

Higher Secondary Course

Computer Science



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 e-mail : 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.**

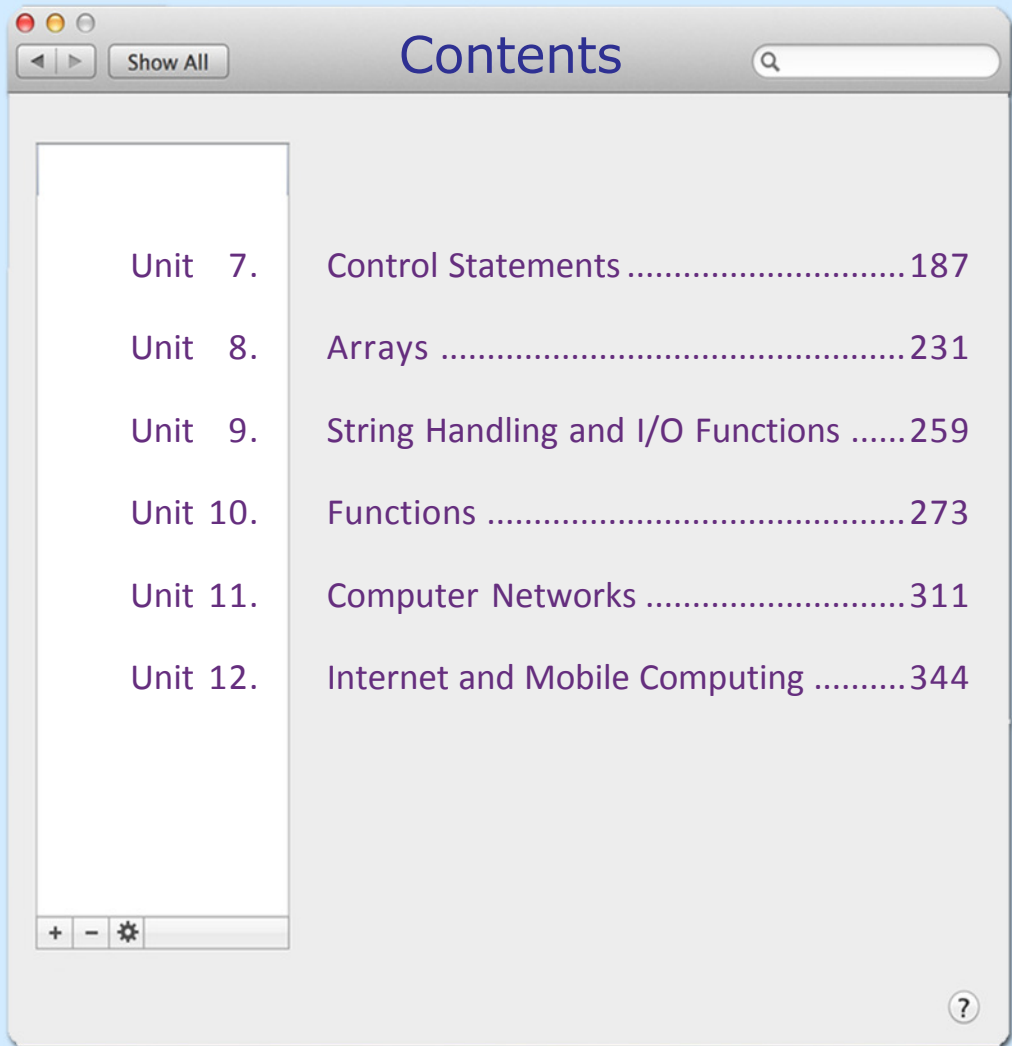
Academic Co-ordinator

Dr Meena S.

Research Officer, SCERT

Jancy Rani A. K.

Research Officer, SCERT



Unit 7.		Control Statements187
Unit 8.		Arrays231
Unit 9.		String Handling and I/O Functions259
Unit 10.		Functions273
Unit 11.		Computer Networks311
Unit 12.		Internet and Mobile Computing344

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.

