig. 4.18 : Translation process

Once the object code is obtained it should be present in the system as long as you want the program to be used.

### 4.3.5 Debugging

Debugging is the stage where programming errors are discovered and corrected. As long as computers are programmed by human beings, the programs will be subject to errors. Programming errors are known as 'bugs' and the process of detecting and correcting these errors is called **debugging**. In general there are two types of errors that occur in a program - syntax errors and logical errors. **Syntax errors** result when the rules or syntax of the programming language are not followed. Such program errors typically involve incorrect punctuation, incorrect word sequence, undefined term, or illegal use of terms or constructs. Almost all language processors detect syntax errors when the program is given for translation. They print error messages that include the line number of the statement having errors and give hints about the nature of the error. In the case of interpreters, the syntax errors will be detected and displayed during execution. The programmer's efficiency in using the language decides the time and effort for the debugging process. The object program will be generated only if all the syntax errors are rectified.

The second type of error, named **logical error**, is due to improper planning of the program's logic. The language processor successfully translates the source code into machine code if there are no syntax errors. During the execution of the program, computer actually follows the program instructions and gives the output as per the instructions. But the output may not be correct. This is known as logical error. When a logical error occurs, all you know is that the computer is not giving the correct output. The computers do not tell us what is wrong. It should be identified by the programmer or user. In order to determine whether or not there is a logical error, the program must be tested. So, let us move on to the next stage of programming.

### 4.3.6 Execution and testing

As we have seen in the previous section, the program is said to be error-free only when logical errors are also rectified. Hence when the compiled version of the program is formed, it should be executed for testing. The purpose of testing is to determine whether the results are correct. The testing procedure involves running the program to process the test data that will produce 'known results'. That is, the operations involved in the program should be done manually and the output thus obtained should be compared with the one given by the computer. The accuracy of the program logic can be determined by this testing. While selecting the test data, we should ensure that all aspects of the program logic will be tested. Hence the selection of proper test data is important in program testing.



Till now, we have discussed incorrect outputs due to incorrect logic. But there is a chance of another type of error, which will interrupt the program execution. This may be due to the inappropriate data that is encountered in an operation. For example consider an instruction  $A = B/C$ . This statement causes interruption in execution if the value of  $C$  happens to be zero. In such a situation, the error messages may be displayed by the error-handling function of the language. These errors are known as **Run-time error**. These errors can be rectified by providing instructions for checking the validity of the data before it gets processed by the subsequent instructions in the program.

### 4.3.7 Documentation

A computerised system cannot be considered to be complete until it is properly documented. In fact documentation is an on-going process that starts in the problem-study phase of the system and continues till its implementation and operation. We can write comments in the source code as part of documentation. It is known as **internal documentation**. It helps the debugging process as well as program modification at a later stage. The logic that we applied in the program may not be remembered when we go through our own program at a later stage. Besides, the program written by one person may need to be modified by some other person in future. If the program is documented, it will help to understand the logic we applied, the reason why a particular statement has been used and so on. However, the documentation part of the program will not be considered by the language processor when you give the program for translation.

Writing comments in programs is only a part of documentation. Another version of documentation is the preparation of system manual and user manual. These are hard copy documents that contain functioning of the system, its requirements etc. and the procedure for installing and using the programs. While developing software for various applications, these manuals are mandatory. This kind of documentation is known as **external documentation**.

Now you have analysed the problem, derived the logic of the solution, expressed in a flow chart, developed the code in a programming language, translated it after removing the syntax errors, checked the accuracy of the output after removing all the possible logical and run-time errors, and we have documented the program.

### Check yourself



1. What is an algorithm?
2. Pictorial representation of algorithm is known as \_\_\_\_\_.
3. Which flow chart symbol is always used in pair?
4. Which flow chart symbol has one entry flow and two exit flows?
5. Program written in HLL is known as \_\_\_\_\_.
6. What is debugging?
7. What is an object code?



## 4.4 Performance evaluation of algorithms

We have developed algorithms for solving various problems. You may think that some of these problems would have been solved by following a different logic. Of course, it is true that the same problem can be solved by different sets of instructions. But an efficient programmer is the one who develops algorithms that require minimum computer resources for execution and give results with high accuracy in lesser time. The performance of an algorithm is evaluated based on the concept of time and space complexity. The algorithm which will be executed faster with minimum amount of memory space is considered as the best algorithm for the problem.

<b>Algorithm-1</b>	<b>Algorithm-2</b>
Step 1: Start	Step 1: Start
Step 2: Input A, B, C	Step 2: Input A, B, C
Step 3: $S = A + B + C$	Step 3: $S = A + B + C$
Step 4: $Avg = S / 3$	Step 4: $Avg = (A + B + C) / 3$
Step 5: Print S, Avg	Step 5: Print S, Avg
Step 6: Stop	Step 6: Stop

*Table 4.2 : Algorithms to find the sum and average of three numbers*

Let us compare the two algorithms given in Table 4.2, developed to find the sum and average of three numbers. The two algorithms differ in step 4. Algorithm-2 uses two steps (steps 3 and 4) for addition operation on the same data. Naturally, that algorithm will take more time for execution than Algorithm-1. So Algorithm-1 is better for coding.

Now let us take another case, where comparison operations are involved for the selection of a statement. We have already discussed an algorithm to find the largest among three numbers in Example 4.4. The two algorithms given in Table 4.3 can also be used for solving the same problem.

<b>Algorithm-1</b>	<b>Algorithm-2</b>
Step 1: Start	Step 1: Start
Step 2: Input M1, M2, M3	Step 2: Input M1, M2, M3
Step 3: If $M1 > M2$ And $M1 > M3$ Then	Step 3: If $M1 > M2$ Then
Step 4: Print M1	Step 4: Big = M1
Step 5: If $M2 > M1$ And $M2 > M3$ Then	Step 5: Else
Step 6: Print M2	Step 6: Big = M2
Step 7: If $M3 > M1$ And $M3 > M2$ Then	Step 7: If $M3 > Big$ Then Big = M3
Step 8: Print M3	Step 8: Print Big
Step 9: Stop	Step 9: Stop

*Table 4.3 : Algorithms to find the largest among three numbers*



The algorithm in Example 4.4 has three comparison operations and one logical operation altogether. All these operations are to be carried out only when the largest value is in M3 (the third variable). To identify the speed of execution in each case, let us assume that 1 second is required for one comparison operation. We can see that fastest result will be in 3 seconds and slowest in 4 seconds. So the average speed is 3.5 seconds.

Now let us analyse Algorithm-1 in Table 4.3. There are three **If** statements, each with three comparison operations. If we follow the assumptions specified above, we can see that the result will be obtained in 9 seconds, irrespective of the values in the variables. So the average speed is 9 seconds. But the Algorithm-2 in Table 4.3 uses two **If** structures. The algorithm shows that whatever be the values in the variables, the time required for comparison will be 2 seconds. Thus the average speed is 2 seconds. So, we can say that the Algorithm-2 is better than the other two.

Let us consider one more case where loop is involved. The two algorithms given in Table 4.4 find the sum of all even numbers and sum of all odd numbers between 100 and 200.

Algorithm-1	Algorithm-2
Step 1: Start	Step 1: Start
Step 2: N = 100, ES = 0	Step 2: N = 100, ES = 0, OS = 0
Step 3: Repeat Steps 4 to 6 While (N <= 200)	Step 3: Repeat Steps 4 to 8 While (N <= 200)
Step 4: If Remainder of N/2 = 0 Then	Step 4: If Remainder of N/2 = 0 Then
Step 5: ES = ES + N	Step 5: ES = ES + N
Step 6: N = N + 1	Step 6: Else
Step 7: Print ES	Step 7: OS = OS + N
Step 8: N = 100, OS = 0	Step 8: N = N + 1
Step 9: Repeat Steps 10 to 12 While (N <= 200)	Step 9: Print ES
Step 10: If Remainder of N/2 = 1 Then	Step 10: Print OS
Step 11: OS = OS + N	Step 11: Stop
Step 12: N = N + 1	
Step 13: Print OS	
Step 14: Stop	

Table 4.4 : Algorithms to find sum of even and odd numbers

Algorithm-1 uses two loops. Obviously, time taken will be double for the initialisation, testing and updation of loop control variable compared to Algorithm-2. From the table, it is clear that Algorithm-2 is better and efficient. So, think divergently and differently to develop logic for solving problems.



## Let us sum up

Program is a sequence of instructions written in a computer language. The process of programming proceeds through some stages. Preparation of algorithms and flowcharts help develop the logic. The program written in HLL, is known as source code and it is to be converted into machine language. The resultant code is known as object code. The errors occurred in a program has to be removed through a process known as debugging. The translated version is executed by the computer. Proper documentation of the program helps us to modify it at a later stage. While solving problems different logic may be applied, but the performance is measured in terms of time and space complexity.



## Learning outcomes

After the completion of this chapter the learner will be able to

- explain various aspects of problem solving.
- develop algorithms for solving problems.
- draw flowcharts to ensure the correctness of algorithms.
- select the best algorithm for solving a problem.

## Sample questions

### Very short answer type

1. What is an algorithm?
2. What is the role of a computer in problem solving?
3. What is the use of connector in a flow chart?
4. What do you mean by logical errors in a program?

### Short answer type

1. What is a computer program? How does an algorithm help to write a program?
2. Write an algorithm to find the sum and average of 3 numbers.
3. Draw a flowchart to display the first 100 natural numbers.
4. What are the limitations of a flow chart?
5. What is debugging?
6. What is the need of documentation for a program?

### Long answer type

1. What are the characteristics of an algorithm?
2. What are the advantages of using a flowchart?
3. Briefly explain different phases in programming.



## Key concepts

- **C++ character set**
- **Tokens**
  - Keywords
  - Identifiers
  - Literals
  - Punctuators
  - Operators
- **Integrated Development Environment (IDE)**
  - Geany IDE

## Introduction to C++ Programming

C++ (pronounced "C plus plus") is a powerful, popular object oriented programming (OOP) language developed by Bjarne Stroustrup. The idea of C++ comes from the C increment operator ++, thereby suggesting that C++ is an added (incremented) version of C language.

The C++ language can be used to practice various programming concepts such as sequence, selection and iteration which we have already discussed in Chapter 4. In this chapter, we will have a brief overview of the fundamentals of C++. We will also familiarise different language processor packages that are used to write C++ programs.

Just like any other language, the learning of C++ language begins with the familiarisation of its basic symbols called characters. The learning hierarchy proceeds through words, phrases (expressions), statements, etc. Let us begin with the learning of characters.

### 5.1 Character set

As we know, the study of any language, such as English, Malayalam or Hindi begins with the alphabet. Similarly, the C++ language also has its own alphabet. With regard to a programming language the alphabet is known as character set. It is a set of valid symbols, called characters that a language can recognize. A character represents





**Dr. Bjarne Stroustrup** developed C++ at AT&T Bell Laboratories in Murray Hill, New Jersey, USA. Now he is a visiting Professor at Columbia University and holder of the College of Engineering Chair in Computer Science at Texas A&M University. He has received numerous honours. Initial name of this language was 'C with classes'. Later it was renamed to C++, in 1983.



*Bjarne  
Stroustrup*

any letter, digit, or any other symbol. The set of valid characters in a language which is the fundamental units of that language, is collectively known as **character set**. The character set of C++ is categorized as follows:

- (i) Letters : A B C D E F G H I J K L M N O P Q R S T U V  
W X Y Z  
a b c d e f g h i j k l m n o p q r s t u v w x y z
- (ii) Digits : 0 1 2 3 4 5 6 7 8 9
- (iii) Special characters : + - \* / ^ \ ( ) [ ] { } = < > . ' " \$  
, ; : % ! & ? \_ (underscore) # @
- (iv) White spaces : Space bar (Blank space), Horizontal Tab (→),  
Carriage Return (↵), Newline, Form feed
- (v) Other characters : C++ can process any of the 256 ASCII characters  
as data or as literals.



Spaces, tabs and newlines (line breaks) are called white spaces. White space is required to separate adjacent words and numbers.

## 5.2 Tokens

After learning the alphabet the second stage is learning words constituted by the alphabet (or characters). The term 'token' in the C++ language is similar to the term 'word' in natural languages. **Tokens** are the fundamental building blocks of the program. They are also known as lexical units. C++ has five types of tokens as listed below:

1. Keywords
2. Identifiers
3. Literals
4. Punctuators
5. Operators



### 5.2.1 Keywords

The words (tokens) that convey a specific meaning to the language compiler are called **keywords**. These are also known as reserved words as they are reserved by the language for special purposes and cannot be redefined for any other purposes. The set of 48 keywords in C++ are listed in Table 5.1. Their meaning will be explained in due course.

asm	continue	float	new	signed	try
auto	default	for	operator	sizeof	typedef
break	delete	friend	private	static	union
case	do	goto	protected	struct	unsigned
catch	double	if	public	switch	virtual
char	else	inline	register	template	void
class	enum	int	return	this	volatile
const	extern	long	short	throw	while

Table 5.1: Keywords of C++

### 5.2.2 Identifiers

We usually assign names to places, people, objects, etc. in our day to day life, to identify them from one another. In C++ we use identifiers for this purpose. **Identifiers** are the user-defined words that are used to name different program elements such as memory locations, statements, functions, objects, classes etc. The identifiers of memory locations are called **variables**. The identifiers assigned to statements are called **labels**. The identifiers used to refer a set of statements are called **function names**.

While constructing identifiers certain rules are to be strictly followed for their validity in the program. The rules are as follows:

- Identifier is an arbitrary long sequence of letters, digits and underscores ( \_ ).
- The first character must be a letter or underscore ( \_ ).
- White space and special characters are not allowed.
- Keywords cannot be used as identifiers.
- Upper and lower case letters are treated differently, i.e. C++ is case sensitive.

Examples for some valid identifiers are Count, Sumof2numbers, Average\_Height, \_1stRank, Main, FOR



The following are some invalid identifiers due to the specified reasons:

Sum of Digits	→ Blank space is used
1styear	→ Digit is used as the first character
First.Jan	→ Special character (.) is used
for	→ It is a keyword



*Identify invalid identifiers from the following list and give reasons:*

`Data_rec, _data, ldata, datal, my.file, asm, switch, goto, break`

**Let us do**

### 5.2.3 Literals

Consider the case of the Single Window System for the admission of Plus One students. You may have given your date of birth in the application form. As an applicant, your date of birth remains the same throughout your life. Once they are assigned their initial values, they never change their value. In mathematics, we know that the value of  $\pi$  is a constant and the value of gravitational constant 'g' never changes, i.e. it remains  $9.8\text{m/s}^2$ . Like that, in C++, we use the type of tokens called **literals** to represent data items that never change their value during the program run. They are often referred to as constants. Literals can be divided into four types as follows:

1. Integer literals
2. Floating point literals
3. Character literals
4. String literals

#### Integer literals

Consider the numbers 1776, 707, -273. They are integer constants that identify integer decimal values. The tokens constituted only by digits are called **integer literals** and they are whole numbers without fractional part. The following are the characteristics of integer literals:

- An integer constant must have at least one digit and must not contain any decimal point.
- It may contain either + or – sign as the first character, which indicates whether the number is positive or negative.
- A number with no sign is treated as positive.
- No other characters are allowed.



*Classify the following into valid and invalid integer constants and give reasons for the invalidity:*

**Let us do**

77,000	70	314.	-5432	+15346
+23267	-.7563	-02281+0	1234E56	-9999



In addition to decimal numbers (base 10), C++ allows the use of octal numbers (base 8) and hexadecimal numbers (base 16) as literals (constants). To express an octal number we have to precede it with a 0 (zero character) and in order to express a hexadecimal number we have to precede it with the characters 0x (zero, x). For example, the integer constants 75, 0113 and 0x4B are all equivalent to each other. All of these represent the same number 75 (seventy-five), expressed as a base-10 numeral, octal numeral and hexadecimal numeral, respectively.

### Floating point literals

You may have come across numbers like  $3.14159$ ,  $3.0 \times 10^8$ ,  $1.6 \times 10^{-19}$  and  $3.0$  during your course of study. These are four valid numbers. The first number is  $\pi$  (Pi), the second one is the speed of light in meter/sec, the third is the electric charge of an electron (an extremely small number) – all of them are approximated, and the last one is the number three expressed as a floating-point numeric literal.

**Floating point literals**, also known as real constants are numbers having fractional parts. These can be written in one of the two forms called fractional form or exponential form.

A real constant in fractional form consists of signed or unsigned digits including a decimal point between digits. The rules for writing a real constant in fractional form are given below:

- A real constant in fractional form must have at least one digit and a decimal point.
- It may also have either + (plus) or – (minus) sign preceding it.
- A real constant with no sign is assumed to be positive.

A real constant in exponential form consists of two parts: *mantissa* and *exponent*. For instance, 5.8 can be written as  $0.58 \times 10^1 = 0.58E1$  where mantissa part is 0.58 (the part appearing before **E**) and exponential part is 1 (the part appearing after **E**). The number E1 represents  $10^1$ . The rules for writing a real constant in exponential form are given below:

- A real constant in exponent form has two parts: a mantissa and an exponent.
- The mantissa must be either an integer or a valid fractional form.



- The mantissa is followed by a letter **E** or **e** and the exponent.
- The exponent must be an integer.

The following are valid real constants.

52.0	107.5	-713.8	-.00925
453.E-5	1.25E08	.212E04	562.0E09
152E+8	1520E04	-0.573E-7	-.097

Some invalid real constants are given along with the reason:

58,250.262 (Comma is used), 5.8E (No exponent part), 0.58E2.3 (Fractional number is used as exponent).



*Classify the following into valid and invalid real constants and justify your answer:*

**Let us do**

77, 00,000	7.0	3.14	-5.0E5.4	+53.45E-6
+532.67.	.756E-3	-0.528E10	1234.56789	34,56.24
4353	+34/2	5.6E	4356	0

## Character literals

When we want to store the letter code for gender usually we use 'f' or 'F' for *female* and 'm' or 'M' for *Male*. Similarly, we may use the letter 'y' or 'Y' to indicate *Yes* and the letter 'n' or 'N' to indicate *No*. These are single characters. When we refer a single character enclosed in single quotes that never changes its value during the program run, we call it a **character literal** or **character constant**.

Note that `x` without single quote is an identifier whereas `'x'` is a character constant. The value of a single character constant is the ASCII value of the character. For instance, the value of `'c'` will be 99 which is the ASCII value of 'c' and the value of `'A'` will be the ASCII value 65.

C++ language has certain non-graphic character constants, which cannot be typed directly from the keyboard. For example, there is no way to express the Carriage Return or Enter key, Tab key and Backspace key. These non-graphic symbols can be represented by using **escape sequences**, which consists of a backslash (\) followed by one or more characters. It should be noted that even though escape sequences contain more than one character enclosed in single quotes, it uses only one corresponding ASCII code to represent it. That is why they are treated as character constants. Table 5.2 lists escape sequences and corresponding characters.





In Table 5.2, we can also see sequences representing `\'`, `\"` and `\?`. These characters can be typed from the keyboard but when used without escape sequence, they carry a special meaning and have a special purpose. However, if these are to be displayed or printed as it is, then escape sequences should be used. Examples of some valid character constants are: `'s'`, `'S'`, `'$'`, `'\n'`, `'+'`, `'9'`

Some invalid character constants are also given with the reason for invalidity:

A (No single quotes), `'82'` (More than one character), `"K"` (Double quotes instead of single quotes), `'\g'` (Invalid escape sequence or Multiple characters).

Escape Sequence	Corresponding Non-graphic character
<code>\a</code>	Audible bell (alert)
<code>\b</code>	Back Space
<code>\f</code>	Form feed
<code>\n</code>	New line or Line feed
<code>\r</code>	Carriage Return
<code>\t</code>	Horizontal Tab
<code>\v</code>	Vertical Tab
<code>\\</code>	Back slash
<code>\'</code>	Single quote
<code>\"</code>	Double quote
<code>\?</code>	Question mark
<code>\0</code>	Null character

Table 5.2 : Escape Sequences



C++ represents Octal Number and Hexadecimal Number with the help of escape sequences. The `\On` and `\xHn` represent a number in the Octal Number System and the Hexadecimal Number System respectively.

## String literals

Nandana is a student and she lives in Bapuji Nagar. Here, "Nandana" is the name of a girl and "Bapuji Nagar" is the name of a place. These kinds of data may need to be processed with the help of programs. Such data are considered as string constants and they are enclosed within double quotes. A sequence of one or more characters enclosed within a pair of double quotes is called **string constant**. For instance, "Hello friends", "123", "C++", "Baby's Day Out", etc. are valid string constants.



Let us do

Classify the following into different categories of literals.

<code>'a'</code>	<code>"rita"</code>	<code>-124</code>	<code>12.5</code>	<code>-12e-1</code>
<code>"raju's pen"</code>	<code>0</code>	<code>-11.999</code>	<code>'\'</code>	<code>32760</code>

### 5.2.4 Punctuators

In languages like English, Malayalam, etc. punctuation marks are used for grammatical perfection of sentences. Consider the statement: *Who developed C++?* Here '?' is the punctuation mark that tells that the statement is a question. Similarly at the end of each sentence we put a full stop (.). In the same way C++ also has some special symbols that have syntactic or semantic meaning to the compiler. These are called **punctuators**. Examples are: # ; ' " ( ) [ ] { }. The purpose of each punctuator will be discussed later.

### 5.2.5 Operators

When we have to add 5 and 3, we express it as  $5 + 3$ . Here + is an operator that represents the addition operation. Similarly, C++ has a rich collection of operators. An **operator** is a symbol that tells the compiler about a specific operation. They are the tokens that trigger some kind of operation. The operator is applied on a set of data called **operands**. C++ provides different types of operators like arithmetic, relational, logical, assignment, conditional, etc. We will discuss more about operators in the next chapter.



*Classify the following into different categories of tokens.*

```
/      -124      +      -12e-1      "KL01"
Sum    "raju\'s pen"  if    rita      '\\\'
Let us do break }
```

## 5.3 Integrated Development Environment (IDE)

Now we have learned the basic elements of a C++ program. Before we start writing C++ programs, we must know where we will type this program. Like other programming languages, a text editor is used to create a C++ program. The compilers such as GCC (GNU Compiler Collection), Turbo C++, Borland C++, and many other similar compilers provide an Integrated Development Environment (IDE) for developing C++ programs. Many of these IDEs provide facilities for typing, editing, searching, compiling, linking and executing a C++ program. We use Geany IDE (T1@School Ubuntu Linux 12.04) for the purpose of illustrating the procedure for coding, compiling and executing C++ programs.

### GCC with Geany IDE

GCC compiler is a free software available with Linux operating system. GCC stands for GNU Compiler Collection and is one of the popular C++ compilers which



works with ISO C++ standard. Geany is a cross-platform IDE for writing, compiling and executing C++ programs.

### A. Opening the edit window

The edit window of Geany IDE can be opened from the **Applications** menu of Ubuntu Linux by proceeding as follows:

**Applications → Programming → Geany**

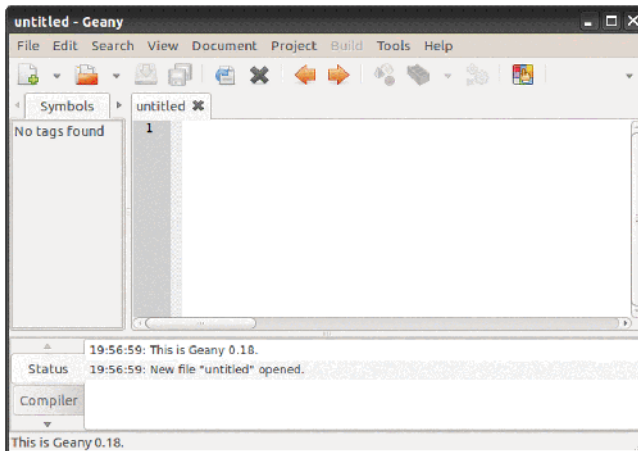



Fig. 5.1: Opening screen of Geany IDE in Ubuntu Linux

In this window, we type the program in a file with the default name **untitled**. To open a new file, choose **File** menu, then select **New** option or click New button  on the toolbar.

### b. Saving the program

Once a file is opened, enter the C++ program and save it with a suitable file name with extension **.cpp**. GCC

being a collection of compilers, the extension decides which compiler is to be selected for the compilation of the code. Therefore we have to specify the extension without fail. If we give the file name before typing the program, GCC provides different colours automatically to distinguish the types of tokens used in the program. It also uses indentation to identify the level of statements in the source code. We will discuss the concept of indentation later.

Geany IDE opens its window as shown in Figure 5.1. It has a title bar, menu bar, toolbar, and a code edit area. We can see a tab named **untitled**, which indicates the file name for the opened edit area. If we use Geany 1.24 on Windows operating system, the opening window will be as shown in Figure 5.2. We can see that both of these are quite the same.

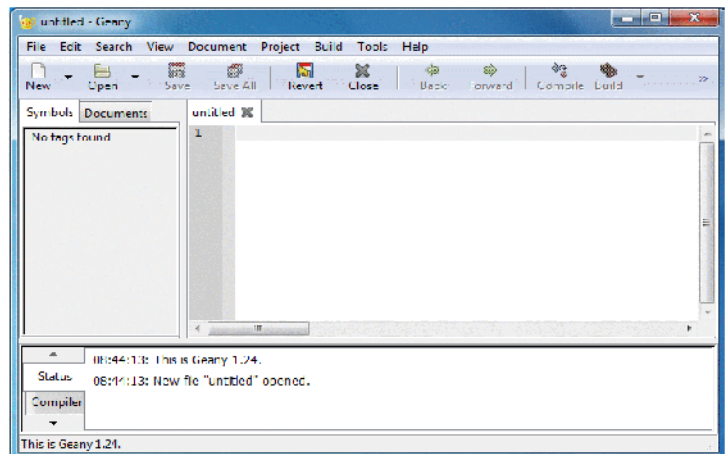


Fig. 5.2: Opening screen of Geany IDE 1.24 in Windows OS



Let us write a simple program given as Program 5.1 and save with the name `welcome.cpp`.

#### Program 5.1: A program to familiarise the IDE

```
// my first C++ program
#include<iostream>
using namespace std;
int main()
{
    cout << "Welcome to the world of C++";
    return 0;
} //end of program
```

The IDE window after entering Program 5.1 is shown in Figure 5.3. Observe the difference in colours used for the tokens.

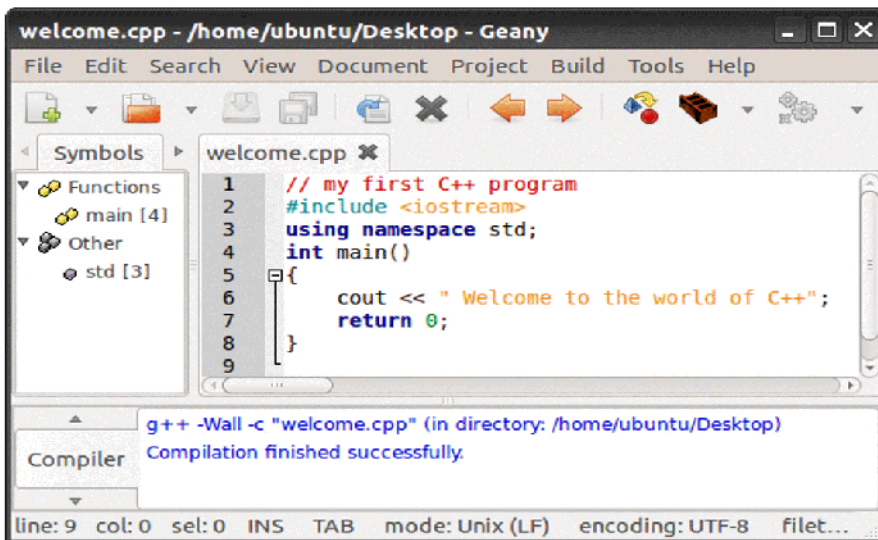



Fig. 5.3: Program saved with a name in Geany IDE


To save the program, choose **File** menu and select **Save** option or use the keyboard shortcut **Ctrl+S**. Alternatively the file can be saved by clicking the Save button  in the toolbar.


It is a good practice to save the program every now and then, just by pressing **Ctrl+S**. This helps to avoid the loss of data due to power failures or due to unexpected system errors. Once the program typing is completed, it is better to save the file before compiling or modifying. Copying the files from the temporary volatile primary memory to permanent non volatile secondary memory for storage is known as saving the program.




C++ program files should have a proper extension depending upon the implementation of C++. Different extensions are followed by different compilers. Examples are `.cpp`, `.cxx`, `.cc`, `.c++`

### C. Compiling and linking the program

The next step is to compile the program and modify it, if errors are detected. For this, choose **Build** menu and select **Compile** option. Alternatively we can also use the Compile button . If there are some errors, those errors will be displayed in the compiler status window at the bottom, otherwise the message **Compilation finished successfully** will be displayed. (refer Figure 5.3).

After successful compilation, choose **Build** menu and select **Build** option for linking or click the Build button  in the toolbar. Now the program is ready for execution.

### D. Running/Executing the program

Running the program is the process by which a computer carries out the instructions of a computer program. To run the program, choose **Build** menu and select **Execute** option. The program can also be executed by clicking the Execute button  in the toolbar. The output will be displayed in a new window as shown in Figure 5.4.

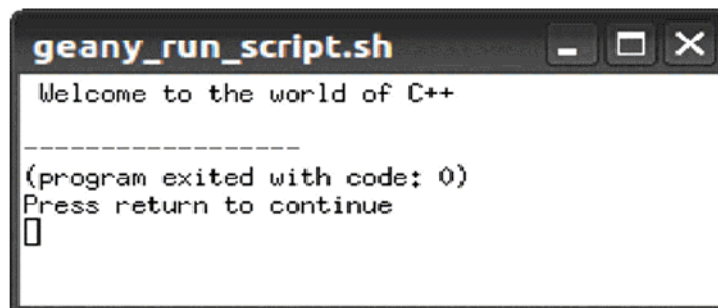



Fig. 5.4: Output window

### E. Closing the IDE

Once we have executed the program and desired output is obtained, it can be closed by selecting **Close** option from **File** menu or by clicking the Close button **X** in the active tab or in the title bar. For writing another program, a new file can be opened by the **New** option from the **File** menu or by clicking the New button  in the tool bar. The key combination **Ctrl+N** can also be used for the same.





After developing program, we can come out of Geany IDE by choosing **File** menu and selecting **Quit** option. The same can be achieved by clicking the Close button of the IDE window or by using the key combination **Ctrl+Q**.

**Let us do**

1. Write a program to print the message "SMOKING IS INJURIOUS TO HEALTH" on screen.
2. Write a program to display the message "TOBACCO CAUSES CANCER" on monitor.

**Let us sum up**

C++ was developed by Bjarne Stroustrup in early 1980s. C++ has its own character set. Tokens are the smallest unit of a program and are constituted by one or more characters in C++. There are five types of tokens namely keywords, identifiers, literals, punctuators and operators. Programs are written in computer with the help of an editor. Software like GCC and Geany IDE provide facilities to enter the source code in the computer, compile it and execute the object code.



### Learning outcomes

After the completion of this chapter the learner will be able to:

- list the C++ character set.
- categorise various tokens.
- identify keywords.
- write valid identifiers.
- classify various literals.
- identify the main components of Geany IDE.
- write, compile and run a simple program.



**Sample questions****Very short answer type**

1. What are the different types of characters in C++ character set?
2. What is meant by escape sequences?
3. Who developed C++?
4. What is meant by tokens? Name the tokens available in C++.
5. What is a character constant in C++?
6. How are non-graphic characters represented in C++? Give an example.
7. Why are the characters \ (slash), ' (single quote), " (double quote) and ? (question mark) typed using escape sequences?
8. Which escape sequences represent newline character and null character?
9. An escape sequence represents \_\_\_\_\_ characters.
10. Which of the following are valid character/string constants in C++?  
'c'      'anu'      "anu"      mine      'main's'      " "  
'char'      '\ \'
11. What is a floating point constant? What are the different ways to represent a floating point constant?
12. What are string-literals in C++? What is the difference between character constants and string literals?
13. What is the extension of C++ program file used for running?
14. Find out the invalid identifiers among the following. Give reason for their invalidity  
a) Principal amount      b) Continuc      c) Area      d) Date-of-join      e) 9B
15. A label in C++ is \_\_\_\_\_.  
a) Keyword      b) Identifier      c) Operator      d) Function
16. The following tokens are taken from a C++ program. Fill up the given table by placing them at the proper places  
(int, cin, %, do, =, "break", 25.7, digit)

Keywords	Identifiers	Literals	Operators

**Short answer type**

1. Write down the rules governing identifiers.
2. What are tokens in C++? How many types of tokens are allowed in C++? List them.
3. Distinguish between identifiers and keywords.
4. How are integer constants represented in C++? Explain with examples.
5. What are character constants in C++? How are they implemented?

**Long answer type**

1. Briefly describe different types of tokens.
2. Explain different types of literals with examples.
3. Briefly describe the Geany IDE and its important features.

## Key concepts

- **Concept of data types**
- **C++ data types**
- **Fundamental data types**
- **Type modifiers**
- **Variables**
- **Operators**
  - Arithmetic
  - Relational
  - Logical
  - Input/Output
  - Assignment
  - Arithmetic assignment
  - Increment and decrement
  - Conditional
  - sizeof
  - Precedence of operators
- **Expressions**
  - Arithmetic
  - Relational
  - Logical
- **Type conversion**
- **Statements**
  - Declaration
  - Assignment
  - Input /Output
- **Structure of a C++ program**
  - Pre-processor directives
  - Header files
  - Concept of namespace
  - The main() function
  - A sample program
- **Guidelines for coding**

## Data Types and Operators

In the previous chapter we familiarised ourselves with the IDE used for the development of C++ programs and also learnt the basic building blocks of C++ language. As we know, data processing is the main activity carried out in computers. All programming languages give importance to data handling. The input data is arranged and stored in computers using some structures. C++ has a predefined template for storing data. The stored data is further processed using operators. C++ also makes provisions for users to define new data types, called user-defined data types.

In this chapter, we will explore the main concepts of the C++ language like data types, operators, expressions and statements in detail.

### 6.1 Concept of data types

Consider the case of preparing the progress card of a student after an examination. We need data like admission number, roll number, name, address, scores in different subjects, the grades obtained in each subject, etc. Further, we need to display the percentage of marks scored by the student and the attendance in percentage. If we consider a case of scientific data processing, it may require data in the form of numbers representing the velocity of light ( $3 \times 10^8$  m/s), acceleration due to gravity (9.8 m/s), electric charge of an electron ( $-1.6 \times 10^{-19}$ ) etc.

From these cases, we can infer that data can be of different types like character, integer number, real number, string, etc. In the last chapter we saw that any valid character of C++ enclosed in single quotes represents character data in C++. Numbers without fractions represent integer data. Numbers with fractions represent floating point data and anything enclosed in double quotes represents a string data. Since the data to be dealt with are of many types, a programming language must provide ways and facilities to handle all types of data. C++ provides facilities to handle different types of data by providing data type names. **Data types** are the means to identify the nature of the data and the set of operations that can be performed on the data. Various data types are defined in C++ to differentiate these data characteristics.

In Chapter 4, we used variables to refer data in algorithms. Variables are also used in programs for referencing data. When we write programs in the C++ language, variables are to be declared before their use. Data types are necessary to declare these variables.

## 6.2 C++ data types

C++ provides a rich set of data types. Based on nature, size and associated operations, they are classified as shown in Figure 6.1. Basically, they are classified into fundamental or built-in data types, derived data types and user-defined data types.

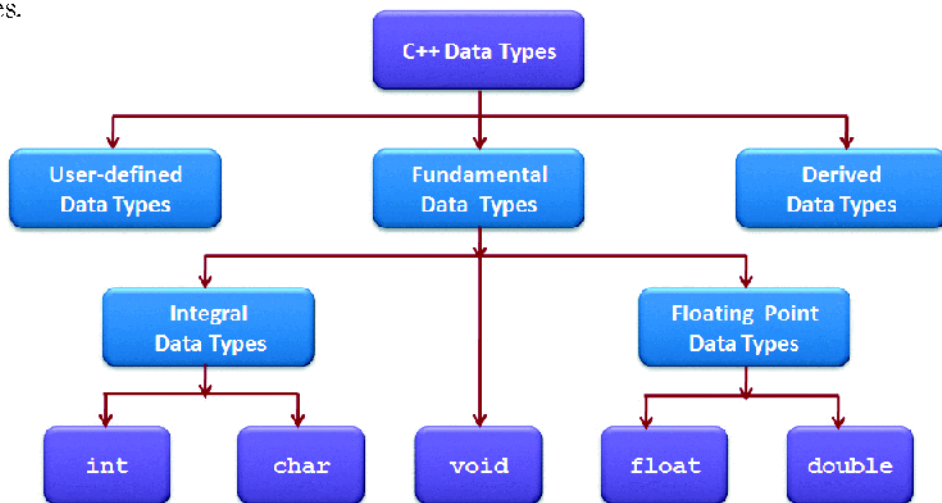


Fig 6.1 : Classification of C++ data types

### Fundamental data types

Fundamental data types are defined in C++ compiler. They are also known as built-in data types. They are atomic in nature and cannot be further decomposed of. The five fundamental data types in C++ are char, int, float, double and void. Among these, int and char comes under integral data type as they can handle



only integers. The numbers with fractions (real numbers) are generally known as floating type and are further divided into `float` and `double` based on precision and range.

### User-defined data types

C++ is flexible enough to allow programmers to define their own data types. Structure (`struct`), enumeration (`enum`), union, class, etc. are examples for such data types.

### Derived data types

Derived data types are constructed from fundamental data types through some grouping or alteration in the size. Arrays, pointers, functions, etc. are examples of derived data types.

## 6.3 Fundamental data types

Fundamental data types are basic in nature. They cannot be further broken into small units. Since these are defined in compiler, the size (memory space allocated) depends on the compiler. We use the compiler available in GCC and hence the size as well as the range of data supported by the data type are given accordingly. It may be different if you use other compilers like Turbo C++ IDE. The five fundamental data types are described below:

### **int** data type *(for integer numbers)*

Integers are whole numbers without a fractional part. They can be positive, zero or negative. The keyword **int** represents integer numbers within a specific range. GCC allows 4 bytes of memory for integers belonging to `int` data type. So the range of values that can be represented by `int` data type is from -2147483648 to +2147483647. The data items 6900100, 0, -112, 17, -32768, +32767, etc. are examples of `int` data type. The numbers 2200000000 and -2147483649 do not belong to `int` data type as they are out of the allowed range.

### **char** data type *(for character constants)*

Characters are the symbols covered by the character set of the C++ language. All letters, digits, special symbols, punctuations, etc. come under this category. When these characters are used as data they are considered as `char` type data in C++. We can say that the keyword **char** represents character literals of C++. Each `char` type data is allowed one byte of memory. The data items 'A', '+', '\t', '0', etc. belong to `char` data type. The `char` data type is internally treated as integers, because computer recognises the character through its ASCII code. Character data is stored in the memory with the corresponding ASCII code. As ASCII codes are integers and need to be stored in one byte (8 bits), the range of `char` data type is from -128 to +127.

**float data type (for floating point numbers)**

Numbers with a fractional part are called floating point numbers. Internally, floating-point numbers are stored in a manner similar to scientific notation. The number 47281.97 is expressed as  $0.4728197 \times 10^5$  in scientific notation. The first part of the number, 0.4728197 is called the mantissa. The power 5 of 10 is called exponent. Computers typically use exponent form (*E notation*) to represent floating-point values. In E notation, the number 47281.97 would be 0.4728197E5. The part of the number before the E is the mantissa, and the part after the E is the exponent. In C++, the keyword **float** is used to denote such numbers. GCC allows 4 bytes of memory for numbers belonging to float data type. The numbers of this data type has normally a precision of 7 digits.

**double data type (for double precision floating point numbers)**

In some cases, floating point numbers require more precision. Such numbers are represented by **double** data type. The range of numbers that can be handled by float type is extended by this data type, because it consumes double the size of float data type. In C++, it is assured that the range and precision of double will be at least as big as float. GCC reserves 8 bytes for storing a double precision value. The precision of double data type is generally 15 digits.

**void data type (for null or empty set of values)**

The data type **void** is a keyword and it indicates an empty set of data. Obviously it does not require any memory space. The use of this data type will be discussed in detail in Chapter 10.

The size of fundamental data types decreases in the order double, float, int and char.

**6.4 Type modifiers**

Have you ever seen travel bags that can alter its size/volume to include extra bit of luggage? Usually we don't use that extra space. But the zipper attached with the bag helps us to alter its volume either by increasing it or by decreasing. In C++ too, we need data types that can accommodate data of slightly bigger/smaller size. C++ provides **data type modifiers** which help us to alter the size, range or precision. Modifiers precede the data type name in the variable declaration. It alters the range of values permitted to a data type by altering the memory size and sign of values. Important modifiers are **signed**, **unsigned**, **long** and **short**.





The exact sizes of these data types depend on the compiler and computer you are using. It is guaranteed that:

- a double is at least as big as a float.
- a long double is at least as big as a double.

Each type and their modifiers are listed in Table 6.1 (based on GCC compiler) with their features.

Name	Description	Size	Range
char	Character	1 byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer	2 bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
float	Floating point number	4 bytes	$-3.4 \times 10^{+/-38}$ to $+3.4 \times 10^{+/-38}$ with approximately 7 significant digits
double	Double precision floating point number	8 bytes	$-1.7 \times 10^{+/-308}$ to $+1.7 \times 10^{+/-308}$ with approximately 15 significant digits
long double	Long double precision floating point number	12 bytes	$-3.4 \times 10^{+/-4932}$ to $+3.4 \times 10^{+/-4932}$ With approximately 19 significant digits

*Table 6.1: Data type and type modifiers*



The values listed in Table 6.1 are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system.

## 6.5 Variables

Memory locations are to be identified to refer data. **Variables** are the names given to memory locations. These are identifiers of C++ by which memory locations are referenced to store or retrieve data. The size and nature of data stored in a variable depends on the data type used to declare it. There are three important aspects for a variable.



### i. Variable name

It is a symbolic name (identifier) given to the memory location through which the content of the location is referred to.

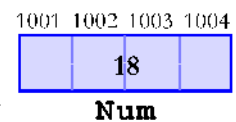
### ii. Memory address

The RAM of a computer consists of collection of cells each of which can store one byte of data. Every cell (or byte) in RAM will be assigned a unique address to refer it. All the variables are connected to one or more memory locations in RAM. The base address of a variable is the starting address of the allocated memory space. In the normal situation, the address is given implicitly by the compiler. The address is also called the L-value of a variable. In figure 6.2 the base address of the variable **Num** is 1001.

### iii. Content

The value stored in the location is called the content of the variable. This is also called the R-value of the variable. Type and size of the content depends on the data type of the variable.

Figure 6.2, shows the memory representation of a variable. Here the variable name is **Num** and it consumes 4 bytes of memory at memory addresses 1001, 1002, 1003 and 1004. The content of this variable is 18. That is the *L-value* of **Num** is 1001 and the *R-value* is 18.



*Fig. 6.2 : Memory representation of a Variable*

## 6.6 Operators

**Operators** are tokens constituted by predefined symbols that trigger computer to carry out operations. The participants of an operation are called **operands**. An operand may be either a constant or a variable.

For example,  $a+b$  triggers an arithmetic operation in which  $+$  (addition) is the operator and  $a, b$  are operands. Operators in C++ are classified based on various criteria. Based on number of operands required for the operation, operators are classified into three. They are unary, binary and ternary.

### Unary operators

A unary operator operates on a single operand. Commonly used unary operators are  $\text{unary}+$  (positive) and  $\text{unary}-$  (negative). These are used to represent the sign of a number. If we apply  $\text{unary}+$  operator on a signed number, the existing sign will not change. If we apply  $\text{unary}-$  operator on a signed number, the sign of the existing



number will be negated. Examples of the use of unary operators are given in Table 6.2.

Some other examples of unary operators are increment (**++**) and decrement (**--**) operators.

### Binary operators

Binary operators operate on two operands. Arithmetic operators, relational operators, logical operators, etc. are commonly used binary operators.

### Ternary operator

Ternary operator operates on three operands. The typical example is the conditional operator (**? :** ).

The operations triggered by the operators mentioned above will be discussed in detail in the coming sections and some of them will be dealt with in Chapter 7.

Based on the nature of operation, operators are classified into arithmetic, relational, logical, input/output, assignment, short-hand, increment/decrement, etc.

## 6.6.1 Arithmetic operators

Arithmetic operators are defined to perform basic arithmetic operations such as addition, subtraction, multiplication and division. The symbols used for this are **+**, **-**, **\*** and **/** respectively. C++ also provides a special operator, **%** (modulus operator) for getting remainder during division. All these operators are binary operators. Note that **+** and **-** are used as unary operators too. The operands required for these operations are numeric data. The result of these operations will also be numeric. Table 6.3 shows some examples of binary arithmetic operations.

Variable <b>x</b>	Variable <b>y</b>	Addition <b>x + y</b>	Subtraction <b>x - y</b>	Multiplication <b>x * y</b>	Division <b>x / y</b>
10	5	15	5	50	2
-11	3	-8	-14	-33	-3.66667
11	-3	8	14	-33	-3.66667
-50	-10	-60	-40	500	5

Table 6.3 : Arithmetic operators

### Modulus operator (%)

The modulus operator, also called as mod operator, gives the remainder value during arithmetic division. This operator can only be applied over integer operands.

Variable <b>x</b>	Unary + <b>+x</b>	Unary- <b>-x</b>
8	8	-8
0	0	0
-9	-9	9

Table 6.2 : Unary operators



Table 6.4 shows some examples of modulus operation. Note that the sign of the result is the sign of the first operand. Here in the table the first operand is **x**.

Variable <b>x</b>	Variable <b>y</b>	Modulus Operation <b>x % y</b>	Variable <b>x</b>	Variable <b>y</b>	Modulus Operation <b>x % y</b>
10	5	0	100	100	0
5	10	5	32	11	10
-5	11	-5	11	-5	1
5	-11	5	-11	5	-1
-11	-5	-1	-5	-11	-5

Table 6.4 : Operations using Modulus operator

### Check yourself



1. Arrange the fundamental data types in ascending order of size.
2. The name given to a storage location is known as \_\_\_\_\_.
3. Name a ternary operator in C++.
4. Predict the output of the following operations if  $x = -5$  and  $y = 3$  initially:
  - a.  $-x$
  - b.  $-y$
  - c.  $-x + -y$
  - d.  $-x - y$
  - e.  $x \% -11$
  - f.  $x + y$
  - g.  $x \% y$
  - h.  $x / y$
  - i.  $x * -y$
  - j.  $-x \% -5$

### 6.6.2 Relational operators

Relational operators are used for comparing numeric data. These are binary operators. The result of any relational operation will be either **True** or **False**. In C++, True is represented by **1** and False is represented by **0**. There are six relational operators in C++. They are **<** (*less than*), **>** (*greater than*), **<=** (*less than or equal to*), **>=** (*greater than or equal to*), **==** (*equal to*) and **!=** (*not equal to*). Note that equality checking requires two equal symbols (**==**). Some examples for the use of various relational operators and their results are shown in Table 6.5.



m	n	m<n	m>n	m<=n	m>=n	m!=n	m==n
12	5	0	1	0	1	1	0
-7	2	1	0	1	0	1	0
4	4	0	0	1	1	0	1

Table 6.5 : Operations using Relational operators

### 6.6.3 Logical operators

Using relational operators, we can compare values. Examples are  $3 < 5$ ,  $\text{num} != 10$ , etc. These comparison operations are called relational expressions in C++. In some cases, two or more comparisons may need to be combined. In Mathematics we may use expressions like  $a > b > c$ . But in C++ it is not possible. We have to separate this into two, as  $a > b$  and  $b > c$  and these are to be combined using the logical operator **&&**, i.e.  $(a > b) \&\& (b > c)$ . The result of such logical combinations will also be either True or False (i.e. 1 or 0). The logical operators are **&&** (logical AND), **||** (logical OR) and **!** (logical NOT).

#### Logical AND (&&) operator

If two relational expressions E1 and E2 are combined using logical AND (**&&**) operator, the result will be 1 (True) only if both E1 and E2 have values 1 (True). In all other cases the result will be 0 (False). The results of evaluation of **&&** operation for different possible combination of inputs are shown in Table 6.6.

E1	E2	E1&&E2
0	0	0
0	1	0
1	0	0
1	1	1

Table 6.6 : Logical AND

Examples:  $10 > 5 \&\& 15 < 25$  evaluates to 1 (True)

$10 > 5 \&\& 100 < 25$  evaluates to 0 (False)

#### Logical OR (||) operator

If two relational expressions E1 and E2 are combined using logical OR (**||**) operator, the result will be 0 (False) only if both E1 and E2 are having value 0 (False). In all other cases the result will be 1 (True). The results of evaluation of **||** operation for different possible combination of inputs are shown in Table 6.7.

E1	E2	E1  E2
0	0	0
0	1	1
1	0	1
1	1	1

Table 6.7 : Logical OR

Examples:  $10 > 5 || 100 < 25$  evaluates to 1 (True)

$10 > 15 || 100 < 90$  evaluates to 0 (False)

### Logical NOT operator (!)

This operator is used to negate the result of a relational expression. This is a unary operation. The results of evaluation of ! operator for different possible inputs are shown in Table 6.8.

E1	!E1
0	1
1	0

Table 6.8 :  
Logical NOT

Example:  $!(100 < 2)$  evaluates to 1 (True)

$!(100 > 2)$  evaluates to 0 (False)

### 6.6.4 Input / Output operators

Usually input operation requires user's intervention. In the process of input operation, the data given through the keyboard is stored in a memory location. C++ provides  $>>$  operator for this operation. This operator is known as **get from** or **extraction** operator. This symbol is constituted by two greater than symbols.

Similarly in output operation, data is transferred from RAM to an output device. Usually the monitor is the standard output device to get the results directly. The operator  $<<$  is used for output operation and is called **put to** or **insertion** operator. It is constituted by two less than symbols.

### 6.6.5 Assignment operator (=)

When we have to store a value in a memory location, assignment operator (=) is used. This is a binary operator and hence two operands are required. The first operand should be a variable where the value of the second operand is to be stored. Some examples are shown in table 6.9.

Item	Description
$a = b$	The value of variable <b>b</b> is stored in <b>a</b>
$a = 3$	The constant <b>3</b> is stored in variable <b>a</b>

Table 6.9 : Assignment operator

We discussed the usage of the relational operator  $==$  in Section 6.6.2. See the difference between these two operators. The  $=$  symbol assigns a value to a variable, whereas  $==$  symbol compares two values and gives True or False as the result.

### 6.6.6 Arithmetic assignment operators

A simple arithmetic statement can be expressed in a more condensed form using arithmetic assignment operators. For example,  $a = a + 10$  can be represented as  **$a += 10$** . Here  **$+=$**  is an arithmetic assignment operator. This method is applicable





to all arithmetic operators and they are shown in Table 6.10. The arithmetic assignment operators in C++ are `+=`, `-=`, `*=`, `/=`, `%=`. These are also known as C++ short-hands. These are all binary operators and the first operand should be a variable. The use of these operators makes the two operations (arithmetic and assignment) faster than the usual method.

Arithmetic assignment operation	Equivalent arithmetic operation
<code>x += 10</code>	<code>x = x + 10</code>
<code>x -= 10</code>	<code>x = x - 10</code>
<code>x *= 10</code>	<code>x = x * 10</code>
<code>x /= 10</code>	<code>x = x / 10</code>
<code>x %= 10</code>	<code>x = x % 10</code>

Table 6.10 : C++ short hands

### 6.6.7 Increment (++) and Decrement (--) operators

Increment and decrement operators are two special operators in C++. These are unary operators and the operand should be a variable. These operators help keeping the source code compact.

#### Increment operator (++)

This operator is used for incrementing the content of an integer variable by one. This can be written in two ways: `++x` (pre increment) and `x++` (post increment). Both are equivalent to `x=x+1` as well as `x+=1`.

#### Decrement operator (--)

As a counterpart of increment operator, there is a decrement operator which decrements the content of an integer variable by one. This operator is also used in two ways: `--x` (pre decrement) and `x--` (post decrement). These are equivalent to `x=x-1` and `x-=1`.

The two usages of these operators are called prefix form and postfix form of increment/decrement operation. Both the forms make the same effect on the operand variable, but the mode of operation will be different when these are used with other operators.

#### Prefix form of increment/decrement operators

In the prefix form, the operator is placed before the operand and the increment/decrement operation is carried out first. The incremented/decremented value is used for the other operations. So, this method is often called *change, then use* method.

Consider the variables `a`, `b`, `c` and `d` with values `a=10`, `b=5`. If an operation is specified as `c=++a`, the value of `a` will be 11 and that of `c` will also be 11. Here the value of `a` is incremented by 1 at first and then the changed value of `a` is assigned



to c. That is why both the variables get the same value. Similarly, after the execution of `d--b` the value of d and b will be 4.

### Postfix form of increment/decrement operators

When increment/decrement operation is performed in postfix form, the operator is placed after the operand. The current value of the variable is used for the remaining operations and after that the increment/decrement operation is carried out. So, this method is often called *use, then change* method.

Consider the same variables used above with the same initial values. After the operation performed with `c=a++`, the value of a will be 11, but that of c will be 10. Here the value of a is assigned to c at first and then a is incremented by 1. That is, before changing the value of a it is used to assign to c. Similarly, after the execution of `d=b--` the value of d will be 5 but that of b will be 4.

### 6.6.8 Conditional operator (?:)

This is a ternary operator applied over three operands. The first operand will be a logical expression (condition) and the remaining two are values. They can be constants, variables or expressions. The condition will be checked first and if it is True, the second operand will be selected to get the value, otherwise the third operand will be selected. Its syntax is:

```
Expression1? Expression2: Expression3
```

Let us see the operation in the following:

```
result = score>50 ? 'p' : 'f'
```

If the value of score is greater than 50 then the value 'p' is assigned to the variable result, else value 'f' is assigned to result. More about this operator will be discussed in Chapter 7.

### 6.6.9 sizeof operator

The operator `sizeof` is a unary compile-time operator that returns the amount of memory space in bytes allocated for the operand. The operand can be a constant, a variable or a data type. The syntax followed is given below:

- `sizeof (data_type)`
- `sizeof variable_name`
- `sizeof constant`

It is to be noted that when data type is used as the operand for `sizeof` operator, it should be given within a pair of parentheses. For the other operands parentheses are not compulsory. Table 6.11 shows different forms of usages of `sizeof` operator.



Item	Description
<code>sizeof(int)</code>	Gives the value 4 (In GCC, size of int data type is 4 bytes)
<code>sizeof 3.2</code>	Returns 8 (A floating point constant will be taken as double type data)
<code>sizeof p;</code>	If p is float type variable, it gives the value 4.

Table 6.11: Various usages of `sizeof` operator

### 6.6.10 Precedence of operators

Let us consider the case where different operators are used with the required operands. We should know in which order the operations will be carried out. C++ gives priority to the operators for execution. During evaluation, pair of parentheses is given the first priority. If the expression is not parenthesised, it is evaluated according to the predefined precedence order. The order of precedence for the operators is given in Table 6.12. In an expression, if the operators of the same priority level occur, the precedence of execution will be from left to right in most of the cases.

Priority	Operations
1	( ) parentheses
2	++, --, !, Unary+, Unary -, sizeof
3	* (multiplication), / (division), % (Modulus)
4	+ (addition), - (subtraction)
5	< (less than), <= (less than or equal to), > (greater than), >= (greater than or equal to)
6	== (equal to), != (not equal to)
7	&& (logical AND)
8	(logical OR)
9	? : (Conditional expression)
10	= (Assignment operator), *=, /=, %=, +=, -= (arithmetic assignment operators)
11	, (Comma)

Table 6.12: Precedence of operators

Consider the variables with values: `a=3, b=5, c=4, d=2, x`

After the operations specified in `x = a + b * c - d`, the value in `x` will be 21. Here \* (multiplication) has higher priority than + (addition) and - (subtraction). Therefore the variables `b` and `c` are multiplied, then that result is added to `a`. From that result, `d` is subtracted to get the final result.



It is important to note that the operator priority can be changed in an expression as per the need of the programmer by using parentheses (). For example, if  $a=5$ ,  $b=4$ ,  $c=3$ ,  $d=2$  then the result of  $a+b-c*d$  will be 3. Suppose the programmer wants to perform subtraction first and then the addition and multiplication, you need to use proper parentheses as  $(a+(b-c)) * d$ . Now the output will be 12. For changing operator priority, brackets [] and braces {} cannot be used.



The operator precedence may be different for different types of compilers. Turbo C++ gives higher precedence to prefix increment / decrement than its postfix form.

For example, if  $a$  is initially 5, the values of  $b$  and  $a$  after  $b=a++ + ++a$  are 12 and 7 respectively. This is equivalent to the set of statements  $a=a+1$  (prefix expansion),  $b=a+a$ , and  $a=a+1$  (postfix expansion).

## 6.7 Expressions

An expression is composed of operators and operands. The operands may be either constants or variables. All expressions can be evaluated to get a result. This result is known as the value returned by the expression. On the basis of the operators used, expressions are mainly classified into arithmetic expressions, relational expressions and logical expressions.

### 6.7.1 Arithmetic expressions

An expression in which only arithmetic operators are used is called arithmetic expression. The operands are numeric data and they may be variables or constants. The value returned by these expressions is also numeric. Arithmetic expressions are further classified into integer expressions, floating point (real) expressions and constant expressions.

#### Integer expressions

If an arithmetic expression contains only integer operands, it is called integer expression and it produces an integer result after performing all the operations given in the expression. For example, if  $x$  and  $y$  are integer variables, some integer expressions and their results are shown in Table 6.13. Note that all the above expressions produce integer values as the results.

$x$	$y$	$x + y$	$x / y$	$-x + x * y$	$5 + x / y$	$x \% y$
5	2	7	2	5	7	1
6	3	9	2	12	7	0

Table 6.13: Integer expressions and their results



### Floating point expressions (Real expressions)

An arithmetic expression that is composed of only floating point data is called floating point or real expression and it returns a floating point result after performing all the operations given in the expression. Table 6.14 shows some real expressions and their results, assuming that  $x$  and  $y$  are floating point variables.

$x$	$y$	$x + y$	$x / y$	$-x + x * y$	$5 + x / y$	$x * x / y$
5.0	2.0	7.0	2.5	5.0	7.5	12.5
6.0	3.0	9.0	2.0	12.0	7.0	12.0

Table 6.14: Floating point expressions and their results

It can be seen that all the above expressions produce floating point values as the results.

In an arithmetic expression, if all the operands are constant values, then it is called **constant expression**. The expression  $20 + 5/2.0$  is an example. The constants like 15, 3.14, 'A' are also known as constant expressions.

### 6.7.2 Relational expressions

When relational operators are used in an expression, it is called relational expression and it produces Boolean type results like True (1) or False (0). In these expressions, the operands are numeric data. Let us see some examples of relational expressions in Table 6.15.

$x$	$y$	$x > y$	$x == y$	$x+y != y$	$x-2 == y+1$	$x*y == 6*y$
5.0	2.0	1 (True)	0 (False)	1 (True)	1 (True)	0 (False)
6	13	0 (False)	0 (False)	1 (True)	0 (False)	1 (True)

Table 6.15: Relational expressions and their results

We know that arithmetic operators have higher priority than relational operators. So when arithmetic expressions are used on either side of a relational operator, arithmetic operations will be carried out first and then the results are compared. The table contains some expressions in which both arithmetic and relational operators are involved. Though they contain mixed type of operators, they are called relational expressions since the final result will be either True or False.

### 6.7.3 Logical expressions

Logical expressions combine two or more relational expressions with logical operators and produce either True or False as the result. A logical expression may contain constants, variables, logical operators and relational operators. Let us see some examples in Table 6.16.



x	y	$x > y \ \&\& \ x == 20$	$x == 5 \    \ y == 0$	$x == y \ \&\& \ y + 2 == 0$	$! (x == y)$
5.0	2.0	0 (False)	1 (True)	0 (False)	1 (True)
20	13	1 (True)	0 (False)	0 (False)	1 (True)

Table 6.16: Logical expressions and their results

As seen in Table 6.16, though some expressions consist of arithmetic and relational operators in addition to logical operators, the expressions are considered as logical expressions. This is because the operation carried out at last will be the logical operation and the result will be either 'True' or 'False'.

### Check yourself



- Predict the output of the following operations if  $x=5$  and  $y=3$ .
  - $x >= 10 \ \&\& \ y >= 4$
  - $x >= 1 \ \&\& \ y >= 3$
  - $x >= 1 \ || \ y >= 4$
  - $x >= 1 \ || \ y >= 3$
- Predict the output if  $p=5$ ,  $q=3$ ,  $r=2$ 
  - $++p - q * r / 2$
  - $p * q -- + r$
  - $p - q - r * 2 + p$
  - $p += 5 * q + r * r / 2$

## 6.8 Type conversion

As discussed earlier arithmetic expressions are of two types, integer expressions and real expressions. In both cases, the operands involved in the arithmetic operation are of the same data type. But there are situations where different types of numeric data may be involved. For example in C++, the integer expression  $5/2$  gives 2 and the real expression  $5.0/2.0$  gives 2.5. But what will the result of  $5/2.0$  or  $5.0/2$  be? Conversion techniques are applied in such situations. The data type of one operand will be converted to another. It is called **type conversion** and can be done in two ways: implicitly and explicitly.

### 6.8.1 Implicit type conversion (Type promotion)

Implicit type conversion is performed by C++ compiler internally. In expressions where different types of data are involved, C++ converts the lower sized operands to the data type of highest sized operand. Since the conversion is always from lower type to higher, it is also known as **type promotion**. Data types in the decreasing order of size are as follows: long double, double, float, unsigned long, long int and unsigned int / short int. The type of the result will also be the type of the highest sized operand.





For example, the expression  $5 / 2 * 3 + 2.5$  gives the result 8.5. The evaluation steps are as follows:

Step 1:  $5 / 2 \rightarrow 2$  (Integer division)

Step 2:  $2 * 3 \rightarrow 6$  (Integer multiplication)

Step 3:  $6 + 2.5 \rightarrow 8.5$  (Floating point addition, 6 is converted into 6.0)

### 6.8.2 Explicit type conversion (Type casting)

Unlike implicit type conversion, sometimes the programmer may decide the data type of the result of evaluation. This is done by the programmer by specifying the data type within parentheses to the left of the operand. Since the programmer explicitly casts a data to the desired type, it is known as explicit type conversion or **type casting**. Usually, type casting is applied on the variables in the expressions. More examples will be discussed in Section 6.9.2.

## 6.9 Statements

Can you recollect the learning hierarchy of a natural language? Alphabet, words, phrases, sentences, paragraphs and so on. In the learning process of C++ language we have covered character set, tokens and expressions. Now we have come to the stage where we start communication with the computer sensibly and meaningfully with the help of statements. **Statements** are the smallest executable unit of a programming language. C++ uses the symbol semicolon ( ; ) as the delimiter of a statement. Different types of statements used in C++ are declaration statements, assignment statements, input statements, output statements, control statements etc. Each statement has its own purpose in a C++ program. All these statements except declaration statements are executable statements as they possess some operations to be done by the computer. Executable statements are the instructions to the computer. The execution of control statements will be discussed in Chapter 7. Let us discuss the other statements.

### 6.9.1 Declaration statements

Every user-defined word should be defined in the program before it is used. We have seen that a variable is a user-defined word and it is an identifier of a memory location. It must be declared in the program before its use. When we declare a variable, we tell the compiler about the type of data that will be stored in it. The syntax of variable declaration is:

```
data_type <variable1>[, <variable2>, <variable3>, ...];
```

The `data_type` in the syntax should be any valid data type of C++. The syntax shows that when there are more than one variables in the declaration, they are separated by comma. The declaration statement ends with a semicolon. Typically, variables are declared either just before they are used or at the beginning of the



program. In the syntax, everything given inside the symbols [ and ] are optional. The following statements are examples for variable declaration:

```
int rollnumber;
double wgpa, avg_score;
```

The first statement declares the variable `rollnumber` as **int** type so that it will be allocated four bytes of memory (*as per GCC*) and it can hold an integer number within the range from -2147483648 to +2147483647. The second statement defines the identifiers `wgpa` and `avg_score` as variables to hold data of **double** type. Each of them will be allocated 8 bytes of memory. The memory is allocated to the variables during the compilation of the program.

### Variable initialisation

We saw, in Section 6.5, that a variable is associated with two values: L-value (its address) and R-value (its content). When a variable is declared, a memory location with an address will be allocated for it. What will its content be? It is not blank or 0 or space! If the variable is declared with `int` data type, the content or R-value will be any integer within the allowed range. But this number cannot be predicted or will not always be the same. So we call it *garbage value*. When we store a value into the variable, the existing content will be replaced by the new one. The value can be stored in the variable either at the time of compilation or execution. Supplying value to a variable at the time of its declaration is called **variable initialisation**. This value will be stored in the respective memory location during compile-time. The assignment operator (=) is used for this. It can be done in two ways as given below:

```
data_type variable = value;
OR
data_type variable(value)
```

The statements: `int xyz =120;` and `int xyz(120);` are examples of variable initialisation statements. Both of these statements declare an integer variable `xyz` and store the value 120 in it as shown in Figure 6.3.

More examples are:

```
float val=0.12, b=5.234;
char k='A';
```

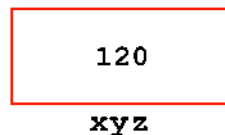


Fig. 6.3: Variable initialisation

A variable can also be initialised during the execution of the program and is known as dynamic initialisation. This is done by assigning an expression to a variable as shown in the following statements:

```
float product = x * y;
float interest = p*n*r/100.0;
```



In the first statement, the variable `product` is initialised with the product of the values stored in `x` and `y` at runtime. In the second case, the expression `p*n*r/100.0` is evaluated and the value returned by it will be stored in the variable `interest`.

Note that during dynamic initialisation, the variables included in the expression at the right of assignment operator should have valid data. Otherwise it will produce unexpected results.

### **const – The access modifier**

It is a good practice to use symbolic constants rather than using numeric constants directly. For example, we can use symbolic names like `Pi` instead of using `22.0/7.0` or `3.14`. The keyword **const** is used to create such symbolic constants whose value can never be changed during execution. Consider the following statement:

```
float pi=3.14;
```

The floating point variable **pi** is initialised with the value 3.14. The content of **pi** can be changed during the execution of the program. But if we modify the declaration as:

```
const float pi=3.14;
```

the value of **pi** remains constant (unaltered) throughout the execution of the program. The read/write accessibility of the variable is modified as read only. Thus, the **const** acts as an access modifier.



During software development, larger programs are developed using collaborative effort. Several people may work together on different portions of the same program. They may share the same variable. In these situations, there may be occasions where one may modify the content of the variable which will adversely affect other person's coding. In these situations we have to keep the content of variables unaffected by the activity of others. This can be done by using 'const'.

### **6.9.2 Assignment statements**

When the assignment operator (`=`) is used to assign a value to a variable, it forms an assignment statement. It can take any of the following syntax:

```
variable = constant;
variable1 = variable2;
variable = expression;
variable = function();
```

In the third case, the result of the expression is stored in the variable. Similarly, in the fourth case, the value returned by the function is stored. The concept of functions will be discussed in Chapter 10.



Some examples of assignment statements are given below:

```
A = 15;           b = 5.8;
c = a + b;        c = a * b;
d = (a + b) * (c + d);  r = sqrt(25);
```

In the last example, `sqrt()` is a function that assigns the square root of 25 to the variable `r`.

The left hand side (LHS) of an assignment statement must be a variable. During execution, the expression at the right hand side (RHS) is evaluated first. The result is then assigned (stored) to the variable at LHS.

Assignment statement can be chained for doing multiple assignments at a time. For instance, the statement `x=y=z=13;` assigns the value 13 in three variables in the order of `z`, `y` and `x`. The variables should be declared before this assignment. If we assign a value to a variable, the previous value in it, if any, will be replaced by the new value.

### Type compatibility

During the execution of an assignment statement, if the data type of the RHS expression is different from that of the LHS variable, there are two possibilities.

- The size of the data type of the variable at LHS is higher than that of the variable or expression at RHS. In this case data type of the value at RHS is promoted (type promotion) to that of the variable at LHS. Consider the following code snippet:

```
int a=5, b=2;
float p, q;
p = b;
q = a / p;
```

Here the data type of `b` is promoted to `float` and 2.0 is stored in `p`. When the expression `a/p` is evaluated, the result will be 2.5 due to the type promotion of `a`. So, `q` will be assigned with 2.5.

- The second possibility is that the size of the data type of LHS variable is smaller than the size of RHS value. In this case, the higher order bits of the result will be truncated to fit in the variable location of LHS. The following code illustrates this.

```
float a=2.6;
int p, q;
p = a;
q = a * 4;
```

Here the value of `p` will be 2 and that of `q` will be 10. The expression `a*4` is evaluated to 10.4, but `q` being `int` type it will hold only 10.



Programmer can apply the explicit conversion technique to get the desired results when there are mismatches in the data types of operands. Consider the following code segment.

```
int p=5, q=2;
float x, y;
x = p/q;
y = (x+p)/q;
```

After executing the above code, the value of  $x$  will be 2.0 and that of  $y$  will be 3.5. The expression  $p/q$  being an integer expression gives 2 as the result and is stored in  $x$  as floating point value. In the last statement, the pair of parentheses gives priority to  $x+p$  and the result will be 7.0 due to the type promotion of  $p$ . Then the result 7.0 will be the first operand for the division operation and hence the result will be 3.5 since  $q$  is converted into float. If we have to get the floating point result from  $p/q$  to store in  $x$ , the statement should be modified as  $x = (\text{float}) p/q$ ; or  $x = p/(\text{float}) q$ ; by applying type casting.

### 6.9.3 Input statements

Input statement is a means that allows the user to store data in the memory during the execution of the program. We saw that the *get from* or *extraction* operator ( $\gg$ ) specifies the input operation. The operands required for this operator are the input device and a location in RAM where data is to be stored. Keyboard being a standard console device, the stream (sequence) of data is extracted from the keyboard and stored in memory locations identified by variables. Since C++ is an object oriented language, keyboard is considered as the standard input stream device and is identified as an object by the name **cin** (pronounced as 'see in'). The simplest form of an input statement is:

```
streamobject >> variable;
```

Since we use keyboard as the input device, the **streamobject** in the syntax will be substituted by **cin**. The operand after the  $\gg$  operator should strictly be a variable. For example, the following statement reads data from the keyboard and stores in the variable **num**.

```
cin >> num;
```

Figure 6.4 shows how data is extracted from keyboard and stored in the variable.

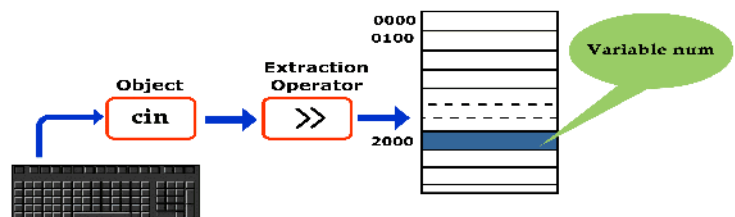


Fig 6.4 : Input procedure in C++



### 6.9.2 Output statements

Output statements make the results available to users through any output device. The *put to* or *insertion* operator ( $\ll$ ) is used to specify this operation. The operands in this case are the output device and the data for the output. The syntax of an output statement is:

```
streamobject << data;
```

The *streamobject* may be any output device and the *data* may be a constant, a variable or an expression. We use *monitor* as the commonly used output device and C++ identifies it as an object by the name **cout** (pronounced as 'see out'). The following are some examples of output statement with *monitor* as the output device:

```
cout << num;
cout << "hello friends";
cout << num+12;
```

The first statement displays the content of the variable **num**. The second statement displays the string constant "hello friends" and the last statement shows the value returned by the expression  $\text{num}+12$  (assuming that *num*

contains numeric value). Figure 6.5 shows how data is inserted into the output stream object (*monitor*) from the memory location **num**.

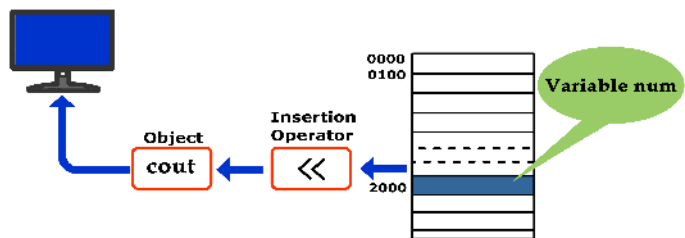


Fig. 6.5: Output procedure in C++



The tokens *cin* and *cout* are not keywords. They are predefined words that are not part of the core C++ language, and you are allowed to redefine them. They are defined in libraries required by the C++ language standard. Needless to say, using a predefined identifier for anything other than its standard meaning can be confusing and dangerous and such practice should be avoided. The safest and easiest practice is to treat all predefined identifiers as if they were keywords.

### Cascading of I/O operators

Suppose you want to input three values to different variables, say *x*, *y*, and *z*. You may use the following statements:

```
cin>>x;
cin>>y;
cin>>z;
```





But these three statements can be combined to form a single statement as given below:

```
cin>>x>>y>>z;
```

The multiple use of input or output operators in a single statement is called **cascading of I/O operators**. In the use of cascading of input operators, the values input are assigned to the variables from left to right. In the example `cin>>x>>y>>z;` the first value is assigned to `x`, the second to `y` and the third to `z`. While entering values to the variables `x`, `y` and `z` during execution the values should be separated by space bar, tab, or carriage return.

Similarly, if you want to display the contents of different variables (say `x`, `y`, `z`), use the following statement:

```
cout<<x<<y<<z;
```

If variables, constants and expressions appear together for output operations, the above technique can be applied as in the following example:

```
cout<<"The number is "<<z;
```

While cascading output operators, the values for the output will be retrieved from right to left. Consider the code fragment given below:

```
int x=5;
cout<<x<<' \t'<<++x;
```

The output of this code will be:      6      6

It will not be:    5      6

It is to be noted that both `<<` and `>>` operators cannot be used in a single statement.

In the statement `x=y=z=5;` the `=` operator is cascaded. Here also the cascading is from right to left.

## 6.10 Structure of a C++ program

We are now in a position to solve simple problems by using the statements we discussed so far. But a set of statements alone does not constitute a program. A C++ program has a typical structure. It is a collection of one or more functions. A function means the set of instructions to perform a particular task referred to by a name. Since there can be many functions in a C++ program, they are usually identified by unique names. The most essential function needed for every C++ program is the **main()** function.



The structure of a simple C++ program is given below:

```
#include <header file>
using namespace identifier;
int main()
{
    statements;
    :
    :
    :
    return 0;
}
```

The first line is called preprocessor directive and the second line is the namespace statement. The third line is the function header which is followed by a set of statements enclosed by a pair of braces. Let us discuss each of these parts of the program.

### 6.10.1 Preprocessor directives

A C++ program starts with pre-processor directives. Preprocessors are the compiler directive statements which give instruction to the compiler to process the information provided before actual compilation starts. Preprocessor directives are lines included in the code that are not program statements. These lines always start with a # (hash) symbol. The pre-processor directive `#include` is used to link the header files available in the C++ library by which the facilities required in the program can be obtained. No semicolon (;) is needed at the end of such lines. Separate `#include` statements should be used for different header files. There are some other pre-processor directives such as `#define`, `#undef`, etc.

### 6.10.2 Header files

Header files contain the information about functions, objects and predefined derived data types and they are available along with compiler. There are a number of such files to support C++ programs and they are kept in the standard library. Whichever program requires the support of any of these resources, the concerned header file is to be included. For example, if we want to use the predefined objects `cin` and `cout`, we have to use the following statement at the beginning of the program.

```
#include <iostream>
```

The header file `iostream` contains the information about the objects `cin` and `cout`. Even though header files have the extension `.h`, it should not be specified for GCC. But the extension is essential for some other compilers like Turbo C++ IDE.



### 6.10.3 Concept of namespace

A program cannot have the same name for more than one identifier (variables or functions) in the same scope. For example, in our home two or more persons (or even living beings) will not have the same name. If there are, it will surely make conflicts in the identity within the home. So, within the scope of our home, a name should be unique. But our neighbouring home may have a person (or any living being) with the same name as that of one of us. It will not make any confusion of identity within the respective scopes. But an outsider cannot access a particular person by simply using the name; but the house name is also to be mentioned.

The concept of namespace is similar to a house name. Different identifiers are associated to a particular namespace. It is actually a group name in which each item is unique in its name. User is allowed to create own namespaces for variables and functions. We can use an identifier to give name to a namespace. The keyword `using` technically tells the compiler about a namespace where it should search for the elements used in the program. In C++, `std` is an abbreviation of 'standard' and it is the standard namespace in which `cout`, `cin` and a lot of other objects are defined. So, when we want to use them in a program, we need to follow the format `std::cout` and `std::cin`. This kind of explicit referencing can be avoided with the statement `using namespace std;` in the program. In such a case, the compiler searches this namespace for the elements `cin`, `cout`, `endl`, etc. So whenever the computer comes across `cin`, `cout`, `endl` or anything of that matter in the program, it will read it as `std::cout`, `std::cin` or `std::endl`.

The statement `using namespace std;` doesn't really add a function, it is the `#include <iostream>` that "loads" `cin`, `cout`, `endl` and all the like.

### 6.10.4 The `main()` function

Every C++ program consists of a function named `main()`. The execution starts at `main()` and ends within `main()`. If we use any other function in the program, it is called (or invoked) from `main()`. Usually a data type precedes the `main()` and in GCC, it should be `int`.

The function header `main()` is followed by its body, which is a set of one or more statements within a pair of braces `{ }`. This structure is known as the definition of the `main()` function. Each statement is delimited by a semicolon `;`. The statements may be executable and non-executable. The executable statements represent instructions to be carried out by the computer. The non-executable statements are intended for compiler or programmer. They are informative statements. The last statement in the body of `main()` is `return 0;`. Even though we do not use this statement, it will not make any error. Its relevance will be discussed in Chapter 10.



C++ is a free form language in the sense that it is not necessary to write each statement in new lines. Also a single statement can take more than one line.

### 6.10.5 A sample program

Let us look at a complete program and familiarise ourselves with its features, in detail. This program on execution will display a text on the screen.

```
#include <iostream>
using namespace std;
int main()
{
    cout<<"Hello, Welcome to C++";
    return 0;
}
```

The program has seven lines as detailed below:

Line 1: The preprocessor directive `#include` is used to link the header file `iostream` with the program.

Line 2: The `using namespace` statement makes available the identifier `cout` in the program.

Line 3: The header of the essential function for a C++ program, i.e., `int main()`.

Line 4: An opening brace `{` that marks the beginning of the instruction set (program).

Line 5: An output statement, which will be executed when we run the program, to display the text "Hello, Welcome to C++" on the monitor. The header file `iostream` is included in this program to use `cout` in this statement.

Line 6: The `return` statement stops the execution of the `main()` function. This statement is optional as far as `main()` is concerned.

Line 7: A closing brace `}` that marks the end of the program.

### 6.11 Guidelines for coding

A source code looks good when the coding is legible, logic is communicative and errors if any are easily detectable. These features can be experienced if certain styles are followed while writing programs. Some guidelines are discussed in this section to write stylistic programs.

**Use suitable naming convention for identifiers**

Suppose we have to calculate the salary for an employee after deductions. We may code it as:  $A = B - C;$

where A is the net salary, B the total salary and C total deduction. The variable names A, B and C do not reflect the quantities they denote. If the same instruction is expressed as follows, it would be better:

```
Net_salary = Gross_salary - Deduction;
```

The variable names used in this case help us to remember the quantity they possess. They readily reflect their purpose. These kinds of identifiers are called ***mnemonic names***. The following points are to be remembered in the choice of names:

- Choose good mnemonic names for all variables, functions and procedures.  
e.g. avg\_hgt, Roll\_No, emp\_code, SumOfDigits, etc.
- Use standardized prefixes and suffixes for related variables.  
e.g. num1, num2, num3 for three numbers
- Assign names to constants in the beginning of the program.  
e.g. float PI = 3.14;

**Use clear and simple expressions**

Some people have a tendency to reduce the execution time by sacrificing simplicity. This should be avoided. Consider the following example. To find out the remainder after division of x by n, we can code as:  $y = x - (x/n) * n;$

The same thing is achieved by a simpler and more elegant piece of code as shown below:

```
y = x % n;
```

So it is better to use simpler codes in programming to make the program more simple and clear.

**Use comments wherever needed**

Comments play a very important role as they provide internal documentation of a program. They are lines in code that are added to describe the program. They are ignored by the compiler. There are two ways to write comments in C++:

**Single line comment:** The characters `//` (two slashes) is used to write single line comments. The text appearing after `//` in a line is treated as a comment by the C++ compiler.

**Multiline comments:** Anything written within `/*` and `*/` is treated as comment so that the comment can take any number of lines.



But care should be taken that no relevant code of the program is included accidentally inside the comment. The following points are to be noted while commenting:

- Always insert prologues, the comments in the beginning of a program that summarises the purpose of the program.
- Comment each variable and constant declaration.
- Use comments to explain complex program steps.
- It is better to include comments while writing the program itself.
- Write short and clear comments.

### Relevance of indentation

In computer programming, an indent style is a convention governing the indentation of blocks of code to convey the program's structure, for good visibility and better clarity. An indentation makes the statements clear and readable. It shows the levels of statements in the program.

The usage of these guidelines can be observed in the programs given in the next section.

### Program gallery

Let us now write programs to solve some problems following the coding guidelines. The call-outs given are not part of the program. Program 6.1 displays a message.

#### Program 6.1: To display a message

```

/* This program displays the message
   "Smoking is injurious to health"
   on the monitor */
#include <iostreamh> // To use the cout object
using namespace std; // To access cout
int main() //program begins here
{
    //The following output statement displays a message
    cout << "Smoking is injurious to health";
    return 0;
} //end of the program

```

Multiline comment

Single line comment

Indentation

On executing Program 6.1, the output will be as follows:

Smoking is injurious to health

More illustrations on the usage of indentation can be seen in the examples given in Chapter 7.





Program 6.2 accepts two integer numbers from the user, finds its sum and displays the result.

**Program 6.2: To find the sum of two integer numbers**

```
#include <iostream>
using namespace std;
int main()
{ //Program begins
/* Two variables are declared to read user inputs and the
variable sum is declared to store the result
*/
    int num1, num2, sum;
    cout<<"Enter two numbers: "; //Prompt for input
    cin>>num1>>num2;    //Cascading to get two numbers
    sum=num1+num2;    //Assignment statement to find the sum
    cout<<"Sum of the entered numbers = "<<sum;
/* The result is displayed with proper message.
Cascading of output operator is utilized */
    return 0;
}
```

A sample output of Program 6.2 is given below:

Enter two numbers: 5 7

User inputs  
separated by spaces

Sum of the entered numbers = 12

Let us consider another problem. A student is awarded with three scores obtained in three Continuous Evaluation (CE) activities. The maximum score of an activity is 20. Find the average score of the student.

**Program 6.3: To find the average of three CE scores**

```
#include <iostream>
using namespace std;
int main()
{
    int score_1, score_2, score_3;
    float avg;
    //Average of 3 numbers can be a floating point value
    cout << "Enter the three CE scores: ";
    cin >> score_1 >> score_2 >> score_3;
    avg = (score_1 + score_2 + score_3) / 3.0;
```



```

/* The result of addition will be an integer value. If 3
is written instead of 3.0, integer division will be
performed and will not get the correct result */
    cout << "Average CE score is: " << avg;
    return 0;
}

```

Program 6.3 gives the following output for the CE scores 17, 19 and 20.

```

Enter the three CE scores: 17    19    20
Average CE score is: 18.666666

```

The assignment statement to find the average value uses an expression to the right of assignment operator (=). This expression has two + operators and one / operator. The precedence of / over + is changed by using parentheses for addition. The operands for the addition operators are all int type data and hence the result will be an integer. When this integer result is divided by 3, the output will again be an integer. If it was so, the output of Program 6.3 would have been 18, which is not accurate. Hence floating point constant 3.0 is used as the second operand for / operator. It makes the integer numerator float by type promotion.

Suppose the radius of a circle 'r' is given and you are requested to compute its area and the perimeter. As you know, area of a circle is calculated using the formula  $\pi r^2$  and perimeter by  $2\pi r$ , where  $\pi = 3.14$ . Program 6.4 solves this problem.

#### Program 6.4: To find the area and perimeter of a circle for a given radius

```

#include <iostream>
using namespace std;
int main()
{
    const float PI = 22.0/7; //Use of const access modifier
    float radius, area, perimeter;
    cout<<"Enter the radius of the circle: ";
    cin>>radius;
    area = PI * radius * radius;
    perimeter = 2 * PI * radius;
    cout<<"Area of the circle = "<<area<<"\n";
    cout<<"Perimeter of the circle = "<<perimeter;
    return 0;
}

```

Escape sequence  
'\n' prints a new  
line after displaying  
the value of Area

A sample output of Program 6.4 is as follows:



```
Enter the radius of the circle: 2.5
```

```
Area of the circle = 19.642857
```

```
Perimeter of the circle = 15.714285
```

The last two output statements of Program 6.4 displays both the results in separate lines. The escape sequence character '\n' brings the cursor to the new line before the last output statement gets executed.

Let us develop another program to find simple interest. As you know, principal amount, rate of interest and period are to be given as input to get the result.

#### Program 6.5: To find the simple interest

```
#include <iostream>
using namespace std;
int main()
{
    float p_Amount, n_Year, i_Rate, int_Amount;
    cout<<"Enter the principal amount in Rupees: ";
    cin>>p_Amount;
    cout<<"Enter the number of years for the deposit: ";
    cin>>n_Year;
    cout<<"Enter the rate of interest in percentage: ";
    cin>>i_Rate;
    int_Amount = p_Amount * n_Year * i_Rate /100;
    cout<<"Simple interest for the principal amount "
        <<p_Amount<<" Rupees for a period of "<<n_Year
        <<" years at the rate of interest "<<i_Rate
        <<" is "<<int_Amount<<" Rupees";
    return 0;
}
```

A sample output of Program 6.5 is given below:

```
Enter the principal amount in Rupees: 100
```

```
Enter the number of years for the deposit: 2
```

```
Enter the rate of interest in percentage: 10
```

```
Simple interest for the principal amount 100 Rupees for a
period of 2 years at the rate of interest 10 is 20 Rupees
```



The last statement in Program 6.5 is the output statement and it spans over four lines. Note that there is no semi colon at the end of each line and so it is considered a single statement. On execution of the program the result may be displayed in multiple lines depending on the size and resolution of the monitor of your computer.

Program 6.6 solves a temperature conversion problem. The temperature in degree celsius will be given as input and the output will be its equivalent in fahrenheit.

#### Program 6.6: To convert temperature from Celsius to Fahrenheit

```
#include <iostream>
using namespace std;
int main()
{
    float celsius, fahrenheit;
    cout<<"Enter the Temperature in Celsius: ";
    cin>>celsius;
    fahrenheit=1.8*celsius+32;
    cout<< celsius<<" Degree Celsius = "
        << fahrenheit<<" Degree Fahrenheit";
    return 0;
}
```

Program 6.6 gives a sample output as follows:

```
Enter the Temperature in Celsius: 37
```

```
37 Degree Celsius = 98.599998 Degree Fahrenheit
```

We know that each character literal in C++ has a unique value called its ASCII code. These values are integers. Let us write a program to find the ASCII code of a given character.

#### Program 6.7: To find the ASCII value of a character

```
#include <iostream>
using namespace std;
int main()
{
    char ch;
    int asc;
    cout << "Enter the character: ";
    cin >> ch;
    asc = ch;
    cout << "ASCII value of "<<ch<<" = " << asc;
    return 0;
}
```



A sampler output of Program 6.7 is given below:

```
Enter the character: A
```

```
ASCII value of A = 65
```



### Let us sum up

Data types are means to identify the type of data and associated operations handling it. Each data type has a specific size and a range of data. Data types are used to declare variables. Type modifiers help handling a higher range of data and are used with data types to declare variables. Different types of operators are available in C++ for various operations. When operators are combined with operands (data), expressions are formed. There are mainly three types of expressions - arithmetic, relational and logical. Type conversion methods are used to get desired results from arithmetic expressions. Statements are the smallest executable unit of a program. Variable declaration statements define the variables in the program and they will be allocated memory space. The executable statements like assignment statements, input statements, output statements, etc. help giving instructions to the computer. Some special operators like arithmetic assignment, increment, decrement, etc. make the expressions and statements compact and the execution faster. C++ program has a typical structure and it must be followed while writing programs. Stylistic guidelines shall be followed to make the program attractive and communicative among humans.



### Learning outcomes

After the completion of this chapter the learner will be able to

- identify the various data types in C++.
- list and choose appropriate data type modifiers.
- choose appropriate variables.
- experiment with various operators.
- apply the various I/O operators.
- write various expressions and statements.
- identify the structure of a simple C++ program.
- identify the need for stylistic guidelines while writing a program.
- write simple programs using C++.



## Lab activity

1. Write a program that asks the user to enter the weight in grams, and then display the equivalent in Kilograms.
2. Write a program to generate the following table

2013	100%
2012	99.9%
2011	95.5%
2010	90.81%
2009	85%

Use a single cout statement for output. (Hint: Make use of `\n` and `\t`)

4. Write a short program that asks for your height in Meter and Centimeter and converts it to Feet and inches. (1 foot = 12 inches, 1 inch = 2.54 cm).
5. Write a program to compute simple interest and compound interest.
6. Write a program to : (i) print ASCII code for a given digit, (ii) print ASCII code for backspace. (Hint : Store escape sequence for backspace in an integer variable).
7. Write a program to accept a time in seconds and convert into hrs: mins: secs format. For example, if 3700 seconds is the input, the output should be 1hr: 1 min: 40 secs.

## Sample questions

### Very short answer type

1. What are data types? List all predefined data types in C++.
2. What is a constant?
3. What is dynamic initialisation of variables?
4. What is type casting?
5. Write the purpose of declaration statement?
6. Name the header file to be included to use cin and cout in programs?
7. What is the input operator ">>" and output operator "<<" called ?
8. What will be the result of  $a = 5/3$  if a is (i) float and (ii) int ?





9. What will be the value of  $P = P++ + ++i$  where  $i$  is 22 and  $P = 3$  initially?
10. Find the value given by the following expression if  $j = 5$  initially.
  - (i)  $(5*++j)\%6$
  - (ii)  $(5*j++)\%6$
11. What will be the order of evaluation for following expressions?
  - (i)  $i+5 > j-6$
  - (ii)  $s+10 < p-2+2*q$
12. What will be the result of the following if  $ans$  is 6 initially?
  - (i) `cout << ans = 8 ;`
  - (ii) `cout << ans == 8`

### Short answer type

1. What is a variable? List the two values associated with it.
2. In how many ways can a variable be declared in C++?
3. Explain the impact of type modifiers of C++ in variable declaration.
5. What is the role of the keyword 'const'?
6. Explain how prefix form of increment operation differs from postfix form.
8. Write down the operation performed by sizeof operator.
9. Explain the two methods of type conversions.
10. What would happen if `main()` is not present in a program?
11. Identify the errors in the following code segments:
  - (a)
 

```
int main()
{ cout << "Enter two numbers"
  cin >> num >> auto
  float area = Length * breadth ; }
```
  - (b)
 

```
#include <iostream>
using namespace std
void Main()
{ int a, b
  cin <<a <<b
  max=(a > b) a:b
  cout>max
}
```
12. Find out the errors, if any, in the following ++ statements:
  - (i) `cout<< "a=" a;`
  - (v) `cin >> "\n" >> y ;`
  - (ii) `m=5,n=12;015`
  - (vi) `cout >> \n "abc"`



(iii) `cout << "x" ; <<x;`      (vii) `a = b + c`  
(iv) `cin >> y`      (viii) `break = x`

13. What is the role of relational operators? Distinguish between `==` and `=`.
14. Comments are useful to enhance readability and understandability of a program. Justify this statement with examples.

### Long answer type

1. Explain the operators of C++ in detail.
2. Explain the different types of expressions in C++ and the methods of type conversions in detail.
3. Write the working of arithmetic assignment operator? Explain all arithmetic assignment operators with the help of examples.



**Class - XI**

**Part - II**



**Government of Kerala  
DEPARTMENT OF EDUCATION**

**State Council of Educational Research and Training (SCERT); Kerala  
2016**

## THE NATIONAL ANTHEM

Jana-gana-mana adhinayaka, jaya he  
Bharatha-bhagya-vidhata.  
Punjab-Sindh-Gujarat-Maratha  
Dravida-Utkala-Banga  
Vindhya-Himachala-Yamuna-Ganga  
Uchchala-Jaladhi-taranga  
Tava subha name jage,  
Tava subha asisa mage,  
Gahe tava jaya gatha.  
Jana-gana-mangala-dayaka jaya he  
Bharatha-bhagya-vidhata.  
Jaya he, jaya he, jaya he,  
Jaya jaya jaya, jaya he!

## PLEDGE

India is my country. All Indians are my brothers and sisters.

I love my country, and I am proud of its rich and varied heritage. I shall always strive to be worthy of it.

I shall give respect to my parents, teachers and all elders and treat everyone with courtesy.

I pledge my devotion to my country and my people. In their well-being and prosperity alone lies my happiness.

*Prepared by :*

**State Council of Educational Research and Training (SCERT)**  
Poojappura, Thiruvananthapuram 695012, Kerala

Website : [www.scertkerala.gov.in](http://www.scertkerala.gov.in) e-mail : [scertkerala@gmail.com](mailto:scertkerala@gmail.com)  
Phone : 0471 - 2341883, Fax : 0471 - 2341869  
Typesetting and Layout : SCERT  
© Department of Education, Government of Kerala

Dear students,

Computer Science, a subject belonging to the discipline of Science and of utmost contemporary relevance, needs continuous updating. The Higher Secondary Computer Science syllabus has been revised with a view to bringing out its real spirit and dimension. The constant and remarkable developments in the field of computing as well as the endless opportunities of research in the field of Computer Science and Technology have been included.

This textbook is prepared strictly in accordance with the revised syllabus for the academic year 2014 - 15. It begins with the historical developments in computing and familiarises the learner with the latest technological advancements in the field of computer hardware, software and network. The advancement in computer network, Internet technology, wireless and mobile communication are also dealt with extensively in the content. In addition to familiarising various services over the Internet, the need to be concerned about the factors that harness morality and the awareness to refrain from cyber security threats are also highlighted.

The major part of the textbook as well as the syllabus establishes a strong foundation to construct and enhance the problem solving and programming skills of the learner. The multi-paradigm programming language C++ is presented to develop programs which enable computers to manage different real life applications effectively. The concepts and constructs of the principles of programming are introduced in such a way that the learner can grasp the logic and implementation methods easily.

I hope this book will meet all the requirements for stepping to levels of higher education in Computer Science and pave your way to the peak of success.

Wish you all success.

**Dr P. A. Fathima**  
Director, SCERT; Kerala

## Textbook Development Team

### COMPUTER SCIENCE

**Joy John**

HSST, St. Joseph's HSS  
Thiruvananthapuram

**Asees V.**

HSST, GHSS Velliyode, Kozhikode

**Roy John**

HSST, St. Aloysius HSS  
Elthuruth, Thrissur

**Aboobacker P.**

HSST, Govt. GHSS Chalappuram,  
Kozhikode

**Shajan Jos N.**

HSST, St. Joseph's HSS, Pavaratty,  
Thrissur

**Afsal K. A.**

HSST, GHSS Sivapuram,  
Kariyathankare P.O., Kozhikode

**Prasanth P. M.**

HSST, St. Joseph's Boys' HSS,  
Kozhikode

**Vinod V.**

HSST, NSS HSS, Prakkulam, Kollam

**Rajamohan C.**

HSST, Nava Mukunda HSS,  
Thirunavaya, Malappuram

**A. S. Ismael**

HSST, Govt. HSS Palapetty,  
Malappuram

**Sunil Kariyatan**

HSST, Govt. Brennen HSS,  
Thalassery

**Sai Prakash S.**

HSST, St. Thomas HSS,  
Poonthura, Thiruvananthapuram

### Experts

**Dr Lajish V. L.**

Assistant Professor, Dept. of Computer Science, University of Calicut

**Dr Madhu S. Nair**

Assistant Professor, Dept. of Computer Science, University of Kerala

**Madhu V. T.**

Director, Computer Centre, University of Calicut

**Dr Binu P. Chacko**

Associate Professor, Dept. of Computer Science,  
Prajyoti Niketan College, Pudukad

**Dr Sushil Kumar R.**

Associate Professor, Dept. of English, D.B. College, Sasthamcotta

**Dr Vineeth K. Paleri**

Professor, Dept. of Computer Science and Engineering, NIT, Kozhikode

**Maheswaran Nair V.**

Sub Divisional Engineer, Regional Telecom Training Centre, Thiruvananthapuram

### Artist

**Sudheer Y.****Vineeth V.**

### Academic Co-ordinator

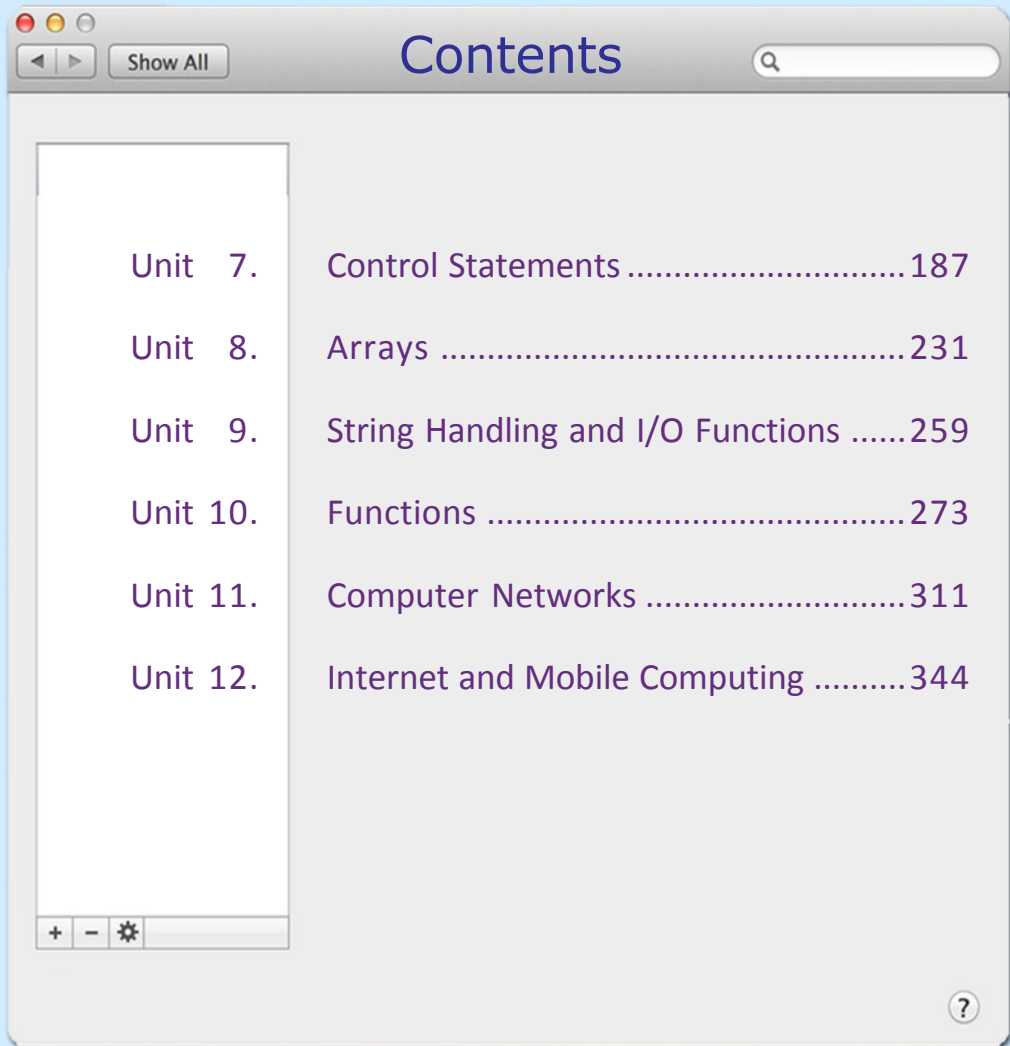
**Dr Meena S.**

Research Officer, SCERT

**Jancy Rani A. K.**

Research Officer, SCERT





Unit 7.	Control Statements .....	187
Unit 8.	Arrays .....	231
Unit 9.	String Handling and I/O Functions .....	259
Unit 10.	Functions .....	273
Unit 11.	Computer Networks .....	311
Unit 12.	Internet and Mobile Computing .....	344

### Icons used in this textbook



Let us do



Check yourself



Information box



Lab activities



Learning outcomes



## Key Concepts

- **Decision making statements**
  - if statement
  - if ... else statement
  - Nested if
  - else if ladder
  - switch statement
  - Conditional operator
- **Iteration statements**
  - while statement
  - for statement
  - do ... while statement
  - Nesting of loops
- **Jump statements**
  - goto
  - break
  - continue

## Control Statements

In the previous chapters we discussed some executable statements of C++ to perform operations such as input, output and assignment. We know how to write simple programs. The execution of these programs is sequential in nature, that is, the statements constituting the program are executed one by one. In this chapter, we discuss C++ statements used for altering the default flow of execution. As we discussed in Chapter 4, selection, skipping or repeated execution of some statements may be required for solving problems. Usually this decision will be based on some condition(s). C++ provides statements to facilitate this requirement with the help of control statements. These statements are used for altering the normal flow of program execution. Control statements are classified into two: (i) decision making/selection statements and (ii) iteration statements. Let us discuss these statements, their syntax and mode of execution.

### 7.1 Decision making statements

At times, it so happens that the computer may not execute all statements while solving problems. Some statements may be executed in a situation, while they may not be executed in another situation. The computer has to take the required decision in this respect. For this, we have to provide appropriate conditions which will be evaluated by the computer. It will then take a





decision on the basis of the result. The decision will be in the form of selecting a particular statement for execution or skipping some statement from being executed. The statements provided by C++ for the selected execution are called **decision making statements** or **selection statements**. `if` and `switch` are the two types of selection statements in C++.

### 7.1.1 `if` statement

The `if` statement is used to select a set of statements for execution based on a condition. In C++, conditions (otherwise known as test expressions) are provided by relational or logical expressions. The syntax (general form) of `if` statement is as follows:

```
if (test expression)
{
    statement block;
}
```

Body of `if` statement that consists of the statement(s) to be executed when the condition is true

Here the `test expression` represents a condition which is either a relational expression or logical expression. If the test expression evaluates to True (non-zero value), a statement or a block of statements associated with `if` is executed. Otherwise, the control moves to the statement following the `if` construct. Figure 7.1 shows the mode of execution of `if` statement. While using `if`, certain points are to be remembered.

- The test expression is always enclosed in parentheses.
- The expression may be a simple expression constituted by relational expression or a compound expression constituted by logical expression.
- The statement block may contain a single statement or multiple statements. If there is a single statement, then it is not mandatory to enclose it in curly braces { }. If there are multiple statements, they must be enclosed in curly braces.

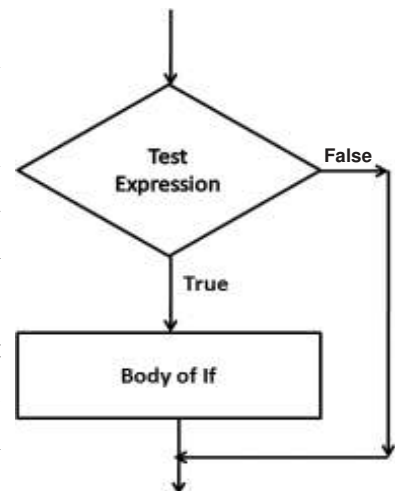


Fig. 7.1 : Working of `if` statement

Program 7.1 accepts the score of a student and displays the text "You have Passed" only if he/she has passed. (Assume that 18 is the minimum score for pass).

**Program 7.1: To display 'You have passed' if score is 18 or more**

```
#include<iostream>
using namespace std;
int main()
{
    int score ;
    cout << "Enter your score: ";
    cin >> score;
    if (score >= 18)
        cout << "You have passed";
    return 0;
}
```

Body of if

The following is a sample output of program 7.1:

```
Enter your score: 25
You have passed
```

In Program 7.1, the score of a student is entered and stored in the variable `score`. The test expression compares the value of `score` with 18. The body of `if` will be executed only if the test expression evaluates to True. That means, when the score is greater than or equal to 18, the output You have Passed will be displayed on the screen. Otherwise, there will be no output.

Note that the statement block associated with `if` is written after a tab space. We call it **indentation**. This is a style of coding which enhances the readability of the source code. Indentation helps the debugging process greatly. But it has no impact on the execution of the program.

Consider the following C++ program segment. It checks whether a given character is an alphabet or a digit.

```
char ch;
cin >> ch;
if (ch >= 'a' && ch <= 'z')
    cout << "You entered an alphabet";
if (ch >= '0' && ch <= '9')
{
    cout << "You entered a digit\n";
    cout << "It is a decimal number ";
}
```

Logical expression is evaluated

Only a single statement; No need of braces { }

More than one statement; Must be enclosed in braces { }



### 7.1.2 if...else statement

Consider the if statement in Program 7.1:

```
if (score >= 18)
    cout << "You have passed";
```

Here, the output is obtained only if the score is greater than or equal to 18. What will happen if the score entered is less than 18? It is clear that there will be no output. Actually we don't have the option of selecting another set of statements if the test expression evaluates to False. If we want to execute some actions when the condition becomes False, we introduce another form of if statement, **if...else**. The syntax is:

```
if (test expression)
{
    statement block 1;
}
else
{
    statement block 2;
}
```

If the test expression evaluates to True, only the statement block 1 is executed. If the test expression evaluates to False statement block 2 is executed. The flowchart shown in Figure 7.2 explains the execution of if...else statement.

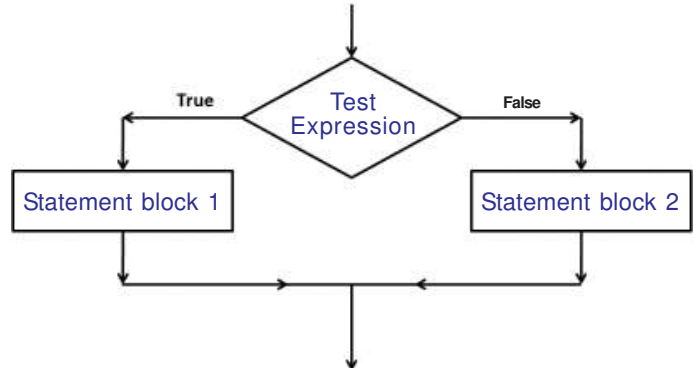


Fig 7.2 : Flowchart of if - else statement

The following code segment illustrates the working of if...else statement.

```
if (score >= 18)
    cout << "Passed";
else
    cout << "Failed";
```

This statement is executed only when score is 18 or more (i.e. when Test expression returns True)

This statement is executed only when score is less than 18 (i.e. when Test expression returns False)

Let us write a program to input the heights of two students and find the taller.



**Program 7.2: To find the taller student by comparing their heights**

```
#include <iostream>
using namespace std;
int main()
{
    int ht1, ht2;
    cout << "Enter heights of the two students: ";
    cin >> ht1 >> ht2;
    if (ht1 > ht2)    //decision making based on condition
        cout<<"Student with height "<<ht1<<" is taller";
    else
        cout<<"Student with height "<<ht2<<" is taller";
    return 0;
}
```

When Program 7.2 is executed, one of the output statements will be displayed. The selection depends upon the relational expression  $ht1 > ht2$ . The following are sample outputs:

**Output 1:** Enter heights of the two students: 170      165  
Student with height 170 is taller

**Output 2:** Enter heights of the two students: 160      171  
Student with height 171 is taller

In the first output, we input 170 for  $ht1$  and 165 for  $ht2$ . So, the test expression,  $(ht1 > ht2)$  is evaluated to True and hence the statement block of `if` is selected and executed. In the second output, we input 160 for  $ht1$  and 171 for  $ht2$ . The test expression,  $(ht1 > ht2)$  is evaluated and found False. Hence the statement block of `else` is selected and executed.