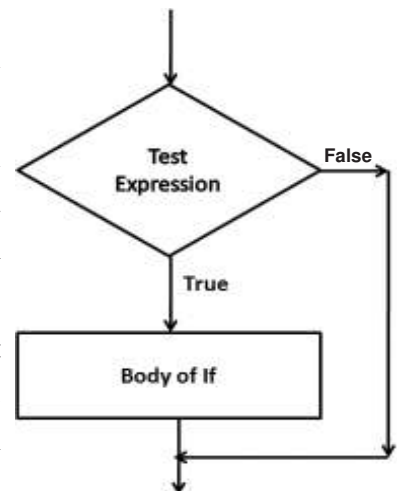


Fig. 7.1 : Working of `if` statement

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.

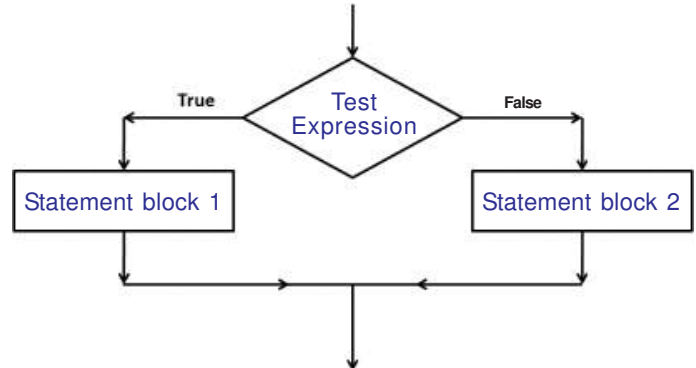


Fig 7.2 : Flowchart of if - else statement

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)  
{
```

Score being int type, the expression returns only integer values

```
    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";  
}
```

Let us have a comparison between `switch` and `else if` ladder as indicated in Table 7.1.

switch statement	else if ladder
<ul style="list-style-type: none"> Permits multiple branching 	<ul style="list-style-type: none"> Permits multiple branching
<ul style="list-style-type: none"> Evaluates conditions with equality operator only 	<ul style="list-style-type: none"> Evaluates any relational or logical expression
<ul style="list-style-type: none"> Case constant must be an integer or a character type value 	<ul style="list-style-type: none"> Condition may include range of values and floating point constants
<ul style="list-style-type: none"> When no match is found, default statement is executed 	<ul style="list-style-type: none"> When no expression evaluates to True, else block is executed
<ul style="list-style-type: none"> <code>break</code> statement is required to exit from <code>switch</code> statement 	<ul style="list-style-type: none"> Program control automatically goes out after the completion of a block
<ul style="list-style-type: none"> More efficient when the same variable or expression is compared against a set of values for equality 	<ul style="list-style-type: none"> More flexible and versatile compared to <code>switch</code>

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.

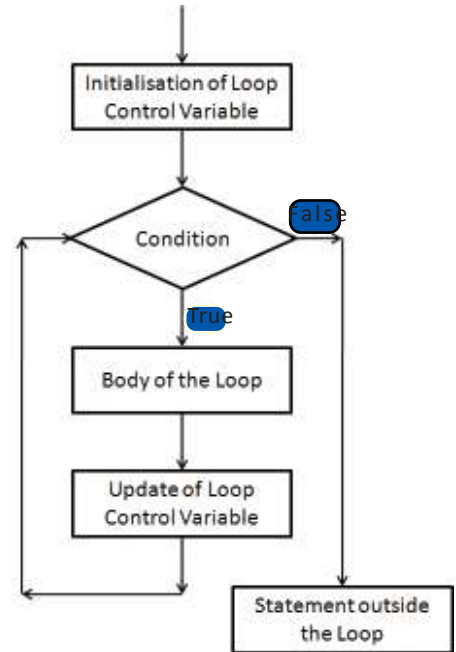


Fig. 7.3: Working of while loop

```

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

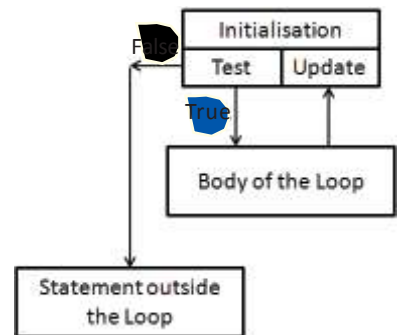


Fig. 7.4: Execution of **for** loop

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

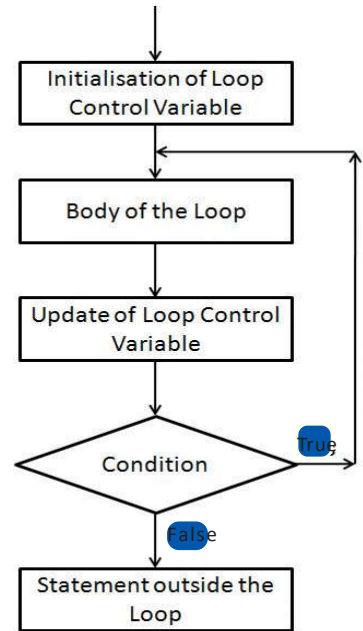


Fig. 7.5: Execution of do..while loop

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.

for loop	while loop	do...while loop
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  
25 + 4 = 29
```

User Input

```
Number 1: 25          Number 2: 4  
          Operator Menu
```

1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit (E)

```
Enter Option number or operator: /  
25 / 4 = 6.25
```

User Input

```
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  
Thank You for using the program
```

User Input

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.

break statement	continue statement
<ul style="list-style-type: none">• Used with <code>switch</code> and loops.• Brings the program control outside the <code>switch</code> or loop by skipping the rest of the statements within the block.• Program control goes out of the loop even though the test expression is <code>True</code>.	<ul style="list-style-type: none">• Used only with loops.• Brings the program control to the beginning of the loop by skipping the rest of the statements within the block.• Program control goes out of the loop only when the test expression becomes <code>False</code>.

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

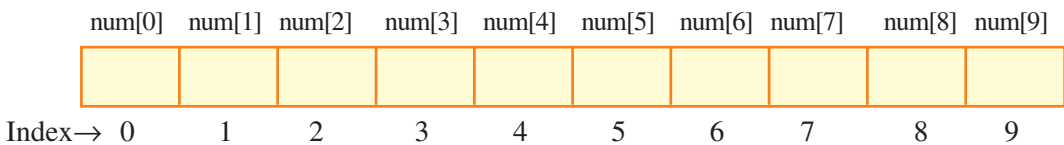
Just like the ordinary variable, the array is to be declared properly before it is used. The syntax for declaring an array in C++ is as follows.

```
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.

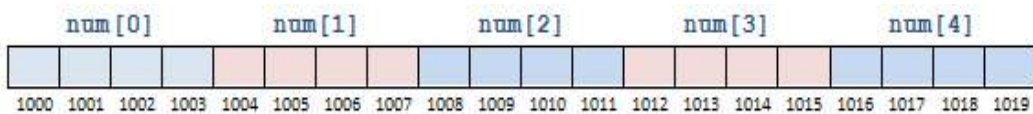


Fig. 8.2: Memory allocation for an integer array

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

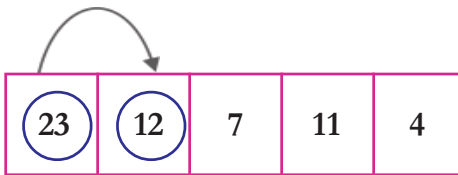
## b. Bubble sort

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.