In `if...else` statement, either the code associated with `if` (statement block 1) or the code associated with `else` (statement block 2) is executed.

Let us see another program that uses an arithmetic expression as one of the operands in the test expression. Program 7.3 uses this concept to check whether an input number is even or odd.

**Program 7.3: To check whether a given number is even or odd**

```
#include <iostream>
using namespace std;
int main()
{
    int num;
```



```

cout << "Enter the number: ";
cin >> num;
if (num%2 == 0)
    cout << "The given number is Even";
else
    cout << "The given number is Odd";
return 0;
}

```

Some sample outputs of Program 7.3 are shown below:

Output 1:

```

Enter the number: 7
The given number is Odd

```

Output 2:

```

Enter the number: 10
The given number is Even

```

In this program, the expression  $(num \% 2)$  finds the remainder when  $num$  is divided by 2 and compares it with the value 0. If they are equal, the `if` block is executed, otherwise the `else` block is executed.



**Let us do**

1. Write a program to check whether a given number is a non-zero integer number and is positive or negative.
2. Write a program to enter a single character for sex and display the gender. If the input is 'M' display "Male" and if the input is 'F', display "Female".
3. Write a program to input your age and check whether you are eligible to cast vote (the eligibility is 18 years and above).

### 7.1.3 Nested if

In some situations there may arise the need to take a decision within `if` block. When we write an `if` statement inside another `if` block, it is called **nesting**. Nested means one inside another. Consider the following program segment:

```

if (score >= 60)
{
    if (age >= 18)
        cout<<"You are selected for the course!";
}

```

Diagram labels: "outer if" points to the first `if` block, and "inner if" points to the nested `if` block.

In this code fragment, if the value of `score` is greater than or equal to 60, the flow of control enters the statement block of outer **if**. Then the test expression of the inner **if** is evaluated (i.e. whether the value of `age` is greater than or equal to 18). If it is



evaluated to True, the code displays the message "You are selected for the course!". Then the program continues to execute the statement following the outer if statement. An if statement, inside another if statement is termed as a **nested if** statement. The following is an expanded form of nested if.

```
if (test expression 1)
{
    if (test expression 2)
        statement 1;
    else
        statement 2;
}
else
{
    body of else ;
}
```

It will be executed if both the test expressions are True.

It will be executed if test expression 1 is True, but test expression 2 is False.

It will be executed if test expression 1 is False. The test expression 2 is not evaluated.

The important point to remember about nested if is that an else statement always refers to the nearest if statement within the same block. Let us discuss this case with an example. Consider the following program segment:

```
cout<<"Enter your score in Computer Science exam: ";
cin>>score;
if (score >= 18)
    cout<<"You have passed";
    if(score >= 54)
        cout<<" with A+ grade !";
else
    cout<<"\nYou have failed";
```

If we input the value 45 for score, the output will be as follows:

You have passed

You have failed

We know that this is logically not correct. Though the indentation of the code is proper, that doesn't matter in execution. The second if statement will not be considered as nested if, rather it is counted as an independent if with an else block. So, when the first if statement is executed, the if block is selected for execution since the test expression is evaluated to True. It causes the first line in the output. After that, while considering the second if statement, the test expression is evaluated to False and hence the second line in the output is obtained. So to get the correct output, the code should be modified as follows:



```

cout<<"Enter your score in Computer Science exam: ";
cin>>score;
if (score >= 18)
{
    cout<<"You have passed";
    if(score >= 54)
        cout<<" with A+ grade !";
}
else
    cout<<"\nYou have failed";

```

Nesting is enforced by putting a pair of braces

The else is now associated with the outer if

If we input the same value 45 as in the case of previous example, the output will be as follows:

You have passed

Program 7.4 uses nested if to find the largest among three given numbers. In this program, if statement is used in both the if block and else block.

#### Program 7.4: To find the largest among three numbers

```

#include <iostream>
using namespace std;
int main()
{
    int x, y, z;
    cout << "Enter three different numbers: ";
    cin >> x >> y >> z ;
    if (x > y)
    {
        if (x > z)
            cout << "The largest number is: " << x;
        else
            cout << "The largest number is: " << z;
    }
    else
    {
        if (y > z)
            cout << "The largest number is: " << y;
        else
            cout << "The largest number is: " << z;
    }
    return 0;
}

```



A sample output of Program 7.4 is given below:

```
Enter three different numbers: 6    2    7
The largest number is: 7
```

As per the input given above, the test expression  $(x > y)$  in the outer `if` is evaluated to `True` and hence the control enters the inner `if`. Here the test expression  $(x > z)$  is evaluated to `False` and so its `else` block is executed. Thus the value of `z` is displayed as the output.

### Check yourself



1. Write a program to input an integer and check whether it is positive, negative or zero.
2. Write a program to input three numbers and print the smallest one.

#### 7.1.4 The `else if` ladder

There are situations where an `if` statement is used within an `else` block. It is used in programs when multiple branching is required. Different conditions will be given and each condition will decide which statement is to be selected for execution. A common programming construct based on `if` statement is the **`else if`** ladder, also referred to as the **`else if`** staircase because of its appearance. It is also known as `if...else if` statement. The general form of `else if` ladder is:

```
if (test expression 1)
    statement block 1;
else if (test expression 2)
    statement block 2;
else if (test expression 3)
    statement block 3;
.....
else
    statement block n;
```

At first, the test expression 1 is evaluated and if it is `True`, the statement block 1 is executed and the control comes out of the ladder. That means, the rest of the ladder is bypassed. If test expression 1 evaluates to `False`, then the test expression 2 is evaluated and so on. If any one of the test expressions evaluates to `True`, the corresponding statement block is executed and control comes out of the ladder. If all the test expressions are evaluated to `False`, the statement block `n`



after the final `else` is executed. Observe the indentation provided in the syntax and follow this style to use `else if` ladder.

Let us illustrate the working of the `else if` ladder by using a program to find the grade of a student in a subject when the score out of 100 is given. The grade is found out by the criteria given in the following table:

Scores	Grade
80 or more	A
From 60 to 79	B
From 40 to 59	C
From 30 to 39	D
Below 30	E

#### Program 7.5: To find the grade of a student for a given score

```
#include <iostream>
using namespace std;
int main()
{
    int score;
    cout << "Enter your score: ";
    cin >> score;
    if (score >= 80)
        cout << "A Grade";
    else if (score >= 60)
        cout << "B Grade ";
    else if (score >= 40)
        cout << "C grade";
    else if (score >= 30)
        cout << "D grade";
    else
        cout << "E Grade";
    return 0;
}
```

The following are the sample outputs of Program 7.5:

Output 1:

```
Enter your score: 73
B Grade
```

Output 2:

```
Enter your score: 25
E Grade
```





In Program 7.5, initially the test expression `score >= 80` is evaluated. Since the input is 73 in Output 1, the test expression is evaluated to False, and the next test expression `score >= 60` is evaluated. Here it is True, and hence "B Grade" is displayed and the remaining part of the `else if` ladder is bypassed. But in the case of Output 2, all the test expressions are evaluated to False and so the final `else` block is executed which makes "E Grade" as the output.

Let us write a program to check whether the given year is a leap year or not. The input value should be checked to know whether it is century year (year divisible by 100). If it is a century year, it becomes a leap year only if it is divisible by 400 also. If the input value is not a century year, then we have to check whether it is divisible by 4. If it is divisible the given year is a leap year, otherwise it is not a leap year.

#### Program 7.6: To check whether the given year is leap year or not

```
#include <iostream>
using namespace std;
void main()
{
    int year ;
    cout << "Enter the year (in 4-digits): ";
    cin >> year;
    if (year%100 == 0)    // Checks for century year
    {
        if (year%400 == 0)
            cout << "Leap year\n";
        else
            cout<< "Not a leap year\n";
    }
    else if (year%4 == 0)
        cout << "Leap year\n";
    else
        cout<< "Not a leap year\n";
    return 0;
}
```

Non - century year  
is leap year only if it  
is divisible by 4

Let us see some sample outputs of Program 7.6:

Output 1:

```
Enter the year (in 4-digits): 2000
Leap year
```

Output 2:

```
Enter the year (in 4-digits): 2014
Not a leap year
```

**Output 3:**

Enter the year (in 4-digits): 2100  
Not a leap year

**Output 4:**

Enter the year (in 4-digits): 2004  
Leap year

Let us write one more program to illustrate the use of `else if` ladder. Program 7.7 allows to input a number between 1 and 7 to denote the day of a week and display the name of the corresponding day. The input 1 will give you “Sunday” as output, 2 will give “Monday” and so on. If the input is outside the range 1 to 7, the output will be “Wrong input”.

**Program 7.7: To display the name of the day for a given day number**

```
#include <iostream>
using namespace std;
int main()
{
    int day;
    cout << "Enter the day number (1-7): ";
    cin >> day;
    if (day == 1)
        cout << "Sunday";
    else if (day == 2)
        cout << "Monday";
    else if (day == 3)
        cout << "Tuesday";
    else if (day == 4)
        cout << "Wednesday";
    else if (day == 5)
        cout << "Thursday";
    else if (day == 6)
        cout << "Friday";
    else if (day == 7)
        cout << "Saturday";
    else
        cout << "Wrong input";

    return 0;
}
```



The following are some sample outputs of Program 7.7:

Output 1:

```
Enter the day number (1-7): 5
Thursday
```

Output 2:

```
Enter day number (1-7): 9
Wrong input
```

### Check yourself



1. Write a program to input an integer number and check whether it is positive, negative or zero using `if ... else if` statement.
2. Write a program to input a character (a, b, c or d) and print as follows:  
a - abacus, b - boolean, c - computer, d - debugging.
3. Write a program to input a character and print whether it is an alphabet, digit or any other character.

### 7.1.5 switch statement

We have seen the concept of multiple branching with the help of `else if` ladder. Some of these programs can be written using another construct of C++ known as **switch** statement. This selection statement successively tests the value of a variable or an expression against a list of integers or character constants. The syntax of **switch** statement is as follows:

```
switch(expression)
{
    case constant_1    : statement block 1;
                       break;
    case constant_2    : statement block 2;
                       break;
    case constant_3    : statement block 3;
                       break;
                       :
                       :
    case constant_n-1  : statement block n-1;
                       break;
    default            : statement block n;
}
```



In the syntax `switch`, `case`, `break` and `default` are keywords. The expression is evaluated to get an integer or character constant and it is matched against the constants specified in the `case` statements. When a match is found, the statement block associated with that `case` is executed until the `break` statement or the end of `switch` statement is reached. If no match is found, the statements in the `default` block get executed. The `default` statement is optional and if it is missing, no action takes place when all matches fail.

The `break` statement, used inside `switch`, is one of the jump statements in C++. When a `break` statement is encountered, the program control goes to the statements following the `switch` statement. We will discuss `break` statement in detail in Section 7.3.2. Program 7.7 can be written using `switch` statement. It enhances the readability and effectiveness of the code. Observe the modification in Program 7.8.

#### **Program 7.8: To display the day of a week using `switch` statement**

```
#include <iostream>
using namespace std;
int main()
{
    int day ;
    cout << "Enter a number between 1 and 7: ";
    cin >> day ;
    switch (day)
    {
        case 1: cout << "Sunday";
                break;
        case 2: cout << "Monday";
                break;
        case 3: cout << "Tuesday";
                break;
        case 4: cout << "Wednesday";
                break;
        case 5: cout << "Thursday";
                break;
        case 6: cout << "Friday";
                break;
        case 7: cout << "Saturday";
                break;
        default: cout << "Wrong input";
    }
    return 0;
}
```



The output of Program 7.8 will be the same as in Program 7.7. The following are some samples:

Output 1:

```
Enter a number between 1 and 7: 5
Thursday
```

Output 2:

```
Enter a number between 1 and 7: 8
Wrong input
```

In Program 7.8, value of the variable `day` is compared against the constants specified in the case statements. When a match is found, the output statement associated with that case is executed. If we input the value 5 for the variable `day`, then the match occurs for the fifth case statement and the statement `cout << "Thursday";` is executed. If the input is 8 then no match occurs and hence the `default` block is executed.

Can you predict the output of Program 7.8, if all the `break` statements are omitted? The value returned by `day` is compared with the case constants. When the first match is found the associated statements will be executed and the following statements will also be executed irrespective of the remaining constants. There are situations where we omit the `break` statements purposefully. If the statements associated with all the case in a `switch` are the same, we only need to write the statement against the last case. Program 7.9 illustrates this concept.

#### Program 7.9: To check whether the given character is a vowel or not

```
#include <iostream>
using namespace std;
int main()
{
    char ch;
    cout<<"Enter the character to check: ";
    cin>>ch;
    switch(ch)
    {
        case 'A' :
        case 'a' :
        case 'E' :
        case 'e' :
        case 'I' :
        case 'i' :
        case 'O' :
        case 'o' :
```



```

        case 'U' :
        case 'u' : cout<<"The given character is a vowel";
                    break;
        default  : cout<<"The given character is not a vowel";
    }
    return 0;
}

```

Some of the outputs given by Program 7.9 are shown below:

Output 1:

```

Enter the character to check: E
The given character is a vowel

```

Output 2:

```

Enter the character to check: k
The given character is not a vowel

```

### Suitability and requirements for using **switch**

Though switch statement and else if ladder cause multiple branching, they do not work in the same fashion. In C++ all switch statements can be replaced by else if ladders, but all else if ladders cannot be substituted by switch. The following are the requirements to implement a multi branching using switch statement:

- Conditions involve only equality checking. In other cases, it should be converted into equality expression.
- The first operand in all the equality expressions should be the same variable or expression.
- The second operand in these expressions should be integer or character constants.

Among the programs we have discussed so far in this chapter, only the branching in Programs 7.3 and 7.7 can be replaced by switch. In Program 7.5, we can use switch if we modify the test expressions as `score/10==10`, `score/10==9`, `score/10==8`, and so on. So the following program fragment may be used instead of else if ladder.

```

switch(score/10)
{
    case 10:
    case 9: case 8: cout<< "A Grade"; break;
    case 7: case 6: cout<< "B Grade"; break;
    case 5: case 4: cout<< "C Grade"; break;
    case 3:          cout<< "D Grade"; break;
    default: cout<< "E Grade";
}

```

Score being int type, the expression returns only integer values





Let us have a comparison between `switch` and `else if` ladder as indicated in Table 7.1.

<b>switch statement</b>	<b>else if ladder</b>
<ul style="list-style-type: none"> <li>• Permits multiple branching</li> </ul>	<ul style="list-style-type: none"> <li>• Permits multiple branching</li> </ul>
<ul style="list-style-type: none"> <li>• Evaluates conditions with equality operator only</li> </ul>	<ul style="list-style-type: none"> <li>• Evaluates any relational or logical expression</li> </ul>
<ul style="list-style-type: none"> <li>• Case constant must be an integer or a character type value</li> </ul>	<ul style="list-style-type: none"> <li>• Condition may include range of values and floating point constants</li> </ul>
<ul style="list-style-type: none"> <li>• When no match is found, default statement is executed</li> </ul>	<ul style="list-style-type: none"> <li>• When no expression evaluates to True, else block is executed</li> </ul>
<ul style="list-style-type: none"> <li>• <code>break</code> statement is required to exit from <code>switch</code> statement</li> </ul>	<ul style="list-style-type: none"> <li>• Program control automatically goes out after the completion of a block</li> </ul>
<ul style="list-style-type: none"> <li>• More efficient when the same variable or expression is compared against a set of values for equality</li> </ul>	<ul style="list-style-type: none"> <li>• More flexible and versatile compared to <code>switch</code></li> </ul>

*Table 7.1: Comparison between `switch` and `else if` ladder*

### 7.1.6 The conditional operator (?:)

As we mentioned in Chapter 6, C++ has a ternary operator. It is the **conditional operator** (?:) consisting of the symbols `?` and `:` (a question mark and a colon). It requires three operands to operate upon. It can be used as an alternative to `if...else` statement. Its general form is:

`Test expression ? True_case code : False_case code;`

`Test expression` can be any relational or logical expression and `True_case code` and `False_case code` can be constants, variables, expressions or statement. The operation performed by this operator is shown below with the help of an `if` statement.

```

if (Test expression)
{
    True_case code;
}
else
{
    False_case code;
}

```

The conditional operator works in the same way as `if...else` works. It evaluates the test expression and if it is true, the `True_case code` is executed. Otherwise, `False_case code` is executed. Program 7.10 illustrates the working of conditional operator.

**Program 7.10: To find the larger number using the conditional operator**

```
#include <iostream>
using namespace std;
int main()
{
    int num1, num2;
    cout << "Enter two numbers: ";
    cin >> num1 >> num2 ;
    (num1>num2)? cout<<num1<<" is larger" : cout<<num2<<" is larger";
    return 0;
}
```

The last statement of this program is called conditional statement as it uses conditional operator. This statement may be replaced by the following code segment:

```
int big = (num1>num2)? num1 : num2;
cout<< big << "is larger";
```

If the test expression evaluates to True, the value of num1 will be assigned to big, otherwise that of num2. Here conditional operator is used to construct a conditional expression. The value returned by this expression will be assigned to big. The following is a complex form of conditional expression. It gives the largest among three numbers. If n1, n2, n3 and big are integer variables,

```
big = (n1>n2) ? ( (n1>n3)?n1:n3 ) : ( (n2>n3)?n2:n3 );
```

Refer to program 7.4 and see how the above conditional expression replaces the nesting of if.

**Check yourself**

1. Write a program to input a number in the range 1 to 12 and display the corresponding month of the year (January for the value 1, February for 2, and so on).
2. Write a program to perform arithmetic operations using switch statement. Accept two operands and a binary arithmetic operator as input.
3. What is the significance of break statement within a switch statement?
4. Write a program to enter a digit (0 to 9) and display it in words using switch statement.
5. Write a program to input a number and check whether it is a multiple of 5 using selection statements and conditional operator.
6. Rewrite the following statement using if...else statement

```
result= mark>30 ? 'p' : ' f';
```



## 7.2 Iteration statements

In Chapter 4, we discussed some problems for which the solution contains some tasks that were executed repeatedly. While writing programs, we use some specific constructs of the language to perform the repeated execution of a set of one or more statements. Such constructs are called **iteration statements** or **looping statements**. In C++, we have three iteration statements and all of them allow a set of instructions to be executed repeatedly when a condition is True.

We use the concept of loop in everyday life. Let us consider a situation. Suppose your class teacher has announced a gift for every student securing A+ grade in an examination. You are assigned the duty of wrapping the gifts. The teacher has explained the procedure for wrapping the gifts as follows:

Step 1 : Take the gift

Step 2 : Cut the wrapping paper

Step 3 : Wrap the gift

Step 4 : Tie the cover with a ribbon

Step 5 : Fill up a name card and paste it on the gift pack

If there are 30 students with A+ grade in the examination, you have to repeat the same procedure 30 times. To repeat the wrapping process 30 times, the instructions can be restructured in the following way.

*Repeat the following steps 30 times*

```
{
    Take the next gift
    Cut the wrapping paper
    Wrap the gift
    Tie the cover with a ribbon
    Fill up a name card and paste on the gift pack
}
```

Let us take another example. Suppose we want to find the class average of scores obtained in Computer Science. The following steps are to be performed:

*Initially Total\_Score has no value*

*Repeat the following steps starting from the first student till the last*

```
{
    Add Score of the student to the Total_Score
    Take the Score of the next student
}
```

*Average = Total\_Score /No. of students in the class*



In both the examples, we perform certain steps for a number of times. We use a counter to know how many times the process is executed. The value of this counter decides whether to continue the execution or not. Since loops work on the basis of such conditions, a variable like the counter will be used to construct a loop. This variable is generally known as **loop control variable** because it actually controls the execution of the loop. In Chapter 4, we discussed four elements of a loop. Let us refresh them:

1. **Initialisation:** Before entering a loop, its control variable must be initialized. During initialisation, the loop control variable gets its first value. The initialisation statement is executed only once, at the beginning of the loop.
2. **Test expression:** It is a relational or logical expression whose value is either True or False. It decides whether the loop-body will be executed or not. If the test expression evaluates to True, the loop-body gets executed, otherwise it will not be executed.
3. **Update statement:** The update statement modifies the loop control variable by changing its value. The update statement is executed before the next iteration.
4. **Body of the loop:** The statements that need to be executed repeatedly constitute the body of the loop. It may be a simple statement or a compound statement.

We learnt in Chapter 4 that loops are generally classified into entry-controlled loops and exit-controlled loops. C++ provides three loop statements: **while** loop, **for** loop and **do-while** loop. Let us discuss the working of each one in detail.

### 7.2.1 while statement

**while** loop is an entry-controlled loop. The condition is checked first and if it is found True the body of the loop will be executed. That is the body will be executed as long as the condition is True. The syntax of **while** loop is:

```
initialisation of loop control variable;
while(test expression)
{
    body of the loop;
    updation of loop control variable;
}
```

Here, test expression defines the condition which controls the loop. The body of the loop may be a single statement or a compound statement or without any statement. The body is the set of statements for repeated execution. Update expression refers to a statement that changes the value of the loop control variable. In a while loop, a loop control variable should be initialised before the loop begins

and it should be updated inside the body of the loop. The flowchart in Figure 7.3 illustrates the working of a while loop.

The initialisation of the loop control variable takes place first. Then the test expression is evaluated. If it returns True the body of the loop is executed. That is why while loop is called an **entry controlled loop**. Along with the loop body, the loop control variable is updated. After completing the execution of the loop body, test expression is again evaluated. The process is continued as long as the condition is True. Now, let us consider a code segment to illustrate the execution of while loop.

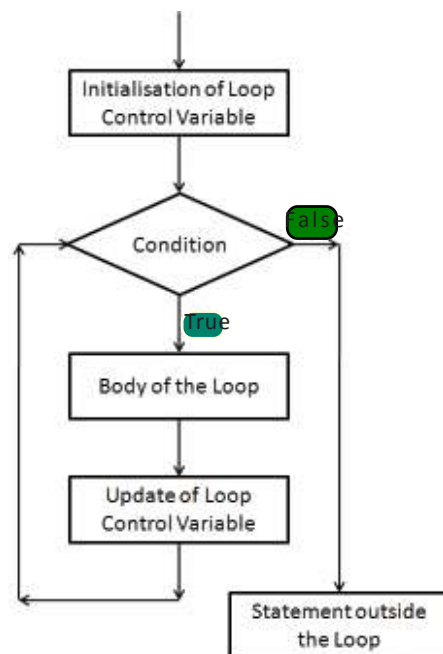


Fig. 7.3: Working of while loop

```

int k=1;
while(k<=3)
{
    cout << k << '\t';
    ++k;
}
  
```

Initialisation before loop

Test expression

Body of loop

Updation inside the loop body

In this code segment, the value 1 is assigned to the variable **k** (loop control variable) at first. Then the test expression **k<=3** is evaluated. Since it is True, the body of the loop is executed. That is the value of **k** is printed as 1 on the screen. After that the update statement **++k** is executed and the value of **k** becomes 2. The condition **k<=3** is checked again and found to be True. Program control enters the body of the loop and prints the value of **k** as 2 on the screen. Again the update statement is executed and the value of **k** is changed to 3. Since the condition is still True, body is executed and 3 is displayed on the screen. The value of **k** is again updated to 4 and now the test expression is evaluated to False. The control comes out of the loop and executes the next statement after the while loop. In short, the output of the code will be:

1      2      3



Imagine what will happen if the initial value of **k** is 5? The test expression is evaluated to False in the first evaluation and the loop body will not be executed. This clearly shows that **while** loop controls the entry into the body of the loop.

Let us see a program that uses **while** loop to print the first 10 natural numbers.

#### Program 7.11: To print the first 10 natural numbers

```
#include<iostream>
using namespace std;
int main()
{
    int n = 1;
    while(n <= 10)
    {
        cout<< n << " ";
        ++n;
    }
    return 0;
}
```

Initialisation of loop variable

Test expression

Body of loop

Updating of loop variable

The output of Program 7.11 will be as follows:

1 2 3 4 5 6 7 8 9 10

Program 7.12 uses **while** loop to find the sum of even numbers upto 20. This program shows that the loop control variable can be updated using any operation.

#### Program 7.12: To find the sum of even numbers upto 20

```
#include<iostream>
using namespace std;
int main()
{
    int i, sum = 0;
    i = 2;
    while( i<= 20)
    {
        sum = sum + i;
        i = i + 2;
    }
    cout<<"\nThe sum of even numbers up to 20 is: "<<sum;
    return 0;
}
```

Loop control variable is updated by adding 2 to the current value

The output of Program 7.12 is given below:

The sum of even numbers up to 20 is: 110



**Let us do**

1. Modify the Program 7.11 to display all odd numbers between 100 and 200.
2. Modify the Program 7.12 to find the average of the first  $N$  natural numbers.



If we put a semi colon (;) after the test expression of **while** statement, there will not be any syntax error. But the statements within the following pair of braces will not be considered as loop body. The worst situation is that, if the test expression is evaluated to be True, neither the code after the **while** loop will be executed nor the program will be terminated. It is a case of infinite loop.

### 7.2.2 for statement

**for** loop is also an entry-controlled loop in C++. All the three loop elements (initialisation, test expression and update statement) are placed together in **for** statement. So it makes the program compact. The syntax is:

```
for (initialisation; test expression; update statement)
{
    body-of-the-loop;
}
```

The execution of **for** loop is the same as that of **while** loop. The flowchart used for **while** can explain the working of **for** loop. Since the three elements come together this statement is more suitable in situations where counting is involved. The flowchart given in Figure 7.4 is commonly used to show the execution of

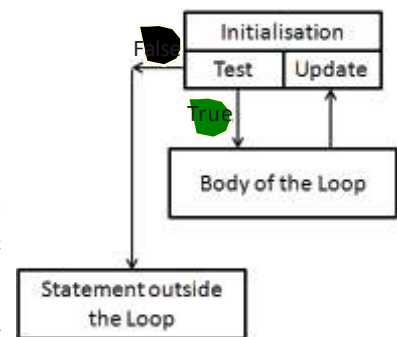


Fig. 7.4: Execution of **for** loop

**for** statement. At first, the initialisation takes place and then the test expression is evaluated. If its result is True, body-of-the-loop is executed, otherwise the program control goes out of the **for** loop. After the execution of the loop body, update expression is executed and again test expression is evaluated. These three steps (test, body, update) are continued until the test expression is evaluated to False.

The loop segment used in Program 7.11 can be replaced with a **for** loop as follows:

```
for (n=1; n<=10; ++n)
    cout << n << " ";
```

This code is executed in the same way as in the case of **while** loop.

**Let us do**

*The steps 1 and 2 in the execution sequence of the `for` loop just mentioned before are given below. Write down the remaining steps.*

*Step 1:  $n = 1$ , Condition is True, 1 is displayed,  $n$  becomes 2*

*Step 2: Condition is True, 2 is displayed,  $n$  becomes 3*

*Step 3: .....*

Let us write a program using `for` loop to find the factorial of a number. Factorial of a number, say  $N$ , represented as  $N!$ , is the product of the first  $N$  natural numbers. For example, factorial of 5 ( $5!$ ) is calculated by  $1 \times 2 \times 3 \times 4 \times 5 = 120$ .

### Program 7.13: To find the factorial of a number using `for` loop

```
#include <iostream>
using namespace std;
int main()
{
    int n, i;
    long fact=1;
    cout<<"Enter the number: ";
    cin>>n;
    for (i=1; i<=n; ++i)
        fact = fact * i;
    cout << "Factorial of " << n << " is " << fact;
    return 0;
}
```

Initialisation; Test Expression; Updation

Loop body

The following is a sample output of program 7.13

```
Enter the number: 6
Factorial of 6 is 720
```

Another program is given below which gives the class average of scores obtained in Computer Science. Program 7.14 accepts the value for  $n$  as the number of students, then reads the scores of each student and prints the average score.

### Program 7.14: To find the average score of $n$ students

```
#include<iostream>
using namespace std;
int main()
{
    int i, sum, score, n;
    float avg;
    cout << "How many students? ";
    cin >> n ;
```



```

for( i=1, sum=0; i<=n; ++i)
{
    cout << "Enter the score of student " << i << ": ";
    cin >> score;
    sum = sum + score;
}
avg = (float)sum / n;
cout << "Class Average: " << avg;
return 0;
}

```

Initialisation contains  
two expressions

Explicit type  
conversion

The following is a sample output of Program 7.14 for 5 students

```

How many students? 5
Enter the score of student 1: 45
Enter the score of student 2: 50
Enter the score of student 3: 52
Enter the score of student 4: 34
Enter the score of student 5: 55

Class Average: 47.2

```

In Program 7.14, the initialisation contains two expressions `i=1` and `sum=0` separated by comma. The initialisation part may contain more than one expression, but they should be separated by comma. Both the variables **i** and **sum** get their first values 1 and 0, respectively. Then, the test expression `i<=n` is evaluated to be True and body of the loop is executed. After the execution of the body of the loop the update expression `++i` is executed. Again the test expression `i<=n` is evaluated, and body of the loop is executed since the condition is True. This process continues till the test expression returns False. It has occurred in the sample output when the value of **i** becomes 6.



*Write a program to display the multiplication table of a given number. Assume that the number will be the input to the variable **n**. The body of the loop is given below:*

**Let us do**

```
cout<<i<<" x "<<n<<" = "<< i * n <<"\n";
```

*Give the output also.*

While using `for` loops certain points are to be noted. The given four code segments explain these special cases. Assume that all the variables used in the codes are declared with `int` data type.



**Code segment 1:**      `for (n=1; n<5; n++);`  
                              `cout<<n;`

A semicolon appears after the parentheses of `for` statement. It is not a syntax error. Can you predict the output? If it is 5, you are correct. This loop has no body. But its process will be completed as usual. The initialisation assigns 1 to **n** and the condition is evaluated to True. Since there is no loop body update takes place and the process continues till **n** becomes 5. At that point, condition is evaluated to be False and the program control comes out of the loop. The output statement then displays 5 on the screen.

**Code segment 2:**      `for (n=1; n<5; )`  
                              `cout<<n;`

In this code, update expression is not present. It does not make any syntax error in the code. But on execution, the loop will never be terminated. The number 1 will be displayed infinitely. We call this an infinite loop.

**Code segment 3:**      `for ( ; n<5; n++)`  
                              `cout<<n;`

The output of this code cannot be predicted. Since there is no initialisation, the control variable **n** gets some integer value. If it is smaller than 5, the body will be executed until the condition becomes False. If the default value of **n** is greater than or equal to 5, the loop will be terminated without executing the loop body.

**Code segment 4:**      `for (n=1; ; n++)`  
                              `cout<<n;`

The test expression is missing in this code. C++ takes this absence as True and obviously the loop becomes an infinite loop.

The four code segments given above reveal that all the elements of a **for** loop are optional. But this is not the case for **while** and **do...while** statements. Test expression is compulsory for these two loops. Other elements are optional, but be cautious about the output.

Another aspect to be noted is that we can provide a number instead of the test expression. If it is zero it will be treated as False, otherwise True.

### Check yourself



1. Write a program to find the sum and average of all even numbers between 1 and 49.
2. Write a program to print the numbers between 10 and 50 which are divisible by both 3 and 5.
3. Predict the output of the following code

```
for(int i=1; i<=10; ++i);
cout << i+2;
```



### 7.2.3 do...while statement

In the case of for loop and while loop, the test expression is evaluated before executing the body of the loop. If the test expression evaluates to False for the first time itself, the body is never executed. But in some situations, it is necessary to execute the loop body at least once, without considering the result of the test expression. In that case the **do...while** loop is the best choice. Its syntax is :

```
initialisation of loop control variable;
do
{
    body of the loop;
    updation of loop control variable;
} while(test expression);
```

Figure 7.5 shows the order of execution of this loop. Here, the test expression is evaluated only after executing body of the loop. So do...while loop is an exit controlled loop. If the test expression evaluates to False, the loop will be terminated. Otherwise, the execution process will be continued. It means that in do...while loop the body will be executed at least once irrespective of the result of the condition.

Let us consider the following program segment to illustrate the execution of do...while loop.

```
int k=1;
do
{
    cout << k << '\t';
    ++k;
} while(k<=3);
```

Initialisation before the loop

Body of loop

Updation inside the loop body

Test expression

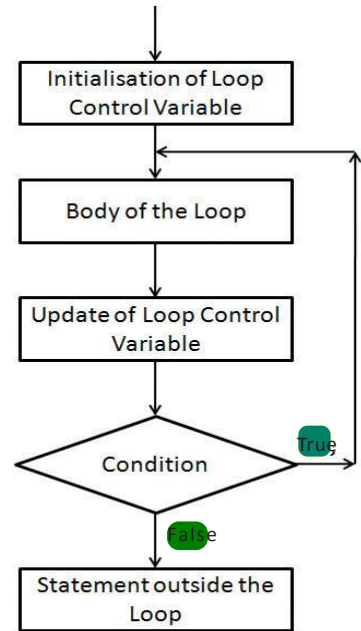


Fig. 7.5: Execution of do..while loop

At first, the value 1 is assigned to the variable **k**. Then body of the loop is executed and the value of **k** is printed as 1. After that the **k** is incremented by 1 (now **k=2**). Then it checks the condition **k<=3**. Since it is found True the body of the loop is executed to print the value of **k**, i.e. 2 on the screen. Again the updation process is carried out,



which makes value of **k** as 3 and the condition **k ≤ 3** is checked again. As it is True, the body of the loop is executed to print the value 3. The variable **k** is again updated to 4 and now the condition is evaluated to be False. It causes the program control to come out of the loop and executes the next statement after the loop body. Thus the output of the code will be:

1          2          3

Now let us see how this loop differs from the other two. Imagine that the initial value of **k** is 5. What will happen? The body of the loop is executed and the value of **k** will be printed on the screen as 5. After that the variable **k** will be updated by incrementing it by 1 and **k** becomes 6. On checking the condition **k ≤ 3**, the test expression is evaluated to False and the control comes out of the loop. This clearly shows that in `do...while` loop there is no restriction to enter the loop body for the first time. So if we want the body to be executed based on the True value of the condition, use `while` or `for` loops.

Let us see an interactive program in the sense that some part of the code will be executed on user's choice. The simplest form of such programs provides facility to accept user's response for executing a code segment repeatedly. Program 7.15 illustrates the use of `do...while` loop to write an interactive program to find the area of rectangles by accepting the length and breadth of each rectangle from the user.

#### Program 7.15: To find the area of rectangles

```
#include <iostream>
using namespace std;
int main()
{
    float length, breadth, area;
    char ch;
    do
    {
        cout << "Enter length and breadth: ";
        cin >> length >> breadth;
        area = length * breadth;
        cout << "Area = " << area;
        cout << "Any more rectangle (Y/N)? ";
        cin >> ch;
    } while (ch == 'Y' || ch == 'y');
    return 0;
}
```



A sample output of Program 7.15 is given below:

```
Enter length and breadth: 3.5      7
Area = 24.5
Any more rectangle (Y/N)? Y
Enter length and breadth: 6      4.5
Area = 27
Any more rectangle (Y/N)? N
```

We have discussed all the three looping statements of C++. Table 7.2 shows a comparison between these statements.

<b>for loop</b>	<b>while loop</b>	<b>do...while loop</b>
Entry controlled loop	Entry controlled loop	Exit controlled loop
Initialization along with loop definition	Initialization before loop definition	Initialization before loop definition
No guarantee to execute the loop body at least once	No guarantee to execute the loop body at least once	Will execute the loop body at least once even though the condition is False

*Table 7.2: Comparison between the looping statements of C++*

### 7.2.4 Nesting of loops

Placing a loop inside the body of another loop is called **nesting** of a loop. When we nest two loops, the outer loop counts the number of completed repetitions of the inner loop. Here the loop control variables for the two loops should be different.

Let us observe how a nested loop works. Take the case of a minute-hand and second-hand of a clock. Have you noticed the working of a clock?. While the minute-hand stands still at a position, the second-hand moves to complete one full rotation (say 1 to 60). The minute hand moves to the next position (that is, the next minute) only after the second hand completes one full rotation. Then the second-hand again completes another full rotation corresponding to the minute-hand's current position. For each position of the minute-hand, second-hand completes one full rotation and the process goes on. Here the second hand movement can be treated as the execution of the inner loop and the minute-hand's movement can be treated as the execution of the outer loop.

All types of loops in C++ allow nesting. An example is given to show the working procedure of a nested `for` loop.

```

for( i=1; i<=2; ++i)
{
    for(j=1; j<=3; ++j)
    {
        cout<< "\n" << i << " and " << j;
    }
}

```

Outer loop

Inner loop

Initially value 1 is assigned to the outer loop variable **i**. Its test expression is evaluated to be True and hence the body of the loop is executed. The body contains the inner loop with the control variable **j** and it begins to execute by assigning the initial value 1 to **j**. The inner loop is executed 3 times, for **j** =1, **j**=2, **j**=3. Each time it evaluates the test expression **j<=3** and displays the output since it is True.

```

1 and 1
1 and 2
1 and 3

```

The first 1 is of **i**  
and the second 1 is  
of **j**

When the test expression **j<=3** is False, the program control comes out of the inner loop. Now the update statement of the outer loop is executed which makes **i**=2. Then the test expression **i<=2** is evaluated to True and once again the loop body (i.e. the inner loop) is executed. Inner loop is again executed 3 times, for **j**=1, **j**=2, **j**=3 and displays the output.

```

2 and 1
2 and 2
2 and 3

```

After completing the execution of the inner loop, the control again goes back to the update expression of the outer loop. Value of **i** is incremented by 1 (Now **i**=3) and the test expression **i<=2** is now evaluated to be False. Hence the loop terminates its execution. Table 7.3 illustrates the execution of the above given program segment:

Iterations	Outer loop	Inner loop	Output
1	1	1	1 and 1
2	1	2	1 and 2
3	1	3	1 and 3
4	2	1	2 and 1
5	2	2	2 and 2
6	2	3	2 and 3

Table 7.3: Execution of a nested loop



When working with nested loops, the control variable of the outer loop changes its value only after the inner loop is terminated. Let us write a program to display the following triangle using nested loop:

```
*
**
***
****
*****
```

### Program 7.16 : To display a triangle of stars

```
#include<iostream>
using namespace std;
int main()
{   int i, j;
    char ch = '*';
    for(i=1; i<=5; ++i)           //outer loop
    {
        cout<< "\n" ;
        for(j=1; j<=i; ++j)       // inner loop
            cout<<ch;
    }
    return 0;
}
```



#### Let us do

1. Predict the output of the following program segment:

```
sum = 0;
for (i=1; i<3; ++i)
{
    for (j=1; j<3; ++j)
        sum = sum + i * j ;
}
cout<<sum;
```

2. Write C++ programs to display the following triangles:

1	1
2 2	1 2
3 3 3	1 2 3
4 4 4	1 2 3 4
5 5 5 5	1 2 3 4 5



### 7.2.5 Nesting of control statements

We have discussed nesting of `if` and nesting of loops. Any control statement can be nested with another control statement. A loop can contain a selection statement such as `if` or `switch`. Similarly, a selection statement can contain a loop such as `while`, `for` or `do...while`. Program 7.17 contains a loop and its body includes a `switch` statement. It is the usual style of a menu driven program.

#### Program 7.17: To input two numbers and perform an arithmetic operation based on user's choice

```
#include<iostream>
using namespace std;
int main()
{
    char ch;
    float n1, n2;
    cout<<"Enter two numbers: ";
    cin>>n1>>n2;
    do
    {
        cout<<"\nNumber 1: "<<n1<<"\tNumber 2: "<<n2;
        cout<<"\n\t\tOperator Menu";
        cout<<"\n\t1. Addition (+)";
        cout<<"\n\t2. Subtraction (-)";
        cout<<"\n\t3. Multiplication (*)";
        cout<<"\n\t4. Division (/)";
        cout<<"\n\t5. Exit (E)";
        cout<<"\nEnter Option number or operator: ";
        cin>>ch;
        switch(ch)
        {
            case '1' :
            case '+' : cout<<n1<<" + "<<n2<<" = "<<n1+n2;
                      break;
            case '2' :
            case '-' : cout<<n1<<" - "<<n2<<" = "<<n1-n2;
                      break;
            case '3' :
            case '*' : cout<<n1<<" * "<<n2<<" = "<<n1*n2;
                      break;
            case '4' :
            case '/' : cout<<n1<<" / "<<n2<<" = "<<n1/n2;
                      break;
```



```

        case '5' :
        case 'E' :
        case 'e' : cout<<"Thank You for using the program";
                    break;
        default  : cout<<"Invalid Choice!!";
    }
} while (ch!='5' && ch!='E' && ch!='e');
return 0;
}

```

The following is a sample output of Program 7.17:

Enter two numbers: 25     4  
 Number 1: 25             Number 2: 4

Operator Menu

1. Addition (+)
2. Subtraction (-)
3. Multiplication (\*)
4. Division (/)
5. Exit (E)

Enter Option number or operator: 1

User Input

25 + 4 = 29

Number 1: 25     Number 2: 4

Operator Menu

1. Addition (+)
2. Subtraction (-)
3. Multiplication (\*)
4. Division (/)
5. Exit (E)

Enter Option number or operator: /

User Input

25 / 4 = 6.25

Number 1: 25     Number 2: 4

Operator Menu

1. Addition (+)
2. Subtraction (-)
3. Multiplication (\*)
4. Division (/)
5. Exit (E)

Enter Option number or operator: 5

User Input

Thank You for using the program



We will discuss more programs using various combinations of nesting of control statements in the Program gallery section.

### 7.3 Jump statements

The statements that facilitate the transfer of program control from one place to another are called **jump statements**. C++ provides four types of jump statements that perform unconditional branching in a program. They are **return**, **goto**, **break** and **continue** statements. In addition, C++ provides a standard library function **exit ()** that helps us to terminate a program.

The return statement is used to transfer control back to the calling program or to come out of a function. It will be explained in detail later in Chapter 10. Now, we will discuss the other jump statements.


#### 7.3.1 goto statement

The **goto** statement can transfer the program control to anywhere in the function. The target destination of a **goto** statement is marked by a *label*, which is an identifier.

The syntax of **goto** statement is:

```
goto label;
.....;
.....;
label: .....;
.....;
```

where the **label** can appear in the program either before or after **goto** statement. The **label** is followed by a colon (:) symbol. For example, consider the following code fragment which prints numbers from 1 to 50.

```
int i=1;
start:  Label
cout<<i;
++i;
if (i<=50)
    goto start;
```

Here, the **cout** statement prints the value 1. After that **i** is incremented by 1 (now **i=2**), then the test expression **i<=50** is evaluated. Since it is True the control is transferred to the statement marked with the label **start**. When the test expression evaluates to False, the process terminates and transfers the program control following the **if** statement.

Let us see another example. Here a number is accepted and tested with a pre-defined value. If it matches, the program continues, otherwise it terminates.






```

int p;
cout<<"Enter the Code: ";
cin>>p;
if(p!=7755)
    goto end;
cout<<"Enter the details";
.....;
.....;
end:
    cout<<"Sorry, the code number is wrong. Try again!";

```



Here the program validates the user input. The program accepts other details only if it is a valid code, otherwise the control goes to the label end. It is to be noted that the usage of `goto` is not encouraged in structured programming.

### 7.3.2 break statement

When a `break` statement is encountered in a program, it takes the program control outside the immediate enclosing loop (`for`, `while`, `do...while`) or `switch` statement. Execution continues from the statement immediately after the control structure. We have already discussed the impact of `break` in `switch` statement. Let us see how it affects the execution of loops. Consider the following two program segments.

#### *Code segment 1:*

```

i=1;
while(i<=10)
{
    cin>>num;
    if (num==0)
        break;
    cout<<"Entered number is: "<<num;
    cout<<"\nInside the loop";
    ++i;
}
cout<<"\nComes out of the loop";

```

The above code fragment allows to input 10 different numbers. During the input if any number happens to be 0, the program control comes out of the loop by skipping the rest of the statements within the loop-body and displays the message "Comes out of the loop" on the screen. Let us consider another code segment that uses `break` within a nested loop.

**Code segment 2:**

```

for (i=1; i<=5; ++i)    //outer loop
{
    cout<<"\n";
    for (j=1; j<=i; ++j)    //inner loop
    {
        cout<<"* ";
        if (j==3)
            break;
    }
}

```

This code segment will display the following pattern:

```

*
* *
* * *
* * *
* * *

```

Whenever **j**  
becomes 3, the inner  
loop terminates

The nested loop executes normally for the value of  $i=1, i=2, i=3$ . For each value of  $i$ , the variable  $j$  takes values from 1 to  $i$ . When the value of  $i$  becomes 4, the inner loop executes for the value of  $j = 1, j=2, j=3$  and comes out from the inner loop on executing the `break` statement.

**7.3.3 continue statement**

**continue** statement is another jump statement used for skipping over a part of the code within the loop-body and forcing the next iteration. The `break` statement forces termination of the loop, but `continue` statement forces next iteration of the loop.

The following program segment explains the working of `continue` statement:

```

for (i=1; i<=10; ++i)
{
    if (i==6)
        continue;
    cout<<i<<"\t";
}

```

This code gives the following output:

```

1      2      3      4      5      7      8      9      10

```

Note that 6 is not in the list. When the value of  $i$  becomes 6 the `continue` statement is executed. As a result, the output statement is skipped and program control goes to the update expression for next iteration.

A `break` statement inside a loop will abort the loop and transfer control to the statement following the loop. A `continue` statement will just abandon the current iteration and let the loop start next iteration. When `continue` statement is used within `while` and `do...while` loops, care should be taken to avoid infinite execution. Table 7.4 shows a comparison between `break` and `continue` statements.

<b>break statement</b>	<b>continue statement</b>
<ul style="list-style-type: none"> <li>Used with <code>switch</code> and loops.</li> <li>Brings the program control outside the <code>switch</code> or loop by skipping the rest of the statements within the block.</li> <li>Program control goes out of the loop even though the test expression is <code>True</code>.</li> </ul>	<ul style="list-style-type: none"> <li>Used only with loops.</li> <li>Brings the program control to the beginning of the loop by skipping the rest of the statements within the block.</li> <li>Program control goes out of the loop only when the test expression becomes <code>False</code>.</li> </ul>

*Table 7.4: Comparison between the `break` and `continue` statements*

C++ provides a built-in function `exit()` which terminates the program itself. The `exit()` function can be used in a program only if we include the header file `cstdlib` (process.h.in Turbo C++). Program 7.18 illustrates the working of this function.

**Program 7.18: To check whether the given number is prime or not**

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
    int i, num;
    cout<<"Enter the number: ";
    cin>>num;
    for(i=2; i<=num/2; ++i)
    {
        if(num%i == 0)
        {
            cout<<"Not a Prime Number";
            exit(0);
        }
    }
    cout<<"Prime Number";
    return 0;
}
```



The test expression in the `for` loop of Program 7.18 can be replaced by `i<=sqrt (num)`, where `sqrt ()` is a function which gives the square root of the given number. If a number has no factors from 2 to its square root, the number will be a prime number. To use `sqrt ()`, we have to include the statement `#include<cmath>`

Some sample outputs of program 7.18 are shown below:

Output 1:

```
Enter the number: 17
Prime Number
```

Output 2:

```
Enter the number: 18
Not a Prime Number
```

### Check yourself



1. The `goto` statement causes control to go to
  - (a) an operator
  - (b) a Label
  - (c) a variable
  - (d) a function
2. A `break` statement causes an exit
  - (a) only from the innermost loop.
  - (b) only from the innermost switch.
  - (c) from all loops and switches.
  - (d) from the innermost loop or switch.
3. The `exit()` function takes the control out of
  - (a) the function it appears in.
  - (b) the loop it appears in.
  - (c) the block it appears in.
  - (d) the program it appears in.
4. Name the header file to be included for using `exit()` function.

### Program gallery

This section contains a collection of programs which use different control statements for solving various problems. The sample outputs of the programs are also given after each program.

Program 7.19 accepts the three coefficients of a quadratic equation of the form  $ax^2 + bx + c = 0$  and calculates its roots. The value of **a** should not be 0 (zero). To solve this problem, the discriminant value of the quadratic equation is to be determined using the formula  $(b^2 - 4ac)$ , to identify the nature of the roots. Formula is also available to find the roots. In this program we use the function `sqrt ()` to get the square root of a number. The header file `math.h` is to be included in the program to use this function.

**Program 7.19: To find the roots of a quadratic equation**

```

#include <iostream>
#include <cmath> // to use sqrt() function
using namespace std;
int main()
{
    float a, b, c, root1, root2, d;
    cout<< "Enter the three coefficients: ";
    cin >> a >> b >> c ;
    if (!a) // equivalent to if (a == 0)
        cout<<"Value of \'a \' should not be zero\n"
            <<"Aborting!!!!\n";
    else
    {
        d =b*b-4*a*c;           //beginning of else block
        if (d > 0)
        {
            root1 = (-b + sqrt(d))/(2*a);
            root2 = (-b - sqrt(d))/(2*a);
            cout<<"Roots are REAL and UNEQUAL\n";
            cout<<"Root1 = "<<root1<<"\tRoot2 = "<<root2;
        }
        else if (d == 0)
        {
            root1 = -b/(2*a);
            cout<<"Roots are REAL and EQUAL\n";
            cout<<"Root1 ="<<root1;
        }
        else
            cout<<"Roots are COMPLEX and IMAGINARY";
    } // end of else block of outer if
    return 0;
}

```

**Output 1:**

```

Enter the three coefficients:      2      3      4
Roots are COMPLEX and IMAGINARY

```

**Output 2:**

```

Enter the three coefficients:      3      5      1
Roots are REAL and UNEQUAL
Root1 =  -0.232408    Root2 = -1.434259

```



Program 7.20 displays the first N terms of the Fibonacci series. This series begins with terms 0 and 1. The next term onwards will be the sum of the last two terms. The series is 0, 1, 1, 2, 3, 5, 8, 13, .....

#### Program 7.20: To print n terms of the Fibonacci series

```
#include <iostream>
using namespace std;
int main()
{
    int first=0, second=1, third, n;
    cout<<"\nEnter number of terms in the series: ";
    cin>>n;
    cout<<first<<"\t"<<second;
    for(int i=3; i<=n; ++i)
    {
        third = first + second;
        cout<<"\t"<<third;
        first = second;
        second = third;
    }
    return 0;
}
```

If the variables `first` and `second` are initialised with the values -1 and +1 respectively, we can avoid the `cout` statement for displaying the first two terms.

Output:

Enter number of terms in the series: 10

0    1        1        2        3        5        8        13       21       34

Program 7.21 reads a number and checks whether it is palindrome or not. A number is said to be palindrome if it is equal to its image. By the term image, we mean a number obtained by reversing the digits of the original number. That is, the image of 163 is 361. Since these two are not equal the number 163 is not palindrome. The number 232 is a palindrome number.

#### Program 7.21: To check whether the given number is palindrome or not

```
#include <iostream>
using namespace std;
int main()
{
    int num, copy, digit, rev=0;
    cout<<"Enter the number: ";
    cin>>num;
    copy=num;
```

The value of `num` will be 0 after the completion of loop. That is why the original value is copied into another variable

```

while(num != 0)
{
    digit = num % 10;
    rev = (rev * 10) + digit;
    num = num/10;
}
cout<<"The reverse of the number is: "<<rev;
if (rev == copy)
    cout<<"\nThe given number is a palindrome.";
else
    cout<<"\nThe given number is not a palindrome.";
return 0;
}

```

Output 1:

```

Enter the number: 363
The reverse of the number is: 363
The given number is a palindrome.

```

Output 2:

```

Enter the number: 257
The reverse of the number is: 752
The given number is not a palindrome.

```

#### Program 7.22: To accept n integers and print the largest among them

```

#include <iostream>
using namespace std;
int main()
{
    int num, big, count;
    cout<<"How many Numbers in the list? ";
    cin >> count;
    cout<<"\nEnter first number: ";
    cin >> num;
    big = num;
    for(int i=2; i<=count; i++)
    {
        cout<<"\nEnter next number: ";
        cin >> num;
        if(num > big) big = num;
    }
    cout<<"\nThe largest number is " << big;
    return 0;
}

```



Output:

```
How many Numbers in the list? 5
Enter first number: 23
Enter next number: 12
Enter next number: -18
Enter next number: 35
Enter next number: 18
The largest number is 35
```



## Let us sum up

The statements providing facilities for taking decisions or for performing repetitive actions in a program are known as control statements. The control statements are the backbones of a computer program. In this chapter we covered the different types of control statements such as selection statements (if, if...else, if...else if, switch), iteration statements (for, while, do...while) and also jump statements (goto, break, continue, exit() function). All these control statements will help us in writing efficient C++ programs.



## Learning outcomes

After the completion of this chapter the learner will be able to

- use control statements in C++ for problem solving.
- identify the situation where control statements are used in a program.
- use correct control statements suitable for the situations.
- categorise different types of control statements.
- identify different types of jump statements in C++.
- write C++ programs using control statements.

## Sample questions

### Very short answer type

1. Write the significance of break statement in switch statement. What is the effect of absence of break in a switch statement?
2. What will the output of the following code fragment be?

```
for (i=1; i<=10; ++i) ;
cout<<i+5;
```



3. Rewrite the following statement using **while** and **do while** loops.

```
for (i=1; i<=10; i++) cout<<i;
```

4. How many times will the following loop execute?

```
int s=0, i=0;
while (i++<5)
    s+=i;
```

5. Write the name of the header file which contains the `exit()` function.
6. Which statement in C++ can transfer control of the program to a named label?
7. Write the purpose of `default` statement in `switch` statement.

### Short answer type

1. Consider two program fragments given below.