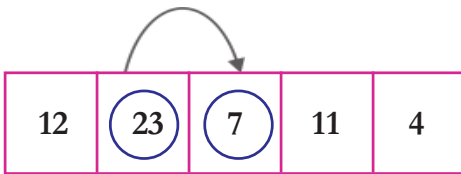
### Initial list

|    |    |   |    |   |
|----|----|---|----|---|
| 23 | 12 | 7 | 11 | 4 |
|----|----|---|----|---|

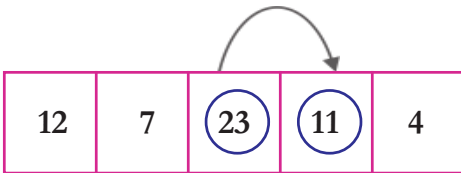
### Pass 1



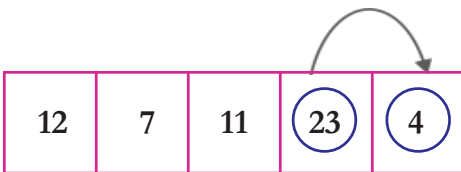
The first comparison results in exchanging the first two elements, 23 and 12.



The second comparison results in exchanging the second and third elements 23 and 7 in the revised list.



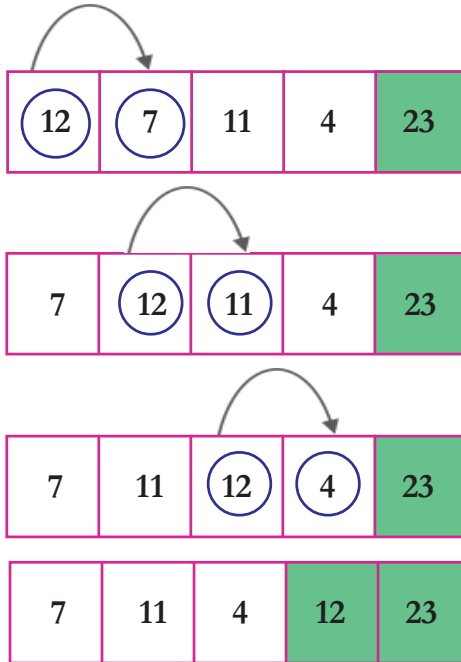
The third comparison results in exchanging the third and fourth elements 23 and 11 in the revised list.



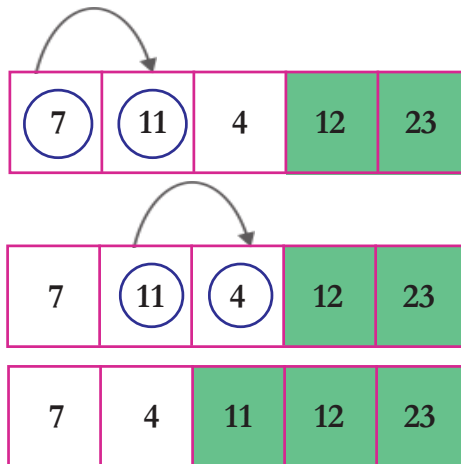
The fourth comparison results in exchanging the fourth and fifth elements 23 and 4 in the revised list.



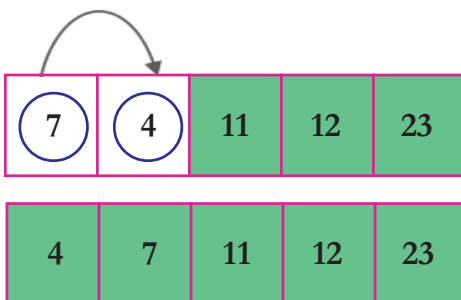
After the end of the first pass, the largest element 23 is bubbled to the last position of the list.

**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 |

**FIRST = 0**  
**LAST = 6**

As **FIRST<=LAST**, let's start iteration

| 0  | 1  | 2  | 3  | 4  | 5  | 6  |
|----|----|----|----|----|----|----|
| 21 | 28 | 33 | 35 | 45 | 58 | 61 |

**MIDDLE = (FIRST+LAST)/2 = (0+6)/2 = 3**  
Here **LIST[3]** is not equal to **45** and **LIST[3]** is less than search element therefore, we take  
**FIRST = MIDDLE + 1 = 3 + 1 = 4, LAST = 6**

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.