```
// version 1
```

```
cin>>mark;
```

```
if (mark >= 90)
```

```
cout<<" A+";
```

```
if (mark >= 80 && mark <90)
```

```
cout<<" A";
```

```
if (mark >= 70 && mark <80)
```

```
cout<<" B+";
```

```
if (mark >= 60 && mark <70)
```

```
cout<<" B";
```

```
//version 2
```

```
cin>>mark;
```

```
if (mark>=90)
```

```
cout<<" A+";
```

```
else if (mark>=80 && mark <90)
```

```
cout<<" A";
```

```
else if (mark>=70 && mark <80)
```

```
cout<<" B+";
```

```
else if (mark>=60 && mark <70)
```

```
cout<<" B";
```

Discuss the advantages of version 2 over version 1.

2. Briefly explain the working of a `for` loop along with its syntax. Give an example of `for` loop to support your answer.
3. Compare and discuss the suitability of three loops in different situations.
4. Consider the following `if else if` statement. Rewrite it with `switch` statement.

```
if (a==1)
```

```
    cout << "One";
```

```
else if (a==0)
```

```
    cout << "Zero";
```

```
else
```

```
    cout << "Not a binary digit";
```

5. What is wrong with the following `while` statement if the value of `z = 3`?

```
while (z>=0)
```

```
    sum+=z;
```



6. What will the output of the following code fragments be?

```
for (outer=10; outer > 5; --outer)
{
    for (inner=1; inner<4; ++inner)
        cout<<outer <<"\t"<<inner <<endl;
}
```

7. What will the output of the given code fragments be? Explain.

```
for (n = 1; n <= 10; ++n)
{
    for ( m=1; m <= 5 ; ++m)
        num = n*m;
    cout<<num <<endl;
}
```

8. Write the importance of a loop control variable. Briefly explain the different parts of a loop.

### Long answer type

1. What output will be produced by the following code fragment?

```
int val, res, n=1000;
cin>>val;
res = n+val > 1750 ? 400 : 200;
```

- If the input is 2000
  - If the input is 500
2. Write a program to find the sum of digits of a number using
- Entry controlled loop.
  - Exit controlled loop.
3. Write a program to print Armstrong numbers less than 1000. (An Armstrong number is a number which is equal to the sum of cubes of its digits. Eg.  $153 = 1^3 + 5^3 + 3^3$ )
4. Explain the different jump statements available in C++.
5. Write a program to produce the following output using nested loop:
- ```
A
A  B
A  B  C
A  B  C  D
A  B  C  D  E
```
8. Suppose you forgot to write the word `else` in an `if...else` statement. Discuss how it will affect the output of your program?

## Key Concepts

- **Array and its need**
  - Declaring arrays
  - Memory allocation for arrays
  - Array initialization
  - Accessing elements of arrays
- **Array operations**
  - Traversal
  - Sorting
    - Selection sort
    - Bubble sort
  - Searching
    - Linear search
    - Binary search
- **Two dimensional (2D) arrays**
  - Declaring 2D arrays
  - Matrices as 2D arrays
- **Multi-dimensional arrays**

## Arrays

We used variables in programs to refer data. If the quantity of data is large, more variables are to be used. This will cause difficulty in accessing the required data. We learned the concept of data types in Chapter 6 and we used basic data types to declare variables and perform type conversion. In this chapter, a derived data type in C++, named 'array' is introduced. The word 'array' is not a data type name, rather it is a kind of data type derived from fundamental data types to handle large number of data easily. We will discuss the creation and initialization of arrays, and some operations like traversal, sorting, and searching. We will also discuss two dimensional arrays and their operations for processing matrices.

### 8.1 Array and its need

An **array** is a collection of elements of the same type placed in contiguous memory locations. Arrays are used to store a set of values of the same type under a single variable name. Each element in an array can be accessed using its position in the list, called index number or subscript.

Why do we need arrays? We will illustrate this with the help of an example. Let us consider a situation where we need to store the scores of 20 students in a class and has to find their class average. If we try to solve this problem by making use of variables, we will need 20 variables to store students' scores. Remembering





and managing these 20 variables is not an easy task and the program may become complex and difficult to understand.

```
int  a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t;
float avg;
cin>>a>>b>>c>>d>>e>>f>>g>>h>>i>>j>>k>>l>>m>>n>>o>>p>>q>>r>>s>>t;
avg = (a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t)/20.0;
```

As it is, this code is fine. However, if you want to modify it to deal with the scores of a large number of students, say 1000, you have a very long and repetitive task at hand. We have to find a way to reduce the complexity of this task.

The concept of array comes as a boon in such situations. As it is a collection of elements, memory locations are to be allocated. We know that declaration statement is needed for memory allocation. So, let us see how arrays are declared and used.

### 8.1.1 Declaring arrays

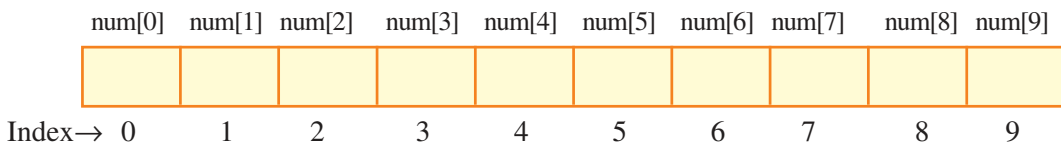
Just like the ordinary variable, the array is to be declared properly before it is used. The syntax for declaring an array in C++ is as follows.

```
data_type array_name[size];
```

In the syntax, `data_type` is the type of data that the array variable can store, `array_name` is an identifier for naming the array and the `size` is a positive integer number that specifies the number of elements in the array. The following is an example:

```
int num[10];
```

The above statement declares an array named `num` that can store 10 integer numbers. Each item in an array is called an `element` of the array. The elements in the array are stored sequentially as shown in Figure 8.1. The first element is stored in the first location; the second element is stored in the second location and so on.



*Fig. 8.1: Arrangement of elements in an array*

Since the elements in the array are stored sequentially, any element can be accessed by giving the array's name and the element's position. This position is called the **index** or **subscript** value. In C++, the array index starts with zero. If an array is



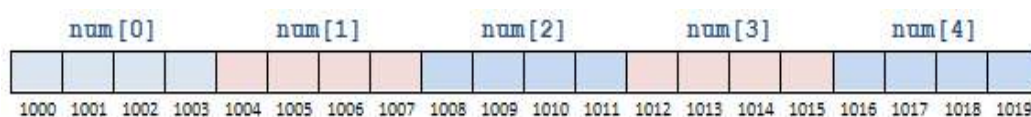
declared as `int num[10]`; then the possible index values are from 0 to 9. In this array, the first element can be referenced as `num[0]` and the last element as `num[9]`. The subscripted variable, `num[0]`, is read as “num of zero” or “num zero”. It’s a shortened way of saying “the num array subscripted by zero”. So, the problem of referring to the scores of 1000 students can be resolved by the following statement:

```
int score[1000];
```

The array, named `score`, can store the scores of 1000 students. The score of the first student is referenced by `score[0]` and that of the last by `score[999]`.

### 8.1.2 Memory allocation for arrays

The amount of storage required to hold an array is directly related to its type and size. Figure 8.2 shows the memory allocation for the first five elements of array **num**, assuming 1000 as the address of the first element. Since `num` is an integer type array, size of each element is 4 bytes (in a system with 32 bit integer representation using GCC) and it will be represented in memory as shown in Figure 2.2.



*Fig. 8.2: Memory allocation for an integer array*

For a single dimensional array, the total size allocated can be computed using the following formula:

$$\text{total\_bytes} = \text{sizeof}(\text{array\_type}) \times \text{size\_of\_array}$$

For example, total bytes allocated for the array declared as `float a[10]`; will be  $4 \times 10 = 40$  bytes.

### 8.1.3 Array initialisation

Array elements can be initialised in their declaration statements in the same manner as in the case of variables, except that the values must be included in braces, as shown in the following examples:

```
int score[5] = {98, 87, 92, 79, 85};
char code[6] = {'s', 'a', 'm', 'p', 'l', 'e'};
float wgpa[7] = {9.60, 6.43, 8.50, 8.65, 5.89, 7.56, 8.22};
```

Initial values are stored in the order they are written, with the first value used to initialize element 0, the second value used to initialize element 1, and so on. In the first example, `score[0]` is initialized to 98, `score[1]` is initialized to 87, `score[2]` is initialized to 92, `score[3]` is initialized to 79, and `score[4]` is initialized to 85.



If the number of initial values is less than the size of the array, they will be stored in the elements starting from the first position and the remaining positions will be initialized with zero, in the case of numeric data types. For char type array, such positions will be initialised with ' ' (space bar) character. When an array is initialized with values, the size can be omitted. For example, the following declaration statement will reserve memory for five elements:

```
int num[] = {16, 12, 10, 14, 11};
```

### 8. 1.4 Accessing elements of arrays

The array elements can be used anywhere in a program as we do in the case of normal variables. We have seen that array is accessed element-wise. That is, only one element can be accessed at a time. The element is specified by the array name with the subscript. The following are some examples of using the elements of the score array:

```
score[0] = 95;
score[1] = score[0] - 11;
cin >> score[2];
score[3] = 79;
cout << score[2];
sum = score[0] + score[1] + score[2] + score[3] + score[4];
```

The subscript in brackets can be a variable, a constant or an expression that evaluates to an integer. In each case, the value of the expression must be within the valid subscript range of the array. An important advantage of using variable and integer expressions as subscripts is that, it allows sequencing through an array by using a loop. This makes statements more structured keeping away from the inappropriate usage as follows:

```
sum = score[0] + score[1] + score[2] + score[3] + score[4];
```

The subscript values in the above statement can be replaced by the control variable of for loop to access each element in the array sequentially. The following code segment illustrates this concept:

```
sum = 0;
for (i=0; i<5; i++)
    sum = sum + score[i];
```

An array element can be assigned a value interactively by using an input statement, as shown below:

```
for(int i=0; i<5; i++)
    cin>>score[i];
```



When this loop is executed, the first value read is stored in the array element `score[0]`, the second in `score[1]` and the last in `score[4]`.

Program 8.1 shows how to read 5 numbers and display them in the reverse order. The program includes two `for` loops. The first one, allows the user to input array values. After five values have been entered, the second `for` loop is used to display the stored values from the last to the first.

**Program 8.1: To input the scores of 5 students and display them in reverse order**

```
#include <iostream>
using namespace std;
int main()
{
    int i, score[5];
    for(i=0; i<5; i++)    // Reads the scores
    {
        cout<<"Enter a score: ";
        cin>>score[i];
    }
    for(i=4; i>=0; i--)    // Prints the scores
        cout<<"score[" << i << "] is " << score[i]<<endl;
    return 0;
}
```

The following is a sample output of program 8.1:

```
Enter a score: 55
Enter a score: 80
Enter a score: 78
Enter a score: 75
Enter a score: 92
score[4] is 92
score[3] is 75
score[2] is 78
score[1] is 80
score[0] is 55
```

**Let us do**

1. Write array declarations for the following:
  - (a) Scores of 100 students
  - (b) English letters
  - (c) A list of 10 years
  - (d) A list of 30 real numbers
2. Write array initialization statements for the following:
  - (a) An array of 10 scores: 89, 75, 82, 93, 78, 95, 81, 88, 77, and 82
  - (b) A list of five amounts: 10.62, 13.98, 18.45, 12.68, and 14.76
  - (c) A list of 100 interest rates, with the first six rates being 6.29, 6.95, 7.25, 7.35, 7.40 and 7.42.
  - (d) An array of 10 marks with value 0.
  - (e) An array with the letters of VIBGYOR.
  - (f) An array with number of days in each month.
3. Write C++ code segment to input values into the array: `int ar[50];`
4. Write C++ code fragment to display the elements in the even positions of the array: `float val[100];`

## 8.2 Array operations

The operations performed on arrays include traversal, searching, insertion, deletion, sorting and merging. Different logics are applied to perform these operations. Let us discuss some of them.

### 8.2.1 Traversal

Basically traversal means accessing each element of the array at least once. We can use this operation to check the correctness of operations during operations like insertion, deletion etc. Displaying all the elements of an array is an example of traversal. If any operation is performed on all the elements in an array, it is a case of traversal. The following program shows how traversal is performed in an array.

#### Program 8.2: Traversal of an array

```
#include <iostream>
using namespace std;
int main()
{
    int a[10], i;
    cout<<"Enter the elements of the array :";
    for(i=0; i<10; i++)
        cin >> a[i];
```

Reading array elements  
from the user

```

for(i=0; i<10; i++)
    a[i] = a[i] + 1;
cout<<"\nEntered elements of the array are...\n";
for(i=0; i<10; i++)
    cout<< a[i]<< "\t";
return 0;
}

```

A case of traversal

Another case of traversal

## 8.2.2 Sorting

Sorting is the process of arranging the elements of the array in some logical order. This logical order may be ascending or descending in case of numeric values or dictionary order in case of strings. There are many algorithms to do this task efficiently. But at this stage, we will discuss two of the algorithms, known as “selection sort” and “bubble sort”.

### a. Selection sort

One of the simplest sorting techniques is the selection sort. To sort an array in ascending order, the selection sort algorithm starts by finding the minimum value in the array and moving it to the first position. At the same time, the element at the first position is shifted to the position of the smallest element. This step is then repeated for the second lowest value by moving it to the second position, and so on until the array is sorted. The process of finding the smallest element and exchanging it with the element at the respective position is known as a *pass*. For ‘n’ number of elements there will be ‘n – 1’ passes. For example, take a look at the list of numbers given below.

**Initial list**

|    |    |    |   |    |
|----|----|----|---|----|
| 32 | 23 | 10 | 2 | 30 |
|----|----|----|---|----|

**Pass 1**

|    |    |    |   |    |
|----|----|----|---|----|
| 32 | 23 | 10 | 2 | 30 |
|----|----|----|---|----|

In Pass 1, the smallest element **2** is selected from the list. It is then exchanged with the first element.

**After Pass 1**

|   |    |    |    |    |
|---|----|----|----|----|
| 2 | 23 | 10 | 32 | 30 |
|---|----|----|----|----|

**Pass 2**

|   |    |    |    |    |
|---|----|----|----|----|
| 2 | 23 | 10 | 32 | 30 |
|---|----|----|----|----|

In Pass 2 excluding the first element, the smallest element **10** is selected and exchanged with the second element.

**After Pass 2**

|   |    |    |    |    |
|---|----|----|----|----|
| 2 | 10 | 23 | 32 | 30 |
|---|----|----|----|----|

**Pass 3**

|   |    |    |    |    |
|---|----|----|----|----|
| 2 | 10 | 23 | 32 | 30 |
|---|----|----|----|----|

In Pass 3 ignoring the first and second elements, the smallest element **23** is selected and exchanged with the third element.

**After Pass 3**

|   |    |    |    |    |
|---|----|----|----|----|
| 2 | 10 | 23 | 32 | 30 |
|---|----|----|----|----|

**Pass 4**

|   |    |    |    |    |
|---|----|----|----|----|
| 2 | 10 | 23 | 32 | 30 |
|---|----|----|----|----|

In Pass 4 ignoring the 1st, 2nd and 3rd elements, the smallest element **30** is selected and exchanged with the fourth element.

**After Pass 4**

|   |    |    |    |    |
|---|----|----|----|----|
| 2 | 10 | 23 | 30 | 32 |
|---|----|----|----|----|

Though each pass is intended for an exchange, no exchange is made in a pass if the smallest value was already in the correct location. This situation is happened in the Pass 3.

### Algorithm for selection sort

- Step 1.** Start
- Step 2.** Accept a value in N as the number of elements of the array
- Step 3.** Accept N elements into the array AR
- Step 4.** Repeat Steps 5 to 9, (N – 1) times
- Step 5.** Assume the first element in the list as the smallest and store it in MIN and its position in POS
- Step 6.** Repeat Step 7 until the last element of the list
- Step 7.** Compare the next element in the list with the value of MIN. If it is found smaller, store it in MIN and its position in POS
- Step 8.** If the first element in the list and the value in MIN are not the same, then swap the first element with the element at position POS
- Step 9.** Revise the list by excluding the first element in the current list

**Step 10.** Print the sorted array AR

**Step 11.** Stop

Program 8.3 uses **AR** as an integer array which can store maximum of 25 numbers, **N** as the number of elements to be sorted, **MIN** as the minimum value and **POS** as the position or index of the minimum value.

**Program 8.3: Selection sort for arranging elements in ascending order**

```
#include <iostream>
using namespace std;
int main()
{   int AR[25], N, I, J, MIN, POS;
    cout<<"How many elements? ";
    cin>>N;
    cout<<"Enter the array elements: ";
    for(I=0; I<N; I++)
        cin>>AR[I];
    for(I=0; I < N-1; i++)
    {
        MIN=AR[I];
        POS=I;
        for(J = I+1; J < N; J++)
            if (AR[J]<MIN)
            {
                MIN=AR[J];
                POS=J;
            }
        if(POS != I)
        {
            AR[POS]=AR[I];
            AR[I]=MIN;
        }
    }
    cout<<"Sorted array is: ";
    for(I=0; I<N; I++)
        cout<<AR[I]<<"\t";
    return 0;
}
```

A sample output of Program 8.3 is given below:

How many elements? 5

Enter the array elements: 12 3 6 1 8

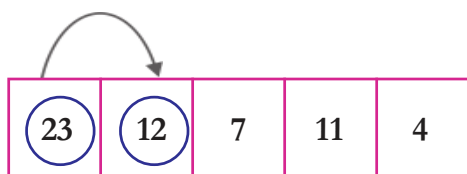
Sorted array is: 1 3 6 8 12

**b. Bubble sort**

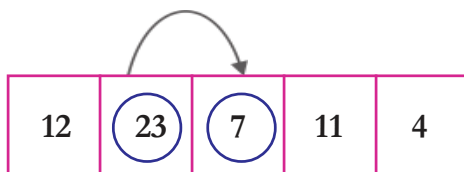
Bubble sort is a sorting algorithm that works by repeatedly stepping through lists that need to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. This passing procedure is repeated until no swaps are required, indicating that the list is sorted. Bubble sort gets its name because larger element bubbles towards the top of the list. To see a specific example, examine the list of numbers.

**Initial list**

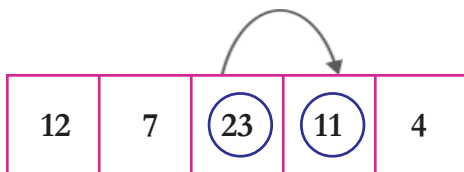
|    |    |   |    |   |
|----|----|---|----|---|
| 23 | 12 | 7 | 11 | 4 |
|----|----|---|----|---|

**Pass 1**

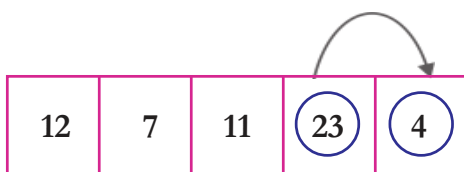
The first comparison results in exchanging the first two elements, 23 and 12.



The second comparison results in exchanging the second and third elements 23 and 7 in the revised list.



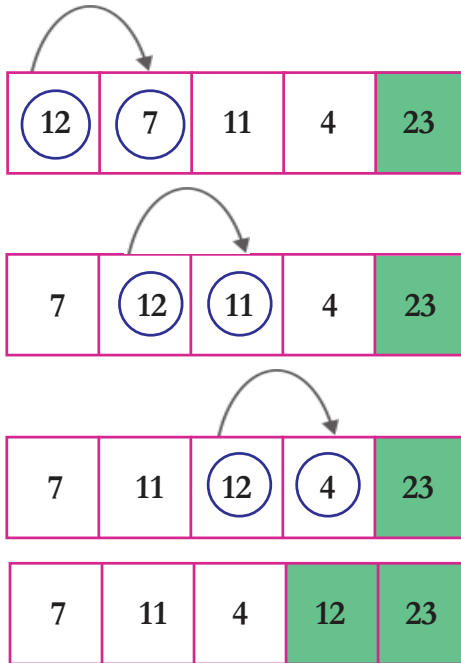
The third comparison results in exchanging the third and fourth elements 23 and 11 in the revised list.



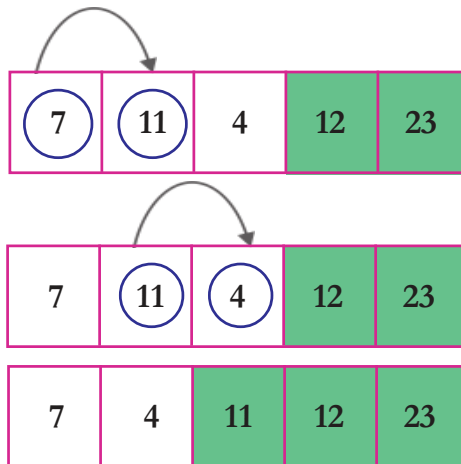
The fourth comparison results in exchanging the fourth and fifth elements 23 and 4 in the revised list.



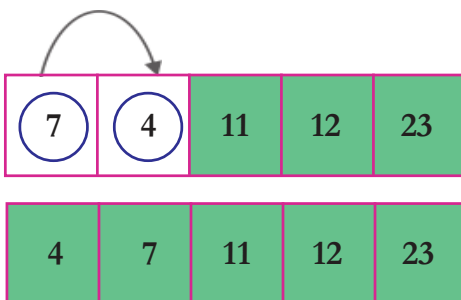
After the end of the first pass, the largest element 23 is bubbled to the last position of the list.

**Pass 2**

In the second pass, we consider only the four elements of the list, excluding 23. The same process is continued as in Pass 1 and as a result of it 12 is bubbled to fourth position, which is the second largest element in the list.

**Pass 3**

In the third pass, we consider only the three elements of the list, excluding 23 and 12. The same process is continued as in the above pass and as a result of it '11' is bubbled to the 3rd position.

**Pass 4**

In the last pass we consider only the two elements of the list, excluding 23, 12 and 11. The same process is continued as in the above pass and as a result of it 7 is bubbled to 2nd position and eventually 4 is placed in the first position.





Now we get the sorted list of elements. In bubble sort, to sort a list of 'N' elements we require (N-1) passes. In each pass the size of the revised list will be reduced by one.

### Algorithm for bubble sort

- Step 1.** Start
- Step 2.** Accept a value in N as the number of elements of the array
- Step 3.** Accept N elements into the array AR
- Step 4.** Repeat Steps 5 to 7, (N - 1) times
- Step 5.** Repeat Step 6 until the second last element of the list
- Step 6.** Starting from the first position, compare two adjacent elements in the list. If they are not in proper order, swap the elements.
- Step 7.** Revise the list by excluding the last element in the current list.
- Step 8.** Print the sorted array AR
- Step 9.** Stop

Program 8.4 uses **AR** as an integer array to store maximum of 25 numbers, **N** as the number of elements to be sorted.

#### Program 8.4: Bubble sort for arranging elements in ascending order

```
#include <iostream>
using namespace std;
int main()
{
    int AR[25], N;
    int I, J, TEMP;
    cout<<"How many elements? ";
    cin>>N;
    cout<<"Enter the array elements: ";
    for(I=0; I<N; I++)
        cin>>AR[I];
    for(I=1; I<N; I++)
        for(J=0; J<N-I; J++)
            if(AR[J] > AR[J+1])
            {
                TEMP = AR[J];
                AR[J] = AR[J+1];
                AR[J+1] = TEMP;
            }
    cout<<"Sorted array is: ";
    for(I=0; I<N; I++)
        cout<<AR[I]<<"\t";
}
```



The following is a sample output of Program 8.4.

How many elements? 5

Enter the array elements: 23 10 -3 7 11

Sorted array is: -3 7 10 11 23

### 8.2.3 Searching

Searching is the process of finding the location of the given element in the array. The search is said to be successful if the given element is found, that is the element exists in the array; otherwise unsuccessful. There are basically two approaches to search operation: linear search and binary Search.

The algorithm that one chooses generally depends on organization of the array elements. If the elements are in random order, the linear search technique is used, and if the array elements are sorted, it is preferable to use the binary search technique. These two search techniques are described below:

#### a. Linear search

Linear search or sequential search is a method for finding a particular value in a list. Linear search consists of checking each element in the list, one at a time in sequence starting from the first element, until the desired one is found or the end of the list is reached.

Assume that the element '45' is to be searched from a sequence of elements 50, 18, 48, 35, 45, 26, 12. Linear search starts from the first element 50, comparing each element until it reaches the 5th position where it finds 45 as shown in Figure 8.3.

| Index | List | Comparison              |
|-------|------|-------------------------|
| 0     | 50   | 50 == 45 : <b>False</b> |
| 1     | 18   | 18 == 45 : <b>False</b> |
| 2     | 48   | 48 == 45 : <b>False</b> |
| 3     | 35   | 35 == 45 : <b>False</b> |
| 4     | 45   | 45 == 45 : <b>True</b>  |
| 5     | 26   |                         |
| 6     | 12   |                         |

*Fig. 8.3: Linear search*

#### Algorithm for Linear Search

- Step 1.** Start
- Step 2.** Accept a value in N as the number of elements of the array
- Step 3.** Accept N elements into the array AR



- Step 4.** Accept the value to be searched in the variable ITEM
- Step 5.** Set LOC = -1
- Step 6.** Starting from the first position, repeat Step 7 until the last element
- Step 7.** Check whether the value in ITEM is found in the current position. If found then store the position in LOC and Go to Step 8, else move to the next position.
- Step 8.** If the value of LOC is less than 0 then display "Not Found", else display the value of LOC + 1 as the position of the search value.
- Step 9.** Stop

Program 8.5 uses **AR** as an integer array to store maximum of 25 numbers, **N** as the number of elements in the array, **ITEM** as the element to be searched and **LOC** as the position or index of the search element.

#### Program 8.5: Linear search to find an item in the array

```
#include <iostream>
using namespace std;
int main()
{
    int AR[25], N;
    int I, ITEM, LOC=-1;
    cout<<"How many elements? ";
    cin>>N;
    cout<<"Enter the array elements: ";
    for(I=0; I<n; I++)
        cin>>AR[I];
    cout<<"Enter the item you are searching for: ";
    cin>>ITEM;
    for(I=0; I<N; I++)
        if(AR[I] == ITEM)
        {
            LOC=I;
            break;
        }
    if(LOC!=-1)
        cout<<"The item is found at position "<<LOC+1;
    else
        cout<<"The item is not found in the array";
    return 0;
}
```



A sample output of program 8.5 is given below:

```
How many Elements? 7
Enter the array elements: 12 18 26 35 45 48 50
Enter the item you are searching for: 35
The item is found at position 4
```

The following output shows the other side:

```
How many Elements? 7
Enter the array elements: 12 18 26 35 45 48 50
Enter the item you are searching for: 25
The item is not found in the array
```

As noted previously, an advantage of the linear search method is that the list need not be in sorted order to perform the search operation. If the search item is towards the front of the list, only a few comparisons are enough. The worst case occurs when the search item is at the end of the list. For example, for a list of 10,000 elements, maximum number of comparisons needed is 10,000.

### **b. Binary search**

The linear search algorithm which we have seen is the most simple and convenient for small arrays. But when the array is large and requires many repeated searches, it makes good sense to have a more efficient algorithm. If the array contains a sorted list then we can use a more efficient search algorithm called Binary Search which can reduce the search time.

For example, suppose you want to find the meaning of the term 'modem' in a dictionary. Obviously, we don't search page by page. We open the dictionary in the middle (roughly) to determine which half contains the term being sought. Then for subsequent search one half is discarded and we search in the other half. This process is continued till we locate the required term or it results in unsuccessful search. The second case concludes that the term is not found in the dictionary. This search method is possible in a dictionary because the words are in sorted order.

Binary search is an algorithm which uses minimum number of searches for locating the position of an element in a sorted list, by checking the middle, eliminating half of the list from consideration, and then performing the search on the remaining half. If the middle element is equal to the searched value, then the position has been found; otherwise the upper half or lower half is chosen for search, based on whether the element is greater than or less than the middle element.



### Algorithm for binary search

- Step 1.** Start
- Step 2.** Accept a value in MAX as the number of elements of the array
- Step 3.** Accept MAX elements into the array LIST
- Step 4.** Accept the value to be searched in the variable ITEM
- Step 5.** Store the position of the first element of the list in FIRST and that of the last in LAST
- Step 6.** Repeat Steps 7 to 11 While (FIRST <= LAST)
- Step 7.** Find the middle position using the formula  $(FIRST + LAST)/2$  and store it in MIDDLE
- Step 8.** Compare the search value in ITEM with the element at the MIDDLE of the list
- Step 9.** If the MIDDLE element contains the search value in ITEM then stop search, display the position and go to Step 12.
- Step 10.** If the search value is smaller than the MIDDLE element  
Then set  $LAST = MIDDLE - 1$
- Step 11.** If the search value is larger than the MIDDLE element  
Then set  $FIRST = MIDDLE + 1$
- Step 12.** Stop

In Program 8.6, **LIST** is used as an integer array to store maximum of 25 numbers, **MAX** as the number of elements in the array, **ITEM** as the element to be searched and **LOC** as the position number or index of the search element. **FIRST**, **LAST** and **MIDDLE** are used to refer the first, last and middle positions respectively of the list under consideration.

#### Program 8.6: Binary search to find an item in the sorted array

```
#include <iostream>
using namespace std;
int main()
{
    int LIST[25], MAX;
    int FIRST, LAST, MIDDLE, I, ITEM, LOC=-1;
    cout<<"How many elements? ";
    cin>>MAX;
    cout<<"Enter array elements in ascending order: ";
    for(I=0; I<MAX; I++)
        cin>>LIST[I];
```



```

cout<<"Enter the item to be searched: ";
cin>>ITEM;
FIRST=0;
LAST=MAX-1;
while (FIRST<=LAST)
{
    MIDDLE=(FIRST+LAST)/2;
    if (ITEM == LIST[MIDDLE])
    {
        LOC = MIDDLE;
        break;
    }
    if (ITEM < LIST[MIDDLE])
        LAST = MIDDLE-1;
    else
        FIRST = MIDDLE+1;
}
if (LOC != -1)
    cout<<"The item is found at position "<<LOC+1;
else
    cout<<"The item is not found in the array";
return 0;
}

```

The following is a sample output of Program 8.6

How many elements? 7

Enter array elements in ascending order: 21 28 33 35 45 58 61

Enter the item to be searched: 35

The item is found at position 4

Let us consider the following sorted array with 7 elements to illustrate the working of the binary search technique. Assume that the element to be searched is 45.

| 0  | 1  | 2  | 3  | 4  | 5  | 6  |                                     |
|----|----|----|----|----|----|----|-------------------------------------|
| 21 | 28 | 33 | 35 | 45 | 58 | 61 | <b>FIRST = 0</b><br><b>LAST = 6</b> |

As **FIRST** ≤ **LAST**, let's start iteration

| 0  | 1  | 2  | 3  | 4  | 5  | 6  |                                                                                                                                                                                                                          |
|----|----|----|----|----|----|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 21 | 28 | 33 | 35 | 45 | 58 | 61 | <b>MIDDLE = (FIRST+LAST)/2 = (0+6)/2 = 3</b><br>Here <b>LIST[3]</b> is not equal to <b>45</b> and<br><b>LIST[3]</b> is less than search element<br>therefore, we take<br><b>FIRST = MIDDLE + 1 = 3 + 1 = 4, LAST = 6</b> |

As **FIRST** ≤ **LAST**, we start next iteration.

|    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  |
| 21 | 28 | 33 | 35 | 45 | 58 | 61 |

**MIDDLE** = (**FIRST**+**LAST**)/2 = (4+6)/2 = 5  
 Here **LIST**[5] is not equal to **45** and **LIST**[5] is greater than the search element therefore, we take **FIRST** = 4, **LAST** = **MIDDLE** - 1 = 5 - 1 = 4,

As **FIRST** ≤ **LAST**, we start next iteration

|    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  |
| 21 | 28 | 33 | 35 | 45 | 58 | 61 |

**MIDDLE** = (**FIRST**+**LAST**)/2 = (4+4)/2 = 4  
 Here **LIST**[4] is equal to **45** and the search terminates successfully.

In Binary search, an array of 100,00,00,000 (hundred crores) elements requires a maximum of only 30 comparisons to search an element. If the number of elements in the array is doubled, only one more comparison is needed.

Table 8.1 shows how linear search method differs from binary search:

| Linear search method                                                                                                                                                                                                   | Binary search method                                                                                                                                                                                                         |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>The elements need not be in any order</li> <li>Takes more time for the process</li> <li>May need to visit all the elements</li> <li>Suitable when the array is small</li> </ul> | <ul style="list-style-type: none"> <li>The elements should be in sorted order</li> <li>Takes very less time for the process</li> <li>All the elements are never visited</li> <li>Suitable when the array is large</li> </ul> |

Table 8.1: Comparison of linear and binary search methods

## 8.3 Two dimensional (2D) arrays

Suppose we have to store marks of 50 students in six different subjects. Here we can use six single dimensional arrays with 50 elements each. But managing these data with this arrangement is not an easy task. In this situation we can use an array of arrays or two dimensional arrays.

A two dimensional array is an array in which each element itself is an array. For instance, an array **AR**[**m**][**n**] is a 2D array, which contains **m** single dimensional arrays, each of which has **n** elements. Otherwise we can say that **AR**[**m**][**n**] is a table containing **m** rows and **n** columns.

### 8.3.1 Declaring 2D arrays

The general form of a two dimensional array declaration in C++ is as follows :

```
data_type array_name[rows][columns];
```

where **data\_type** is any valid data type of C++ and elements of this 2D array will be of this type. The **rows** refers to the number of rows in the array and **columns**





refers to the number of columns in the array. The indices (subscripts) of rows and columns, start at 0 and ends at (rows-1) and (columns-1) respectively. The following declaration declares an array named `marks` of size  $5 \times 4$  (5 rows and 4 columns) of type integer.

```
int marks[5][4];
```

The elements of this array are referred to as `marks[0][0]`, `marks[0][1]`, `marks[0][2]`, `marks[0][3]`, `marks[1][0]`, `marks[1][1]`, ..., `marks[4][3]` as shown in Figure 8.4.

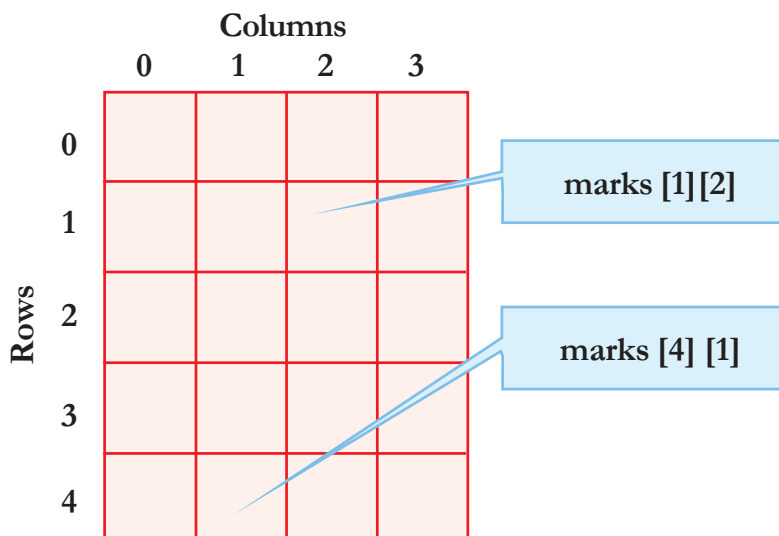


Fig. 8.4: Structure of a 2D array

The amount of storage required to hold a two dimensional array depends upon its base type, number of rows and number of columns. The formula to calculate total number of bytes required for a 2D array is as follows:

$$\text{total\_bytes} = \text{sizeof}(\text{base type}) \times \text{number of rows} \times \text{number of columns}$$

For instance, the above declared array `marks[5][4]` requires  $4 \times 5 \times 4 = 80$  bytes.

### 8.3.2 Matrices as 2D arrays

Matrix is a useful concept in mathematics. We know that a matrix is a set of  $m \times n$  numbers arranged in the form of a table with  $m$  rows and  $n$  columns. Matrices can be represented through 2D arrays. The following program illustrates some operations on matrices. To process a 2D array, you need to use nested loops. One loop processes the rows and other the columns. Normally outer loop is for rows and inner loop is for columns. Program 8.7 creates a matrix `mat` with  $m$  rows and  $n$  columns.

**Program 8.7: To create a matrix with m rows and n columns**

```
#include <iostream>
using namespace std;
int main()
{   int m, n, row, col, mat[10][10];
    cout<< "Enter the order of matrix: ";
    cin>> m >> n;
    cout<<"Enter the elements of matrix\n";
    for (row=0; row<m; row++)
        for (col=0; col<n; col++)
            cin>>mat[row][col];
    cout<<"The given matrix is:";
    for (row=0; row<m; row++)
    {
        cout<<endl;
        for (col=0; col<n; col++)
            cout<<mat[row][col]<<"\t";
    }
    return 0;
}
```

Creation of the matrix

Display the elements in matrix format

A sample output of Program 8.7 is given below:

```
Enter the order of matrix: 3 4
Enter the elements of matrix
1 2 3 4 2 3 4 5 3
The given matrix is:
1 2 3 4
2 3 4 5
3 4 5 6
```

Note that the elements of the matrix are entered sequentially but the output is given with the specified matrix format.

Let us see a program that accepts the order and elements of two matrices and displays their sum. Two matrices can be added only if their order is the same. The elements of the sum matrix are obtained by adding the corresponding elements of the operand matrices. If A and B are two operand matrices, each element in the sum matrix C will be of the form  $C[i][j] = A[i][j] + B[i][j]$ , where i indicates the row position and j the column position.

**Program 8.8: To find the sum of two matrices if conformable**

```

#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
    int m1, n1, m2, n2, row, col;
    int A[10][10], B[10][10], C[10][10];
    cout<<"Enter the order of first matrix: ";
    cin>>m1>>n1;
    cout<<"Enter the order of second matrix: ";
    cin>>m2>>n2;
    if(m1!=m2 || n1!=n2)
    {
        cout<<"Addition is not possible";
        exit(0);
    }
    cout<<"Enter the elements of first matrix\n";
    for (row=0; row<m1; row++)
        for (col=0; col<n1; col++)
            cin>>A[row][col];
    cout<<"Enter the elements of second matrix\n";
    for (row=0; row<m2; row++)
        for (col=0; col<n2; col++)
            cin>>B[row][col];
    for (row=0; row<m1; row++)
        for (col=0; col<n1; col++)
            C[row][col] = A[row][col] + B[row][col];
    cout<<"Sum of the matrices:\n";
    for(row=0; row<m1; row++)
    {
        cout<<endl;
        for (col=0; col<n1; col++)
            cout<<C[row][col]<<"\t";
    }
}

```

To use exit() function

Program terminates

Creation of first matrix

Creation of second matrix

Matrix addition process. Instead of m1 and n1, we can use m2 and n2.

A sample output of Program 8.8 is given below:

```

Enter the order of first matrix: 3    4
Enter the order of second matrix: 3    4
Enter the elements of first matrix
2    5    -3    7
5    12    4    9
-3    0    6    -5

```

Here the elements are entered in matrix form. But it is not essential



Enter the elements of second matrix

1      4      3      5

4      -5      7      13

3      -4      7      9

Sum of the matrices:

3      9      0      12

9      7      11      22

0      -4      13      4

The subtraction operation on matrices can be performed in the same fashion as in Program 8.8 except that the formula is  $C[i][j] = A[i][j] - B[i][j]$ .

Now, let us write a program to find the sum of the diagonal elements of a square matrix. A matrix is said to be a square matrix, if the number of rows and columns are the same. Though there are two diagonals for a square, here we mean the elements  $mat[0][0], mat[1][1], mat[2][2], \dots, mat[n-1][n-1]$ , where **mat** is the 2D array. These diagonal elements are called leading or major diagonal elements. Program 8.9 can be used to find the sum of the major diagonal elements.

#### Program 8.9: To find the sum of major diagonal elements of a matrix

```
#include <iostream>
using namespace std;
int main()
{
    int mat[10][10], n, i, j, s=0;
    cout<<"Enter the rows/columns of square matrix: ";
    cin>>n;
    cout<<"Enter the elements\n";
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            cin>>mat[i][j];
    cout<<"Major diagonal elements are\n";
    for(i=0; i<n; i++)
    {
        cout<<mat[i][i]<<"\t";
        s = s + mat[i][i];
    }
    cout<<"\nSum of major diagonal elements is: ";
    cout<<s;
    return 0;
}
```

Accesses only the  
diagonal elements to  
find the sum

When Program 8.9 is executed the following output is obtained:

```
Enter the rows/columns of square matrix: 3
Enter the elements
3      5      -2
7      4      0
2      8      -1
Major diagonal elements are
3      4      -1
Sum of major diagonal elements is: 6
```

Each matrix has a transpose. It is obtained by converting row elements into column elements or vice versa. Program 8.10 shows this process.

#### Program 8.10: To find the transpose of a matrix

```
#include <iostream>
using namespace std;
int main()
{
    int ar[10][10], m, n, row, col;
    cout<<"Enter the order of matrix: ";
    cin>>m>>n;
    cout<<"Enter the elements\n";
    for(row=0; row<m; row++)
        for(col=0; col<n; col++)
            cin>>ar[row][col];
    cout<<"Original matrix is\n";
    for(row=0; row<m; row++)
    {
        cout<<"\n";
        for(col=0; col<n; col++)
            cout<<ar[row][col]<<"\t";
    }
    cout<<"\nTranspose of the entered matrix is\n";
    for(row=0; row<n; row++)
    {
        cout<<"\n";
        for(col=0; col<m; col++)
            cout<<ar[col][row]<<"\t";
    }
    return 0;
}
```

Note that the positions of row size and column size are changed in loops

Subscripts also changed their positions



A sample output of Program 8.10 is given below:

```
Enter the order of matrix: 4      3
Enter the elements
3      5      -1
2      12     0
6      8      4
7      -5     6
Original matrix is
3      5      -1
2      12     0
6      8      4
7      -5     6
Transpose of the entered matrix is
3      2      6      7
5      12     8      -5
-1     0      4      6
```

These elements can  
be entered in a  
single line

When data is arranged in tabular form, in some situations, we may need sum of elements of each row as well as each column. Program 8.11 helps the computer to perform this task.

#### Program 8.11: To find the row sum and column sum of a matrix

```
#include <iostream>
using namespace std;
int main()
{
    int ar[10][10], rsum[10]={0}, csum[10]={0};
    int m, n, row, col;
    cout<<"Enter the number of rows & columns in the array: ";
    cin>>m>>n;
    cout<<"Enter the elements\n";
    for(row=0; row<m; row++)
        for(col=0; col<n; col++)
            cin>>ar[row][col];
    for(row=0; row<m; row++)
        for(col=0; col<n; col++)
        {
            rsum[row] += ar[row][col];
            csum[col] += ar[row][col];
        }
    cout<<"Row sum of the 2D array is\n";
```

Row elements  
and column elements are  
added separately and each  
sum is stored in respective  
locations of the arrays  
concerned



```
for(row=0; row<m; row++)
    cout<<rsum[row]<<"\t";
cout<<"\nColumn sum of the 2D array is\n";
for(col=0; col<n; col++)
    cout<<csum[col]<<"\t";
return 0;
}
```

A sample output of Program 8.11 is given below:

```
Enter the number of rows & columns in the array: 3    4
Enter the elements
3      12      5      0
4      -6      2      1
5      7       -6      2
Row sum of the 2D array is
20     1       8
Column sum of 2D array is
12     13      1      3
```

## 8.4 Multi-dimensional arrays

Each element of a 2D array may be another array. Such an array is called 3D (Three Dimensional) array. Its declaration is as follows:

```
data_type array_name[size_1][size_2][size_3];
```

The elements of a 3D array are accessed using three subscripts. If `ar[10][5][3]` is declared as a 3D array in C++, the first element is referenced by `ar[0][0][0]` and the last element by `ar[9][4][2]`. This array can contain 150 ( $10 \times 5 \times 3$ ) elements. Similarly more sizes can be specified while declaring arrays so that multi-dimensional arrays are formed.



### Let us sum up

Array is a collection of elements placed in contiguous memory locations identified by a common name. Each element in an array is referenced by specifying its subscript with the array name. Array elements are accessed easily with the help of `for` loop. Operations like traversing, sorting and searching are performed on arrays. Bubble sort and selection sort methods are used to sort the elements. Linear search and binary search techniques are applied to search an element in an array. Two dimensional arrays are used to solve matrix related problems. We need two subscripts to refer to an element of 2D array. Besides 2D arrays, it is possible to create multidimensional arrays in C++.





## Learning outcomes

After the completion of this chapter learner will be able to

- identify scenarios where an array can be used.
- declare and initialize single dimensional and 2D arrays.
- develop logic to perform various operations on arrays like sorting and searching.
- solve matrix related problems with the help of 2D arrays.



## Lab activity

1. Write a C++ program to input the amount of sales for 12 months into an array named `SalesAmt`. After all the input, find the total and average amount of sales.
2. Write a C++ program to create an array of `N` numbers, find the average and display those numbers greater than the average.
3. Write a C++ program that specifies three one-dimensional arrays named `price`, `quantity` and `amount`. Each array should be capable of holding 10 elements. Use a `for` loop to input values to the arrays `price` and `quantity`. The entries in the array `amount` should be the product of the corresponding values in the arrays `price` and `quantity` (i.e.  $\text{amount}[i] = \text{price}[i] \times \text{quantity}[i]$ ). After all the data has been entered, display the following output, with the corresponding value under each column heading as follows:

| Price | Quantity | Amount |
|-------|----------|--------|
| _____ | _____    | _____  |
| _____ | _____    | _____  |

4. Write a C++ program to input 10 integer numbers into an array and determine the maximum and minimum values among them.
5. Write a C++ program which reads a square matrix of order `n` and prints the upper triangular elements. For example, if the matrix is

```

2   3   1
7   1   5
2   5   1

```

The output should be

```

2   3   1
   1   5
       1

```

6. Write a program which reads a square matrix of order **n** and prints the lower triangular elements. For instance, if the matrix is

```

2   3   1
7   1   5
2   5   7

```

The output will be

```

2
7   1
2   5   7

```

7. Write a C++ program to find the sum of leading diagonal elements of a matrix.  
 8. Write a C++ program to find the sum of off diagonal elements of a matrix.  
 9. Write a program to print the Pascal's triangle as shown below:

```

1
1   2   1
1   3   3   1
1   4   6   4   1

```

### Sample questions

#### Very short answer type

- All the elements in an array must be \_\_\_\_\_ data type.
- The elements of an array with ten elements are numbered from \_\_\_\_\_ to \_\_\_\_\_.
- An array element is accessed using \_\_\_\_\_.
- If AR is an array, which element will be referenced using AR[7]?
- Consider the array declaration `int a[3]={2,3,4};` What is the value of `a[1]`?
- Consider the array declaration `int a[ ]={1,2,4};` What is the value of `a[1]`?
- Consider the array declaration `int a[5]={1,2,4};` What is the value of `a[4]`?
- Write array initialization statements for an array of 6 scores: 89, 75, 82, 93, 78, 95.
- Printing all the elements of an array is an example for \_\_\_\_\_ operation.
- How many bytes are required to store the array `int a[2][3];`?
- In an array of *m* elements, Binary search required maximum of *n* search for finding an element. How many search required if the number of elements is doubled?



12. Write the statement to initialize an array of 10 marks with value 0.
13. State True or false: The compiler will complain if you try to access array element 16 in a ten-element array.

### Short answer type

1. Define an Array.
2. What does the declaration `int studlist[1000];` mean?
3. How is memory allocated for a single dimensional array?
4. Write C++ statements to accept an array of 10 elements and display the count of even and odd numbers in it.
5. Write the initialization statement for an array num with values 2, 3, 4, 5.
6. What is meant by traversal?
7. Define sorting.
8. What is searching?
9. What is bubble sort?
10. What is binary search?
11. Define a 2D array.
12. How is memory allocated for a 2D array?

### Long answer type

1. An array AR contains the elements 25, 81, 36, 15, 45, 58, 70. Illustrate the working of binary search technique for searching an element 45.
2. Write C++ statements to accept two single dimensional array of equal length and find the difference between corresponding elements.
3. Illustrate the working of bubble sort method for sorting the elements 32, 25, 44, 16, 37, 12.
4. If 24, 45, 98, 56, 76, 24, 15 are the elements of an array, illustrate the working of selection sort for sorting.
5. Write a program to find the difference between two matrices.
6. Write a program to find the sum and average of elements in a 2D array.
7. Write a program to find the largest element in a 2D array.

## Key Concepts

- String handling using arrays
- Memory allocation for strings
- Input/output operations on strings
- Console functions for Character I/O
  - `getchar()`
  - `putchar()`
- Stream functions for I/O operations
  - Input functions - `get()`, `getline()`
  - Output functions - `put()`, `write()`

## String Handling and I/O Functions

We have learnt that array is the effective mechanism to handle large number of homogeneous type of data. Most of the programs that we discussed are used to process numeric type data. We know that there are string type data also. In this chapter we will see how string data are stored and processed. Also we will discuss some built-in functions to perform input/output operations on string and character data.

### 9.1 String handling using arrays

We know that string is a kind of literal in C++ language. It appears in programs as a sequence of characters within a pair of double quotes. Imagine that you are asked to write a program to store your name and display it. You have learned that variables are required to store data. Let us take the identifier `my_name` as the variable. Remember that in C++, a variable is to be declared before it is used. A declaration statement is required for this and it begins with a data type. Which data type should be used to declare a variable to hold string data? There is no basic data type to represent string data. You may think of **char** data type. But note that the variable declared using `char` can hold only one character. Here we have to input string which is a sequence of characters.



Let us consider a name “Niketh”. It is a string consisting of six characters. So it cannot be stored in a variable of `char` type. But we know that an array of `char` type can hold more than one character. So, we declare an array as follows:

```
char my_name[10];
```

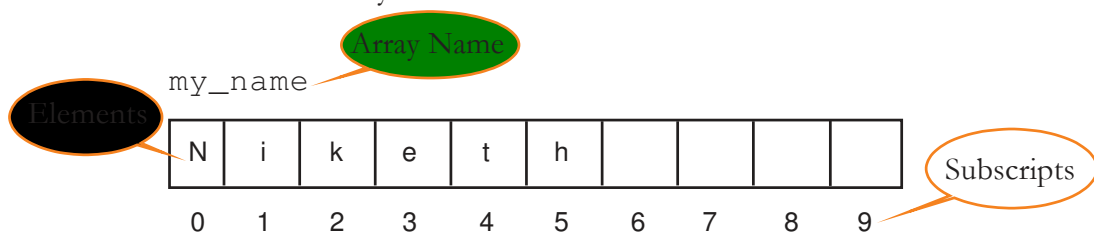
It is sure that ten contiguous locations, each with one byte size, will be allocated for the array named `my_name`. If we follow the usual array initialization method, we can store the characters in the string “Niketh” as follows:

```
char my_name[10]={'N', 'i', 'k', 'e', 't', 'h'};
```

Figure 9.1 shows the memory allocation for the above declared character array. Note that we store the letters in the string, separated by commas. If we want to input the same data, the following C++ statement can be used:

```
for (int i=0; i<6; i++)
    cin >> my_name[i];
```

During the execution of this statement, we have to input six letters of “Niketh” one after the other separated by **Space bar**, **Tab** or **Enter** key. The memory allocation in both of these cases may be shown as follows:



*Fig. 9.1 : Memory allocation for the character array*

So, let us conclude that a character array can be used to store a string, since it is a sequence of characters. However, it is true that we do not get the feel of inputting a string. Instead, we input the characters constituting the string one by one.

In C++, character arrays have some privileges over other arrays. Once we declare a character array, the array name can be considered as an ordinary variable that can hold string data. Let's say that a character array name is equivalent to a string variable. Thus your name can be stored in the variable `my_name` (the array name) using the following statement:

```
cin >> my_name;
```

It is important to note that this kind of usage is wrong in the case of arrays of other data types. Now let us complete the program. It will be like the one given in Program 9.1.

**Program 9.1: To input a string and display**

```
#include <iostream>
using namespace std;
int main()
{
    char my_name[10];
    cout << "Enter your name: ";
    cin >> my_name;
    cout << "Hello " << my_name;
}
```

On executing this program you will get the output as shown below.

```
Enter your name: Niketh
Hello Niketh
```

Note that the string constant is not "Hello", but "Hello " (a white space is given after the letter o).

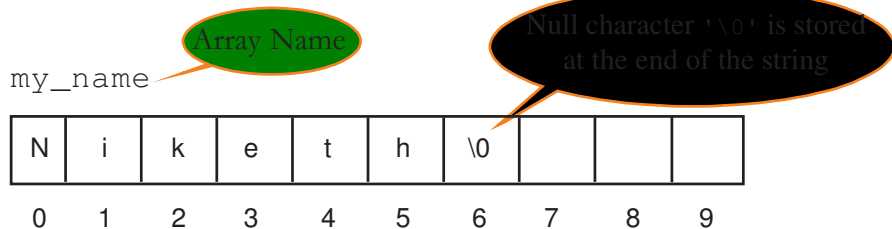


**Let us do**

*Run Program 9.1 and input your full name by expanding the initials if any, and check whether the output is correct or not. If your name contains more than 10 characters, increase the size of the array as needed.*

**9.2 Memory allocation for strings**

We have seen how memory is allocated for an array of characters. As Figure 9.1 shows, the memory required depends upon the number of characters stored. But if we input a string in a character array, the scene will be different. If we run Program 9.1 and input the string *Niketh*, the memory allocation will be as shown in Figure 9.2.



*Fig. 9.2 : Memory allocation for the character array*

Note that a null character '`\0`' is stored at the end of the string. This character is used as the string terminator and added at the end automatically. Thus we can say that memory required to store a string will be equal to the number of characters in the string plus one byte for null character. In the above case, the memory used to store the string *Niketh* is seven bytes, but the number of characters in the string is only six.



As in the case of variable initialization, we can initialize a character array with a string as follows:

```
char my_name[10] = "Niketh";
char str[] = "Hello World";
```

In the first statement 10 memory locations will be allocated and the string will be stored with null character as the delimiter. The last three bytes will be left unused. But for the second statement, size of the array is not specified and hence only 12 bytes will be allocated (11 bytes for the string and 1 for '\0').

### 9.3 Input/Output operations on strings

Program 9.1 contains input and output statements for string data. Let us modify the declaration statement by changing the size of the array to 20. If we run the program by entering the name Maya Mohan, the output will be as follows:

```
Enter your name: Maya Mohan
Hello Maya
```

Note that though there is enough size for the array, we get only the word "Maya" as the output. Why does this happen?

Let us have a close look at the input statement: `cin>>my_name;`. We have experienced that only one data item can be input using this statement. A white space is treated as a separator of data. Thus, the input Maya Mohan is treated as two data items, Maya and Mohan separated by white space. Since there is only one input operator (`>>`) followed by a variable, the first data (i.e., Maya) is stored. The white space after "Maya" is treated as the delimiter.

So, the problem is that we are unable to input strings containing white spaces. C++ language gives a solution to this problem by a function, named **gets()**. The function `gets()` is a console input function used to accept a string of characters including white spaces from the standard input device (keyboard) and store it in a character array.

The string variable (character array name) should be provided to this function as shown below:

```
gets(character_array_name);
```

When we use this function, we have to include the library file `stdio.h` in the program. Let us modify Program 9.1, by including the statement `#include<cstdio>`, and replacing the statement `cin>>my_name;` by `gets(my_name);`. After executing the modified program, the output will be as follows:

```
Enter your name: Maya Mohan
Hello Maya Mohan
```





The output shows the entire string that we input. See the difference between `gets()` and `cin`.

Though we are not using the concept of subscripted variable for the input and output of strings, any element in the array can be accessed by specifying its subscript along with the array name. We can access the first character of the string by `my_name[0]`, fifth character by `my_name[4]` and so on. We can even access the null character (`'\0'`) by its subscript. The following program illustrates this idea.

### Program 9.2: To input a string and count the vowels in a string

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    char str[20];
    int vow=0;
    cout<<"Enter a string: ";
    gets(str);
    for(int i=0; str[i]!='\0'; i++)
        switch(str[i])
        {
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u': vow++;
        }
    cout<<"No. of vowels in the string "<<str<<" is "<<vow;
    return 0;
}
```

Header file for using `gets()` function

Condition will be true as long as the null character is not retrieved

Each character in the array will be compared with the constants for matching

If we run Program 9.2 by inputting the string “hello guys”, the following output can be seen:

Enter a string: hello guys

No. of vowels in the string hello guys is 3

Now, let us analyse the program and see how it works to give this output.