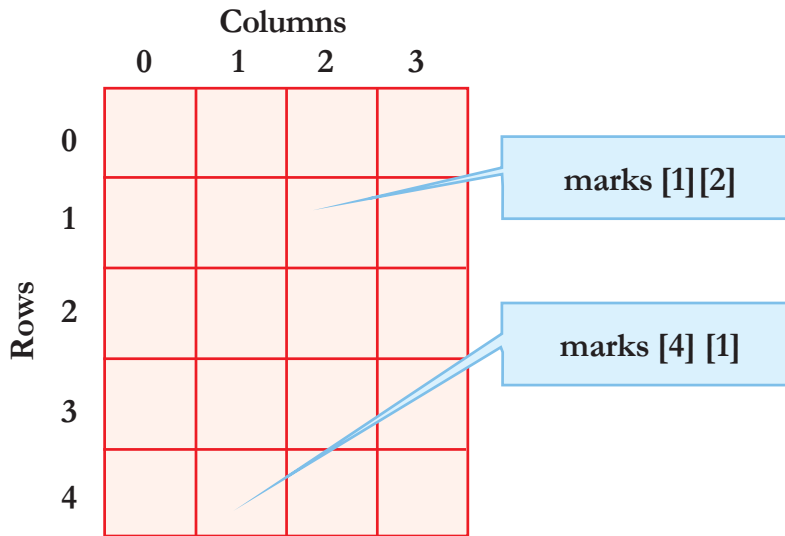


Fig. 8.4: Structure of a 2D array

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]`, ..., `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];
```

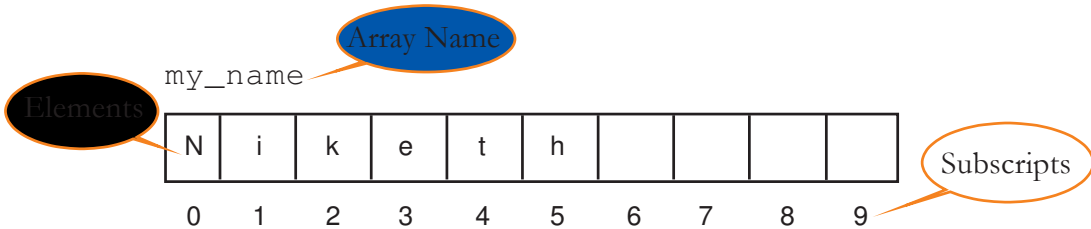
It is sure that ten contiguous locations, each with one byte size, will be allocated for the array named `my_name`. If we follow the usual array initialization method, we can store the characters in the string “Niketh” as follows:

```
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
```

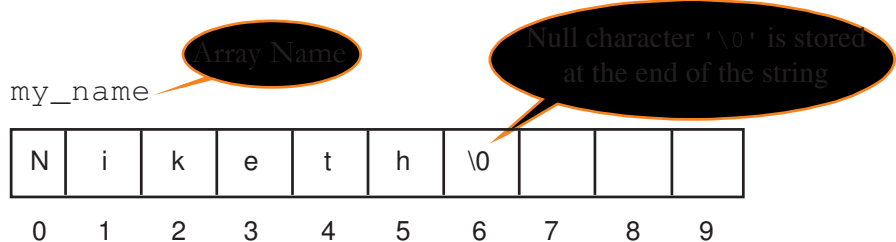
Note that the string constant is not "Hello", but "Hello " (a white space is given after the letter o).

**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.

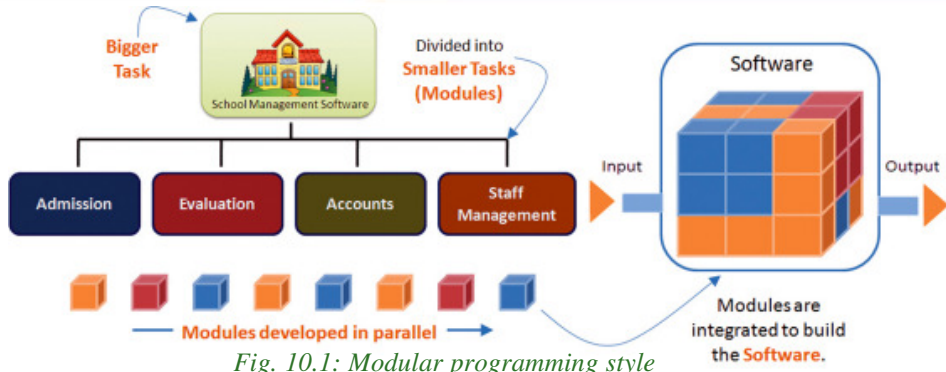


Fig. 10.1: Modular programming style

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)**

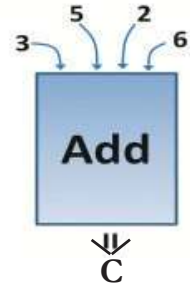


Fig. 10.3. Addition function

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** (`cctype.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 s1, s2 and s3 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><i>x</i></div> <div>20</div> </div> <div>change(int n)</div> <div> <pre>{     ..... }</pre> </div> <div> <div><i>n</i></div> <div></div> </div> | <div>main()</div> <div> <pre>{     .....     ..... }</pre> </div> <div> <div><i>x</i></div> <div>20</div> </div> <div>↓</div> <div> <div><i>n</i></div> <div>20</div> </div> <div>change(int n)</div> <div> <pre>{     ..... }</pre> </div> <div> <div><i>n</i></div> <div>20</div> </div> | <div>main()</div> <div> <pre>{     .....     ..... }</pre> </div> <div> <div><i>x</i></div> <div>20</div> </div> <div>change(int n)</div> <div> <pre>{     ..... }</pre> </div> <div> <div><i>n</i></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.

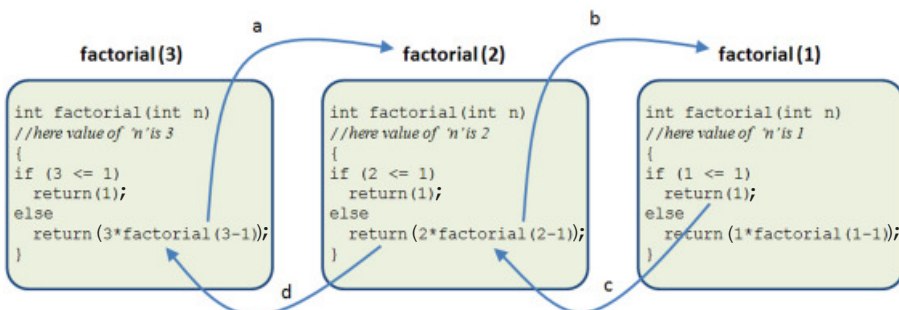
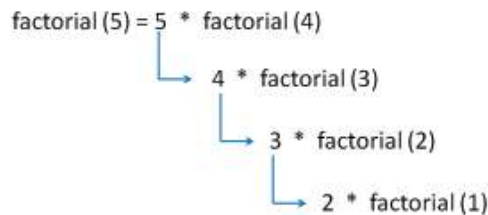


Fig. 10.4: Control flow in a recursive function call

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.

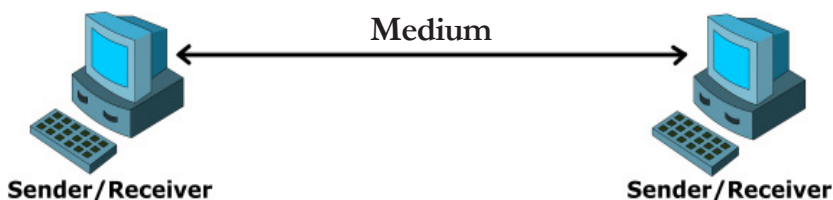


*Make a list of the hardware and software resources shared in your school network.*

**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.

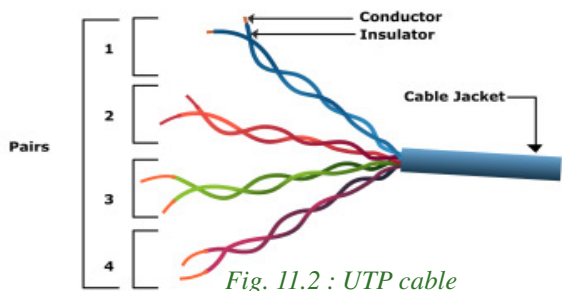


Fig. 11.2 : UTP cable



### *Characteristics of UTP cable*

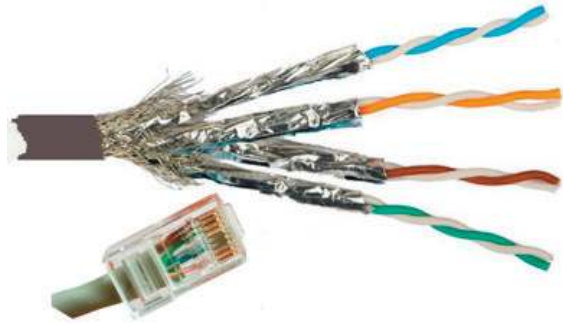
- Low cost cable available for setting up small networks.
- Thin and flexible cable.
- Ease of installation.
- Carries data upto a length of 100 m at a stretch.

**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.



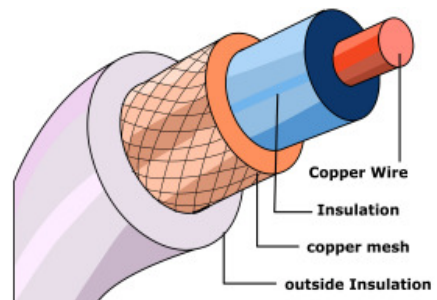