- In the beginning, the `gets()` function is used and so we can input the string “hello guys”.
- The body of the `for` loop will be executed as long as the element in the array, referenced by the subscript `i`, is not the null character (`'\0'`). That is, the body of the loop will be executed till the null character is referenced.



- The body of the loop contains only a `switch` statement. Note that, no statements are given against the first four cases of the `switch`. In the last case, the variable `vow` is incremented by 1. You may think that this is required for all the cases. Yes, you are right. But you should use the `break` statement for each case to exit the `switch` after a match. In this program the action for all the cases are the same and that is why we use this style of code.
- While the `for` loop iterates, the characters will be retrieved one by one for matching against the constants attached to the cases. Whenever a match is found, the variable `vow` is incremented by 1.
- As per the input string, matches occur when the value of `i` becomes 1, 4 and 7. Thus, the variable `vow` is incremented by 1 three times and we get the correct output.

We have seen how `gets()` function facilitates input of strings. Just like the other side of a coin, C++ gives a console function named **`puts()`** to output string data. The function `puts()` is a console output function used to display a string data on the standard output device (monitor). Its syntax is:

```
puts(string_data);
```

The string constant or variable (character array name) to be displayed should be provided to this function. Observe the following C++ code fragment:

```
char str[10] = "friends";
puts("hello");
puts(str);
```

The output of the above code will be as follows:

```
hello
friends
```

Note that the string "friends" in the character array `str[10]` is displayed in a new line. Try this code using `cout<<"hello";` and `cout<<str;` instead of the `puts()` functions and see the difference. The output will be in the same line without a space in between them in the case of `cout` statement.



**Let us do**

*Predict the output, if the input is "HELLO GUYS" in Program 9.2. Execute the program with this input and check whether you get correct output. Find out the reason for difference in output. Modify the program to get the correct output for any given string.*



## 9.4 Console functions for character I/O

We have discussed functions for input/output operations on strings. C++ also provides some functions for performing input/output operations on characters. These functions require the inclusion of header file **cstdio** (`stdio.h` in Turbo C++ IDE) in the program.

### **getchar ()**

This function returns the character that is input through the keyboard. The character can be stored in a variable as shown in the example given below:

```
char ch = getchar();
```

We have seen `puts ()` function and its advantage in string output. Let us have a look at a function used to output character data.

### **putchar ()**

This function displays the character given as the argument on the standard output unit (monitor). The argument may be a character constant or a variable. If an integer value is given as the argument, it will be considered as an ASCII value and the corresponding character will be displayed. The following code segment illustrates the use of `putchar ()` function.

```
char ch = 'B';    //assigns 'B' to the variable ch
putchar(ch);     //displays 'B' on the screen
putchar('c');    //displays 'c' on the screen
putchar(97);     //displays 'a' on the screen
```

Program 9.3 illustrates the working of these functions. This program allows inputting a string and a character to be searched. It displays the number of occurrences of a character in the string.

#### **Program 9.3: To search for a character in a string using console functions**

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    char str[20], ch;
    int i, num=0;
    puts("Enter a string:"); //To print '\n' after the string
    gets(str); //To accept a string with white spaces
```



```

cout<<"Enter the character to be searched: ";
ch=getchar();//To input the character to be searched
/* A loop to search for the character and count its
   occurrences in the string. Search will be
   terminated when a null character is found */
for(i=0; str[i]!='\0'; i++)
    if (str[i]==ch)
        num++;
cout<<"The string \"<str<<\" uses the character \"<";
putchar(ch);
cout<<\" \"<<num<<\" times\";
return 0;
}

```

Program 9.3 uses all the console functions we have discussed. The following is a sample output of this program:

Enter a string:

I have a Dream

Enter the character to be searched: a

The string 'I have a Dream' uses the character 'a' 3 times

### Check yourself



1. Which character is used to delimit the string in memory?
2. Write the statement to declare a variable for storing "Save Earth".
3. Name the header file required for using console I/O functions.
4. How many bytes are required to store the string "Be Positive"?
5. How does `puts("hello");` differ from `cout<<"hello";`?

## 9.5 Stream functions for I/O operations

C++ provides another facility to perform input/output operations on character and strings. It is in the form of functions that are available in the header file **iostream**. These functions are generally called stream functions since they allow a stream of bytes (data) to flow between memory and objects. Devices like the keyboard and the monitor are referenced as objects in C++. Let us discuss some of these functions.

### A. Input functions

These functions allow the input of character and string data. The input functions such as **get()** and **getline()** allow a stream of bytes to flow from input object into the memory. The object **cin** is used to refer to keyboard and hence whenever



we input data using keyboard, these functions are called or invoked using this object as **cin.get()** and **cin.getline()**. Note that a period symbol (.), called *dot operator* is used between the object **cin** and the function.

### i. get ()

It can accept a single character or multiple characters (string) through the keyboard. To accept a string, an array name and size are to be given as arguments. Following code segment illustrates the usage of this function.

```
char ch, str[10];
ch=cin.get(ch); //accepts a character and stores in ch
cin.get(ch);    //equivalent to the above statement
cin.get(str,10); //accepts a string of maximum 10 characters
```

### ii. getline ()

It accepts a string through the keyboard. The delimiter will be Enter key, the number of characters or a specified character. This function uses two syntaxes as shown in the code segment given below.

```
char ch, str[10];
int len;
cin.getline(str,len);           // With 2 arguments
cin.getline(str,len,ch);       // With 3 arguments
```

In the first usage, **getline()** function has two arguments - a character array (here it is, **str**) and an integer (**len**) that represents maximum number of characters that can be stored. In the second usage, a delimiting character (content of **ch**) can also be given along with the number of characters. While inputting the string, only (**len-1**) characters, or characters upto the specified delimiting character, whichever occurs first will be stored in the array.

## B. Output functions

Output functions like **put()** and **write()** allow a stream of bytes to flow from memory into an output object. The object **cout** is used with these functions since we use the monitor for the output.

### i. put ()

It is used to display a character constant or the content of a character variable given as argument.

```
char ch='c';
cout.put(ch);           //character 'c' is displayed
cout.put('B');          //character 'B' is printed
cout.put(65);           //character 'A' is printed
```

**ii. write()**

This function displays the string contained in the argument. For illustration see the example given below.

```
char str[10]="hello";
cout.write(str,10);
```

The above code segment will display the string `hello` followed by 5 white spaces, since the second argument is 10 and the number of characters in the string is 5.

**Program 9.4: To illustrate the working of stream input/output functions**

```
#include <iostream>
#include <cstring> //To use strlen() function
using namespace std;
int main()
{
    char ch, str[20];
    cout<<"Enter a character: ";
    cin.get(ch); //To input a character to the variable ch
    cout<<"Enter a string: ";
    cin.getline(str,10, '.'); //To input the string
    cout<<"Entered character is:\t";
    cout.put(ch); //To display the character
    cout.write("\nEntered string is:",20);
    cout.write(str,strlen(str));
    return 0;
}
```

On executing Program 9.4, the following output will be obtained:

```
Enter a character: p
Enter a string: hello world
Entered character is:      p
Entered string is:
hello wo
```

Let us discuss what happens when the program is executed. In the beginning, `get()` function allows to input a character, say `p`. When the function `getline()` is executed, we can input a string, say `hello world`. The `put()` function is then executed to display the character `p`. Observe that the `write()` function displays only `hello wo` in a new line. In the `getline()` function, we specified the integer 10 as the maximum number of characters to be stored in the array `str`. Usually 9 characters will be stored, since one byte is reserved for `'\0'` character as the string



terminator. But the output shows only 8 characters including white space. This is because, the Enter key followed by the character input (p) for the `get()` function, is stored as the `'\n'` character in the first location of `str`. That is why, the string, `hello wo` is displayed in a new line.

If we run the program, by giving the input `hello.world`, the output will be as follows: Observe the change in the content of `str`.

```
Enter a character: a
Enter a string: hello.world
Entered character is:      a
Entered string is:
hello
```

Note the dot character (.)

The change has occurred because the `getline()` function accepts only the characters that appear before the dot symbol.



Be careful while using these functions, because pressing of any key does matter a lot in the input operation. So you may not get the outputs as you desire. Another point you have to notice is that, `strlen()` function is used in the `write()` function. Instead of using this function, you can provide a number like 10 or 20. But the output will be the string you input, followed by some ASCII characters, if the number of characters is less than the given number. When you use `strlen()`, you are actually specifying the exact number of characters in the string. More about this function will be discussed in chapter 10. You can use this function only if you include **cstring** file.



*The following table compares the stream functions. But it is not complete. Fill up the table and check whether your entries are correct by comparing with that of your friends.*

#### Let us do

| Comparison Aspect    | Console Functions                                                           | Stream Functions                                                                           |
|----------------------|-----------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| Header file required | .....                                                                       | .....                                                                                      |
| Usage format         | Mention the function name with required data or variable within parentheses | Use object name followed by dot operator and function name with required data or variable. |
| Device reference     | Keyboard or monitor is not mentioned                                        | .....                                                                                      |
| Examples             | .....                                                                       | .....                                                                                      |





## Check yourself



1. Name the stream function to input a character data.
2. Write a C++ statement to display the string "Smoking is injurious to health" using `write()` function.
3. Name the header file required for using stream I/O functions.
4. Write down the syntax of `getline()` function.



## Let us sum up

Array of characters is used to handle strings in C++ programs. While allocating memory for string, a null character ('`\0`') is placed as the delimiter. Different console functions are available to perform input/output operations on strings. These functions are available in the header file `cstdio`. The header file `iostream` provides some stream function for the input and output of strings.



## Learning outcomes

After the completion of this chapter the learner will be able to

- use character arrays for string handling.
- use various built-in functions for I/O operations on character and string data.
- compare console functions and stream functions.



## Lab activity

1. Write a program to input a string and find the number of uppercase letters, lowercase letters, digits, special characters and white spaces.
2. Write a program to count the number of words in a sentence.
3. Write a program to input a string and replace all lowercase vowels by the corresponding uppercase letters.
4. Write a program to input a string and display its reversed string using console I/O functions only. For example, if the input is "AND", the output should be "DNA".



5. Write a program to input a word (say COMPUTER) and create a triangle as follows:
 

```

C
C   O
C   O   M
C   O   M   P
C   O   M   P   U
C   O   M   P   U   T
C   O   M   P   U   T   E
C   O   M   P   U   T   E   R
      
```
6. Write a program to input a line of text and display the first characters of each word. Use only console I/O functions. For example, if the input is "Save Water, Save Nature", the output should be "SWSN"
7. Write a program to check whether a string is palindrome or not. (A string is said to be plaindrome if it is the same as the string constituted by reversing the characters of the original string. eg. "MALAYALAM")

**Sample questions****Very short answer type**

1. What will the output of the statement: `putchar (97) ;` be?
2. Distinguish between console functions and stream functions.
3. Write a C++ statement to input the string "Computer" using `get ( )` function.
4. Write down the output of the following code segment:

```
puts("hello");
puts("friends");
```

**Short answer type**

1. Predict the output of the following code segment:

```
char str[] = "Program";
for (int i=0; str[i] != '\0'; ++i)
{
    putchar(str[i]);
    putchar('-');
}
```



2. Identify the errors in the following program and give reason for each.

```
#include<iostream>
using namespace std;
int main()
{
    char ch, str[10];
    write("Enter a character");
    ch=getchar();
    puts("Enter a string");
    cin.getline(str);
    cout<< "The data entered are " <<ch;
    putchar(str);
}
```

3. Observe the following functions. If the statement is valid, explain what happens when they are executed. If invalid, state reasons for it. (*Assume that the variables are valid*)

(a) `getchar(ch);`                      (b) `gets(str[5]);`  
 (c) `putchar("hello");`      (d) `cin.getline(name, 20, ',');`  
 (e) `cout.write("hello world", 10);`

4. Read the following statements:

```
char name[20];
cin>>name;
cout<<name;
```

What will be the output if you input the string "Sachin Tendulkar"? Justify your answer. Modify the code to get the entered string as the output.

### Long answer type

1. Explain the console I/O functions with examples.
2. Briefly describe the stream I/O functions with examples.

## Key Concepts

- **Concept of modular programming**
- **Functions in C++**
- **Predefined functions**
  - String functions
  - Mathematical functions
  - Character functions
  - Conversion functions
  - I/O manipulating functions
- **User-defined functions**
  - Creating user-defined functions
  - Prototype of functions
  - Arguments of functions
  - Functions with default arguments
  - Methods of calling functions
- **Scope and life of variables and functions**
- **Recursive functions**
- **Creation of header Files**

## Functions

We discussed some simple programs in the previous chapters. But to solve complex problems, larger programs running into thousands of lines of code are required. As discussed in Chapter 4, complex problems are divided into smaller ones and programs to solve each of these sub problems are written. In other words, we break the larger programs into smaller sub programs. In C++, function is a way to divide large programs into smaller sub programs. We have already seen functions like `main()`, `sqrt()`, `gets()`, `getchar()`, etc. The functions, except `main()`, are assigned specific tasks and readily available for use. So, these functions are known as built-in functions or predefined functions. Besides such functions, we can define functions for a specific task. These are called user-defined functions. In this chapter we will discuss more predefined functions and learn how to define our own functions. Before going into these, let us familiarise ourselves with a style of programming called modular programming.

### 10.1 Concept of modular programming

Let us consider the case of a school management software. It is a very large and complex software which may contain many programs for different tasks. The complex task of school management can be divided into smaller tasks or modules and developed in parallel, and later integrated to build the complete software as shown in Figure 10.1.



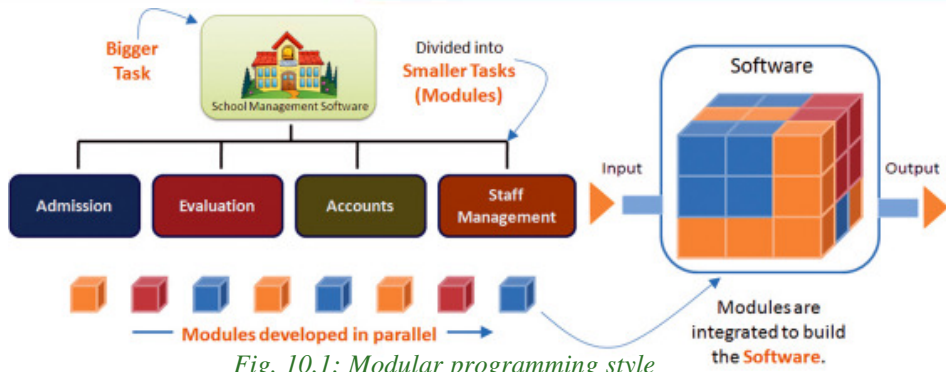


Fig. 10.1: Modular programming style

In programming, the entire problem will be divided into small sub problems that can be solved by writing separate programs. This kind of approach is known as modular programming. Each sub task will be considered a module and we write programs for each module. The process of breaking large programs into smaller sub programs is called **modularisation**. Computer programming languages have different methods to implement modularization. The sub programs are generally called functions. C++ also facilitates modular programming with functions.

### Merits of modular programming

The modular style of programming has several advantages. It reduces the complexity and size of the program, makes the program more readable, improves re-usability and makes the debugging process easy. Let us discuss these features in detail:

**Reduces the size of the program:** In some cases, certain instructions in a program may be repeated at different points of the program. Consider the expression

$\frac{x^5 + y^7}{\sqrt{x} + \sqrt{y}}$ . To evaluate this expression for the given values of x and y, we have to

use instructions for the following:

1. find the 5<sup>th</sup> power of x
2. find the 7<sup>th</sup> power of y
3. add the results obtained in steps 1 and 2
4. find the square root of x
5. find the square root of y
6. add the results obtained in steps 4 and 5
7. divide the result obtained in step 3 by that in step 6

We know that separate loops are needed to find the results of step 1 and 2. Can you imagine the complexity of the logic to find the square root of a number? It is clear that the program requires the same instructions to process different data at different

points. The modular approach helps to isolate the repeating task and write instructions for this. We can assign a name to this set of instructions and this can be invoked by using that name. Thus program size is reduced.

**Less chances of error:** When the size of the program is reduced, naturally syntax errors will be less in number. The chances of logical error will also be minimized. While solving complex problems we have to consider all the aspects of the problem, and hence the logic of the solution will also be complex. But in a modularized program, we need to concentrate only on one module at a time. If any error is identified in the output, we can identify the module concerned and rectify the error in that module only.

**Reduces programming complexity:** The net result of the two advantages discovered above is reducing programming complexity. If we properly divide the problem into smaller conceptual units, the development of logic for the solution will be simpler. Thus modularization reduces the programming complexity by bringing down our mind to a simplified task at a time, reducing the program size and making the debugging process easy.

**Improves reusability:** A function written once may be used later in many other programs, instead of starting from scratch. This reduces the time taken for program development.

### Demerits of modular programming

Though there are significant merits in modular programming, proper breaking down of the problem is a challenging task. Each sub problem must be independent of the others. Utmost care should be taken while setting the hierarchy of the execution of the modules.

## 10.2 Functions in C++

Let us consider the case of a coffee making machine and discuss its functioning based on Figure 10.2. Water, milk, sugar and coffee powder are supplied to the machine. The machine processes it according to a set of predefined instructions stored in it and returns the coffee which is collected in a cup. The instruction-set may be as follows:

1. Get 60 ml milk, 120 ml water, 5 gm coffee powder and 20 gm sugar from the storage of the machine.
2. Boil the mixture
3. Pass it to the outlet.



Fig. 10.2 : Functioning of a coffee making machine





Usually there will be a button in the machine to invoke this procedure. Let us name the button with the word “MakeCoffee”. Symbolically we can represent the invocation as:

**Cup = MakeCoffee (Water, Milk, Sugar, Coffee Powder)**

We can compare all these aspects with functions in programs. The word “MakeCoffee” is the **name** of the function, “Water”, “milk”, “sugar” and “coffee powder” are **parameters** for the function and “coffee” is the result **returned**. It is **stored** in a “Cup”. Instead of cup, we can use a glass, tumbler or any other container.

Similarly, a C++ function accepts parameters, processes it and returns the result. Figure 10.3 can be viewed as a function **Add** that accepts 3, 5, 2 and 6 as parameters, adds them and returns the value which is stored in the variable **C**. It can be written as:

**C = Add (3, 5, 2, 6)**

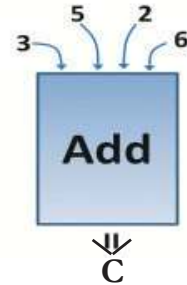


Fig. 10.3. Addition function

We can say that **function** is a named unit of statements in a program to perform a specific task as part of the solution. It is not necessary that all the functions require some parameters and all of them return some value. C++ provides a rich collection of functions ready to use for various tasks. The functions `clrscr()`, `getch()`, `sqrt()`, etc. are some of them. The tasks to be performed by each of these are already written, debugged and compiled, their definitions alone are grouped and stored in files called header files. Such ready-to-use sub programs are called **predefined functions** or **built-in functions**.

While writing large programs, the predefined functions may not suffice to apply modularization. C++ provides the facility to create our own functions for some specific tasks. Everything related to a function such as the task to be carried out, the name and data required are decided by the user and hence they are known as **user-defined functions**.

What is the role of **main()** function then? It may be considered as user-defined in the sense that the task will be defined by the user. We have learnt that it is an essential function in a C++ program because the execution of a program begins in `main()`. Without `main()`, C++ program will not run. All other functions will be executed when they are called or invoked (or used) in a statement.

### 10.3 Predefined functions

C++ provides a number of functions for various tasks. We will discuss only the most commonly used functions. While using these functions, some of them require data for performing the task assigned to it. We call them **parameters** or **arguments**





and are provided within the pair of parentheses of the function name. There are certain functions which give results after performing the task. This result is known as **value returned** by the function. Some functions do not return any value, rather they perform the specified task. In the following sections, we discuss functions for manipulating strings, performing mathematical operations and processing character data. While using these functions the concerned header files are to be included in the program.

### 10.3.1 String Functions

Several string functions are available in C++ for the manipulation of strings. As discussed in Chapter 9, C++ does not have a string data type and hence an array of characters is used to handle strings. So, in the following discussion, wherever the word string comes, assume that it is a character array. Following are the commonly used string functions. We should include the header file **cstring** (`string.h` in Turbo C++) in our C++ program to use these functions.

#### a. **strlen()**

This function is used to find the length of a string. Length of a string means the number of characters in the string. Its syntax is:

```
int strlen(string);
```

This function takes a string as the argument and gives the length of the string as the result. The following code segment illustrates this.

```
char str[] = "Welcome";  
int n;  
n = strlen(str);  
cout << n;
```

Here, the argument for the `strlen()` function is a string variable and it returns the number of characters in the string, i.e. 7 to the variable `n`. Hence the program code will display 7 as the value of the variable `n`. The output will be the same even though the array declaration is as follows.

```
char str[10] = "Welcome";
```

Note that the array size is specified in the declaration. The argument may be a string constant as shown below:

```
n = strlen("Computer");
```

The above statement returns 8 and it will be stored in `n`.

#### b. **strcpy()**

This function is used to copy one string into another. The syntax of the function is:

```
strcpy(string1, string2);
```



The function will copy `string2` to `string1`. Here `string1` and `string2` are array of characters or string constants. These are the arguments for the execution of the function. The following code illustrates its working:

```
char s1[10], s2[10] = "Welcome";
strcpy(s1, s2);
cout << s1;
```

The string "Welcome" contained in the string variable `s1` will be displayed on the screen. The second argument may be a string constant as follows:

```
strcpy(str, "Welcome");
```

Here, the string constant "Welcome" will be stored in the variable `str`. The assignment statement, `str = "Welcome";` is wrong. But we can directly assign value to a character array at the time of declaration as:

```
char str[10] = "Welcome";
```

### c. `strcat()`

This function is used to append one string to another string. The length of the resultant string is the total length of the two strings. The syntax of the functions is:

```
strcat(string1, string2);
```

Here `string1` and `string2` are array of characters or string constants. `string2` is appended to `string1`. So, the size of the first argument should be able to accommodate both the strings together. Let us see an example showing the usage of this function:

```
char s1[20] = "Welcome", s2[10] = " to C++";
strcat(s1, s2);
cout << s1;
```

The above program code will display "Welcome to C++" as the value of the variable `s1`. Note that the string in `s2` begins with a white space.

### d. `strcmp()`

This function is used to compare two strings. In this comparison, the alphabetical order of characters in the strings is considered. The syntax of the function is:

```
strcmp(string1, string2)
```

The function returns any of the following values in three different situations.

- Returns 0 if `string1` and `string2` are same.
- Returns a -ve value if `string1` is alphabetically lower than `string2`.
- Returns a +ve value if `string1` is alphabetically higher than `string2`.

The following code fragment shows the working of this function.

```
char s1[]="Deepthi", s2[]="Divya";
int n;
n = strcmp(s1,s2);
if(n==0)
    cout<<"Both the strings are same";
else if(n < 0)
    cout<<"s1 < s2";
else
    cout<<"s1 > s2";
```

It is clear that the above program code will display "s1 < s2" as the output.

### e. strcmpi()

This function is used to compare two strings ignoring cases. That is, the function will treat both the upper case and lower case letters as the same for comparison. The syntax and working of the function are the same as that of strcmp() except that strcmpi() is not case sensitive. This function also returns values as in the case of strcmp(). Consider the following code segment:

```
char s1[]="SANIL", s2[]="sanil";
int n;
n = strcmpi(s1,s2);
if(n==0)
    cout<<"strings are same";
else if(n < 0)
    cout<<"s1 < s2";
else
    cout<<"s1 > s2";
```

The above program code will display "strings are same" as the output because the uppercase and lowercase letters will be treated as the same during the comparison.

Program 10.1 compares and concatenates two strings. The length of the newly formed string is also displayed.

#### Program 10.1: To combine two strings if they are different and find its length

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char s1[20], s2[20], s3[20];
    cout<<"Enter two strings: ";
```

Header file essential  
for using string  
manipulating functions



```

cin>>s1>>s2;
int n=strcmp(s1, s2);
if (n==0)
    cout<<"\nThe input strings are same";
else
{
    cout<<"\nThe input strings are not same";
    strcpy(s3, s1); //Copies the string in s1 into s3
    strcat(s3, s2); //Appends the string in s2 to that in s3
    cout<<"\nString after concatenation is: "<<s3;
    cout<<"\nLength of the new string is: "<<strlen(s3);
}
return 0;
}

```

### 10.3.2 Mathematical functions

Now, let us discuss the commonly used mathematical functions available in C++. We should include the header file **cmath** (`math.h` in Turbo C++) to use these functions in the program.

#### a. **abs()**

It is used to find the absolute value of an integer. It takes an integer as the argument (+ve or -ve) and returns the absolute value. Its syntax is:

```
int abs(int)
```

The following is an example to show the output of this function:

```
int n = -25;
cout << abs(n);
```

The above program code will display 25. If we want to find the absolute value of a floating point number, we can use **fabs()** function as used above. It will return the floating point value.

#### b. **sqrt()**

It is used to find the square root of a number. The argument to this function can be of type `int`, `float` or `double`. The function returns the non-negative square root of the argument. Its syntax is:

```
double sqrt(double)
```

The following code snippet is an example. This code will display 5.

```
int n = 25;
float b = sqrt(n);
cout << b;
```

**c. pow()**

This function is used to find the power of a number. It takes two arguments **x** and **y**. The argument **x** and **y** are of type `int`, `float` or `double`. The function returns the value of **x<sup>y</sup>**. Its syntax is:

```
double pow(double, int)
```

The following example shows the working of this function.

```
int x = 5, y = 4, z;  
z = pow(x, y);  
cout << z;
```

The above program code will display 625.

**d. sin()**

It is a trigonometric function and it finds the **sine** value of an angle. The argument to this function is `double` type data and it returns the sine value of the argument. The angle must be given in radian measure. Its syntax is:

```
double sin(double)
```

The sine value of  $\angle 30^\circ$  (*angle 30 degree*) can be found out by the following code.

```
float x = 30*3.14/180; //To convert angle into radians  
cout << sin(x);
```

The above program code will display 0.4999770

**e. cos()**

The function is used to find the cosine value of an angle. The argument to this function is `double` type data and it returns the cosine value of the argument. In this case also, the angle must be given in radian measure. The syntax is:

```
double cos(double)
```

The cosine value of  $\angle 30^\circ$  (*angle 30 degree*) can be found out by the following code.

```
double x = cos(30*3.14/180);  
cout << x;
```

Note that the argument provided is an expression that converts angle in degree into radians. The above code will display 0.866158.

Program 10.2 uses different mathematical functions to find the length of the sides of a right angled triangle, if an angle and length of one side is given. We use the following formula for solving this problem.

$$\sin \theta = \frac{\text{Opposite side}}{\text{Hypotenuse}}, \quad \cos \theta = \frac{\text{Adjacent side}}{\text{Hypotenuse}}, \quad \tan \theta = \frac{\text{Opposite side}}{\text{Adjacent side}}$$

$$(\text{Hypotenuse})^2 = (\text{Base})^2 + (\text{Altitude})^2$$



### Program 10.2: To find the length of the sides of a right angled triangle using mathematical functions

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    const float pi=22.0/7;
    int angle, side;
    float radians, length, opp_side, adj_side, hyp;
    cout<<"Enter the angle in degree: ";
    cin>>angle;
    radians=angle*pi/180;
    cout <<"\n1. Opposite Side"
        <<"\n2. Adjacent Side"
        <<"\n3. Hypotenuse";
    cout <<"\nInput 1, 2 or 3 to specify the side: ";
    cin>>side;
    cout<<"Enter the length: ";
    cin>>length;
    switch(side)
    {
        case 1: opp_side=length;
                adj_side=opp_side / tan(radians);
                hyp= sqrt(pow(opp_side,2) + pow(adj_side,2));
                break;
        case 2: adj_side=length;
                hyp=adj_side / cos(radians);
                opp_side=sqrt(pow(hyp,2) - pow(adj_side,2));
                break;
        case 3: hyp=length;
                opp_side=hyp * sin(radians);
                adj_side=sqrt(pow(hyp,2) - pow(opp_side,2));
    }
    cout<<"Angle in degree = "<<angle;
    cout<<"\nOpposite Side = "<<opp_side;
    cout<<"\nAdjacent Side = "<<adj_side;
    cout<<"\nHypotenuse      = "<<hyp;
    return 0;
}
```

Header file essential for using mathematical functions

Conversion of angle in degree into radians



The following is a sample output of the above program:

```
Enter the angle in degree: 30
1. Opposite Side
2. Adjacent Side
3. Hypotenuse
Input 1, 2 or 3 to specify the side: 1
Enter the length: 6
Angle in degree = 30
Opposite Side   = 6
Adjacent Side   = 10.38725
Hypotenuse      = 11.995623
```

### 10.3.3 Character functions

These functions are used to perform various operations of characters. The following are the various character functions available in C++. The header file **cctype** (`ctype.h` for Turbo C++) is to be included to use these functions in a program.

#### a. **isupper()**

This function is used to check whether a character is in the upper case or not. The syntax of the function is:

```
int isupper(char c)
```

The function returns 1 if the given character is in the uppercase, and 0 otherwise.

The following statement assigns 0 to the variable n.

```
int n = isupper('x');
```

Consider the following statements:

```
char c = 'A';
int n = isupper(c);
```

The value of the variable n, after the execution of the above statements will be 1, since the given character is in upper case.

#### b. **islower()**

This function is used to check whether a character is in the lower case or not. The syntax of the function is:

```
int islower(char c)
```

The function returns 1 if the given character is in the lower case, and 0 otherwise.





After executing the following statements, the value of the variable `n` will be 1 since the given character is in the lower case.

```
char ch = 'x';  
int n = islower(ch);
```

But the statement given below assigns 0 to the variable `n`, since the given character is in the uppercase.

```
int n = islower('A');
```

### c. `isalpha()`

This function is used to check whether the given character is an alphabet or not. The syntax of the function is:

```
int isalpha(char c)
```

The function returns 1 if the given character is an alphabet, and 0 otherwise.

The following statement assigns 0 to the variable `n`, since the given character is not an alphabet.

```
int n = isalpha('3');
```

But the statement given below displays 1, since the given character is an alphabet.

```
cout << isalpha('a');
```

### d. `isdigit()`

This function is used to check whether the given character is a digit or not. The syntax of the function is:

```
int isdigit(char c)
```

The function returns 1 if the given character is a digit, and 0 otherwise.

After executing the following statement, the value of the variable `n` will be 1 since the given character is a digit.

```
n = isdigit('3');
```

When the following statements are executed, the value of the variable `n` will be 0, since the given character is not a digit.

```
char c = 'b';  
int n = isdigit(c);
```

### e. `isalnum()`

This function is used to check whether a character is an alphanumeric or not. The syntax of the function is:

```
int isalnum (char c)
```



The function returns 1 if the given character is an alphanumeric, and 0 otherwise.

Each of the following statements returns 1 after the execution.

```
n = isalnum('3');
cout << isalnum('A');
```

But the statements given below assigns 0 to the variable n, since the given character is neither an alphabet nor a digit.

```
char c = '-';
int n = isalnum(c);
```

### f. toupper()

This function is used to convert the given character into its uppercase. The syntax of the function is:

```
char toupper(char c)
```

The function returns the upper case of the given character. If the given character is in the upper case, the output will be the same.

The following statement assigns the character constant 'A' to the variable c.

```
char c = toupper('a');
```

But the output of the statement given below will be 'A' itself.

```
cout << (char)toupper('A');
```

Note that type conversion using (char) is used in this statement. If conversion method is not used, the output will be 65, which is the ASCII code of 'A'.

### g. tolower()

This function is used to convert the given character into its lower case. The syntax of the function is:

```
char tolower(char c)
```

The function returns the lower case of the given character. If the given character is in the lowercase, the output will be the same.

Consider the statement: `c = tolower('A');`

After executing the above statement, the value of the variable c will be 'a'. But when the following statements are executed, the value of the variable c will be 'a'.

```
char x = 'a';
char c = tolower(x);
```

In the case of functions tolower() and toupper(), if the argument is other than an alphabet, the given character itself will be returned on execution.



Program 10.3 illustrates the use of character functions. This program accepts a line of text and counts the lowercase letters, uppercase letters and digits in the string. It also displays the entire string both in the upper and lower cases.

**Program 10.3: To count different types of characters in the given string**

```
#include <iostream>
#include <stdio>
#include <cctype>
using namespace std;
int main()
{
    char text[80];
    int Ucase=0, Lcase=0, Digit=0, i;
    cout << "Enter a line of text: ";
    gets(text);
    for(i=0; text[i]!='\0'; i++)
        if (isupper(text[i])) Ucase++;
        else if (islower(text[i])) Lcase++;
        else if (isdigit(text[i])) Digit++;
    cout << "\nNo. of uppercase letters = " << Ucase;
    cout << "\nNo. of lowercase letters = " << Lcase;
    cout << "\nNo. of digits = " << Digit;
    cout << "\nThe string in uppercase form is\n";
    i=0;
    while (text[i]!='\0')
    {
        putchar(toupper(text[i]));
        i++;
    }
    cout << "\nThe string in lowercase form is\n";
    i=0;
    do
    {
        putchar(tolower(text[i]));
        i++;
    } while(text[i]!='\0');
    return 0;
}
```

Loop will be terminated when the value of `i` points to the null character is reached

If `cout<<` is used instead of `putchar()`, the ASCII code of the characters will be displayed



A sample output is given below:

```
Enter a line of text : The vehicle ID is KL01 AB101
No. of uppercase letters = 7
No. of lowercase letters = 11
No. of digits = 5
The string in uppercase form is
THE VEHICLE ID IS KL01 AB101
The string in lowercase form is
the vehicle id is kl01 ab101
```

The input by  
the user

### 10.3.4 Conversion functions

These functions are used to convert a string to integer and an integer to string. Following are the different conversion functions available in C++. The header file **cstdlib** (**stdlib.h** in Turbo C++) is to be included to use these functions in a program.

#### a. itoa()

This function is used to convert an integer value to string type. The syntax of the function is:

```
itoa(int n, char c[], int len)
```

From the syntax, we can see that the function requires three arguments. The first one is the number to be converted. The second argument is the character array where the converted string value is to be stored and the last argument is the size of the character array. The following code segment illustrates this function:

```
int n = 2345;
char c[10];
itoa(n, c, 10);
cout << c;
```

The above program code will display "2345" on the screen.

#### b. atoi()

This function is used to convert a string value to integer. The syntax of the function is:

```
int atoi(char c[]);
```

The function takes a string as argument returns the integer value of the string. The following code converts the string "2345" into integer number 2345.

```
int n;
char c[10] = "2345";
n = atoi(c);
cout << n;
```



If the string consists of characters other than digits, the output will be 0. But if the string begins with digits, only that part will be converted into integer. Some usages of this function and their outputs are given below:

- (i) `atoi("Computer")` returns 0
- (ii) `atoi("12.56")` returns 12
- (iii) `atoi("a2b")` returns 0
- (iv) `atoi("2ab")` returns 2
- (v) `atoi(".25")` returns 0
- (vi) `atoi("5+3")` returns 5

Program 10.4 illustrates the use of these functions in problem solving. It accepts the three parts (day, month and year) of a date of birth and displays it in date format.

#### Program 10.4: To display date of birth in date format

```
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;
int main()
{
    char dd[10], mm[10], yy[10], dob[30];
    int d, m, y;
    cout<<"Enter day, month and year in your Date of Birth: ";
    cin>>d>>m>>y;
    itoa(d, dd, 10);
    itoa(m, mm, 10);
    itoa(y, yy, 10);
    strcpy(dob, dd);
    strcat(dob, "-");
    strcat(dob, mm);
    strcat(dob, "-");
    strcat(dob, yy);
    cout<<"Your Date of Birth is "<<dob;
    return 0;
}
```

A sample output of Program10.4 is given below:

```
Enter day, month and year in your Date of Birth: 26 11 1999
Your Date of Birth is 26-11-1999
```

### 10.3.5 I/O Manipulating function

These functions are used to manipulate the input and output operations in C++. The header file **iomanip** is to be included to use these functions in a program.

#### setw()

This function is used to set the width for the subsequent string. The following code shows its impact in the output:

```
char s[]="hello";
cout<<setw(10)<<s<<setw(10)<<"friends";
```

The output of the above code will be as follows:

```
hello    friends
```

The word `hello` will be displayed right aligned within a span of 10 character positions. Similarly, the word `friends` will also be displayed right aligned within a span of 10 character positions.



**Let us do**

*Prepare a chart in the following format and fill up the columns with all predefined functions we have discussed so far.*

| Function | Usage | Syntax | Example | Output |
|----------|-------|--------|---------|--------|
|          |       |        |         |        |

### Check yourself



1. What is modular programming?
2. What is a function in C++?
3. Name the header file required for using character functions.
4. Name the function that displays the subsequent data in the specified width.
5. Pick the odd one out and give reason:  
(a) `strlen()`   (b) `itoa()`   (c) `strcpy()`   (d) `strcat()`

## 10.4 User-defined functions

All the programs that we have discussed so far contain a function named **main()**. We know that the first line is a pre-processor directive statement. The remaining part is actually the definition of a function. The `void main()` in the programs is called **function header** (or function heading) of the function and the statements within the pair of braces immediately after the header is called its **body**.



The syntax of a function definition is given below:

```
data_type  function_name(argument_list)
{
    statements in the body;
}
```

The `data_type` is any valid data type of C++. The `function_name` is a user-defined word (identifier). The `argument_list`, which is optional, is a list of parameters, i.e. a list of variables preceded by data types and separated by commas. The body comprises C++ statements required to perform the task assigned to the function. Once we have decided to create a function, we have to answer certain questions.

- (i) Which data type will be used in the function header?
- (ii) How many arguments are required and what should be the preceding data type of each?

Let us recollect how we have used the predefined functions `getchar()`, `strcpy()` and `sqrt()`. We have seen that these functions will be executed when they are called (or used) in a C++ statement. The function `getchar()` does not take any argument. But for `strcpy()`, two strings are provided as arguments or parameters. Without these arguments this function will not work, because it is defined with two string (character array) arguments. In the case of `sqrt()`, it requires a numeric data as argument and gives a result (of double type) after performing the predefined operations on the given argument. This result, as mentioned earlier, is called the **return-value** of the function. The data type of a function depends on this value. In other words, we can say that the function should return a value which is of the same data type of the function. So, the data type of a function is also known as the **return type** of the function. Note that we use `return 0;` statement in `main()` function, since it is defined with `int` data type as per the requirement of GCC.

The number and type of arguments depend upon the data required by the function for processing. But some functions like `setw()` and `gets()` do not return any value. The header of such functions uses `void` as the return type. A function either returns one value or nothing at all.

### 10.4.1 Creating user-defined functions

Based on the syntax discussed above, let us create some functions. The following is a function to display a message.

```
void saywelcome()
{
    cout<<"Welcome to the world of functions";
}
```



The name of the function is `saywelcome()`, its data type (return type) is `void` and it does not have any argument. The body contains only one statement.

Now, let us define a function to find the sum of two numbers. Four different types of definitions are given for the same task, but they vary in definition style and hence the usage of each is different from others.

| <i>Function 1</i>                                                                                                                                       | <i>Function 2</i>                                                                                                                   |
|---------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <pre>void sum1() {     int a, b, s;     cout&lt;&lt;"Enter 2 numbers: ";     cin&gt;&gt;a&gt;&gt;b;     s=a+b;     cout&lt;&lt;"Sum="&lt;&lt;s; }</pre> | <pre>int sum2() {     int a, b, s;     cout&lt;&lt;"Enter 2 numbers: ";     cin&gt;&gt;a&gt;&gt;b;     s=a+b;     return s; }</pre> |
| <i>Function 3</i>                                                                                                                                       | <i>Function 4</i>                                                                                                                   |
| <pre>void sum3(int a, int b) {     int s;     s=a+b;     cout&lt;&lt;"Sum="&lt;&lt;s; }</pre>                                                           | <pre>int sum4(int a, int b) {     int s;     s=a+b;     return s; }</pre>                                                           |

Let us analyse these functions and see how they are different. The task is the same in all these functions, but they differ in the number of parameters and return type.

Table 10.1 shows that the function defined with a data type other than **void** should return a value in accordance with the data type. The **return** statement is used for this purpose (Refer functions 2 and 4). The **return** statement returns a value to the calling function and transfers the program control back to the calling function. So, remember that if a `return` statement is executed in a function, the remaining statements within that function will not be executed. In most of the functions,

| Name                | Arguments             | Return value              |
|---------------------|-----------------------|---------------------------|
| <code>sum1()</code> | No arguments          | Does not return any value |
| <code>sum2()</code> | No arguments          | Returns an integer value  |
| <code>sum3()</code> | Two integer arguments | Does not return any value |
| <code>sum4()</code> | Two integer arguments | Returns an integer value  |

Table 10.1 : Analysis of functions



return is placed at the end of the function. The functions defined with void data type may have a return statement within the body, but we cannot provide any value to it. The return type of main() function is either void or int.

Now, let us see how these functions are to be called and how they are executed. We know that no function other than main() is executed automatically. The sub functions, either predefined or user-defined, will be executed only when they are called from main() function or other user-defined function. The code segments within the rectangles of the following program shows the function calls. Here the main() is the calling function and sum1(), sum2(), sum3(), and sum4() are the called functions.

```
int main()
{
    int x, y, z=5, result;
    cout << "\nCalling the first function\n";
    sum1();
    cout << "\nCalling the second function\n";
    result = sum2();
    cout << "Sum given by function 2 is " << result;
    cout << "\nEnter values for x and y : ";
    cin >> x >> y;
    cout << "\nCalling the third function\n";
    sum3(x, y);
    cout << "\nCalling the fourth function\n";
    result = sum4(z, 12);
    cout << "Sum given by function 4 is " << result;
    cout << "\nEnd of main function"
}
```

The output of the program is as follows:

```
Calling the first function
Enter 2 numbers: 10  25
Sum=35
Calling the second function
Enter 2 numbers: 5   7
Sum given by function 2 is 12
Enter values for x and y : 8   13
Calling the third function
Sum=21
Calling the fourth function
Sum given by function 4 is 17
End of main function
```

Input for a and b  
of sum1()

Input for a and b  
of sum2()

Input for x and y  
of main()



Function 4 requires two numbers for the task assigned and hence we provide two arguments. The function performs some calculations and gives a result. As there is only one result, it can be returned. Comparatively this function is a better option to find the sum of any two numbers.

Now, let us write a complete C++ program to check whether a given number is perfect or not. A number is said to be perfect if it is equal to the sum of its factors other than the number itself. For example, 28 is a perfect number, because the factors other than 28 are 1, 2, 4, 7 and 14. Sum of these factors is 28. For solving this problem, let us define a function that accepts a number as argument and returns the sum of its factors. Using this function, we will write the program. But where do we write the user-defined function in a C++ program? The following table shows two styles to place the user-defined function:

**Program 10.5****- Perfect number checking -****Program 10.6**

| <i>Function before main()</i>                                                                                                                                                                                                                                                                                                                                                                     | <i>Function after main()</i>                                                                                                                                                                                                                                                                                                                                                                      |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> #include &lt;iostream&gt; using namespace std; int sumfact(int N) { int i, s = 0;   for(i=1; i&lt;=N/2; i++)     if (N%i == 0)       s = s + i;   return s; } //Definition above main() int main() {   int num;   cout&lt;&lt;"Enter the Number: ";   cin&gt;&gt;num;   if (num==sumfact(num))     cout&lt;&lt;"Perfect number";   else     cout&lt;&lt;"Not Perfect";   return 0; } </pre> | <pre> #include &lt;iostream&gt; using namespace std; int main() {   int num;   cout&lt;&lt;"Enter the Number: ";   cin&gt;&gt;num;   if (num==sumfact(num))     cout&lt;&lt;"Perfect number";   else     cout&lt;&lt;"Not Perfect";   return 0; } //Definition below main() int sumfact(int N) { int i, s = 0;   for(i=1; i&lt;=N/2; i++)     if (N%i == 0)       s = s + i;   return s; } </pre> |

When we compile Program 10.5, there will be no error and we will get the following output on execution:

```

Enter the Number: 28
Perfect number

```



If we compile Program 10.6, there will be an error 'sumfact was not declared in this scope'. Let us see what this error means.

### 10.4.2 Prototype of functions

We have seen that a C++ program can contain any number of functions. But it must have a `main()` function to begin the execution. We can write the definitions of functions in any order as we wish. We can define the `main()` function first and all other functions after that or vice versa. Program 10.5 contains the `main()` function after the user-defined function, but in Program 10.6, the `main()` is defined before the user-defined function. When we compile this program, it will give an error - "sumfact was not declared in this scope". This is because the function `sumfact()` is called in the program, before it is defined. During the compilation of the `main()` function, when the compiler encounters the function call `sumfact()`, it is not aware of such a function. Compiler is unable to check whether there is such a function and whether its usage is correct or not. So it reports an error arising out of the absence of the function prototype. A **function prototype** is the declaration of a function by which compiler is provided with the information about the function such as the name of the function, its return type, the number and type of arguments, and its accessibility. This information is essential for the compiler to verify the correctness of the function call in the program. This information is available in the function header and hence the header alone will be written as a statement before the function call. The following is the format:

```
data_type function_name(argument_list);
```

In the prototype, the argument names need not be specified.

So, the error in Program 10.6 can be rectified by inserting the following statement before the function call in the `main()` function.

```
int sumfact(int);
```

Like a variable declaration, a function must be declared before it is used in the program. If a function is defined before it is used in the program, there is no need to declare the function separately. The declaration statement may be given outside the `main()` function. The position of the prototype differs in the accessibility of the function. We will discuss this later in this chapter. Wherever be the position of the function definition, execution of the program begins in `main()`.



### 10.4.3 Arguments of functions

We have seen that functions have arguments or parameters for getting data for processing. Let us see the role of arguments in function call.

Consider the function given below:

```
float SimpleInterest(long P, int N, float R)
{
    float amt;
    amt = P * N * R / 100;
    return amt;
}
```

This function gives the simple interest of a given principal amount for a given period at a given rate of interest.

The following code segment illustrates different function calls:

```
cout << SimpleInterest(1000,3,2); //Function call 1
int x, y; float z=3.5, a;
cin >> x >> y;
a = SimpleInterest(x, y, z);      //Function call 2
```

When the first statement is executed, the values 1000, 3 and 2 are passed to the argument list in the function definition. The arguments *P*, *N* and *R* get the values 1000, 3 and 2 respectively. Similarly, when the last statement is executed, the values of the variables *x*, *y* and *z* are passed to the arguments *P*, *N* and *R*.

The variables *x*, *y* and *z* are called actual (original) arguments or actual parameters since they are the actual data passed to the function for processing. The variables *P*, *N* and *R* used in the function header are known as formal arguments or formal parameters. These arguments are intended to receive data from the calling function.

**Arguments or parameters** are the means to pass values from the calling function to the called function. The variables used in the function definition as arguments are known as **formal arguments**. The constants, variables or expressions used in the function call are known as **actual (original) arguments**. If variables are used in function prototype, they are known as dummy arguments.

Now, let us write a program that uses a function `fact ()` that returns the factorial of a given number to find the value of  $nCr$ . As we know, factorial of a number *N* is the product of the first *N* natural numbers. The value of  $nCr$  is calculated by the

formula  $\frac{n!}{r!(n-r)!}$  where  $n!$  denotes the factorial of *n*.

**Program 10.7: To find the value of  $nCr$** 

```

#include<iostream>
using namespace std;
int fact(int);
int main()
{
    int n,r;
    int ncr;
    cout<<"Enter the values of n and r : ";
    cin>>n>>r;
    ncr=fact(n)/((fact(r)*fact(n-r)));
    cout<<n<<"C"<<r<<" = "<<ncr;
    return 0;
}
int fact(int N)
{
    int f;
    for(f=1; N>0; N--)
        f=f*N;
    return f;
}

```

Function prototype

Function call according to the formula

Function header

Formal argument

Actual arguments

Factorial is returned

The following is a sample output:

```

Enter the values of n and r : 5    2
5C2 = 10

```

User inputs

**10.4.4 Functions with default arguments**

Let us consider a function `TimeSec()` with the argument list as follows. This function accepts three numbers that represent time in hours, minutes and seconds. The function converts this into seconds.

```

long TimeSec(int H, int M=0, int S=0)
{
    long sec = H * 3600 + M * 60 + S;
    return sec;
}

```

Note that the two arguments `M` and `S` are given default value 0. So, this function can be invoked in the following ways.

```

long s1 = TimeSec(2,10,40);
long s2 = TimeSec(1,30);
long s3 = TimeSec(3);

```



It is important to note that all the default arguments must be placed from the right to the left in the argument list. When a function is called, actual arguments are passed to the formal arguments from left onwards.

When the first statement is executed, the function is called by passing the values 2, 10 and 40 to the formal parameters H, M and S respectively. The initial values of M and S are over-written by the actual arguments. During the function call in the second statement, H and M get values from actual arguments, but S works with its default value 0. Similarly, when the third statement is executed, H gets the value from calling function, but M and S use the default values. So, after the function calls, the values of s<sub>1</sub>, s<sub>2</sub> and s<sub>3</sub> will be 7840, 5400 and 10800 respectively.

We have seen that functions can be defined with arguments assigned with initial values. The initialized formal arguments are called **default arguments** which allow the programmer to call a function with different number of arguments. That is, we can call the function with or without giving values to the default arguments.

#### 10.4.5 Methods of calling functions

Suppose your teacher asks you to prepare invitation letters for the parents of all students in your class, requesting them to attend a function in your school. The teacher can give you a blank format of the invitation letter and also a list containing the names of all the parents. The teacher can give you the list of names in two ways. She can take a photocopy of the list and give it to you. Otherwise, she can give the original list itself. What difference would you feel in getting the name list in these two ways? If the teacher gives the original name list, you will be careful not to mark or write anything in the name list because the teacher may want the same name list for future use. But if you are given a photocopy of the list, you can mark or write in the list, because the change will not affect the original name list.

Let us consider the task of preparing the invitation letter as a function. The name list is an argument for the function. The argument can be passed to the function in two ways. One is to pass a copy of the name list and the other is to pass the original name list. If the original name list is passed, the changes made in the name list, while preparing the invitation letter, will affect the original name list. Similarly, in C++, an argument can be passed to a function in two ways. Based on the method of passing the arguments, the function calling methods can be classified as Call by Value method and Call by Reference method. The following section describes the methods of argument passing in detail.





### a. Call by value (Pass by value) method

In this method, the value contained in the actual argument is passed to the formal argument. In other words, a copy of the actual argument is passed to the function. Hence, if the formal argument is modified within the function, the change is not reflected in the actual argument at the calling place. In all previous functions, we passed the argument by value only. See the following example:

```
void change(int n)
{
    n = n + 1;
    cout << "n = " << n << '\n';
}

int main()
{
    int x = 20;
    change(x);
    cout << "x = " << x;
}
```

The parameter *n* has its own memory location to store the value 20 in `change()`.

The value of *x* is passed to *n* in `change()`

When we pass an argument as specified in the above program segment, only a copy of the variable *x* is passed to the function. In other words, we can say that only the value of the variable *x* is passed to the function. Thus the formal parameter *n* in the function will get the value 20. When we increase the value of *n*, it will not affect the value of the variable *x*. The following will be the output of the above code:

```
n = 21
x = 20
```

Table 10.2 shows what happens to the arguments when a function is called using call-by-value method:

| Before function call                                                                                                                                                                                     | After function call                                                                                                                                                                                                                                             | After function execution                                                                                                                                                                                   |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <div>main()</div> <div> <pre>{   .....   ..... }</pre> </div> <div> <div>x</div> <div>20</div> </div> <div>change(int n)</div> <div> <pre>{   ..... }</pre> </div> <div> <div>n</div> <div></div> </div> | <div>main()</div> <div> <pre>{   .....   ..... }</pre> </div> <div> <div>x</div> <div>20</div> </div> <div>↓</div> <div> <div>n</div> <div>20</div> </div> <div>change(int n)</div> <div> <pre>{   ..... }</pre> </div> <div> <div>n</div> <div>20</div> </div> | <div>main()</div> <div> <pre>{   .....   ..... }</pre> </div> <div> <div>x</div> <div>20</div> </div> <div>change(int n)</div> <div> <pre>{   ..... }</pre> </div> <div> <div>n</div> <div>21</div> </div> |

Table 10.2: Call by value procedure



### b. Call by reference (Pass by reference) method

When an argument is passed by reference, the reference of the actual argument is passed to the function. As a result, the memory location allocated to the actual argument will be shared by the formal argument. So, if the formal argument is modified within the function, the change will be reflected in the actual argument at the calling place. In C++, to pass an argument by reference we use reference variable as formal parameter. A **reference variable** is an alias name of another variable. An ampersand symbol (&) is placed in between the data type and the variable in the function header. Reference variables will not be allocated memory exclusively like the other variables. Instead, it will share the memory allocated to the actual arguments. The following function uses reference variable as formal parameter and hence call by reference method is implemented for function call.

```
void change(int & n)
{
    n = n + 1;
    cout << "n = " << n << '\n';
}

int main()
{
    int x=20;
    change(x);
    cout << "x = " << x;
}
```

The parameter n is a reference variable and hence there is no exclusive memory allocation for it

The reference of x will be passed to n of the change() function, which results into the sharing of memory.

Note that the only change in the change () function is in the function header. The & symbol in the declaration of the parameter n means that the argument is a reference variable and hence the function will be called by passing reference. Hence when the argument x is passed to the change () function, the variable n gets the address of x so that the location will be shared. In other words, the variables x and n refer to the same memory location. We use the name x in the main () function, and the name n in the change () function to refer the same storage location. So, when we change the value of n, we are actually changing the value of x. If we run the above program, we will get the following output:

n = 21

x = 21

Table 10.3 depicts the changes in the arguments when call-by-reference is applied for the function call.

| Before function call                                                            | After function call                                                                   | After function execution                                                              |
|---------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| <pre>main() { ..... ..... }  change(int &amp;n) { ..... }</pre> <p>x<br/>20</p> | <pre>main() { ..... ..... }  change(int &amp;n) { ..... }</pre> <p>x<br/>20<br/>n</p> | <pre>main() { ..... ..... }  change(int &amp;n) { ..... }</pre> <p>x<br/>21<br/>n</p> |

Table 10.3: Call by reference procedure

These two methods of function call differ as shown in Table 10.4.

| Call by Value Method                                                                                                                                                                                                                                                                                                                                   | Call by Reference Method                                                                                                                                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>• Ordinary variables are used as formal parameters.</li> <li>• Actual parameters may be constants, variables or expressions.</li> <li>• The changes made in the formal arguments are not reflected in actual arguments.</li> <li>• Exclusive memory allocation is required for the formal arguments.</li> </ul> | <ul style="list-style-type: none"> <li>• Reference variables are used as formal parameters.</li> <li>• Actual parameters will be variables only.</li> <li>• The changes made in the formal arguments are reflected in actual arguments.</li> <li>• Memory of actual arguments is shared by formal arguments.</li> </ul> |

Table 10.4 : Call by value v/s Call by reference

Let us discuss a typical example for call by reference method. This program uses a function that can be called by reference method for exchanging the values of the two variables in the `main()` function. The process of exchanging values of two variables is known as swapping.

#### Program 10.8: To swap the values of two variables

```
#include <iostream>
using namespace std;
void swap(int &x, int &y)
{
    int t = x;
    x = y;
    y = t;
}
```



```
int main()
{
    int m, n;
    m = 10;
    n = 20;
    cout<<"Before swapping m= "<< m <<" and n= "<<n;
    swap(m, n);
    cout<<"\nAfter swapping m= "<< m <<" and n= "<<n;
    return 0;
}
```

Let us go through the statements in Program 10.8. The actual arguments *m* and *n* are passed to the function by reference. Within the `swap()` function, the values of *x* and *y* are interchanged. When the values of *x* and *y* are changed, actually the change takes place in *m* and *n*. Therefore the output of the above program code is:

Before swapping *m*= 10 and *n*= 20

After swapping *m*= 20 and *n*= 10

Modify the above program by replacing the formal arguments with ordinary variables and predict the output. Check your answer by executing the code in the lab.

### Check yourself



1. Identify the most essential function in C++ programs.
2. List the three elements of a function header.
3. What is function prototype?
4. Which component is used for data transfer from calling function to called function?
5. What are the two parameter passing techniques used in C++?

## 10.5 Scope and life of variables and functions

We have discussed C++ programs consisting of more than one function. Predefined functions are used by including the header file concerned. User-defined functions are placed before or after the `main()` function. We have seen the relevance of function prototypes while defining functions. We have also used variables in the function body and as arguments. Now, let us discuss the availability or accessibility of the variables and functions throughout the program. Program 10.9 illustrates the accessibility of local variables in a program.

**Program 10.9: To illustrate the scope and life of variables**

```
#include <iostream>
using namespace std;
int cube(int n)
{
    int cb;
    cout<< "The value of x passed to n is " << x;
    cb = n * n * n;
    return cb;
}
int main()
{
    int x, result;
    cout << "Enter a number : ";
    cin >> x;
    result = cube(x);
    cout << "Cube = " << result;
    cout << "\nCube = "<<cb;
}
```

This is an error because the variable `x` is declared within the `main()` function. So it cannot be used in other functions.

This is an error because the variable `cb` is declared within the `cube()` function. So it cannot be used in other functions.

When we compile the program, there will be two errors because of the reasons shown in the call-outs. The concept of availability or accessibility of variables and functions is termed as their scope and life time. **Scope** of a variable is that part of the program in which it is used. In the above program, scope of the variable `cb` is in the `cube()` function because it is declared within that function. Hence this variable cannot be used outside the function. This scope is known as **local scope**. On completing the execution of a function, memory allocated for all the variables within a function is freed. In other words, we can say that the life of a variable, declared within a function, ends with the execution of the last instruction of the function. So, if we use a variable `n` within the `main()`, that will be different from the argument `n` of a called function or a variable `n` within the called function. The variables used as formal arguments and/or declared within a function have local scope.

Just like variables, functions also have scope. A function can be used within the function where it is declared. That is the function is said to have local scope. If it is declared before the `main()` function and not within any other function, the scope of the function is the entire program. That is the function can be used at any place in the program. This scope is known as **global scope**. Variables can also be declared with global scope. Such declarations will be outside all the functions in the program.

Look at Program 10.10 to get more clarity on the scope and life of variables and functions throughout the program.

**Program 10.10 : To illustrate the scope and life of variables and functions**

```
#include <iostream>
using namespace std;
int cb;           //global variable
void test()//global function since defined above other functions
{
    int cube(int n); //It is a local function
    cb=cube(x); //Invalid call. x is local to main()
    cout<<cb;
}
int main() // beginning of main() function
{
    int x=5; //local variable
    test(); //valid call since test() is a global function
    cb=cube(x); //Invalid call. cube() is local to test()
    cout<<cb;
}
int cube(int n)//Argument n is local variable
{
    int val= n*n*n; //val is local variable
    return val;
}
```

| Scope & life     | Local                                                                                                                                                                                                                                                                                               | Global                                                                                                                                                                                                                                                          |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Variables</b> | <ul style="list-style-type: none"> <li>Declared within a function or a block of statements.</li> <li>Available only within that function or block.</li> <li>Memory is allocated when the function or block is active and freed when the execution of the function or block is completed.</li> </ul> | <ul style="list-style-type: none"> <li>Declared outside all the functions.</li> <li>Available to all the functions of the program.</li> <li>Memory is allocated just before the execution of the program and freed when the program stops execution.</li> </ul> |
| <b>Functions</b> | <ul style="list-style-type: none"> <li>Declared within a function or a block of statements and defined after the calling function.</li> <li>Accessible only within that function or the block.</li> </ul>                                                                                           | <ul style="list-style-type: none"> <li>Declared or defined outside all other functions.</li> <li>Accessible by all the functions in the program</li> </ul>                                                                                                      |

*Table 10.5 : Scope and life of variables and functions*



The given call-outs explain the scope and life of functions. A function which is declared inside the function body of another function is called a **local function** as it can be used within that function only. A function declared outside the function body of any other function is called a **global function**. A global function can be used throughout the program. In other words, the scope of a global function is the entire program and that of a local function is only the function where it is declared. Table 10.5 summarises the scope and life time of variables and functions.

## 10.6 Recursive functions

Usually a function is called by another function. Now let us define a function that calls itself. The process of calling a function by itself is known as **recursion** and the function is known as **recursive function**. Some of the complex algorithms can be easily simplified by using the method of recursion. A typical recursive function will be as follows:

```
int function1()
{
    .....
    .....
    int n = function1();    //calling itself
    .....
    .....
}
```

In recursive functions, the function is usually called based on some condition only. Following is an example for recursive function by which the factorial of a number can be found. Note that factorial of a negative number is not defined. Therefore 'n' can take the value either zero or a positive integer.

```
int factorial(int n)
{
    if ((n==1) || (n==0))
        return 1;
    else if (n>1)
        return (n * factorial(n-1));
    else
        return 0;
}
```

Let us discuss how this function is executed when the user calls this function as:

```
f = factorial(3);
```

When the function is called for the first time, the value of the parameter n is 3. The condition in the if statement is evaluated to be false. So, the else block is executed.



When the value of  $n$  is 3, the instruction in the `else` block becomes

```
return (3 * factorial(3-1));
```

which is simplified as: `return (3 * factorial(2));` ..... (i)

In order to find `3 * factorial(2)`, it needs to find the value of `factorial(2)`. The `factorial()` function is called again with 2 as the argument. The function is called within the same function. When the `factorial()` function is executed with 2 as the value of the parameter  $n$ , the condition in the `if` block becomes false. So only the `else` block is executed. The instruction in the `else` block is:

```
return (2 * factorial(2-1));
```

which is simplified as: `return (2 * factorial(1));` ..... (ii)

In order to find this returning value, it calls the `factorial()` function again with 1 as the value of the parameter  $n$ . Now, the condition in the `if` statement becomes true, hence it will return 1 as the value of the function call `factorial(1)`;

Now, the program can find the return value in the instruction numbered (ii). The instruction (ii) will become: `return 2 * 1;`  
which is same as: `return 2;`

That is, the value 2 is returned as the value of the function call `factorial(2)` in the instruction numbered (i). Thus the instruction (i) becomes: `return 3 * 2;`  
which is same as: `return 6;`

Now the value 6 is returned as the value of the function call `factorial(3)`;

The execution of the instruction `f=factorial(3)` is summarised in Figure 10.4.

The execution of the function call `factorial(3)` is delayed till it gets the value of the function `factorial(2)`. The execution of this function is delayed till it gets the value of the function `factorial(1)`. Once this function call gets 1 as its return value, it returns the value to the previous function calls. Figure 10.4 shows what happens to the arguments and return value on each call of the function.

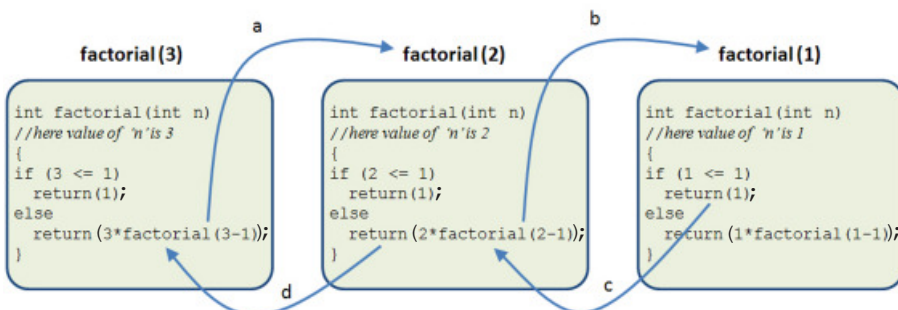
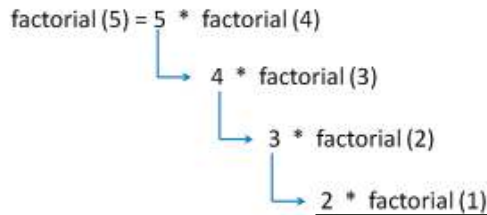


Fig. 10.4: Control flow in a recursive function call



Note that if we call the function `factorial()` with a negative integer as argument, the function will return 0. It doesn't mean that the factorial of a negative number is 0. In mathematics the factorial of a negative number is undefined. So, for example, the value returned by the function call `factorial(-3)` should be interpreted as an invalid call in the calling function.

The procedure that takes place when the function `factorial()` is called with 5 is shown in Figure 10.5.



*Fig. 10.5: Recursion process for factorial(5)*

A function to find the factorial of a number can also be defined as follows, without using recursion.

```

int factorial(int n)
{
    int f=1;
    /* The formula n×(n-1)×(n-2)× ... ×2×1 is applied
    instead of 1 × 2 × 3×...×(n-1)×n to find the factorial
    */
    for(int i=n; i>1; i--)
        f *= i;
    return f;
}
  
```

We can compare the difference between the two functions – the one that uses recursion and the other that does not use recursion. Every function that uses recursion can also be written without using recursion. Then why do we use recursion? Some programmers find the use of recursion much simpler than the other. The recursion method cannot be used in all the functions. How do we understand whether recursion can be applied to a function or not? If we can express a function by performing some operation on the output of the same function, we can use recursion for that. For example, to find the sum of first  $n$  natural numbers, we can write the `sum(n)` as:

$$\text{sum}(n) = n + \text{sum}(n-1)$$

Let us discuss a program that converts a given decimal number into its equivalent binary number. We discussed the conversion procedure in Chapter 2.

**Program 10.11: To display the binary equivalent of a decimal number**

```
#include <iostream>
using namespace std;
void Binary(int);
int main()
{
    int decimal;
    cout<<"Enter an integer number: ";
    cin>>decimal;
    cout<<"Binary equivalent of "<<decimal<<" is ";
    Binary(decimal);
    return 0;
}

void Binary(int n)    //Definition of a recursive function
{
    if (n>1)
        Binary(n/2);
    cout<<n%2;
}
```

A sample output of Program 10.11 is given below:

```
Enter an integer number: 19
Binary equivalent of 19 is 10011
```

**10.7 Creation of header files**

All of us know that we always use `#include <iostream>` in the beginning of all the programs that we have discussed so far. Why do we use this statement? Actually `iostream` is a header file that contains the declarations and definitions of so many variables or objects that we use in C++ programs. The objects `cout` and `cin` that we use in the program are declared in this header file. So when we include this header file in a program, the definitions and declarations of objects and functions are available to the compiler during compilation. Then the executable code of these functions and objects will be linked with the program and will be executed when and where they are called. We can create similar header files containing our own variables and functions. Suppose we want to write a function to find the factorial of a number and use this function in a number of programs. Instead of defining the factorial function in all the programs, we can place the function in a header file and then include this header file in all other programs.



The following example shows how we can create a header file. Enter the following program code in any IDE editor.

```
int factorial(int n)
{
    int f=1;
    for (int i=1; i<=n; i++)
        f *= i;
    return f;
}
```

Save this in a file with the name **factorial.h** and then create a C++ program as follows:

```
#include <iostream>
#include "factorial.h"//includes user-defined headerfile
using namespace std;
int main()
{
    int n;
    cout<<"Enter a number : ";
    cin >> n;
    cout<<"Factorial : " << factorial(n);
}
```

We can compile and run the program successfully. Note that the statement `#include "factorial.h"` uses double quotes instead of angular brackets `< >`. This is because, when we use angular brackets for including a file, the compiler will search for the file in the include directory. But if we use double quotes, the compiler will search for the file in the current working directory only. Usually when we save the `factorial.h` file, it will be saved in the working directory, where the main C++ program is saved. So, we must use double quotes to include the file. Now, whenever we want to include factorial function in any C++ program, we only need to include header file `factorial.h` in the program by using the statement `#include "factorial.h"`. In the same way we can place any number of functions in a single header file and include that header file in any program to make use of these functions.

### Check yourself



1. If the prototype of a function is given immediately after the preprocessor directive, its scope will be \_\_\_\_\_.
2. What is recursion?
3. What is the scope of predefined functions in C++?
4. The arguments of a function have \_\_\_\_\_ scope.



## Let us sum up

Modular programming is an approach to make programming simpler. C++ facilitates modularization with functions. Function is a named unit of program to perform a specific task. There are two types of functions in C++: predefined functions and user-defined functions. Predefined functions can be used in a program only if we include the header file concerned in the program. User-defined functions may need to be declared if the definition appears after the calling function. During function call, data may be transferred from calling function to the called function through arguments. Arguments may be classified as actual arguments and formal arguments. Either call by value method or call by reference method can be used to invoke functions. The variables and functions in a program have scope and life depending on the place of their declaration. Though a function is called by another function, C++ allows recursion which is a process of calling a function by itself. New header files can be created to store the user-defined functions so that these functions can be used in any program.



## Learning outcomes

After completing this chapter the learner will be able to

- recognise modular programming style and its merits.
- use predefined functions for problem solving.
- define sub functions for performing particular tasks involved in problem solving.
- use the sub functions defined by the user.
- define the recursive functions and use them for problem solving.



## Lab activity

1. Define a function to accept a number and return 1 if it is prime, 0 otherwise. Using this function write a program to display all prime numbers between 100 and 200.
2. Write a program to find the smallest of three or two given numbers using a function (use the concept of default arguments).
3. With the help of a user-defined function find the sum of digits of a number. That is, if the given number is 3245, the result should be  $3+2+4+5 = 14$ .
4. Using a function, write a program to find the LCM of two given numbers.
5. Write a program to display all palindrome numbers between a given range using a function. The function should accept number and return 1 if it is palindrome, 0 otherwise.



## Sample questions

### Very short answer type

1. How do top down design and bottom up design differ in programming?
2. What is a function in C++?
3. The ability of a function to call itself is \_\_\_\_\_.
4. Write down the role of header files in C++ programs.
5. When will you use void data type for function definition?

### Short answer type

1. Distinguish between actual parameters and formal parameters.
2. Construct the function prototypes for the following functions
  - (a) `Total()` takes two double arguments and returns a double
  - (b) `Math()` takes no arguments and has no return value
3. Distinguish between `exit()` function and `return` statement.
4. Discuss the scope of global and local variables with examples.
5. Distinguish between Call-by-value method and Call-by-reference method used for function calls.
6. In C++, function can be invoked without specifying all its arguments. How?
7. Write down the process involved in recursion.

### Long answer type

1. Look at the following functions:

```
int sum(int a, int b=0, int c=0)
{ return (a + b + c); }
```

- (a) What is the speciality of the function regarding the parameter list?
  - (b) Give the outputs of the following function calls by explaining its working and give reason if the function call is wrong.
    - (i) `cout<<sum(1, 2, 3);`      (ii) `cout<<sum(5, 2);`
    - (iii) `cout<<sum();`      (iv) `cout<<sum(0);`
2. The prototype of a function is: `int fun(int, int);`  
The following function calls are invalid. Give reason for each.
    - (a) `fun(2, 4);`      (b) `cout<<fun();`      (c) `val=fun(2.5, 3.3);`
    - (d) `cin>>fun(a, b);`      (e) `z=fun(3);`



## Key Concepts

- **Computer network**
  - Need for network
  - Some key terms
- **Data communication system**
- **Communication medium**
  - Guided medium
  - Unguided medium
  - Wireless communication technologies using radio waves
- **Data communication devices**
  - NIC, Hub, Switch, Repeater, Bridge, Router, Gateway
- **Data terminal equipments**
  - Modem, Multiplexer/ Demultiplexer
- **Network topologies**
  - Bus, Star, Ring, Mesh
- **Types of network**
  - PAN, LAN, MAN, WAN
- **Logical classification of networks**
  - Peer-to-peer
  - Client-Server
- **Identification of computers and users over a network**
  - MAC address
  - IP address
- **Network protocols**
  - TCP/IP (HTTP, FTP, DNS)
- **Uniform Resource Locator**

## Computer Networks

Have you surfed the Internet to search your examination result or to know whether you got admission to the Plus One course in a school of your choice? Have you visited an ATM counter to draw money? Have you transferred songs, images or movie clips from your computer to a cell phone or booked a train ticket using Internet? If your answer to any of these questions is 'yes', you have accessed the services of a computer network. In this chapter, we will learn more about the working of networks and their advantages. We will also discuss different media and devices, different types of networks and the rules to be followed in data communication using these devices.

### 11.1 Computer network

Computer network is a group of computers and other computing hardware devices (such as printers, scanners, modems, CD drives, etc.) connected to each other electronically through a communication medium. They can communicate with each other, exchange commands, share data, hardware and other resources. Computers on a network may be linked through cables, telephone lines, radio waves, satellites or infrared light beams.

#### 11.1.1 Need for Network

Internet is a good example for a computer network. It is impossible to imagine a world



without e-mails, online newspapers, blogs, chat and other services offered through the Internet. Apart from these, there are many other advantages in using networked computers instead of stand-alone computers. Some of them are listed below.

- Resource sharing
- Price-performance ratio
- Communication
- Reliability
- Scalability

**Resource sharing:** The sharing of available hardware and software resources in a computer network is called **resource sharing**. For example, the content of a DVD placed in a DVD drive of one computer can be read in another computer. Similarly, other hardware resources like hard disk, printer, scanner, etc. and software resources like application software, anti-virus tools, etc. can also be shared through computer networks.

**Price-performance ratio:** One can easily share the resources available in one computer with other computers. The cost of purchasing licensed software for each computer can be reduced by purchasing network versions of such software. This will least affect the performance of such resources and lead to considerable savings in cost.

**Communication:** Computer network helps user to communicate with any other user of the network through its services like e-mail, chatting, video conferencing etc. For example, one can send or receive messages within no time irrespective of the distance.

**Reliability:** It is possible to replicate or backup data/information in multiple computers using the network. For example, the C++ files, photos or songs saved in one computer can also be saved in other computers in the same network. These can be retrieved from other computers in which they are saved in case of disasters (malfunctioning of computers, accidental deletion of files, etc.)

**Scalability:** Computing capacity can be increased or decreased easily by adding or removing computers to the network. In addition to this, the storage capacity of networks can also be increased by including more storage devices to the network.

### 11.1.2 Some key terms

Some of the key terms related to computer network are explained below:

**Bandwidth :** Bandwidth measures the amount of data that can be sent over a specific connection in a given amount of time. Imagine you are in a

highway or a public road. The bigger the road, the more will be the number of vehicles that can travel on it. Moreover, the traffic here is faster than on a narrow road. On a narrow road, the traffic is likely to be congested. We can say that the bandwidth of a bigger road is higher than a narrow road.

Bandwidth describes the maximum data-transfer rate between computers in a network. In digital systems, bandwidth is measured in bits per second (bps). If the bandwidth is more, data travels faster and hence large amounts of data can be transferred within a particular time span across the network. For example, an Internet connection via cable modem may provide 25 Mbps of bandwidth.

- Noise** : Noise is unwanted electrical or electromagnetic energy that lowers the quality of data signals. It occurs from nearby radio transmitters, motors or other cables. The transfer of all types of data including texts, programs, images and audio over a network is adversely affected by noise.
- Node** : Any device (computer, scanner, printer, etc.) which is directly connected to a computer network is called a **node**. For example, computers linked to the computer network in the school are nodes. When we connect the Internet to our computer, our computer becomes a node of the Internet.

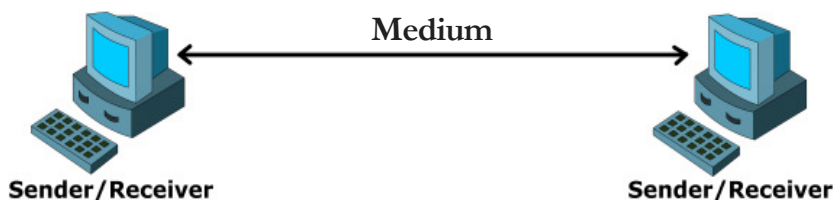


*Make a list of the hardware and software resources shared in your school network.*

**Let us do**

## 11.2 Data communication system

In a computer network, computing devices are connected in various ways, to communicate and share resources. **Data communication** is the exchange of digital data between any two devices through a medium of transmission. Figure 11.1 shows the representation of a general data communication system.



*Fig. 11.1 : Data communication system*

The following five basic elements are necessary for building any data communication system.

- Message** : It is the information to be communicated. Major forms of information include text, picture, audio, video, etc.
- Sender** : The computer or device that is used for sending messages is called the sender, source or transmitter.
- Receiver** : The computer or device that receives the messages is called the receiver.
- Medium** : It is the physical path through which a message travels from the sender to the receiver. It refers to the way in which nodes are connected.
- Protocol** : The rules under which message transmission takes place between the sender and the receiver is called a protocol.

### 11.3 Communication medium

Data communication is possible only if there is a medium through which data can travel from one device to another. The medium for data transmission over a computer network is called **communication channel** or **communication medium**. The communication medium between computers in a network are of two types: guided and unguided. In guided or wired medium physical wires or cables are used and in unguided or wireless medium radio waves, microwaves or infrared signals are used for data transmission.

#### 11.3.1 Guided medium (Wired)

The coaxial cable, twisted pair cable (Ethernet cable) and optical fibre cable are the different types of cables used to transfer data through computer networks.

##### a. Twisted pair cable (Ethernet cable)

This is the most widely used media in small computer networks. It consists of four twisted pairs which are enclosed in an outer shield. These pairs are colour coded. Twisted pair cables are of two types:

- (i) Unshielded Twisted Pair (UTP) cables and
- (ii) Shielded Twisted Pair (STP) cables.

**Unshielded Twisted Pair (UTP) cable:** As its name suggests, the individual pairs in UTP cables are not shielded. Figure 11.2 shows the components of a UTP cable.

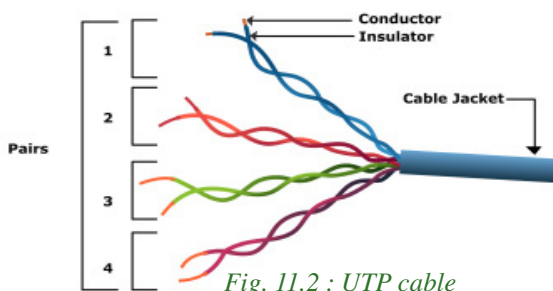


Fig. 11.2 : UTP cable



### *Characteristics of UTP cable*

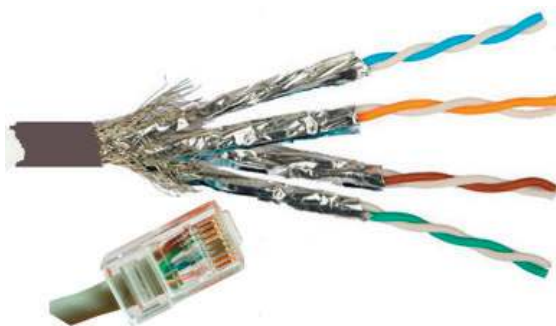
- Low cost cable available for setting up small networks.
- Thin and flexible cable.
- Ease of installation.
- Carries data upto a length of 100 m at a stretch.

**Shielded Twisted Pair (STP) cable:** It is the same cable as the UTP, but with each pair shielded individually. An outer shield then covers all the pairs like in UTP.

### *Characteristics of STP cable*

- Shielding in STP offers better immunity against noise.
- It is more expensive than UTP cable.
- Compared to UTP cable, STP cable is difficult to install.

An RJ-45 connector is used to connect UTP/STP twisted pair cable to a computer. Figure 11.3 shows the STP cable and RJ-45 connector.



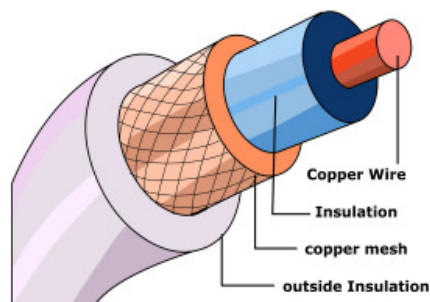
*Fig. 11.3 : STP cable and RJ-45 connector*

## **b. Coaxial cable**

A coaxial cable consists of an inner conductor surrounded by a tubular insulating layer which is further covered by a tubular conducting shield. It has an outer insulation to protect the cable too. Figure 11.4 shows the construction of a coaxial cable.

### *Characteristics of coaxial cable*

- Carries data to longer distances (185 m - 500 m approx.) at a stretch.
- High bandwidth.
- Less electromagnetic interference due to the outer shield.
- Thicker than twisted pair.
- Less flexible than twisted pair.
- Difficult to install than twisted pair cable.

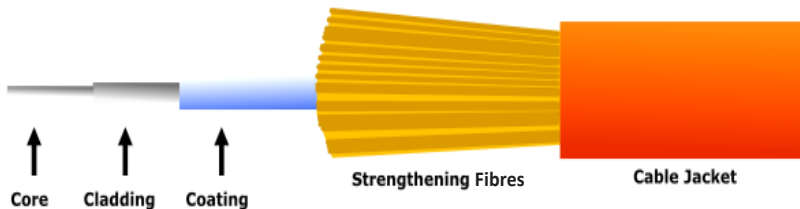


*Fig. 11.4 : Coaxial Cable*



### c. Optical fibre cable

Optical fibres are long thin glass fibres through which data is transmitted as light signals. Data travels as fast as light and can be transmitted to far off distances. Figure 11.5 shows the major parts of an optical fibre cable.



*Fig. 11.5 : Optical fibre*

Optical fibre has the following parts:

- Core - the thin glass rod at the centre through which the light travels.
- Cladding - the outer optical material surrounding the core that reflects the light back into the core.
- Coating - the plastic coating that protects the cable from damage and moisture.

These optical fibres are arranged in bundles of hundreds and thousands and are protected by the outer covering of the cable called jacket.

At the source end, the optical transmitter converts electrical signals into optical signals (modulation) using semiconductor devices such as light-emitting diodes (LEDs) or laser diodes. At the destination end, the optical receiver, consisting of a photo detector, converts light back to electric signals (demodulation) using the photoelectric effect. The speed of transmission and the distance of signals are higher for laser diodes than for LEDs.

#### *Characteristics of optical fibre cable*

- High bandwidth for voice, video and data applications.
- Carries data over a very long distance at a stretch.
- Not susceptible to electromagnetic fields, as it uses light for data transmission.
- The most expensive and the most efficient communication media available for computer networks.
- Installation and maintenance are difficult and complex.



### 11.3.2 Unguided medium (Wireless)

Electromagnetic waves are used for wireless communication on computer networks. Frequencies of waves are measured in Hertz (Hz). Based on their frequencies, electromagnetic waves are categorised into various types as shown in Figure 11.6. From this category we can infer that only radio waves, microwaves and infrared rays are used for wireless communication.

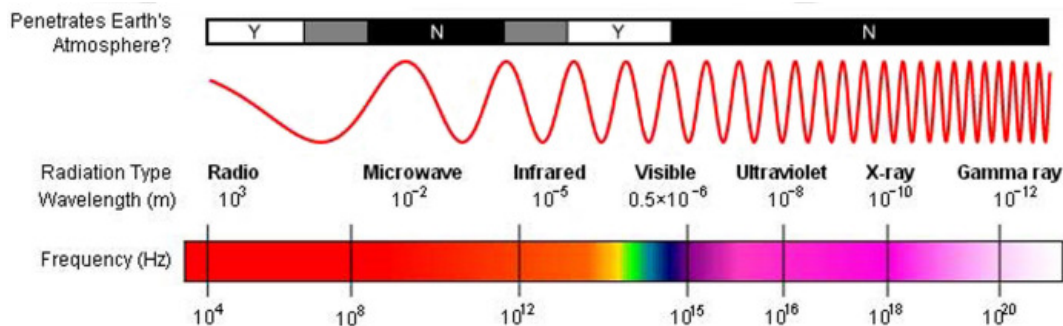


Fig. 11.6: Electromagnetic Spectrum

#### a. Radio waves

Radio waves have a frequency range of 3 KHz to 3 GHz. Radio waves can be used for short and long distance communication. These waves are easy to generate and can go over the walls of a building easily. That is why radio waves are widely used for communication-both indoors and outdoors. Cordless phones, AM and FM radio broadcast and mobile phones make use of radio wave transmission.

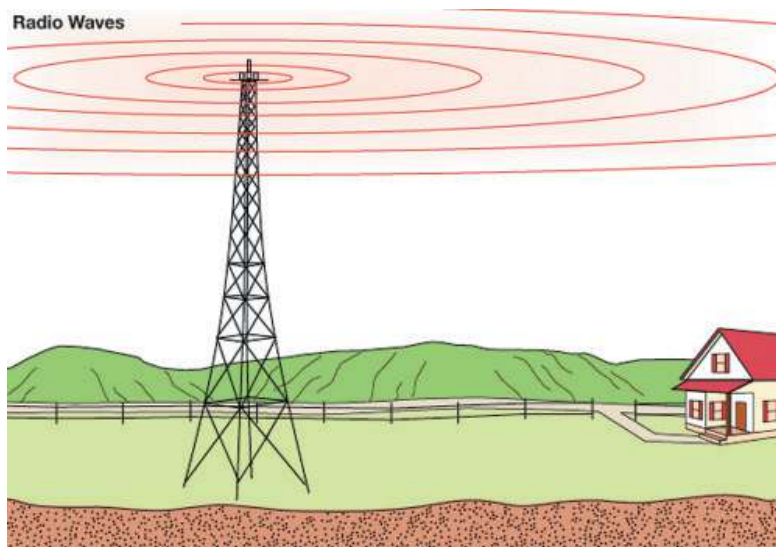


Fig. 11.7 : Radio wave transmission



### Characteristics of radio wave transmission

- Waves are transmitted in all directions, therefore transmitting and receiving antennas need not be aligned face to face.
- Relatively inexpensive than wired media.
- Can penetrate through most objects.
- Transmission can be affected by motors or other electrical equipment.
- Less secure mode of transmission.
- Permissions from authorities concerned are required for the use of radio wave transmission.

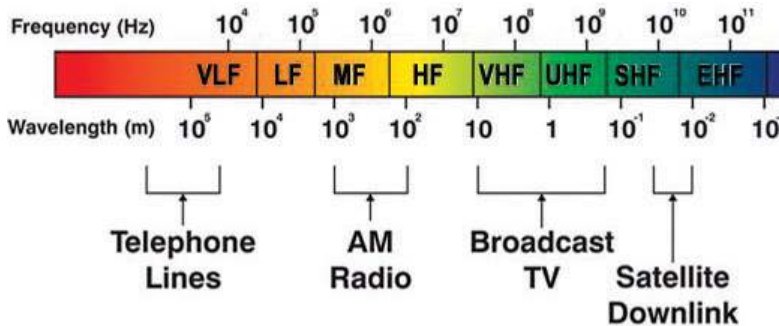


Fig. 11.8 : Spectrum of radio communication band

### b. Micro waves

Micro waves have a frequency range of 300 MHz (0.3 GHz) to 300 GHz. Microwaves travel in straight lines and cannot penetrate any solid object. Therefore, high towers are built and microwave antennas are fixed on their top for long distance microwave communication. As these waves travel in straight lines the antennas used for transmitting and receiving messages have to be aligned with each other. The distance between two microwave towers depends on many factors including frequency of the waves being used and heights of the towers. Figure 11.9 shows the components of a microwave transmission system.

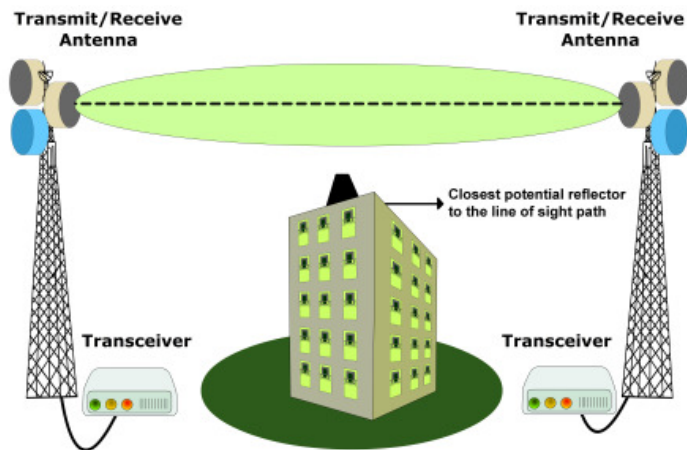


Fig. 11.9 : Microwave transmission



### *Characteristics of micro wave transmission*

- Relatively inexpensive than wired media.
- Offers ease of communication over difficult terrain.
- The transmission is in straight lines. Therefore, the transmitting and receiving antennas need to be properly aligned (line of sight transmission).

### **c. Infrared waves**

Infrared waves have a frequency range of 300 GHz to 400 THz. These waves are used for short range communication (approx. 5 m) in a variety of wireless communications, monitoring and control applications.

Home entertainment remote control devices, cordless mouse and intrusion detectors are some of the devices that utilise infrared communication (refer Figure 11.10).



*Fig. 11.10 : Infrared transmission*

### *Characteristics of infrared wave transmission*

- A line of sight transmission; hence information passed to one device is not leaked.
- Only two devices can communicate at a time.
- The waves cannot cross solid objects. (You may stand between the remote control and your television set and check whether the remote control works.)
- The longer the distance the weaker the performance.

## **11.3.3 Wireless communication technologies using radio waves**

### **a. Bluetooth**

Bluetooth technology uses radio waves in the frequency range of 2.402 GHz to 2.480 GHz. This technology is used for short range communication (approx. 10 m) in a variety of devices for wireless communication. Cell phones, laptops, mouse, keyboard, tablets, headsets, cameras, etc. are some of the devices that utilise bluetooth communication (refer Figure 11.11).



*Fig. 11.11 : Bluetooth transmission*



### *Characteristics of bluetooth transmission*

- Line of sight between communicating devices is not required.
- Bluetooth can connect upto eight devices simultaneously.
- Slow data transfer rate (upto 1 Mbps).

### **b. Wi-Fi**

Wi-Fi network makes use of radio waves to transmit information across a network like cell phones, televisions and radios. The radio waves used in Wi-Fi ranges from a frequency of 2.4 GHz to 5 GHz. Communication across a wireless network is two-way radio communication. The wireless adapter in a computer translates data into radio signal and transmits it using an antenna. A wireless router receives the signal and decodes it. Once decoded, the data will be sent to the Internet or network through a wired Ethernet /wireless connection. Similarly, the data received from the Internet/network will also pass through the router and coded into radio signals that will be received by the wireless adapter in a computer as indicated in Figure 11.12. Nowadays, this technology is widely used to share Internet connection with laptops or desktops.



*Fig. 11.12 : Wi-Fi transmission*

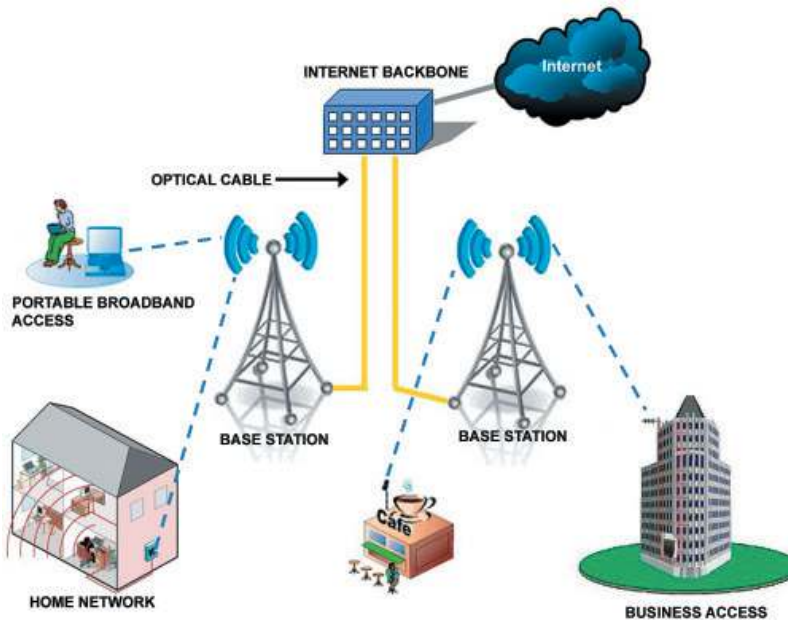
### *Characteristics of Wi-Fi transmission*

- Line of sight between communicating devices is not required.
- Data transmission speed is upto 54 Mbps.
- Wi-Fi can connect more number of devices simultaneously.
- Used for communication upto 375 ft (114 m).

### **c. Wi-MAX**

Worldwide Interoperability for Microwave Access (Wi-MAX) originally based on 802.16e, combines the benefits of broadband and wireless. Wi-MAX has a frequency

range of 2 GHz to 11 GHz. Wi-MAX can provide high-speed wireless Internet access over very long distances (a whole city). Wi-MAX equipment exists in two basic forms - base stations, installed by service providers to deploy the technology in a coverage area, and receivers, installed by clients. Figure 11.13 shows the basic components of a Wi-MAX transmission.



*Fig. 11.13 : WiMAX transmission*

### *Characteristics of Wi-MAX transmission*

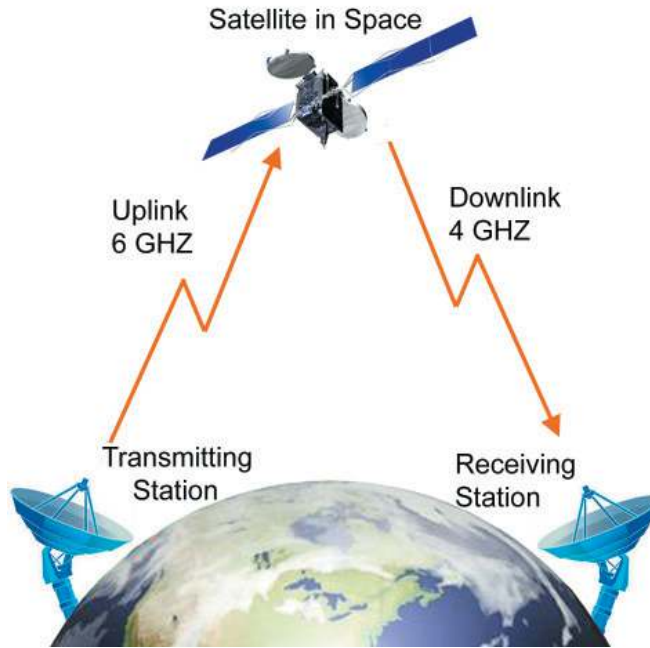
- Hundreds of users can connect to a single station.
- Provides higher speed connection upto 70 Mbps over an area of 45 Kilometres.
- Line of sight between communicating devices is not required.
- Weather conditions like rain, storm, etc. could interrupt the signal.
- Very high power consumption.
- High costs of installation and operation.

### **d. Satellite link**

Long distance wireless communication systems use satellite links for transmitting signals. Usually, a signal travels in a straight line and is not able to bend around the globe to reach a destination far away. Signals can be sent to geostationary satellites in space and then redirected to another satellite or directly to a far away destination.



A geostationary satellite orbits the earth in the same direction and amount of time it takes to revolve the earth once. From the earth, therefore, the satellite appears to be stationary, always above the same area of the earth. These satellites carry electronic devices called transponders for receiving, amplifying, and re-broadcasting signals to the earth.



*Fig. 11.14 : Satellite link*

Transmission of signals from the earth to a satellite is called **uplink** and from a satellite to the earth is called **downlink**. There are multiple micro wave frequency bands which are used for satellites links. Frequency used for uplink varies from 1.6 GHz to 30.0 GHz and that for downlink varies from 1.5 GHz to 20.0 GHz. Downlink frequency is always lower than the uplink frequency.

The satellite system is very expensive, but its coverage area is very large. Communication satellites are normally owned by governments or by government approved organisations of various countries.

#### *Characteristics of transmission using satellite link*

- Satellites cover a large area of the earth.
- This system is expensive.
- Requires legal permission and authorisation.

**Check yourself**

1. Name the basic elements needed for a data communication system.
2. Define resource sharing.
3. Name two classifications of communication channels between computers in a network.
4. Name the connector used to connect UTP/STP cable to a computer.
5. The cable media that use light to transmit data signals to very long distances is \_\_\_\_\_.
6. AM and FM radio broadcast and mobile phones make use of \_\_\_\_\_ medium for transmission.
7. The medium for communication used in home entertainment remote control devices, certain mouse, etc. is \_\_\_\_\_.
8. A short range communication technology that does not require line of sight between communicating devices is \_\_\_\_\_.
9. A communication system that is very expensive, but has a large coverage area when compared to other wireless communication systems is \_\_\_\_\_.

**11.4 Data communication devices**

A data communication device provides an interface between computer and the communication channel. These devices are used to transmit, receive, amplify and route data signals across a network through various communication media.

**11.4.1 Network Interface Card (NIC)**

Network Interface Card (NIC) is a device that enables a computer to connect to a network and communicate. It provides hardware interface between a computer and a network. It can be a separate circuit board that is installed in a computer or a circuit already integrated with the motherboard. NIC can prepare, send, receive and control data on the network. It breaks up data into manageable units, translates the protocols of the computer to that of the communication medium and supplies address recognition capabilities.





Fig. 11.15 (a) : NIC card



Fig. 11.15 (b) : Wireless NIC card

Figure 11.15 (a, b) shows the NIC card and wireless NIC card. Some NIC cards have wired connections (Ethernet), while others are wireless (Wi-Fi). Ethernet NICs include jacks for network cables, while Wi-Fi NICs contain built-in transmitters/receivers (transceivers) and an antenna. NICs can transfer data at a speed of 1 Gbps.

### 11.4.2 Hub

A hub is a device used in a wired network to connect computers/devices of the same network. It is a small, simple, passive and inexpensive device (refer Figure 11.16).



Fig. 11.16 : Hub

Computers/devices are connected to ports of the hub using Ethernet cable. When NIC of one computer sends data packets to hub, the hub transmits the packets to all other computers connected to it. Each computer is responsible for determining its data packets. The computer for which the data packets are intended accepts it. Other computers on the network discards these data packets. The main disadvantage of hub is that it increases the network traffic and reduces the effective bandwidth, as it transmits data packets to all devices connected to it.

### 11.4.3 Switch

A switch is an intelligent device that connects several computers to form a network. It is a higher performance alternative to a hub. It looks exactly like a hub. Switches are capable of determining the destination and redirect the data only to the intended node. Switch performs this by storing the addresses of all the devices connected to it in a table. When a data packet is send by one device, the switch reads the destination address on the packet and transmits the packet to the destination device with the

help of the table. A switch performs better than a hub on busy networks, since it generates less network traffic in delivering messages.

#### 11.4.4 Repeater

A repeater is a device that regenerates incoming electrical, wireless or optical signals through a communication medium (refer Figure 11.17). Data transmissions through wired or wireless medium can travel only a limited distance as the quality of the signal degrades due to noise. Repeater receives incoming data signals, amplifies the signals to their original strength and retransmits them to the destination.



Fig. 11.17 : Wireless repeater

#### 11.4.5 Bridge

A bridge is a device used to segmentise a network. An existing network can be split into different segments and can be interconnected using a bridge. This reduces the amount of traffic on a network. When a data packet reaches the bridge, it inspects the incoming packet's address and finds out to which side of the bridge it is addressed (to nodes on the same side or the other side). Only those packets addressed to the nodes on the other side, will be allowed to pass the bridge. Others will be discarded. The packet that passes the bridge will be broadcast to all nodes on the other side and is only accepted by the intended destination node. Figure 11.18 shows the function of a bridge.

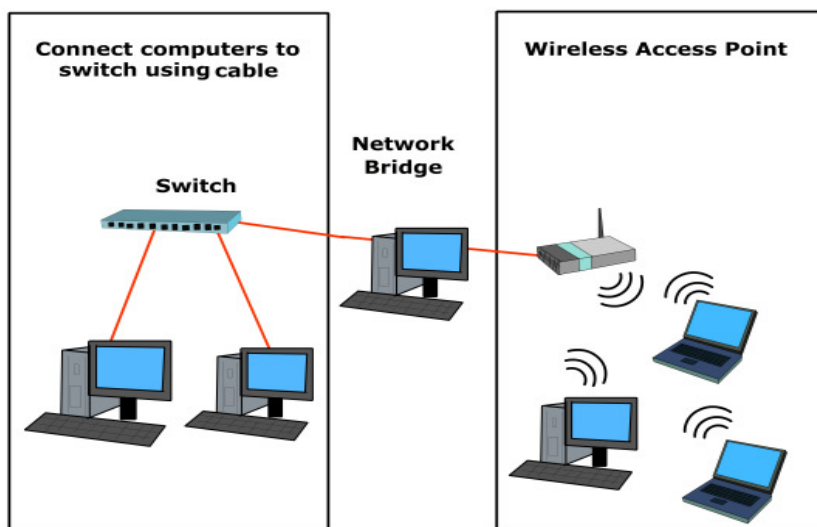


Fig. 11.18 : Bridge





### 11.4.6 Router

A router is a device that can interconnect two networks of the same type using the same protocol. It can find the optimal path for data packets to travel and reduce the amount of traffic on a network. Even though its operations are similar to a bridge, it is more intelligent. The router can check the device address and the network address and can use algorithms to find the best path for packets to reach the destination. Figure 11.19 shows the role of a router.

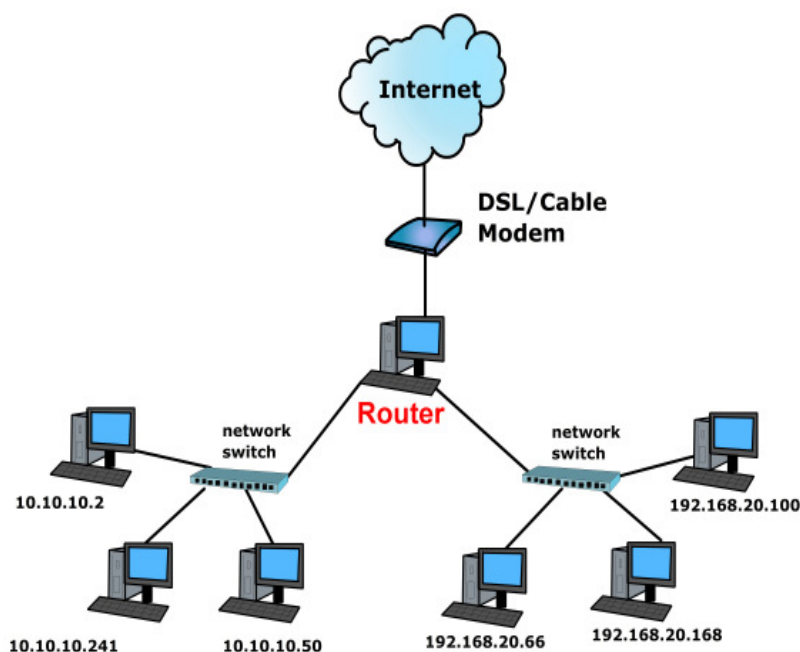


Fig. 11.19 : Router

### 11.4.7 Gateway

A gateway is a device that can interconnect two different networks having different protocols (refer Figure 11.20). It can translate one protocol to another protocol. It is a network point that acts as an entrance to another network. Its operations are similar to that of a router. It can check the device address and the network address and can use algorithms to find the

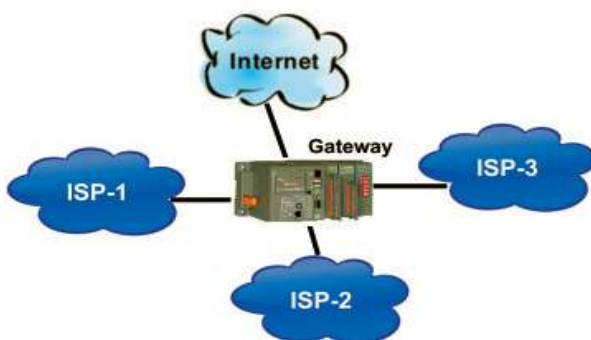


Fig. 11.20 : Gateway



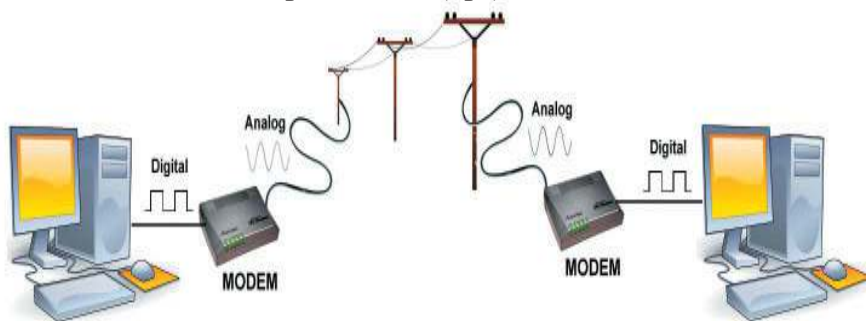
best path for packets to reach the destination. Further, while interconnecting two networks with different protocols, there must be some mutual understanding between the networks. A gateway is capable of understanding the address structure used in different networks and seamlessly translate the data packet between these networks.

## 11.5 Data terminal equipments

A data terminal equipment (DTE) is a device that controls data flowing to or from a computer. It is connected to the transmission medium at the end of a telecommunication link. Here we discuss the most commonly used DTEs - modem and multiplexer.

### 11.5.1 Modem

A modem is an electronic device used for communication between computers through telephone lines (refer Figure 11.21). The name is formed from the first three letters of the two words modulator and demodulator. It converts digital signals received from a computer to analog signals (modulation) for telephone lines. It also converts the analog signals received back from telephone lines to digital signals (demodulation) for the computer. The speed of the modem determines how fast it can send and receive information through telephone lines. The speed of modem is measured in bits per second (bps).



*Fig. 11.21 : Communication using modem*

### 11.5.2 Multiplexer/Demultiplexer

Have you ever wondered how 200 or more TV channels are transmitted through a single cable in a television network? It is called multiplexing. Similar is the case with data transmission over networks. Multiplexing is sending multiple signals on a physical medium at the same time in the form of a single, complex signal and then recovering the separate signals at the receiving end. Multiplexing divides the physical medium into logical segments called frequency channels. Multiplexer combines (multiplexes) the inputs from different sources and sends them through different channels of a medium. The combined data travels over the medium simultaneously.

At the destination, a demultiplexer separates (demultiplexes) the signals and sends them to destinations. Figure 11.22 shows the function of a multiplexer and demultiplexer.

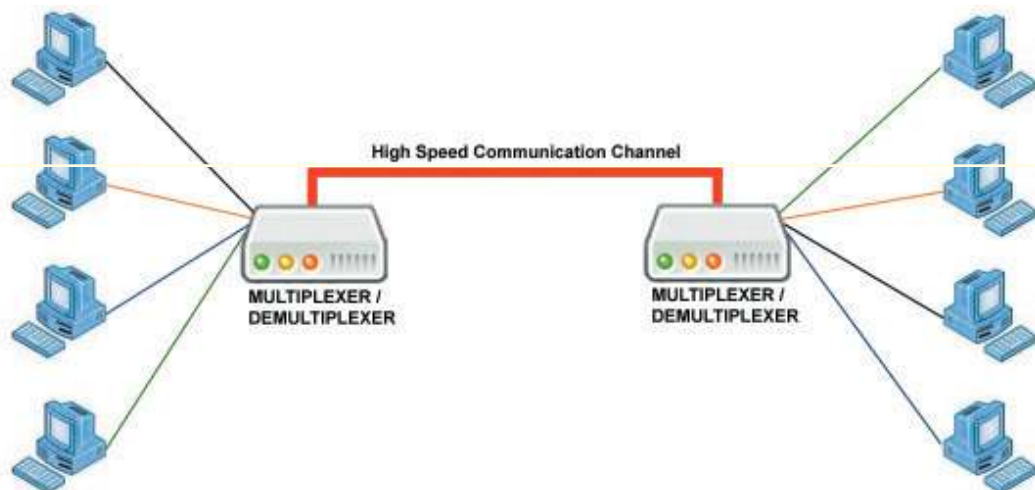


Fig. 11.22 : Multiplexer/De-multiplexer



Let us do

*Make a list of networking devices and communication medium necessary to create a small computer network having a maximum of 10 nodes.*

### Check yourself



1. Compare hub and switch.
2. What is the use of a repeater?
3. The devices used to interconnect two networks of same type is \_\_\_\_\_.
4. Differentiate between router and bridge.
5. A device that can interconnect two different networks having different protocols is \_\_\_\_\_.
6. An electronic device used for communication between computers through telephone lines is \_\_\_\_\_.

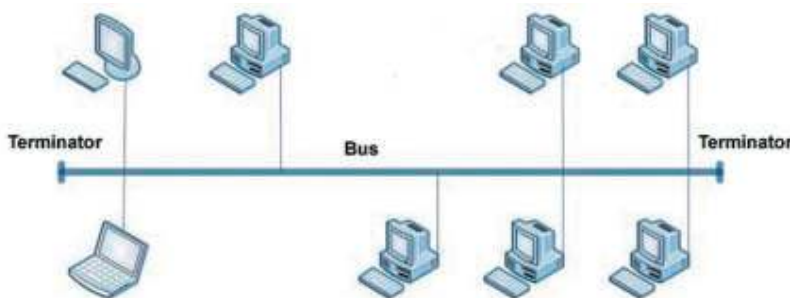
## 11.6 Network topologies

Imagine that we have ten computers and we need to interconnect them to form a network. What are the ways by which we can interconnect them?

Using available media and based on certain conditions, there are different ways of interconnecting the nodes. The way in which the nodes are physically interconnected to form a network is called a **Topology**. Major topologies are bus, star, ring and mesh.

### 11.6.1 Bus topology

In bus topology (refer Figure 11.23) all the nodes are connected to a main cable called bus. If a node has to send data to another node, it sends data to the bus. The signal travels through the entire length of the bus. All nodes check the bus, and only the node for which data is addressed accepts it. A small device called terminator is attached at each end of the bus. When the signal reaches the end of the bus, the terminator absorbs the signal and removes it from the bus. Now the bus is free to carry another signal. This prevents the reflection of a signal back on the cable and hence eliminates the chances of signal interference. The process of transmitting data from one node to all other nodes is called broadcasting.



*Fig. 11.23 : Bus topology*

#### *Characteristics of bus topology*

- Easy to install.
- Requires less cable length and hence it is cost effective.
- Failure of a node does not affect the network.
- Failure of cable (bus) or terminator leads to a break down of the entire network.
- Fault diagnosis is difficult.
- Only one node can transmit data at a time.

### 11.6.2 Star topology

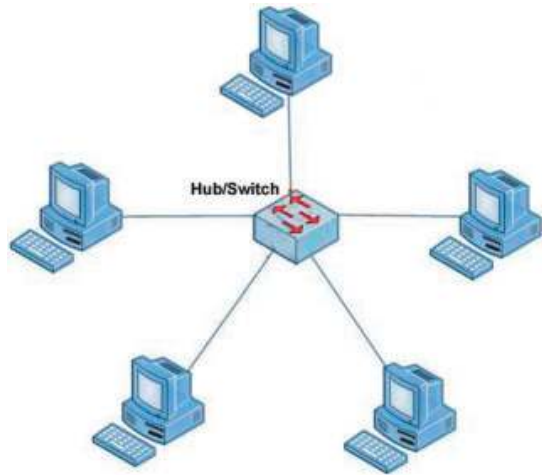
In star topology each node is directly connected to a hub/switch as shown in Figure 11.24. If any node has to send some information to any other node, it sends the



signal to the hub/switch. This signal is then broadcasted (in case of a hub) to all the nodes but is accepted only by the intended node. In the case of a switch, the signal is sent only to the intended node.

### ***Characteristics of star topology***

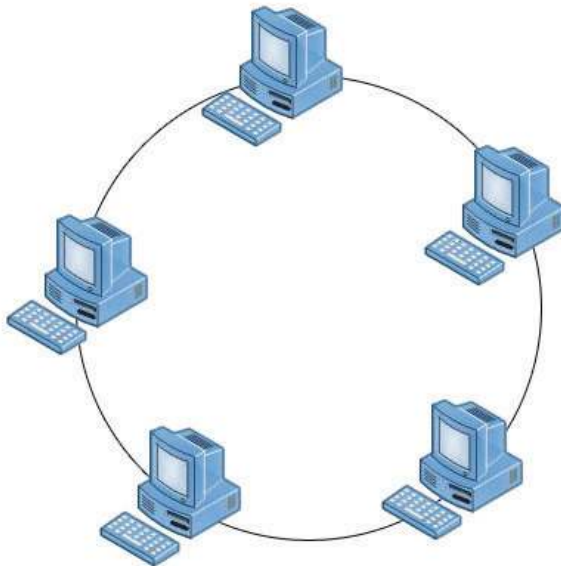
- More efficient compared to bus topology.
- Easy to install.
- Easy to diagnose faults.
- Easy to expand depending on the specifications of central hub/switch.
- Failure of hub/switch leads to failure of entire network.
- Requires more cable length compared to bus topology.



*Fig. 11.24 : Star topology*

### **11.6.3 Ring topology**

In ring topology, all nodes are connected using a cable that loops in a ring or circle. A ring topology is in the form of a circle that has no start and no end (refer Figure 11.25). Terminators are not necessary in a ring topology. Data travels only in one direction in a ring. While they are passed from one node to the next, each node regenerates the signal. The node for which the signal is intended reads the signal. After travelling through each node, the signal reaches back to the sending node from where it is removed.



*Fig. 11.25 : Ring topology*



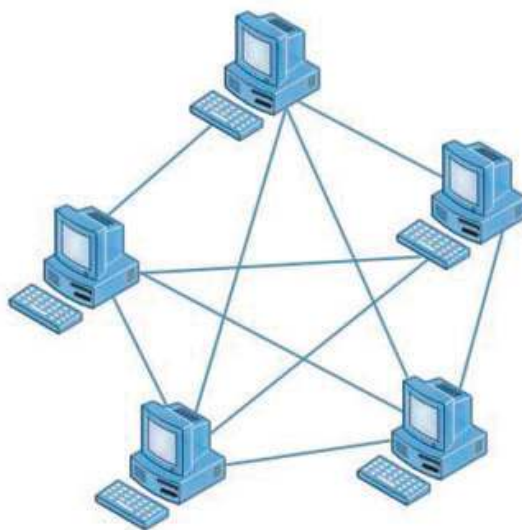


### *Characteristics of ring topology*

- No signal amplification is required as each node amplifies the signal.
- Requires less cable length and hence is cost effective.
- If one node fails, entire network will fail.
- Addition of nodes to the network is difficult.

### **11.6.4 Mesh topology**

In mesh topology, every node is connected to other nodes. So there will be more than one path between two nodes as shown in Figure 11.26. If one path fails, the data will take another path and reach the destination.



*Fig. 11.26 : Mesh topology*

### *Characteristics of mesh topology*

- Network will not fail even if one path between the nodes fails.
- Expensive because of the extra cables needed.
- Very complex and difficult to manage.



*Identify the network topology used in your school lab.*

**Let us do**

## **11.7 Types of networks**

A computer network may span any amount of geographical area. It can be on a table, in a room, in a building, in a city, in a country, across continents or around the world. On the basis of the area covered, computer networks are classified as:

- PAN - Personal Area Network
- LAN - Local Area Network
- MAN - Metropolitan Area Network
- WAN -Wide Area Network

### 11.7.1 Personal Area Network (PAN)

PAN is a network of communicating devices (computer, mobile, tablet, printer, etc.) in the proximity of an individual. It can cover an area of a radius with few meters (refer Figure 11.27).

When we transfer songs from one cell phone to another or from a PC to an MP3 player, a PAN is set up between the two. PAN can be set up using guided media (USB) or unguided media (Bluetooth, infrared).



Fig. 11.27 : Personal Area Network

### 11.7.2 Local Area Network (LAN)

LAN is a network of computing and communicating devices in a room, building, or campus. It can cover an area of radius with a few meters to a few Kilometers. A networked office building, school or home usually contains a single LAN, though sometimes one building can contain a few small LANs (Like some schools have independent LANs in each computer lab) as shown in Figure 11.28. Occasionally a LAN can span a group of nearby buildings.

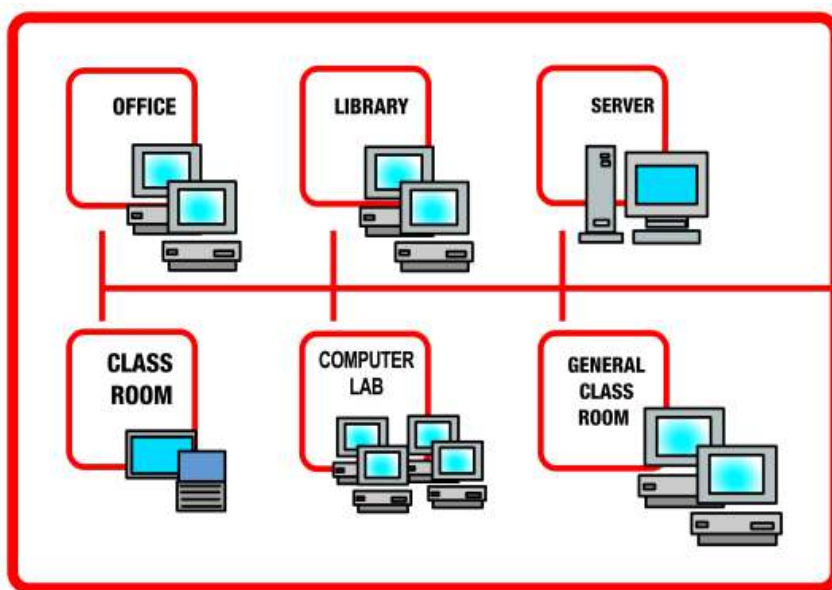


Fig. 11.28 : Local Area Network

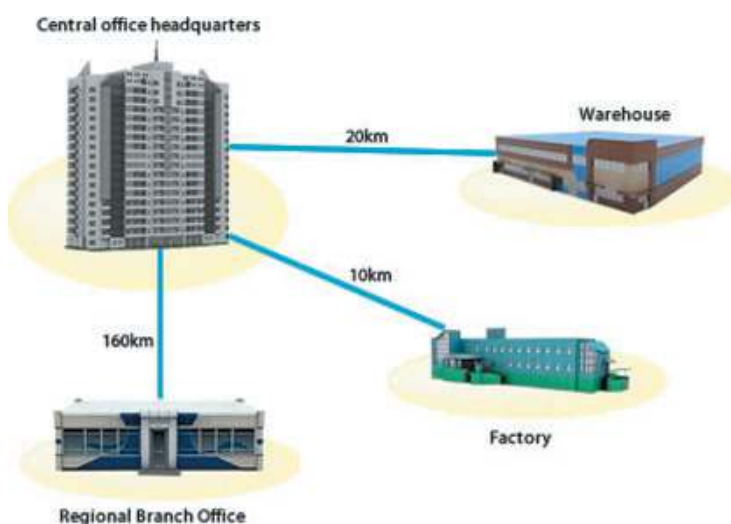


In addition to operating in a limited space, a LAN is owned, controlled and managed by a single person or an organisation.

LAN can be set up using wired media (UTP cables, coaxial cables, etc.) or wireless media (infrared, radio waves, etc.). If a LAN is setup using unguided media, it is known as WLAN (Wireless LAN).

### 11.7.3 Metropolitan Area Network (MAN)

MAN is a network of computing and communicating devices within a city. It can cover an area of a few Kilometers to a few hundred Kilometers radius. MAN is usually formed by interconnecting a number of LANs and individual computers. All types of communication media (guided and unguided) are used to set up a MAN. MAN is typically owned and operated by a single entity such as a government body or a large corporation (refer Figure 11.29).



*Fig. 11.29 : Metropolitan Area Network*

### 11.7.4 Wide Area Network (WAN)

WAN is a network of computing and communicating devices crossing the limits of a city, country or continent. It can cover an area of over hundreds of Kilometers in radius. WAN usually contain a number of interconnected individual computers, LANs, MANs and maybe other WANs. All types of communication media (guided and unguided) are used to set up a WAN as shown in Figure 11.30. The best known example of a WAN is the Internet. Internet is considered as the largest WAN in the world. A network of ATMs, banks, government offices, international organisations,



offices, etc. spread over a country, continent or covering many continents are examples of WAN.

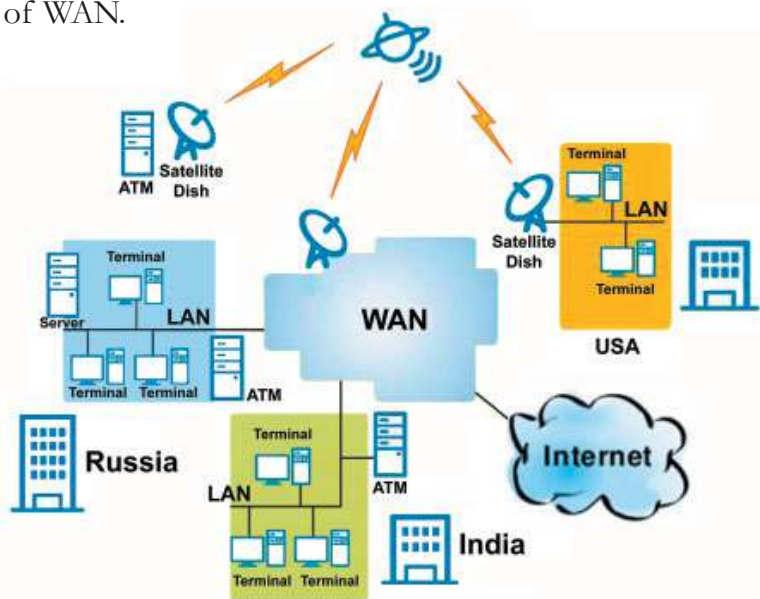


Fig. 11.30 : Wide Area Network

Table 11.1 summarises the characteristics of PAN, LAN, MAN and WAN.

| Parameter             | PAN                               | LAN                                                         | MAN                                                 | WAN                                       |
|-----------------------|-----------------------------------|-------------------------------------------------------------|-----------------------------------------------------|-------------------------------------------|
| Area covered          | Small area<br>(Up to 10 m radius) | A few meters to<br>a few Kilometers<br>(Up to 10 Km radius) | A city and its<br>vicinity (Up to<br>100 Km radius) | Entire country,<br>continent,<br>or globe |
| Transmission<br>speed | High speed                        | High speed                                                  | Moderate<br>speed                                   | Low speed                                 |
| Networking cost       | Negligible                        | Inexpensive                                                 | Moderately<br>expensive                             | Expensive                                 |

Table 11.1 : Characteristics summary of PAN, LAN, MAN, WAN

## 11.8 Logical classification of networks

This classification is based on the role of computers in the network and division falls into two categories: peer-to-peer and client-server.

### 11.8.1 Peer-to-Peer

A peer-to-peer network has no dedicated servers. Here a number of computers are connected together for the purpose of sharing information or devices. All the computers are considered equal. Any computer can act as a client or as a server at any instance. This network is ideal for small networks where there is no need for dedicated servers, like home network or small business establishments or shops.

### 11.8.2 Client-Server

The client-server concept is the driving force behind most of the networks. It is similar to going to a restaurant, reading the menu, calling the waiter (server) and then ordering one's preference from the menu. If the ordered item is available in the restaurant at that time, it is 'served' to whom the order was placed (client), else the request is refused.

In a network, the client-server architecture consists of high-end computer (called server) serving lower configuration machines called clients. A server provides clients with specific services (responses) upon client's request. The services include sharing of data, software and hardware resources. Figure 11.31 shows the general client-server architecture.

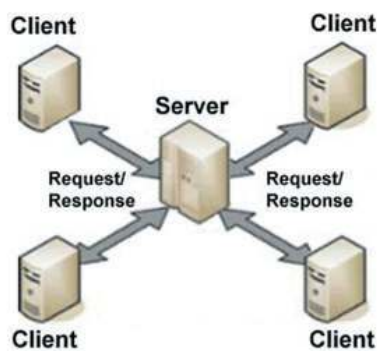


Fig. 11.31 : Client - Server

Client-server architecture is an example for centralised software management. When software is loaded on the server and shared among the clients, changes made to the software in the server will reflect in the clients also. So there is no need to spend time and energy for installing updates and tracking files independently on the clients.

*Classifications for servers are*

- File server** - A computer that stores and manages files for multiple users on a network.
- Web server** - A computer dedicated to responding to requests for web pages.
- Print server** - Redirects print jobs from clients to specific printers.
- Database server** - Allows authorised clients to view, modify and/or delete data in a common database.

#### Check yourself



- In bus topology, when the signal reaches the end of the bus, \_\_\_\_\_ absorbs the signal and removes it from the bus.
- In \_\_\_\_\_ topology each node is directly connected to a hub/switch.
- In which topology is every node connected to other nodes?
- Categorise and classify the different types of networks given below:  
ATM network, Cable television network, Network within the school, Network at home using bluetooth, Telephone network, Railway network
- What is PAN?
- What is a peer-to-peer network?

## 11.9 Identification of computers over a network

Imagine that you are in India and you wish to write a letter to your friend in America. What would you do? You write a letter, put it in an envelop, write your friend's address on it and write your address on the back. When the letter is posted in a post office in India, it is stamped with a unique seal and date. After going through a feasible route the letter reaches the post office in America, where it is stamped again with a unique seal and date. Then, the postman makes sure that it reaches the specified addressee. In a network, data is sent in the form of packets in a similar way.

Once a network has been set up, the nodes can communicate among themselves. But for proper communication, the nodes should be uniquely identifiable. If node X sends some information to node Y on a network, then it is mandatory that nodes X and Y are uniquely identifiable on the network. Let us see how this is achieved.

### 11.9.1 Media Access Control (MAC) address

A Media Access Control (MAC) address is a universally unique address (12 digit hexadecimal number) assigned to each NIC (Network Interface Card) by its manufacturer. This address is known as the MAC address. It means that a machine with an NIC can be identified uniquely through the MAC address of its NIC. MAC address of an NIC is permanent and never changes.

MAC addresses are 12-digit hexadecimal (or 48 bit binary) numbers. By convention, MAC addresses are usually written in one of the following two formats:

**MM : MM : MM : SS : SS : SS** or **MM – MM – MM – SS – SS – SS**

The first half (MM:MM:MM) of a MAC address contains the ID number of the adapter manufacturer. The second half (SS:SS:SS) of a MAC address represents the serial number assigned to the adapter (NIC) by its manufacturer. For example, in the following MAC address,

00:A0:C9 : 14:C8:35

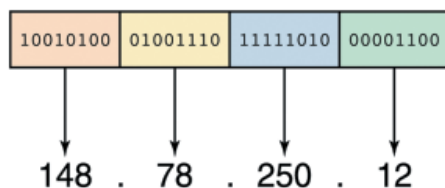
The prefix 00:A0:C9 indicates that the manufacturer is Intel Corporation. And the last three numbers 14:C8:35 are given by the manufacturer (Intel in this example) to this NIC.

### 11.9.2 Internet Protocol (IP) address

An IP address is a unique 4 part numeric address assigned to each node on a network, for their unique identification. IP address is assigned to each machine by the network administrator or the Internet Service Provider. An IP address is a group of four bytes (or 32 bits) each of which can be a number from 0 to 255.

To make it easier for us to remember, IP addresses are normally expressed in decimal format as a “dotted decimal number” as indicated in Figure 11.32.

On a network, the IP address of a machine is used to identify it. IP protocol identifies a machine with its IP address to route the packets.



*Fig. 11.32 : IP address*

There are two versions of IP addresses: version 4 (IPv4) and version 6 (IPv6). IPv4 uses 32 bits and IPv6 uses 128 bits for an IP address. Using IPv4 only  $2^{32}$  (approximately 4 billion) distinct devices can be addressed.

As the number of devices which need to be networked (mobile phones, home appliances, personal communication devices, etc.) is increasing at a very fast pace, IPv4 addresses are being exhausted. To address this problem IPv6 was developed and it is now being deployed. Using IPv6,  $2^{128}$  (approximately 4 billion  $\times$  4 billion 4 billion) distinct devices can be addressed.



*Identify the IP and MAC Id of each networked machine in your school and prepare a table as follows. (Use ipconfig /all at command prompt).*

**Let us do**

| Sl No | Computer Name | IP | MAC |
|-------|---------------|----|-----|
| 1.    |               |    |     |
| 2.    |               |    |     |
| 3.    |               |    |     |

## 11.10 Network protocols

A network protocol is the special set of rules to be followed in a network when devices in the network exchange data with each other. Each protocol specifies its own rules for formatting data, compressing data, error checking, identifying and making connections and making sure that data packets reach its destination.

Several computer network protocols have been developed for specific purposes and environments. Some commonly used protocols are TCP/IP, SPx/IPx, etc.

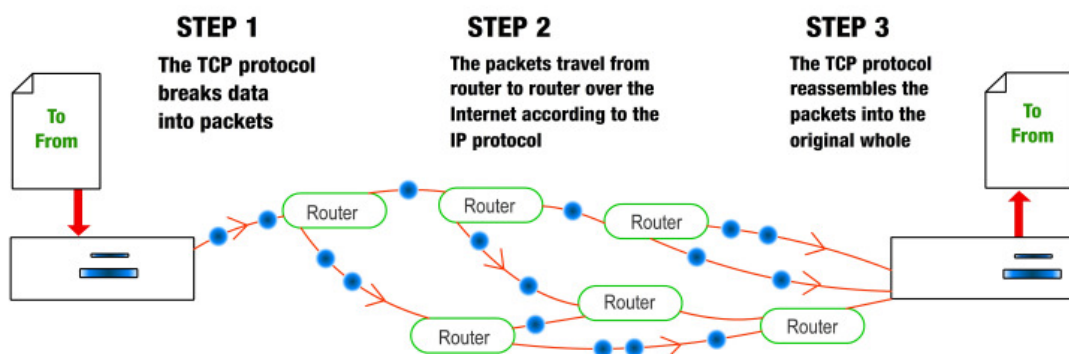
### TCP/IP

TCP/IP, Transmission Control Protocol/Internet Protocol, is a suite of communications protocols used to interconnect network devices on the local networks and the Internet. TCP/IP defines rules for how electronic devices (like



computers) should be connected to the Internet and how data should be transmitted between them.

When data is to be sent from one computer to another over Internet, it is first broken into smaller packets by TCP and then sent. When these packets are received by the receiving computer, TCP checks packets for errors. If errors are found, TCP submits requests for retransmission; else packets are assembled into the original message according to the rules specified in TCP protocol. Figure 11.33 shows the steps involved in the working of TCP/IP protocol. Delivery of each of these packets to the right destinations is done by Internet protocol (IP). Even though different packets of the same message may be routed differently, they will reach the same destination and get reassembled there. HTTP, FTP and DNS are three sub protocols of TCP/IP protocol suite.

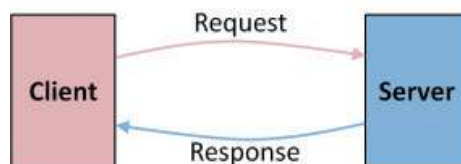


*Fig. 11.33 : How TCP/IP works*

### a. HTTP

HTTP stands for Hypertext Transfer Protocol. It is a standard protocol for transferring requests from client-side and to receive responses from the server-side. The HTTP client (browser) sends a HTTP request to the HTTP server (web server) and server responds with a HTTP response. This pair of request and response is called an HTTP session (refer Figure 11.34).

The response from the server can be static such as a file already stored on the server, or dynamic, such as the result of executing a piece of code by the server as per the request from the client.



*Fig. 11.34 : An HTTP session*



*The two important characteristics of HTTP are*

- HTTP is transmission medium independent.
- HTTP is stateless (The server and client are aware of each other only during a request or response. Afterwards, each forgets the other).

### **b. FTP**

FTP stands for File Transfer Protocol. It is a standard for exchanging of data and program files across a network. FTP is the easiest way to transfer files between computers via the Internet. It uses TCP and IP to perform uploading and downloading. A FTP client program installed in the computer can help in the uploading (sending files to another computer) and downloading (receiving files from another computer) of files.

FTP uses client–server architecture in servers with security features, username and password protection for file transfer. An FTP client program (Filezilla, Cute FTP, etc.) installed in the computer can help in the easy uploading and downloading of files.

### **c. DNS**

DNS stands for Domain Name System. DNS returns the IP address of the domain name, that we type in our web browser's address bar. (like mobile phone automatically dialing the phone number when we select a name from contact list).

The DNS system has its own network. DNS implements a database to store domain names and IP address information of all web sites on the Internet. DNS assumes that IP addresses do not change (statically assigned). If one DNS server does not know how to translate a particular domain name, it asks another one, and so on, until the correct IP address is returned.



*Find and prepare notes on five protocols other than TCP/IP, HTTP, FTP, DNS.*

**Let us do**

## **11.11 Uniform Resource Locator (URL)**

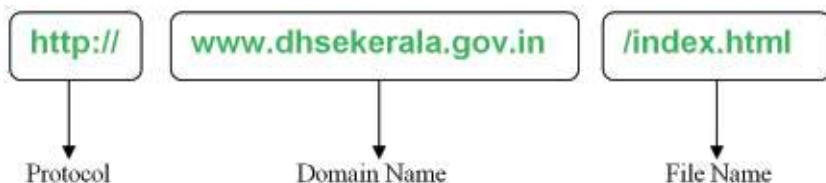
URL stands for Uniform Resource Locator. URL is a formatted text string used by web browsers, e-mail clients and other software to identify a network resource on the Internet. Every resource on the Internet has a unique URL. Network resources are files that can be plain web pages, other text documents, graphics, programs, etc.



URL consists of letters, numbers and punctuations. URL string can be divided into three parts.

- Network protocol (also called the scheme)
- Domain name (Host name or address)
- File name

For example the URL <http://www.dhsekerala.gov.in/index.html> has three parts as shown in Figure 11.35. The detailed description of these three parts are given below:



*Fig: 11.35 : Components of an URL*

#### a. Protocol

The protocol enables the browser to know what protocol is used to access the information specified in the domain.

#### b. Domain name

Domain name is the name assigned to a server through the Domain Name System (DNS). Domain names are used in URLs to identify the particular web server. Domain names provide Internet users with a short name that is easy to remember. Whenever we have to communicate with a computer on Internet, we can do so by using its IP address. But it is practically impossible for a person to remember the IP addresses of all the computers, one may have to communicate with. Therefore, a system has been developed which assigns names to computers (web servers) and maintains a database of these names and their corresponding IP addresses. These names are called domain names. Examples of some domain names are [dhsekerala.gov.in](http://dhsekerala.gov.in), [keralareresults.nic.in](http://keralareresults.nic.in), [google.com](http://google.com), [gmail.com](http://gmail.com), etc.

A domain name usually has more than one part: top level domain name or primary domain name and sub-domain name(s). For example, in the domain name above, 'in' is the primary domain name; 'gov' is the sub-domain of in and 'dhsekerala' is the sub-domain of 'gov'. There are only a limited number of top level domains and these are divided into two categories: Generic Domain Names and Country-Specific Domain Names. Examples of generic and country specific domain names are given in Table 11.2.



| Generic Domain Names                  | Country Specific Domain Names       |
|---------------------------------------|-------------------------------------|
| <b>.com</b> Commercial business       | <b>.in</b> India                    |
| <b>.edu</b> Educational institutions  | <b>.au</b> Australia                |
| <b>.gov</b> Government agencies       | <b>.ca</b> Canada                   |
| <b>.mil</b> Military                  | <b>.ch</b> China                    |
| <b>.net</b> Network organizations     | <b>.jp</b> Japan                    |
| <b>.org</b> Organizations (nonprofit) | <b>.us</b> United States of America |

*Table 11.2 : Generic and country specific domain names*

### c. File name

It is the file to be opened. In the example given in Figure 11.35, 'index.html' is the file that is to be accessed from the web server specified by the domain name.



**Let us do**

*Make a list of valid URL, containing two examples for each generic domain name and country specific domain name. Also note down the file name opened by default (for file look at the URL in address bar after the site is opened).*



### Let us sum up

We learned about computer networks, the essential technology of the century, in this chapter. Importance of network was discussed by highlighting the various advantages it provides. We discussed the various communication media and their pros and cons. The devices used at various situations while forming a network was also discussed. Before discussing the types of network, we learned the different ways a network could be formed by discussing various topologies. We then discussed protocol and how TCP/IP send/receive data across the network. Methods to uniquely identify a node in the network were introduced and finally we concluded discussing URL.



## Learning outcomes

After the completion of this chapter the learner will be able to

- identify and choose a communication medium.
- compare different types of network.
- explain the logical classification of networks.
- identify how data is sent across networks.
- design small networks.
- explain how a node is identified in a network.
- identify the various parts of a URL.

## Sample questions

### Very short answer type

1. The transmission media which carry information in the form of light signals is called \_\_\_\_\_.
  - a. Coaxial
  - b. Twisted
  - c. WiFi
  - d. Optical Fiber
2. Different networks with different protocols are connected by a device called \_\_\_\_\_.
  - a. Router
  - b. Bridge
  - c. Switch
  - d. Gateway
3. In \_\_\_\_\_ topology, the failure of any one computer will affect the network operation.
  - a. Bus
  - b. Ring
  - c. Star
  - d. none of these
4. To transmit signals from multiple devices through a single communication channel simultaneously, we use \_\_\_\_\_ device.
  - a. Modem
  - b. Switch
  - c. Router
  - d. Multiplexer
5. Bluetooth can be used for
  - a. long distance communication
  - b. short distance communication
  - c. in mobile phones only
  - d. none of the above
6. Satellite links are generally used for
  - a. PANs
  - b. LANs
  - c. MANs
  - d. all of the above



7. A domain name maps to
- URL
  - an IP address
  - website
  - all of the above

### Short answer type

- Define bandwidth.
- Switch and Hub are two devices associated with networking. Differentiate them.
- What is an IP address? Give an example.
- What is TCP/IP? What is its importance?
- Define a computer network.
- What is Bluetooth?
- What is a Modem?
- Distinguish between router and gateway.
- Explain the need for establishing computer networks.
- What are the uses of computer networks?
- What is the limitation of microwave transmission? How is it eliminated?
- Briefly describe the characteristics of Wi-Fi.
- An International School is planning to connect all computers, spread over distance of 45 meters. Suggest an economical cable type having high-speed data transfer, which can be used to connect these computers.
- What is NIC? What is its importance in networking?
- Suppose that you are the administrator of network lab in one Institution. Your manager directed you to replace 10 Mbps switch by 10 Mbps Ethernet hub for better service. Will you agree with this decision? Justify your answer.
- You need to transfer a biodata file stored in your computer to your friend's computer that is 10 kms away using telephone network
  - Name the device used for this at both ends.
  - Explain how the file is send and received inside the device, once a connection between two computers is established.
- When is a repeater used in a computer network?
- Compare infrared and Bluetooth transmission.
- Identify and explain the device used for connecting a computer to a telephone network.
- Briefly explain LAN topologies.



21. Briefly describe TCP/IP protocol.
22. What is a MAC address? What is the difference between a MAC address and an IP address?

### Long answer type

1. How are computer networks classified, based on size?
2. Compare different LAN topologies.
3. Explain various types of guided communication channels.
4. Compare different unguided media.
5. Define the term protocol. Briefly describe any two communication protocols.
6. Briefly describe the various communication devices used in computer networks.
7. Which is/are communication channel(s) suitable for the following situations?
  - a. Setting up a LAN.
  - b. Transfer of data from a laptop to a mobile phone.
  - c. Transfer of data from one mobile phone to another.
  - d. Creating a remote control that can control multiple devices in a home.
  - e. Very fast communication between two offices in two different countries.
  - f. Communication in a hilly area.
  - g. Communication within a city and its vicinity where cost of cabling is too high.

## Key Concepts

- **History of the Internet**
- **Connecting the computer to the Internet**
- **Types of connectivity**
  - Dial up
  - Wired broadband
  - Wireless broadband
  - Internet access sharing methods
- **Services on Internet**
  - www
  - Search engines
  - Email
  - Social media
- **Cyber security**
  - Computer virus, worm, Trojan horse, spams, hacking, phishing, denial of service attack, man-in-the-middle attacks
- **Preventing network attacks**
  - Firewall, antivirus scanners, cookies
- **Guidelines for using computers over Internet**
- **Mobile computing**
- **Mobile communication**
  - Generations in mobile communication
  - Mobile communication services
- **Mobile operating system**

## Internet and Mobile Computing

In the previous chapter we saw that the Internet is the largest computer network in the world. We use the Internet to check SSLC results, to submit applications for higher secondary school admissions and check its status, to get information about various types of scholarships and to submit applications for receiving them, etc. Can you imagine life without the Internet today? It would be difficult for us to manage all the above tasks without it. Internet has definitely made life easier for us. It has influenced our daily life to a great extent. Because of its wide popularity and increase in use even the television sets with facilities for Internet connectivity have come up in markets.

People generally use the Internet to search information, access e-mails, make bill payments, for online shopping, online banking, to connect with people in social networking sites, etc. The reach of Internet is very vast and it helps in reducing cost and time. Issues like online intrusion to privacy, online fraud, cyber-attacks, etc. are becoming common now. Apart from the Internet and its access methods, let us discuss the various services provided by the Internet like search engines, e-mail, social media and about the threats and preventive measures while using Internet in this chapter.







## 12.1 History of the Internet

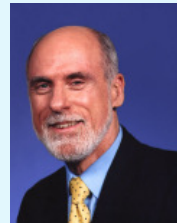
The Internet started as a small network through a project by the United States Department of Defence by the name ARPANET (Advanced Research Projects Agency Network). During 1970s this military network was connected to the computers of Universities and companies that worked for the Department of Defence. In 1984 the military network split from ARPANET to form MILNET to be used by the American military only. ARPANET which used TCP/IP protocol for communication was thereafter used for scientific research and information sharing. Later, several other networks merged with ARPANET to form a large network. ARPANET is considered as the first wide area network (WAN). Vinton Gray Cerf who was instrumental in the development of Internet and TCP/IP protocol, is considered as the father of Internet.



Vinton Gray Cerf (1943 - ) popularly called Vint Cerf, an American computer scientist, is widely known as 'Father of the Internet'. He was instrumental in the initial development of Internet

along with his fellow American computer scientist Bob Kahn.

He worked for the United States Department of Defence Advanced Research Projects Agency (DARPA) and had a key role in the development of TCP/IP protocol. He was also involved in the formation of ICANN.



In 1989, Tim Berners Lee, a researcher, proposed the idea of World Wide Web (WWW). Tim Berners Lee and his team are credited with inventing Hyper Text Transfer Protocol (HTTP), HTML and the technology for a web server and a web browser. Using hyperlinks embedded in hypertext the web developers were able to connect web pages. They could design attractive webpages containing text, sound and graphics. This change witnessed a massive expansion of the Internet in the 1990s.



*Fig. 12.1: Tim Berners Lee (1955 - )*

Various types of computers loaded with diverse operating systems in different organisations at geographically distant locations joined this network making it a global phenomenon. TCP/IP protocol is used as the communication protocol for Internet. Any computer that joins Internet should follow the TCP/IP protocol. In 1998, Internet Corporation for Assigned Names and Numbers (ICANN) was established. ICANN does not control the Internet content; rather it develops policies on the Internet's Uniform Resource Locators (URL).





Today, Internet is the largest public network that connects billions of computers all over the world and provides several services like searching, e-mail, file transfer, social networking, etc. The **Internet** is an interconnected system of computer networks that serves the users all over the world.

An **intranet** is considered as a private computer network similar to Internet that uses TCP/IP protocol to share information, software or services within an organisation. An intranet can host websites, provide e-mail service, file transfer and other services available on Internet.

When an intranet is made accessible to some computers that are not part of a company's private network it is called an **extranet**. A network that allows vendors and business partners to access a company resource can be considered as an example of extranet.

## 12.2 Connecting the computer to the Internet

As we know today, the Internet has become very popular and almost all organisations and people around the world are joining it. Earlier, people used the Internet to search for information and check e-mails only, but today It is used to book train tickets, recharge mobile phones, Internet banking and a lot more. Therefore almost all of us require an Internet connection in our computers or mobile devices.

The following are the hardware and software requirements for connecting a computer to the Internet:

- A computer with Network Interface Card (wired/wireless) facility and an operating system that supports TCP/IP protocol
- Modem
- Telephone connection
- An Internet account given by an Internet Service Provider (ISP)
- Software like browser, client application for e-mail, chat, etc.

Nowadays desktop computers or laptops are not the only devices that we use to connect to the Internet. People have also started using tablets, smart phones, etc. to browse the Internet. Some of these devices come with built-in modems, whereas others use a wireless dongle or wireless connection from a modem to access the Internet.

## 12.3 Types of connectivity

Today most websites use images and multimedia content to make webpages more attractive. Several websites provide videos that can be downloaded or viewed on



the Internet. Instead of distributing software in CDs or other storage media, it is now distributed online by various vendors. The latest trend shows that software like word processors, spreadsheets, antivirus, etc. are used online on a rental basis instead of installing it on each computer. In all these cases, a large volume of data is transferred online. Therefore the speed or data transfer rate of the Internet is an important aspect. **Data transfer rate** is the average number of bits transferred between devices in unit time.

1 kbps = 1000 bits per second

1 Mbps = 1000 kbps

1 Gbps = 1000 Mbps



**Difference between unit symbols b and B**

b stands for bit

B stands for Byte

**Difference between unit symbols k and K**

$$k = 1000 = 10^3$$

$$K = 1024 = 2^{10}$$

Here 'k' is a decimal unit and 'K' is a binary unit of measurement. But for Mega, Giga and Tera, both decimal and binary units use 'M', 'G' and 'T' as symbols respectively. They are differentiated from the context in which they are used.

Note that data transfer rate is measured in decimal units and memory is measured in binary.

The main factor that decides Internet access speed is the type of connectivity we choose to link to the Internet. Internet connectivity is classified based on the speed of the connection and the technology used. They can be broadly classified as dial-up connectivity, wired broadband connectivity and wireless broadband connectivity. The data transfer rates of each type of connectivity may vary as technology advances.

### 12.3.1 Dial-up connectivity

A dial-up connection uses the conventional telephone line and a dial-up modem to dial and connect to the server at the Internet Service Provider (ISP). Figure 12.2 shows the dial-up connectivity system. As the connection is made by dialing, it takes time to connect to the server at the ISP. This connection commonly uses a 56 kbps modem that can transmit data up to a maximum speed of 56 kbps. This slow

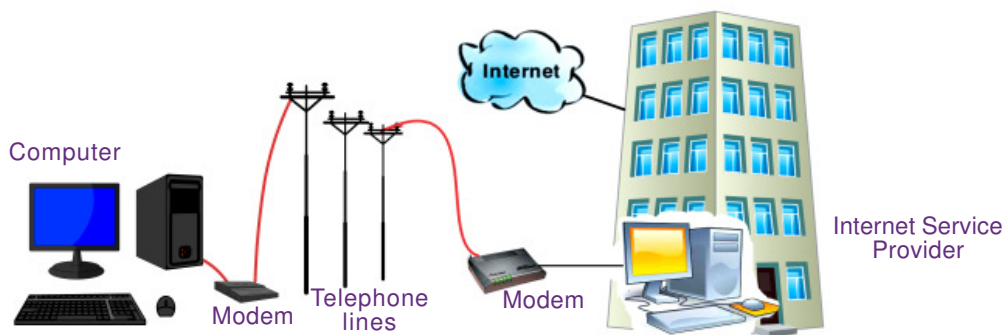


Fig. 12.2: Dial-up connectivity

connection is comparatively less costly when compared to other types of connections. Another disadvantage is that a dial-up connection requires exclusive use of the telephone line, i.e., while accessing Internet, we cannot make or receive telephone calls (voice calls). Nowadays, broadband connections that have a higher speed are replacing dial-up connections.

### 12.3.2 Wired broadband connectivity

The term broadband refers to a broad range of technologies that help us to connect to the Internet at a higher data rate (speed). Wired broadband connections are 'always on' connections that do not need to be dialled and connected. Broadband connections use a broadband modem (refer Figure 12.3) and allow us to use the telephone even while using the Internet. Table 12.1 shows the comparison between dial-up and wired broadband connections.



Fig. 12.3: Broadband modem

| Dial-up connection                                                                                                                                                                                             | Wired broadband connection                                                                                                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>• Slow connection, speed upto 56 kbps</li> <li>• Requires dialing to connect to ISP</li> <li>• Uses telephone line exclusively</li> <li>• Uses dial-up modem</li> </ul> | <ul style="list-style-type: none"> <li>• High speed connection, speed usually higher than 256 kbps</li> <li>• Always on connection</li> <li>• Simultaneous use of voice and Internet</li> <li>• Uses broadband modem</li> </ul> |

Table 12.1: Comparison between dial-up and wired broadband connections

Popular broadband technologies are Integrated Services Digital Network (ISDN), Cable Internet, Digital Subscriber Line (DSL), Leased Lines and Fiber to the Home (FTTH).

#### a. Integrated Services Digital Network (ISDN)

ISDN is a broadband service capable of transporting voice and digital data. Most ISDN lines offered by telephone companies give users two lines. The users can use



one line for voice and the other for data, or they can use both lines for data. ISDN lines are capable of offering data transfer rates upto 2 Mbps.

#### **b. Cable Internet**

Cable Internet access provides Internet access using coaxial cables laid for television signal transmission to our homes. The service provider uses a cable modem at our home to connect our computer to cable network. Cable TV systems are designed to carry large bandwidth and therefore cable Internet can provide speeds between 1 Mbps to 10 Mbps.

#### **c. Digital Subscriber Line (DSL)**

DSL is another broadband service that provides connection to the Internet through standard telephone lines. DSL allows the user to use copper telephone lines for both Internet communication and for making voice calls simultaneously. It is composed of several subcategories, the most common being Asymmetric Digital Subscriber Line (ADSL). ADSL is a communication technology that allows faster flow of information over a telephone line. The down stream speed of ADSL services typically ranges from 256 kbps to 24 Mbps. This connection requires an ADSL modem at our homes/offices. ADSL is the most popular broadband service available in India.

#### **d. Leased Line**

Leased lines are dedicated lines used to provide Internet facility to ISPs, business, and other large enterprises. An Internet leased line is a premium Internet connection that provides speed in the range from 2 Mbps to 100 Mbps and is comparatively costly. This is why leased lines are used only for connecting large campus of organisations like educational institutions to Internet.

#### **e. Fibre to the Home (FTTH)**

Fibre to the Home (FTTH) uses optical fibres for data transmission. Optical fibres are laid from the ISP to our homes. FTTH technology has been accepted worldwide to implement high speed Internet to the home. Since optical fibres are known to have high bandwidth and low error rates, they provide very high speed connectivity. A Network Termination Unit (NTU) is installed in our homes, which is connected to our computer through an FTTH modem.

### **12.3.3 Wireless broadband connectivity**

Wireless broadband connectivity provides almost the same speed as that of a wired broadband connection. The popular wireless broadband accesses are Mobile Broadband, Wi-MAX, Satellite Broadband and Wi-Fi. Some of the wireless modems available for use to connect to Internet are shown in Figure 12.4.



*Fig. 12.4 : Wireless broadband modems*

**a. Mobile broadband**

Mobile broadband is wireless Internet access using mobile phone, USB wireless modem, tablet or other mobile devices. The modem is built into mobile phones, tablets, USB dongles, etc. Mobile broadband offers the freedom to allow the user to access the Internet from anywhere on the move. This technology uses the cellular network of mobile phones for data transmission. The mobile technology for data transmission has been changing from 2<sup>nd</sup> Generation (2G) through 3<sup>rd</sup> Generation (3G) to the current 4<sup>th</sup> Generation (4G). The speed of data transmission increases with the progression of generations of mobile technology.

**b. Wi-MAX**

In the previous chapter we learned that Worldwide Interoperability for Microwave Access (Wi-MAX) is used as an alternative for wired broadband. Wi-MAX offers a Metropolitan Area Network which can provide wireless Internet upto a distance of 50 Km. Connectivity is provided using devices like Wi-MAX handsets, USB dongles, devices embedded in laptops, etc. that have a Wi-MAX modem integrated in it. This technology provides a maximum connection speed of upto 70 Mbps.

**c. Satellite broadband**

Satellite broadband technology is a method by which Internet connectivity is provided through a satellite. A Very Small Aperture Terminal (VSAT) dish antenna and a transceiver (transmitter and receiver) are required at the user's location. A modem at the user's end links the user's computer with the transceiver. Download speed is upto 1 Gbps for this technology. It is among the most expensive forms of broadband Internet access. They are used by banks, stock exchanges, governments, etc. and also for Internet access in remote areas.

**12.3.4 Internet access sharing methods**

An Internet connection can be shared among several computers using a LAN, Wi-Fi network or Li-Fi network.

**a. Using LAN**

The Internet connected to a computer in a Local Area Network (LAN) can be shared among other computers in the network. This can be done either using features available in the operating system or using any proxy server software available in the market. Sharing can also be done by connecting computers directly to the router using a cable.



### b. Using Wi-Fi network

We have heard of Wi-Fi campuses in large educational institutions, coffee shops, shopping malls, hotels, etc. We also know that some of the broadband modems at our homes and schools offer Wi-Fi Internet access. Wi-Fi is a popular short distance data transmission technology that is used for network access, mostly Internet. Wi-Fi locations receive Internet connection through any one of the above mentioned wired or wireless broadband access methods, as discussed in the previous section. They provide us Internet connectivity through a Wi-Fi router or a wireless network access point. Such an access point, popularly called hotspot, has a range of about 100 meters indoors and a greater range outdoors. We access Internet in our Wi-Fi enabled devices like laptops, tablets, mobile phones, etc. through these hotspots. A drawback of Wi-Fi is that it is less secure than wired connections.



Fig. 12.5 : Wi-Fi network

### c. Using Li-Fi network

Li-Fi (Light Fidelity) is a fast optical version of Wi-Fi, which uses visible light for data transmission. The main component of this communication is a bright LED (Light Emitting Diode) lamp that can transmit data and a photo diode that serves as the receiver. LEDs can be switched on and off to generate a binary string of 1s and 0s. The flickering of this LED is so fast that the human eye cannot detect it. A data rate of over 100 Mbps is possible using this technique as light offers very high bandwidth. Another advantage is that since Li-Fi uses light, it can be used in aircrafts and hospitals where radio waves may cause interference. It can also be used underwater where Wi-Fi does not work. It provides greater security as light cannot penetrate walls when compared to Wi-Fi. One of the shortcomings of Li-Fi is that it works only in direct line-of-sight. In future this technology can be further developed to use light bulbs as a source of Internet.

### Check yourself



1. ARPANET stands for \_\_\_\_\_.
2. Who proposed the idea of www?
3. The protocol for Internet communication is \_\_\_\_\_.
4. What do you mean by an 'always on' connection?
5. A short distance wireless Internet access method is \_\_\_\_\_.

**Let us do**

*Prepare a comparison chart on the different methods of Internet connection.*

## 12.4 Services on Internet

The Internet offers a variety of services. Services like WWW, e-mail, search engines, social media, etc. are widely used throughout the globe. In this section we shall discuss some of the services of Internet.

### 12.4.1 World Wide Web (WWW)

The World Wide Web (WWW) is a system of interlinked hypertext documents accessed via the Internet. It is a service on the Internet that uses Internet infrastructure. WWW is a huge client-server system consisting of millions of clients and servers connected together. Each server maintains a collection of documents and they can be accessed using a reference called Uniform Resource Locator (URL). These documents may contain text, images, videos and other multimedia content. It may also contain hyperlinks to documents on different servers. Selecting a hyperlink results in a request to fetch that document/web page from the server and display it. The WWW works by establishing hypertext links between documents anywhere on the network. Clients can access the documents on the servers using software called browser. A browser is responsible for properly displaying the documents.

#### a. Browser

A web browser is a software that we use to retrieve or present information and to navigate through web pages in the World Wide Web. The document to be displayed is identified using a URL. A URL consists of its DNS name and the file name to be retrieved. It also specifies the protocol for transferring the document across the network. A browser is capable of displaying text, images, hypertext links, videos, sounds, scripts (program code inside a web page), etc. in a web document/page. Most of WWW documents are created using Hyper Text Markup Language (HTML) tags and are called web pages. The web browser interprets these tags and displays a formatted page. It allows us to navigate through web pages using the hyperlinks available in web pages. Some common browsers are Google Chrome, Internet Explorer, Mozilla Firefox, Opera, and Safari. Icons of some popular browsers are shown in Figure 12.6. Some of these browsers have a mobile version that can be used in mobile operating systems.





Google Chrome

Internet Explorer

Mozilla Firefox

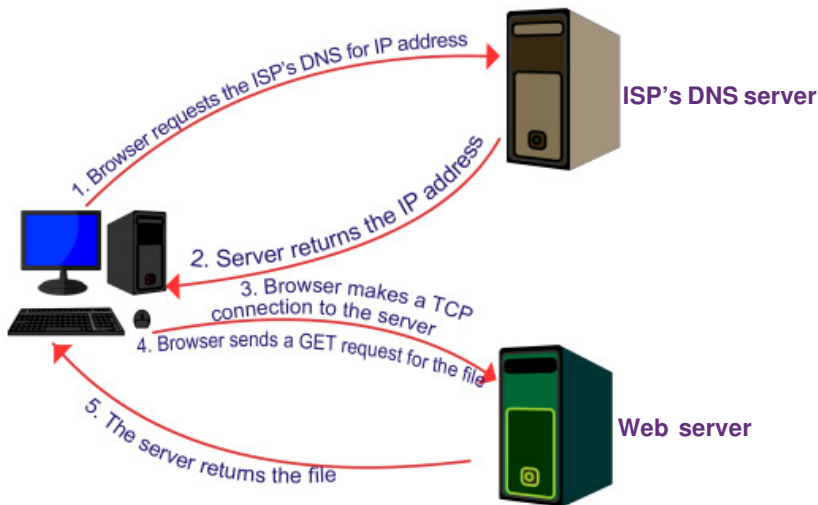
Opera

Safari

*Fig. 12.6 : Icons of popular browsers*

## b. Web browsing

All of us have visited web sites by entering the website address (URL) into web browsers and then using the hyperlinks in it to move through the web pages. Traversing through the web pages of World Wide Web is called web browsing. Major operations performed while web browsing are shown in Figure 12.7.

*Fig. 12.7 : Web browsing*

Suppose you wish to visit the website 'www.kerala.gov.in'. What will you do? You will enter this URL in the address box of the web browser and press **Enter** key. The steps a browser will follow to display a webpage may be summarised as follows.

1. The browser determines the URL (<http://www.kerala.gov.in>) entered.
2. The browser then sends a request to the DNS server of the user's ISP to get the IP address of the URL.
3. The ISP's DNS server replies with the IP address.
4. The browser then makes a TCP connection to the web server at the IP address ([www.kerala.gov.in](http://www.kerala.gov.in)).
5. Then it sends a GET request for the required file (web page) to the web server.
6. The web server returns the web page.

7. The TCP connection is released.
8. The browser processes the contents of the webpage and displays it.

### 12.4.2 Search engines

There are millions of pages available on the Internet that contain information on a variety of topics. But it is very difficult to search for a topic in this large collection of web pages. Internet search engine websites are special programs that are designed to help people to find information available in World Wide Web. Search engine programs search documents available on World Wide Web for specified keywords and return a list of the documents/web pages matching the keywords.

Let us discuss the technology behind these websites. Search engine websites use programs called web crawlers or spiders or robots to search the web. Web crawlers search the web pages stored in the different web servers and find possible keywords. The search engine website stores these keywords along with their URLs to form an index in the search engine's web servers. When we use the search engine website to search a particular

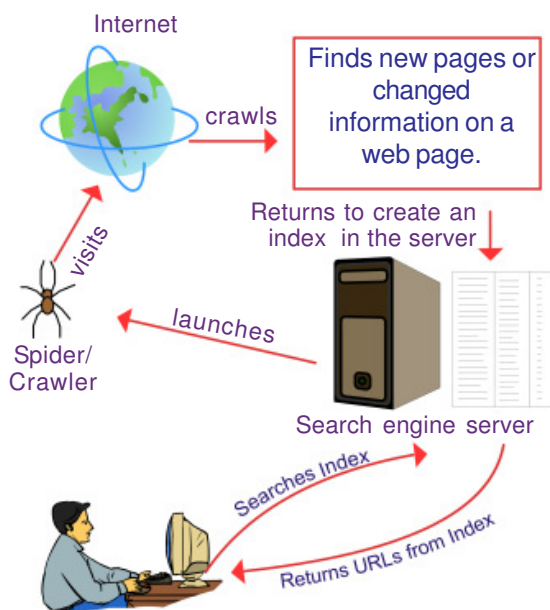


Fig. 12.8 : Working of a search engine

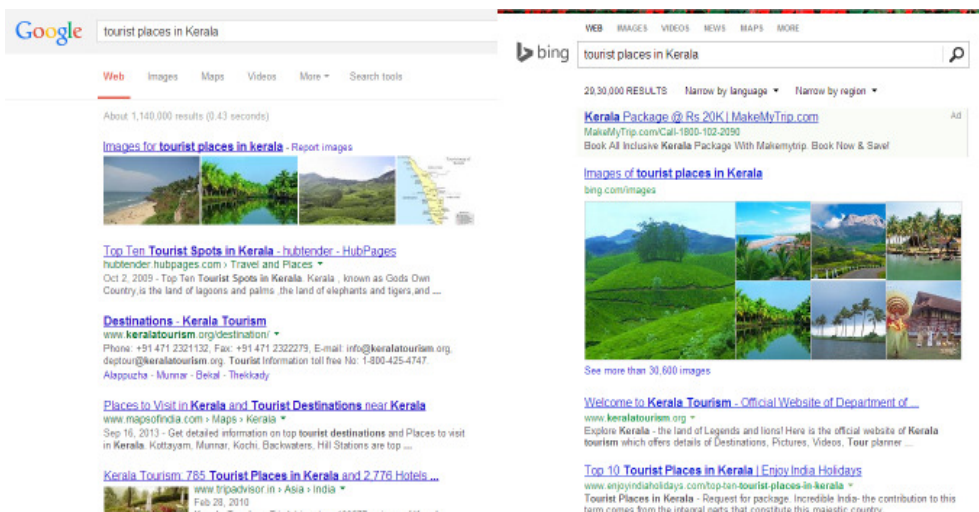


Fig. 12.9 : Search results of different search engines

topic (keyword), it does not search the World Wide Web. It only searches the index, which the web crawler programs have created in the search engine's web server for the topic/keyword. Search engines select a list of URLs where the particular topic is found from the index and displays it as the result. Figure 12.8 shows the working of a search engine.

Some of the most popular web search engine sites are Google, Bing, Yahoo Search, Ask, etc. Figure 12.9 shows the search results of different search engines.

### 12.4.3 E-mail

E-mail enables us to contact any person in the world in a matter of seconds. Billions of e-mail messages are sent over the Internet every day. **Electronic mail or e-mail** is a method of exchanging digital messages between computers over Internet.

E-mail has become an extremely popular communication tool. The e-mail will be delivered almost instantly in the recipient's mail box (Inbox). Apart from text matter, we can send files, documents, pictures, etc. as attachment along with e-mail. The same e-mail can be sent to any number of people simultaneously. Figure 12.10 shows a sample e-mail message.

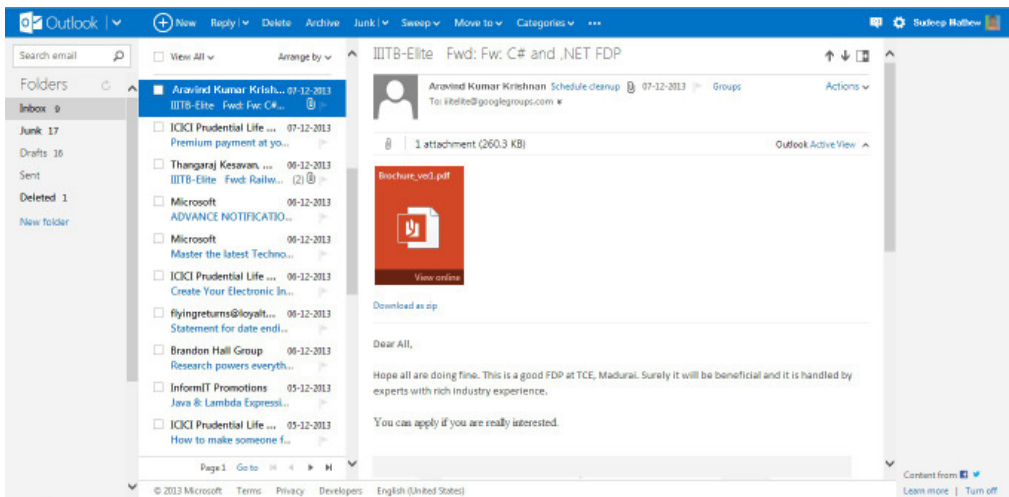


Fig. 12.10 – A sample e-mail message

Most of you will have an e-mail address. The structure of an e-mail address is: user name@domain name. An example of an e-mail address is

scertkerala@gmail.com

An e-mail address consists of two parts separated by @ symbol. The first part *scertkerala* is the username that identifies the addressee and the second part *gmail.com* is the domain name of the e-mail server, i.e., the name of the e-mail service provider.

E-mails can be accessed using websites like *gmail.com*, *hotmail.com*, etc. that provide web applications consisting of functions to send, receive, forward, reply and organise e-mails. Such a facility is popular and is commonly referred to as web mail.

E-mails can also be accessed using e-mail client software that is installed in our computers. Such software uses our e-mail address and password to retrieve e-mails from the e-mail service provider's server and store it in our computer. An e-mail client allows to send, receive and organise e-mail. The messages sent when the computer is offline are stored in the program and send later when computer is online. For receiving messages, e-mail client applications usually use either the Post Office Protocol (POP) or the Internet Message Access Protocol (IMAP). The popular e-mail client applications are Microsoft Outlook and Mozilla Thunderbird.

### a. Sections of an e-mail

A client software gives provisions to enter the following sections. Figure 12.11 shows the major sections of an e-mail.

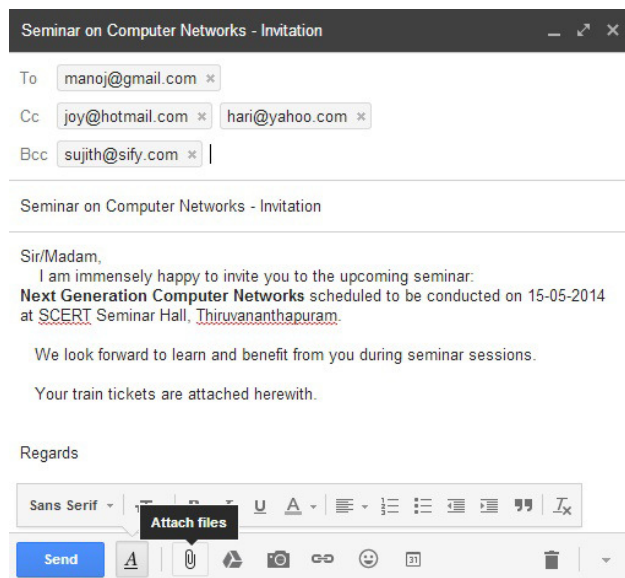


Fig. 12.11 : Composing an e-mail

**To** (Receipient Address) – A box to provide the e-mail addresses of the primary recipients to whom the e-mail has to be sent.

**Cc** (Carbon copy) – Write the e-mail addresses of the secondary recipients to whom the message has to be sent.

**Bcc** (Blind carbon copy) - Write the e-mail addresses of the tertiary recipients who receive the message. When the message is received the primary and secondary recipients cannot see the e-mail addresses of the tertiary recipients in the message. Depending on e-mail service used, the tertiary recipients may only see their own e-mail address in Bcc, or they may see the e-mail addresses of all recipients.

**Subject** – Provide a meaningful subject for your conversation here. This helps you to identify a conversation with a particular person when you search your e-mails later.

**Content** – Type your message here. Today most of the e-mail service providers offer features to create an attractive message by giving colours, changing font styles, size, etc.

Attachment facility allows us to send files like documents, pictures, etc. along with an e-mail. The 'Send' button is used to send the message to the recipients. 'Reply' button allows you to send a reply back to the sender of the message received. 'Forward' button helps you to send a message received by you to other people.

## b. Working of e-mail

Have you ever wondered how e-mail is sent from your computer to a friend on the other side of the world? When an e-mail is sent from your computer using web mail or e-mail client software, it reaches the e-mail server of our e-mail service provider. From there the message is routed from sender's e-mail server all the way to the recipient's e-mail server. The recipient's e-mail server then delivers the e-mail to the recipient's mail box (inbox), which stores the e-mail and waits for the user to read it. Simple Mail Transfer Protocol (SMTP) is used for e-mail transmission across Internet. Figure 12.12 shows the working of e-mail.

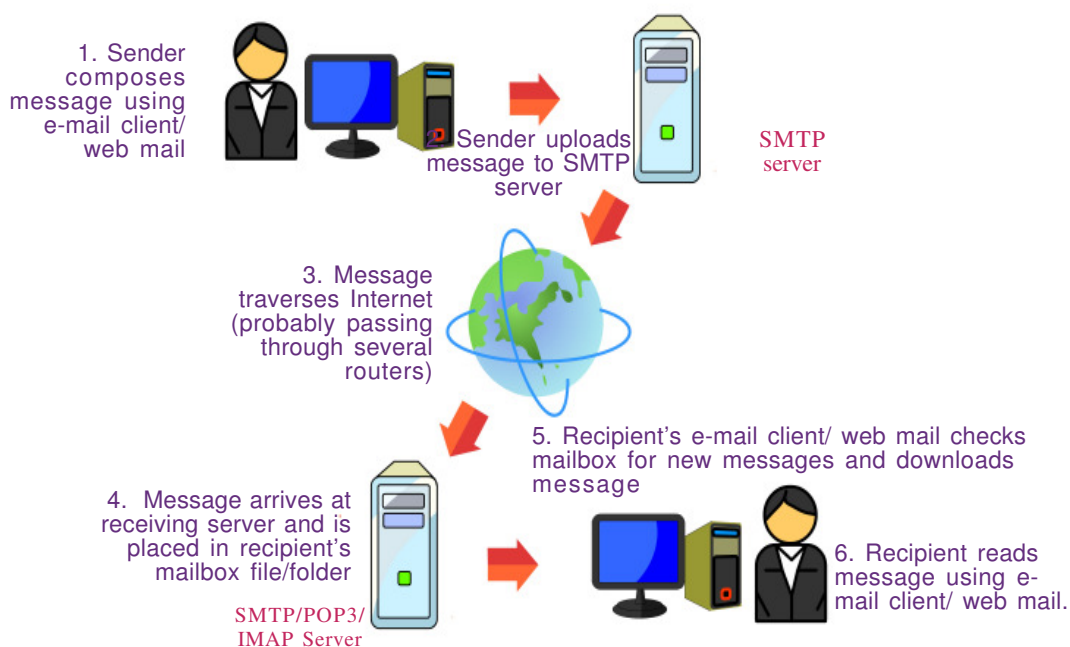


Fig. 12.12: Working of email





### c. Advantages of using e-mail

The benefits of using e-mail facility are listed below.

- **Speed:** An e-mail is delivered instantly to any location across the globe. We can send the same e-mail to multiple users simultaneously.
- **Easy to use:** We can send and receive e-mails, organise our daily conversations and save them easily on our computer.
- **Provision of attachments:** The attachment feature allows to send pictures, files, documents, etc. along with e-mail.
- **Environment friendly:** e-mails do not use paper and save a lot of trees from being cut down.
- **Reply to an e-mail:** When we need to reply to an e-mail, we can use the provision of attaching previous e-mails as reference. It helps to refresh the recipient about the subject.
- **Cost-effective:** When compared to fax or conventional mail, e-mail is less expensive.
- **Available anywhere anytime:** Messages can be read at user's convenience. Access to mail box is available anytime.

The e-mail service, though beneficial in our daily life, can be misused in different ways as listed below.

- **E-mails may carry viruses:** Viruses send along with e-mail can harm our computer system. Viruses can also illegally access our e-mail address book and spread virus infected messages to all e-mail addresses in it.
- **Junk mails:** Checking and deleting unwanted mails consume a lot of time.



#### Internet of Things (IoT)

Can you imagine a fridge which checks its egg tray and reminds you to buy eggs in your mobile phone or orders the nearby grocery store to supply eggs to your home; an air conditioner that can be switched on or off using your mobile phone; or a car that automatically reminds you about filling fuel as you approach a fuel pump? This is being made possible using Internet of Things (IoT). IoT is the concept of connecting all devices like mobile phones, fridges, cars, air conditioners, lamps, wearable devices, etc. to the Internet. Each device is provided with a unique IP address which identifies it and allows it to transfer data over Internet without human intervention. The huge increase in the number of IP addresses due to the implementation of IPv6 supports the introduction of this technology. The IoT can be used to monitor health of patients and inform the doctor about an urgency, applied to things which help us reduce wastage like power, water, etc. and improve the way we work and live.



### 12.4.4 Social media

All of us are familiar with wikipedia, the free encyclopedia in Internet. We have also heard about people responding to social issues through facebook, twitter, etc. Also we know that people use youtube to share videos and for promotion of products or business. All of these are part of social media which is changing the way we communicate, entertain and live. Social media refers to the use of mobile and web-based technologies through which individuals and communities can create, share, discuss and modify content.

In social media, interactions among people happen in virtual communities and networks over Internet. These digital technologies influence the formation and activities of civil communities to a great extent.

#### a. Classification of social media

The various types of social media that exist on the Internet are: Internet forums, social blogs, microblogs, wikis, social networks, content communities and a lot more. Figure 12.13 displays logos of popular social media websites. Here we discuss the most popular classifications of social media.



*Fig. 12.13 Logo of popular social media websites*

#### 1. Internet forums

An Internet forum is an online discussion web site where people can engage in conversations in the form of posted messages. Each Internet forum will have sub forums which may have several topics. Each discussion on a topic is called a thread. People can login and start a thread or respond to discussion in a thread. Some forums allow anonymous login also. Discussions can be about programming, social/political issues, fashion, etc. These discussions help us to learn and find solutions to problems. Ubuntu Forum – a community that provides help on Ubuntu is a popular forum.

#### 2. Social blogs

A blog (web log) is a discussion or informational website consisting of entries or posts displayed in the reverse chronological order i.e., the most recent post appears first. Some blogs provide comments on a particular subject; others function as personal online diaries and some others as online brand advertising for a particular individual or company. Initially blogs were created by a single user only. But now there are multi-author blogs that are professionally edited. Blogger.com and Wordpress.com are popular sites that offer blogging facility.

#### 3. Microblogs

Microblogs allow users to exchange short sentences, individual images or video links. People use microblogs to share what they observe in their surroundings –





information about events and opinions about topics from a wide range of fields. Microblogging offers a communication mode that is spontaneous and can influence public opinion. Twitter.com is a popular microblogging site.

#### 4. Wikis

Wikis allow people to add content or edit existing information in a web page, to form a community document. Wiki is a type of content management system. Editing done by users is very closely monitored by other editors and therefore incorrect information, advertising, etc. are removed immediately. wikipedia.org – the free online encyclopedia is the most popular wiki on web.



WIKIPEDIA  
The Free Encyclopedia



Wikipedia is a free online encyclopedia to which anyone can add content and edit. Wikipedia was formally launched on 15<sup>th</sup> January 2001 by Jimmy Wales and Larry

Sanger using the concept and technology of a wiki. Wikipedia consists of over 3 crore articles in around 300 languages. The english edition alone includes around 44 lakhs articles and is one of the most visited websites on Internet. Articles on topics range from very broad to highly specific. Each article consists of a number of links to Wikipedia itself and other external resources. Since users are able to create and edit articles, the quality of the content in the articles depends on the person who contributes and edits it. The Malayalam edition of Wikipedia is available at [ml.wikipedia.org](http://ml.wikipedia.org).

#### 5. Social networks

Social networking sites allow people to build personal web pages and then connect with friends to communicate and share content. We can share text, pictures, videos, etc. and comment to the posts. A social networking site can be for general topics or for a specific area like professional networking. Public opinion is greatly influenced by the discussions and posts in these websites. Popular social networking sites are facebook.com and linkedin.com.

#### 6. Content communities

Content communities are websites that organise and share contents like photos, videos, etc. Youtube.com is a popular video sharing site and flickr.com shares pictures.

Most of today's social media websites offer more than one type of service, i.e., social networking and microblogging; blogging and internet forum; etc. Studies have revealed that social media is now recognised as a social influencer.

#### b. Advantages of social media

- **Bring people together:** Social networking allows people to find long-lost childhood friends and make new ones.



- **Plan and organise events:** These sites help users to organise and participate in events.
- **Business promotion:** Social media offers opportunities for businesses to connect with customers, implement marketing campaigns, manage reputation, etc.
- **Social skills:** These sites allow people to express their views over a particular issue and become an agent for social change.

### c. Limitations in use of social media

- **Intrusion to privacy:** The personal information of users can be used for illegal activities. Information like the e-mail address, name, location and age can be used to commit online crimes.
- **Addiction:** Addiction to these sites wastes our valuable time. It will negatively affect our mental states and may lead to depression and tension. It can reduce the productivity of workers in an organisation. Students may lose concentration and this in turn may affect their studies.
- **Spread rumours:** Social media will spread the news very quickly. It can facilitate or worsen a crisis by spreading negative information or misinformation at an incredible speed.

### d. Social media interaction – Best practices

- Avoid unnecessary uploading of personal data like e-mail address, telephone number, address, pictures and videos.
- Setting time schedule for using these sites can save wastage of time.
- In social media websites like wikis and blogs, photo and video sharing are public. What you contribute is available for all to see. Be aware of what you post online. Avoid posting content you may regret later.
- Set your privacy levels in such a way that you know exactly who can see your posts and who can share them. The three basic privacy levels in social media are private, friends and public.



#### Let us do

- *Prepare a chart on the different social networking websites and their uses.*
- *Create a blog of your class and update the activities like achievements in sports, arts, class tests, assignments, etc.*
- *Conduct a survey in your school to find the most popular Internet browser. Also prepare a chart based on the collected data.*

**Check yourself**

1. Give an example for an e-mail address.
2. Which of the following is not a search engine?  
(a) Google      (b) Bing      (c) Facebook      (d) Ask
3. Name the protocol used for e-mail transmission across Internet.
4. What is a blog?
5. Name two services over Internet.
6. Each document on the web is referred using \_\_\_\_.

**12.5 Cyber security**

Today, we know that people use Internet to transfer personal and confidential information or make payments, organisations like banks perform all their financial transactions using their computer network, railways do business – selling tickets, information on running trains, etc. using the railway's computer network. Can you imagine the volume of financial loss and other issues that may occur if these computer networks are not available, even for a short time?

Security to computer networks is vital because important data can be lost and privacy can be violated. Further, work or business can be interrupted for several hours or even days if a network comes under attack. With the arrival of the Internet, security has become a major concern as people started using Internet as a tool for communication and doing business. Every organisation should monitor its network for possible intrusion and other attacks. Here we discuss the common threats that affect a computer network.

**12.5.1 Computer virus**

A computer virus is a program that attaches itself to another program or file enabling it to spread from one computer to another without our knowledge and interferes with the normal operation of a computer. A virus might corrupt or delete data on our computer, replicate itself and spread to other computers or even erase everything on the hard disk. Almost all viruses are attached to executable files. A virus may exist on a computer, but it cannot infect the computer unless this malicious program is run or opened. Viruses spread when the file they are attached to, is transferred from one computer to another using a portable storage media (USB drives, portable hard disks, etc.), file sharing, or through e-mail attachments. Viruses have become a huge problem on the Internet and have caused damage worth billions.



### 12.5.2 Worm

A computer worm is a stand alone malware (malicious software) program that replicates itself in order to spread to other computers. Worms spread from computer to computer on its own. Unlike a virus, it does not need to attach itself to a program to propagate. A worm takes advantage of the data transport features of the computer system to travel without help. Worms always slow data traffic on the network by consuming bandwidth, whereas viruses almost always corrupt or modify files on a computer. The most destructive effect that a worm can cause is through e-mails. A worm can send a copy of itself to every address in an e-mail address book. Then, the worm sends itself to everyone listed in each of the receiver's address book and so on.



#### **I LOVE YOU worm**

This worm affected computers in 2000 by overwriting most of the files. Users received this worm as an e-mail with a subject line "ILOVEYOU" and with a file attachment LOVE-LETTER-FOR-YOU.TXT.vbs. Those who clicked the attachment got their computers affected by the worm and lost their files.

### 12.5.3 Trojan horse

A Trojan horse, will appear to be a useful software but will actually do damage once installed or run on the computer. Users are typically tricked into loading and executing it on their systems. When a Trojan is activated on a computer, they can cause serious damage by deleting files and destroying information on the system. Some Trojans create a backdoor on the computer. This gives malicious users access to confidential or personal information in the computer through the network. Unlike viruses and worms, Trojans do not reproduce by infecting files nor do they self-replicate.



#### **Ie0199.exe Trojan**

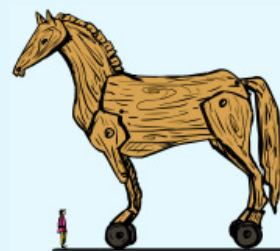
E-mail users received a message that offered a free upgrade to Internet Explorer that contained an executable file Ie0199.exe as attachment. This e-mail instructed the user to download and install this program for the upgrade. The users who followed these instructions got their files infected.



### Trojan War

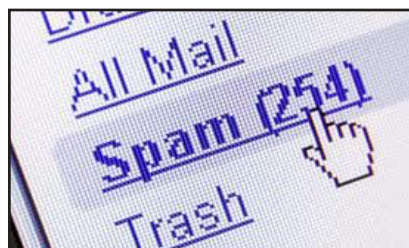


In Greek mythology, the Trojan War was waged against the city of Troy by the Greeks after Prince Paris of Troy stole away the Greek Queen Helen. The Greeks fought a battle with the city of Troy for ten long years. The Greek soldiers got fed up and wanted to return to their homes. Then, Athena, the Goddess of war gave the Greeks an idea to end the war. According to the plan, they built a big hollow wooden horse. The hollow horse was filled with soldiers and they left it as a gift for the Trojans. All other soldiers pretended to abandon their camp. Trojans thought that they had won the war. They pulled the huge horse to their city. They started celebrations of their victory. In the night when everyone was asleep, the Greek soldiers opened the horse and came out. They killed the sleeping soldiers of Troy and rescued Queen Helen.



#### 12.5.4 Spams

Spams or junk mails are unsolicited e-mails sent indiscriminately to persons to promote a product or service. Spammers collect e-mail addresses from chat rooms, websites, customer lists, newsgroups, etc. Clicking on links in spams may send users to websites that host certain viruses. Today most e-mail service providers provide e-mail filters that can successfully separate genuine e-mail from spams as indicated in Figure 12.14.



*Fig. 12.14 Collection of spams in the e-mail menu*

#### 12.5.5 Hacking

In computer networking, hacking is a technical effort to manipulate the normal behavior of network connections and connected systems. Hacking is performed both by computer security experts and by computer criminals. Computer experts perform hacking to test the security and find the vulnerabilities in computer networks and computer systems. Such computer experts are often called 'white hats' and such hacking is called ethical hacking.

Computer criminals break into secure networks to destroy data or make the network unusable for those who are authorised to use the network. They do this with the intent of stealing confidential data or destroying files. Such criminals are called 'black hats'.



There is another category of hackers called grey hat hackers, who fall between white and black hackers. They sometimes act illegally, though with good intentions, to identify the vulnerabilities. Grey hat hackers do this to achieve better security.

### 12.5.6 Phishing

Phishing is a type of identity theft that occurs online. Phishing is an attempt to acquire information such as usernames, passwords and credit card details by posing as the original website, mostly that of banks and other financial institutions. Phishing websites have URLs and home pages similar to their original ones. The act of creating such a misleading website is called spoofing. People are persuaded to visit these spoofed websites through e-mails. Users are tempted to type their usernames, passwords, credit card numbers, etc. in these web pages and lose them to these websites. These frauds use this information to steal money. Phishing is currently the most widespread financial threat on the Internet. The URL in Figure 12.15 indicates that it is a phishing website.

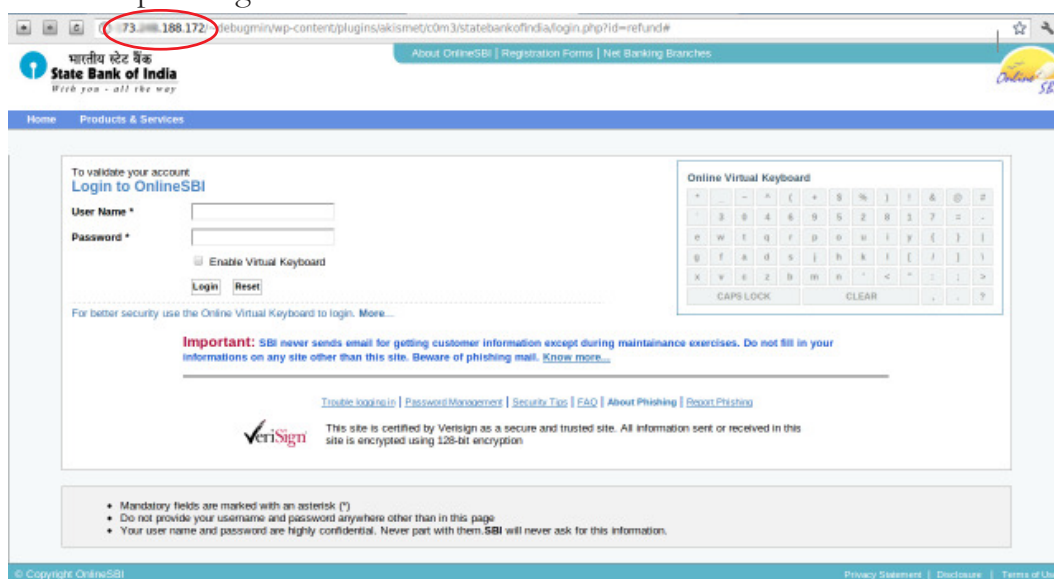


Fig. 12.15: A phishing website

### 12.5.7 Denial of Service (DoS) attack

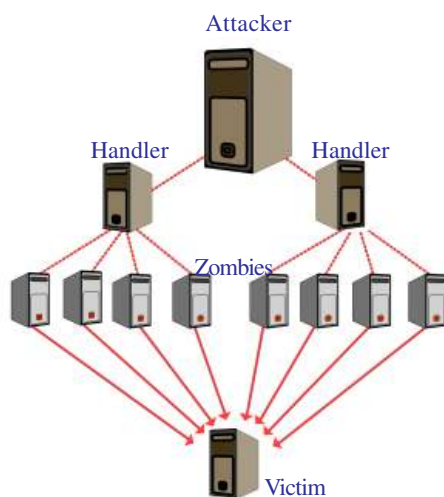
A Denial of Service (DoS) attack is usually aimed at a web server. Such an attack forces the server/computer to restart. An attack in which the attackers' goal is to shut down the target server rather than stealing data is called DoS attacks. This prevents genuine users of a service/website on the web server from using that service. This attack can be done using a single computer called Denial of Service (DoS) attack or using more than one computer called Distributed Denial of Service (DDoS) attack.



We have learned that when we type a website address in the browser and press the **Enter** key, the browser requests for that web page from the server. DoS attacks sends large number of such requests to the server until it collapses under the load and stops functioning. A DoS attack using a computer on a network slows down the network by flooding a server with a large number of requests. A DDoS attack uses multiple computers in the network that it has previously infected. These infected computers called 'zombies', work together and send out large quantities of fake messages/requests to the target server. Figure 12.16 shows the Distributed Denial of Service

attack. This increases the amount of data traffic to the target server. This

leads to server overload and the server is unable to provide services to its users. The target computer is thus forced to reset / restart leading to unavailability of its service for a period of time. A DoS attack interrupts network service for some period, but it does not cause severe damage to files as in the case of a virus attack.



*Fig. 12.16: Distributed Denial of Service (DDoS) attacks*

### 12.5.8 Man-in-the-Middle attacks

A man-in-the-middle attack refers to an attack in which an attacker secretly intercepts electronic messages between the sender and the receiver and then captures, inserts and modifies messages during message transmission. If sender transmits messages without appropriate security, the attacker may exploit the vulnerabilities in the network to capture and modify the messages and send the modified messages to the receiver. Since the network transmission still works properly, both the sender and receiver will find it difficult to notice that the messages have been trapped or modified by an intruder. If we use such a computer for online transactions, the man in the middle may capture our bank account number and password to steal money, leading to financial loss. Encrypted connections such as HTTPS (HTTP Secure), SFTP (Secure FTP) etc. should be used for secure transactions, so that intruders cannot modify the messages.

## 12.6 Preventing network attacks

Threats to computers and networks are a major issue as long as information is accessible and transferred across the Internet. Different defense and detection mechanisms are developed to deal with these attacks.



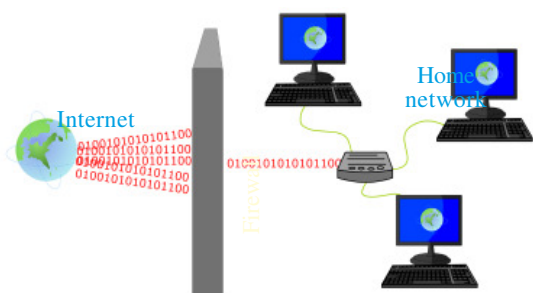


Fig. 12.17: Firewall

### 12.6.1 Firewall

A firewall is a system of computer hardware and software that provides security to the computer network in an organisation. A firewall controls the incoming and outgoing network traffic by analysing the data and determining whether they should be allowed through

or not, based on a rule set. Firewalls deny malicious data from entering into the computer networks as shown in Figure 12.17.



#### Sandboxing

Sandboxing is a technique through which programs that are suspected to be infected with a virus can be run. Through sandboxing such programs are run in a separate memory area and therefore cannot damage our operating system.

### 12.6.2 Anti-virus scanners

Viruses, worms and Trojan horses are all examples of malicious software (malware). Antivirus tools are used to detect them and cure the infected system. Anti-virus software scans files in the computer system for known viruses and removes them if found. The anti-virus software uses virus definition files containing signatures (details) of viruses and other malware that are known. When an antivirus program scans a file and notices that the file matches a known piece of malware, the antivirus program stops the file from running, and puts it into 'quarantine'. Quarantine is a special area for storing files probably infected with viruses. These files can later be deleted or the virus can be removed. For effective use of anti-virus software, virus definitions must be updated regularly.

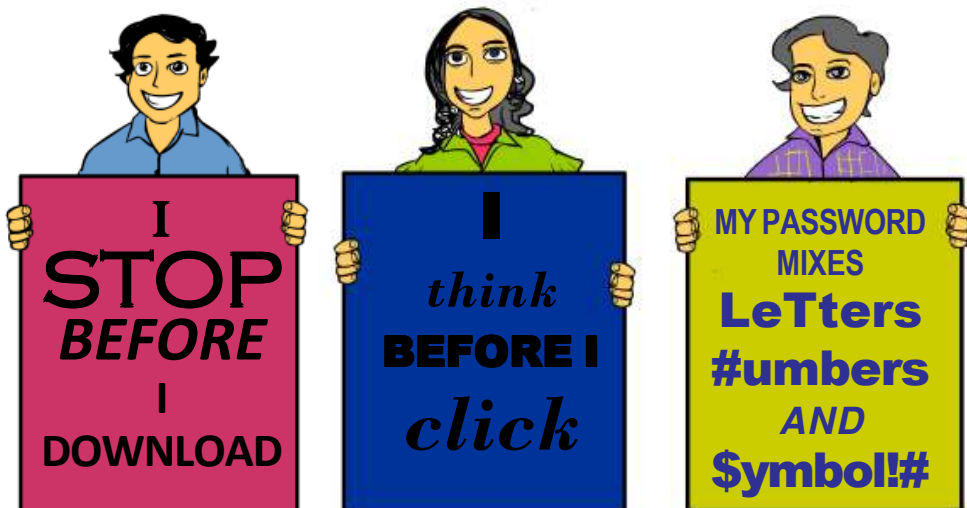
### 12.6.3 Cookies

Cookies are small text files that are created when we use a browser to visit a website. Cookies keep track of our movements within the website – remembers our user name, preferences, e-mail address, etc. Browsers store cookies for an interval of time, usually in a cookie folder on the client's computer. Cookies are text files and so they are not executable programs. Websites use cookies mainly because they save time and make browsing efficient.

Cookies are treated as data and so it is not a virus, but it is always possible for a hacker to use it for malicious purposes. Cookies can be used to act as a spyware.

There are harmful cookies that are used by different websites to compromise our privacy. Such websites store a special cookie in our computer that will keep track of our activities like, websites visited, products purchased or the forms that are filled. Most browsers provide facilities to manage/delete harmful cookies. Frequent deletion of cookies helps to prevent illegal access and use of personal information.

## 12.7 Guidelines for using computers over Internet



Following are the major guidelines for using computers over Internet.

- Most of the computer viruses are spread through e-mail attachments. Do not open any e-mail attachment that you are not sure about the sender.
- Download files only from reputed sources. Do not use/copy software that you cannot confirm the origin.
- Avoid clicking on pop-up advertisements. Close them instead.
- Use USB drives with caution. Plugging someone else's USB storage into your computer or plugging your own USB storage into a computer at an Internet cafe/unsafe computer, can spread an infection through the USB storage.
- Make sure the firewall is set and turned on.
- Use strong passwords. Change passwords at regular intervals.
- Update the virus definitions of your antivirus program periodically online.
- Keep a regular backup of your important files (on DVD, another hard disk, etc.)
- Be careful about giving personal data online. If you see e-mail message requests for personal data such as telephone number, address, credit card number, etc. from unknown persons, ignore it.



### Guidelines for setting up a strong password

- A password should have atleast 8 characters.
- A password should contain
  - Upper case letters
  - Lower case letters
  - Numbers
  - Symbols like @, #, \$, etc.
- A password should not be personal information like name, date of birth, etc. or common words.
- Never disclose your password to others.
- Do not write it on a paper or store it in a file in your computer.
- Do not use the same password for all logins.
- Change password often.

- Visit banks' websites by typing the URL into the address bar. Do not click on links within e-mails to go to bank websites. Banks or any of its representatives never sends you e-mail/SMS or phone calls to get your personal information, usernames or password. Never reveal your passwords or ATM card details to anyone.
- Check whether the website you are visiting is secure while performing financial transactions. The web address in the address bar should start with 'https://'. Also look for a lock icon on the browser's address bar.
- Keep a regular check on your online accounts. Regularly login to your online accounts, and check your statements. If you see any suspicious transaction, report them to your bank or credit card provider.

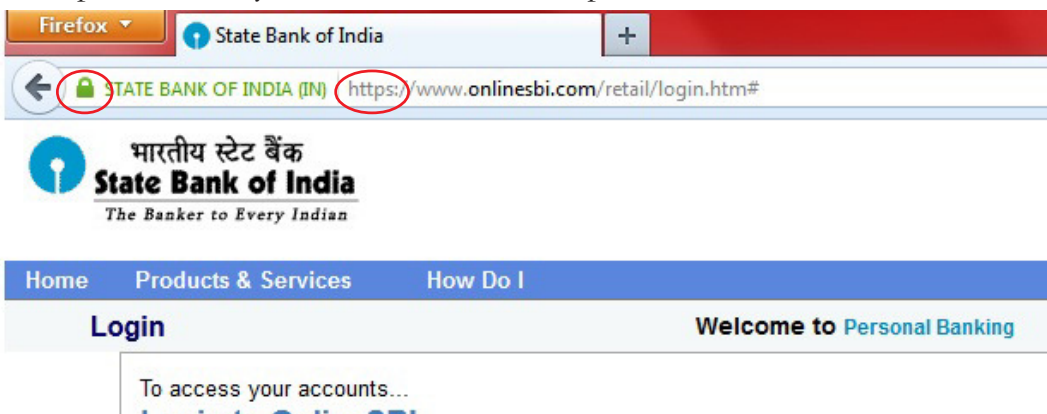


Fig 12.18: Secure banking - lock icon and https

**Let us do**

- Conduct a group discussion on the topic “security threats / cyber attacks to your system”, and draw a bar diagram based on information arrived in the event.
- List the names of various viruses and their features in the form of a chart.

**Check yourself**

1. What is a virus?
2. What do you mean by phishing?
3. The small text files used by browsers to remember our email ids, user names, etc. are known as \_\_\_\_\_.
4. The act of breaking into secure networks to destroy data is called \_\_\_\_\_ hacking.
5. What is quarantine?

**12.8 Mobile computing**

The advances in computer technology have led to the development of more computing power in small devices. Light weight computing devices with low power consumption is now cheap and is common. Devices like laptops, tablets, smart phones, etc. changed the lifestyle and work culture of people drastically. Today people are able to connect to others, send and receive files or other information anywhere anytime. This increased the computing demands day by day.

Mobile computing is a technology that has computing capability and can transmit/ receive data while in motion. Mobile computing requires portable computing devices like laptops, tablets, smart phones, etc., wireless communication networks and connectivity to Internet. The demand for mobile computing started the growth and development of mobile communication technology.

**12.9 Mobile communication**

The term ‘mobile’ has completely revolutionised communication by opening up innovative ways for exchanging information. Today, mobile communication has become the backbone of the society. Mobile system technologies have improved the living standards. Mobile communication networks does not depend on any physical connection to communicate between two devices.



### 12.9.1 Generations in mobile communication

The mobile phone was introduced in the year 1946. In the initial stage, the growth in mobile communication was very slow. With the increase in the number of users, accommodating them within the limited available frequency spectrum became a major problem. To solve this problem, the concept of cellular communication was evolved. The cellular concept was developed in the 1960's at Bell Laboratories. However, in the late 1990's, when the Government of India offered spectrum licenses for mobile communication, cellular technology became a common standard in our country. The different generations in mobile communication are given below:



#### a. First Generation networks (1G)

1G refers to the first-generation of wireless telephone technology (mobile telecommunications) developed around 1980. 1G mobile phones were based on the analog system and provided basic voice facility only.

#### b. Second Generation networks (2G)

2G networks follow digital system for communication. This improved the audio quality in transmission. In 2G networks phone conversations are digitally encrypted. These networks provided far greater mobile phone coverage. 2G networks also introduced data services for mobile. Picture messages and MMS (Multimedia Messaging Service) were introduced. The two popular standards introduced by 2G systems are GSM and CDMA. A detailed discussion on the GSM and CDMA standards are given below.

##### i. Global System for Mobile (GSM)

GSM is a globally accepted standard for digital cellular communication. GSM uses narrowband TDMA (Time Division Multiple Access), which allows simultaneous calls on the same radio frequency. It is a digital, circuit-switched network for voice telephony. The frequency band for GSM is 900 MHz to 1800 MHz. GSM follows a uniform international standard that made it possible to use a mobile device around the world. The network is identified using the SIM (Subscriber Identity Module). The users can select a handset of their choice. GSM is the most successful family of cellular standards.





### **GPRS and EDGE**

2G network was expanded later to include improved data communication features using GPRS (General Packet Radio Services) and EDGE (Enhanced Data rates for GSM Evolution).

GPRS is a packet oriented mobile data service on GSM. When compared to conventional GSM, users of GPRS benefitted from shorter access time and higher data rates. GPRS allows billing based on volume of data transferred. Although GPRS is a data only technology, it helps to improve GSM's voice quality.

EDGE is a digital mobile phone technology that allows improved data transmission rates for GSM. EDGE is a superset to GPRS and can function on any network with GPRS deployed on it. It provides nearly three times faster speeds than the GPRS system. Both phone and network must support EDGE, otherwise the phone will revert automatically to GPRS.

#### **ii. Code Division Multiple Access (CDMA)**

In CDMA or Code Division Multiple Access system, several transmitters can send information simultaneously over a single communication channel. CDMA provides wider coverage than GSM and provides better reception even in low signal strength conditions. The voice quality in CDMA is better than GSM. It has a signal with wider bandwidth and increased resistance to interference. CDMA technology provides better security to the mobile network when compared to GSM.

#### **c. Third Generation networks (3G)**

3G wireless network technology provides high data transfer rates for handheld devices. The high data transfer rates allows 3G networks to offer multimedia services combining voice and data. 3G is also referred to as wireless broadband as it has the facility to send and receive large amounts of data using a mobile phone. The access part in 3G networks uses WCDMA (Wideband Code Division Multiple Access). It requires upgrading the base stations (mobile towers) and mobile phones. Also the base stations need to be close to each other.

#### **d. Fourth Generation networks (4G)**

A 4G system, also called Long Term Evolution (L.T.E.), provides mobile ultra-broadband Internet access to mobile devices. 4G networks offer very high speeds and provides excellent performance for bandwidth intensive applications such as high quality streaming video. One of the key requirements for 4G is a wireless IP-based access system. The access part in 4G networks uses OFDMA (Orthogonal Frequency Division Multiple Access). 4G provides good quality images and videos than TV.

## 12.9.2 Mobile communication services

Mobile communication industry uses a number of acronyms for the various technologies that are being developed. Here we discuss a few popular mobile communication technologies like SMS, MMS, GPS and smart cards.

### a. Short Message Service (SMS)

Short Message Service (SMS) is a text messaging service in mobile communication systems that allows exchanging short text messages. SMS is the most widely used data application by mobile phone users. The GSM standard allows to send a message containing upto 160 characters. When a message is sent, it reaches a Short Message Service Center (SMSC), which provides a 'store and forward' mechanism. SMSC attempts to send messages to the recipients. If a recipient is not reachable, the SMSC waits and then retries later. Some SMSC's also provide a 'forward and forget' option where transmission is tried only once and if it fails, the message is not sent again. SMS messages are exchanged using the protocol called SS7 (Signalling System No # 7).



#### The First SMS

The first SMS message was sent on 3<sup>rd</sup> December 1992 from a personal computer to a cellular phone on the Vodafone GSM network in the UK. The content of the message was 'Merry Christmas'.

### b. Multimedia Messaging Service (MMS)

Multimedia Messaging Service (MMS) is a standard way to send and receive messages that consists of multimedia content using mobile phones. It is an extension of the capability of SMS to send text messages. Unlike SMS, MMS does not specify a maximum size for a multimedia message. MMS supports contents such as text, graphics, music, video clips and more. An MMS server is responsible for storing and handling incoming and outgoing messages. Associated with the MMS server is the MMS proxy relay, which is responsible for transferring messages between different messaging systems.

### c. Global Positioning System (GPS)

The Global Positioning System (GPS) is a satellite based navigation system that is used to locate a geographical position anywhere on earth, using its longitude and latitude. GPS is designed and operated by the U.S. Department of Defence and it consists of satellites, control and monitoring stations, and receivers.

The basis of the GPS is a group of satellites that are continuously orbiting the earth. These satellites transmit radio signals that contain their exact location, time,



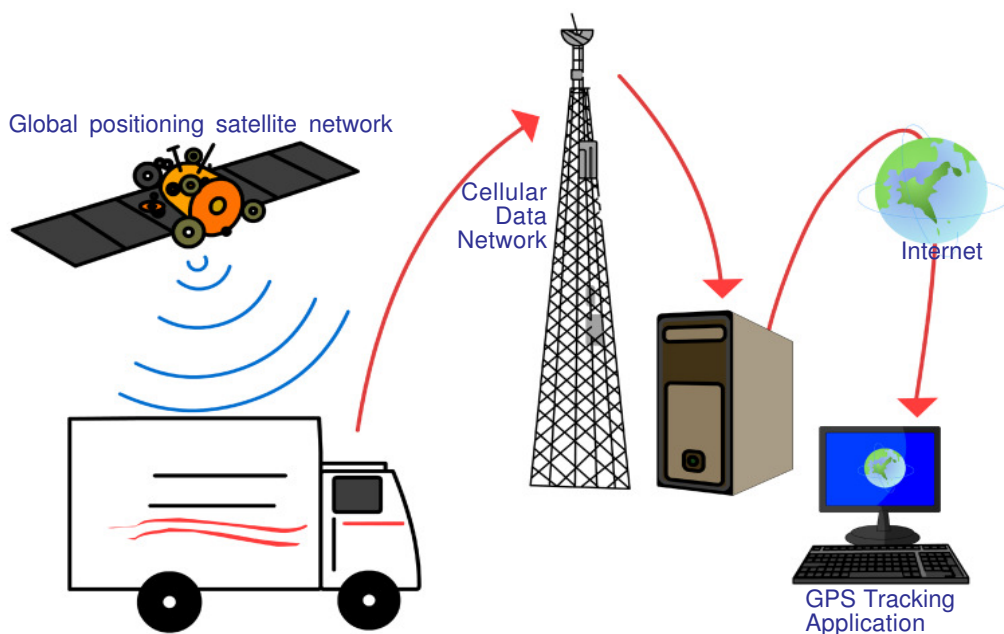


Fig. 12.19: GPS fleet tracking

and other information. The radio signals from the satellites, which are monitored and corrected by control stations, are picked up by the GPS receiver. GPS receivers take information transmitted from the satellites to calculate a user's exact location on earth. A GPS receiver needs only three satellites to plot a 2D position, which will not be very accurate. Ideally, four or more satellites are needed to plot a 3D position, which is much more accurate.

GPS is used for vehicle fleet tracking by transporting companies to track the movement of their trucks. Figure 12.19 depicts the working of a truck tracking application using GPS. Vehicle navigation systems will direct the driver to his or her destination through the best route. In commercial aviation GPS is used for aircraft navigation. GPS is also used in oil exploration, farming, atmospheric studies, etc. GPS receivers are now integrated in many mobile phones for implementing various tracking applications.

#### d. Smart cards

Let us recollect about smart cards and smart card readers that we discussed in Chapter 3. A smart card is a plastic card embedded with a computer chip / memory that stores and transacts data. The advantages of using smart cards is that it is secure (data is protected), intelligent (it can store and process



Fig. 12.20: A model smart card



data) and that it is convenient (it is easy to carry). That is why businesses and other organisations use smart cards for authentication and storing data. A model of smart card issued by Government of India for RSBY scheme is shown in Figure 12.20.

In mobile communication the smart card technology is used in Subscriber Identity Modules (SIM) for GSM phone systems (refer Figure 12.21). The smart card is inserted or integrated into the mobile handset. The card stores personal subscriber information and preferences. SIM cards help to identify a subscriber, roam across networks and provide security to value added services like Internet browsing, mobile commerce, mobile banking, etc. Smart cards also work as credit cards, ATM cards, fuel cards, authorisation cards for television receiver, high-security identification cards, etc.



*Fig. 12.21 GSM SIM card*

## 12.10 Mobile operating system

A mobile operating system is the operating system used in a mobile device (smart phone, tablet, etc.), similar to an operating system used in a desktop or laptop computer.



*Fig. 12.22: Icons of popular mobile operating systems*

A mobile OS manages the hardware, multimedia functions, Internet connectivity, etc. in a mobile device. It is the software platform on which other programs, called application programs, are running. Popular mobile operating systems are Android from Google, iOS from Apple, BlackBerry OS from BlackBerry and Windows Phone from Microsoft.

### Android operating system

Android is a Linux-based operating system designed mainly for touch screen mobile devices such as smart phones and tablet computers. It was originally developed by Android Inc. that was founded in Palo Alto, California in 2003 by Andy Rubin and his friends. In 2005, Google acquired Android Inc. making it a wholly owned subsidiary of Google. At Google, the team led by Rubin developed a mobile device platform powered by the Linux kernel. In 2007, the Open Handset Alliance, a consortium of several companies which include Google, HTC, Intel, Motorola,



etc. was established with a goal to develop open standards for mobile devices. Android was released along with the formation of the Open Handset Alliance (OHA).

The user interface of Android is based on touch inputs like swiping, tapping, pinching and reverse pinching to manipulate on-screen objects. Android allows users to customise their home screens with shortcuts to applications and widgets. Since its launch in 2007, the Android operating system's market share has been growing at a fast pace to make it the most widely used mobile operating system today. Major Android versions have been developed under a codename and released according to alphabetical order. Table 12.2 shows the Android version names.

The Android OS consists of a kernel based on Linux kernel. Android uses Linux kernel as it has a powerful memory and process management system, permissions-based security structure and open source nature. An Application Framework for developers to build applications using the Android Software Development Kit is available in Android. Applications like Google Maps, Facebook, etc. that run on Android are built using this.

| Version | Code name                 |
|---------|---------------------------|
| 4.4     | <i>KitKat</i>             |
| 4.1     | <i>Jelly Bean</i>         |
| 4.0.3   | <i>Ice Cream Sandwich</i> |
| 3.1     | <i>Honeycomb</i>          |
| 2.3     | <i>Gingerbread</i>        |
| 2.2     | <i>Froyo</i>              |
| 2.0     | <i>Eclair</i>             |
| 1.6     | <i>Donut</i>              |
| 1.5     | <i>Cupcake</i>            |

*Table 12.2: Android version names*

The Android code is released under the Apache License. Apache licensing allows the software to be freely modified and distributed by device manufacturers and developers. Android has a large community of developers writing applications called 'apps' that enhance the functionality of devices. Apps are written using a customised version of the Java programming language.

Android's acceptance is due to factors like its open source nature. This makes it easy for developers to make programs/applications for Android OS called Android apps. Most of these apps are available for free download from the Android Play Store. It adds the popularity of this OS.

With remarkable advances in mobile hardware and software, these trimmed-down operating systems may be heading in a direction that could replace a desktop OS. It is also likely that mobile OS could be further developed to be included in electronic devices like televisions, washing machines, watches, etc.

**Let us do**

- *Prepare a chart displaying the different mobile operating systems available in the market and their features.*

**Check yourself**

1. SIM is
  - (a) Subscriber Identity Module
  - (b) Subscriber Identity Mobile
  - (c) Subscription Identification Module
  - (d) Subscription Identification Mobile
2. What is a GPS?
3. The protocol used to send SMS messages is \_\_\_\_\_.
4. How can multimedia content be sent using mobile phones?
5. What are the functions of a mobile operating system?

**Let us sum up**

The Internet, which was started as a defence project of the US government has become a part of our life. Today the Internet is accessed using mobile devices like mobile phones, tablets, etc. than using a desktop computer. Therefore speed of Internet access has become an important factor. New technologies connect to Internet focus on data transmission speed. Internet services like e-mail, social media, searching etc. have changed the way we communicate. Each of the above services has its own benefits and risks. Computer networks today play an important role in providing the above services. It has increased the risk factors for networks, like viruses, worms, Trojan horse, phishing, etc. Antivirus software, firewalls, etc. are used to protect computer networks from different kinds of attacks. The risks for a network attack can be reduced by following certain guidelines while using computers on Internet.

The convergence of a mobile phone and a computer has shifted the focus from desktops to mobile computing devices. These devices are always connected to the Internet and therefore mobile communication technology



has gained importance. The mobile communication technology has evolved through generations from 1G to 4G, the prime focus being speed. This also has led to the development of features like SMS, MMS, GPS, etc. Android is one of the popular mobile operating systems, developed by Google based on Linux kernel. The advances in the development of this OS is in such a way that it will soon replace desktop operating systems and could be used in devices like television, washing machines, etc.



## Learning outcomes

After the completion of this chapter the learner will be able to

- recognise the people behind the evolution of Internet.
- identify the hardware and software requirements for Internet connection.
- use the services available on Internet.
- classify the different types of social media.
- judge the risks while interacting with social media.
- recognise the threats to network security.
- identify the various mobile computing terminologies.
- recognise the features of mobile operating systems.
- discover the features of Android operating system.

## Sample questions

### Very short answer type

1. Why is the invention of HTTP and HTML considered an important land mark in the expansion of Internet?
2. Compare intranet and extranet.
3. Write short notes on
  - a. Mobile broadband
  - b. Wi-MAX
4. Explain the terms web browser and web browsing.
5. Compare blogs and microblogs.
6. What are wikis?
7. What is firewall?

### Short answer type

1. Your neighbour Ravi purchased a new PC for his personal use. Mention the components required to connect this PC to Internet.
2. What are the advantages of using broadband connection over a dial-up connection?
3. XYZ engineering college has advertised that its campus is Wi-Fi enabled. What is Wi-Fi? How is the Wi-Fi facility implemented in the campus?
4. Madhu needs to prepare a presentation. For this, he uses [www.google.com](http://www.google.com) to search for information. How does google display information when he types 'Phishing' in the search box and clicks search button?
5. Manoj's e-mail id is [manoj@gmail.com](mailto:manoj@gmail.com). He sends an email to Joseph whose e-mail id is [joseph@yahoo.com](mailto:joseph@yahoo.com). How is the mail sent from Manoj's computer to Joseph's computer?
6. How does a Trojan horse affect a computer?
7. Explain a few threats that affect a computer network.
8. Compare GSM and CDMA standards.
9. Write short notes on SMS.
10. What is a smart card? How is it useful?

### Long answer type

1. Suppose you wish to visit the website of kerala school kalolsavam, [www.schoolkalolsavam.in](http://www.schoolkalolsavam.in) and you have entered the URL in the address bar. Write the steps that follow until the home page is displayed.
2. Write the disadvantages of social media. What are the different ways to avoid the disadvantages of social media?
3. Explain the features of Android OS.
4. Explain the various broadband technologies available for Internet access.





## References

- Pradeep K. Sinha, Priti Sinha. *Computer Fundamentals* : BPB Publication
- V. Rajaraman (2010). *Fundamentals of Computers* : PHI Publication
- Thomas L. Floyd (2011). *Digital Fundamentals* : Pearson Education
- Craig Zacker, John Rourke (2008). *PC Hardware: The Complete Reference* : TMH Publication
- Abraham Silberschatz, Greg Gagne, Peter B. Galvin (2005) *Operating System Concepts* : John Wiley & Sons
- Herbert Schildt (2003). *C++ A beginners Guide* : McGraw-Hill Publication
- Bjarne Stroustrup (2013). *The C++ Programming Language* : Addison-Wesley Professional
- Robert Lafore (2009). *Object-Oriented Programming in C++* : Sams Publishing
- E. Balagurusamy (2008). *Object Oriented Programming with C++* : Tata McGraw-Hill Education
- Yashavant P Kanetkar(2000). *Let Us C++*: BPB Publication
- Andrew S. Tanenbaum, David J. Wetherall (2010). *Computer Networks* : Prentice Hall