*Fig. 11.3 : STP cable and RJ-45 connector*

### **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*

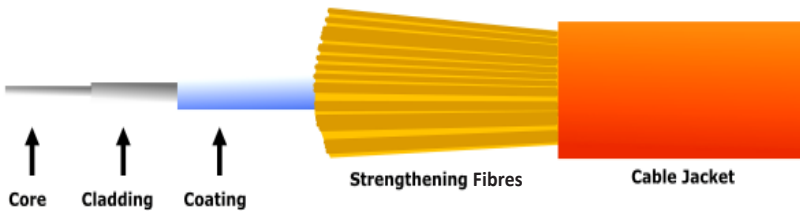
- Carries data to longer distances (185 m - 500 m approx.) at a stretch.
- High bandwidth.
- Less electromagnetic interference due to the outer shield.
- Thicker than twisted pair.
- Less flexible than twisted pair.
- Difficult to install than twisted pair cable.



*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.

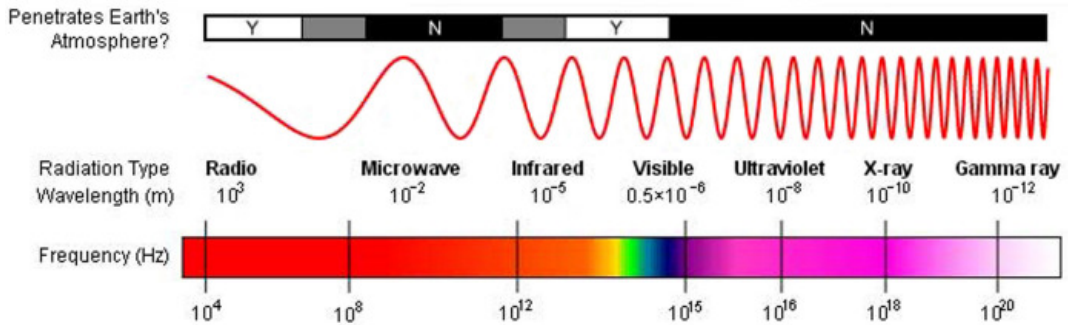


Fig. 11.6: Electromagnetic Spectrum

#### 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.

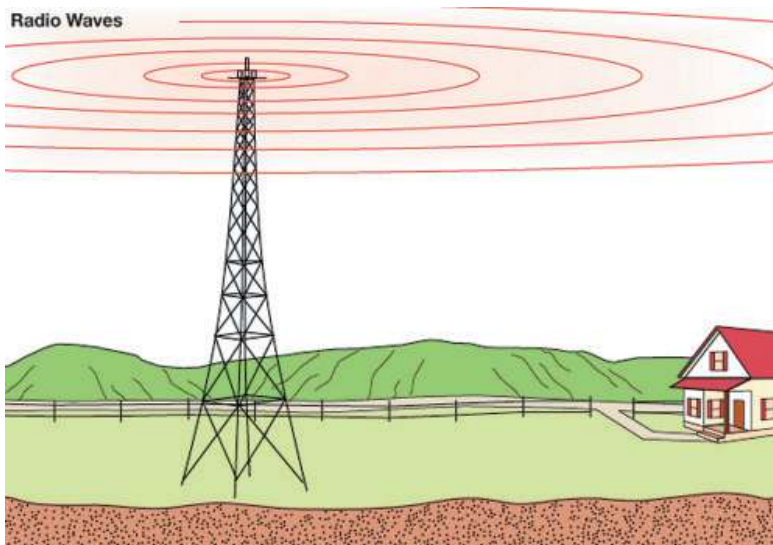


Fig. 11.7 : Radio wave transmission

### 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.

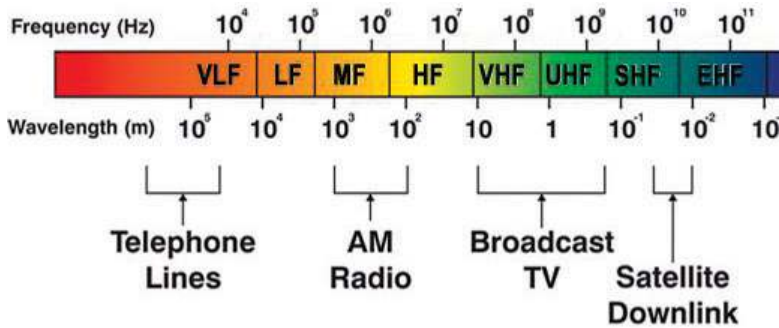


Fig. 11.8 : Spectrum of radio communication band

### 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.

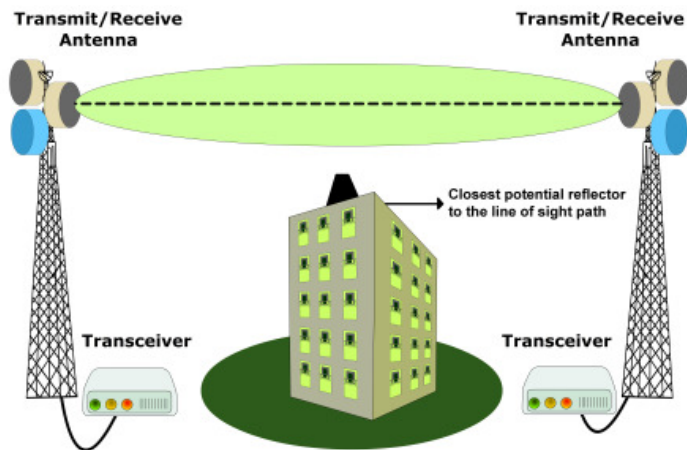


Fig. 11.9 : Microwave transmission

### *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*

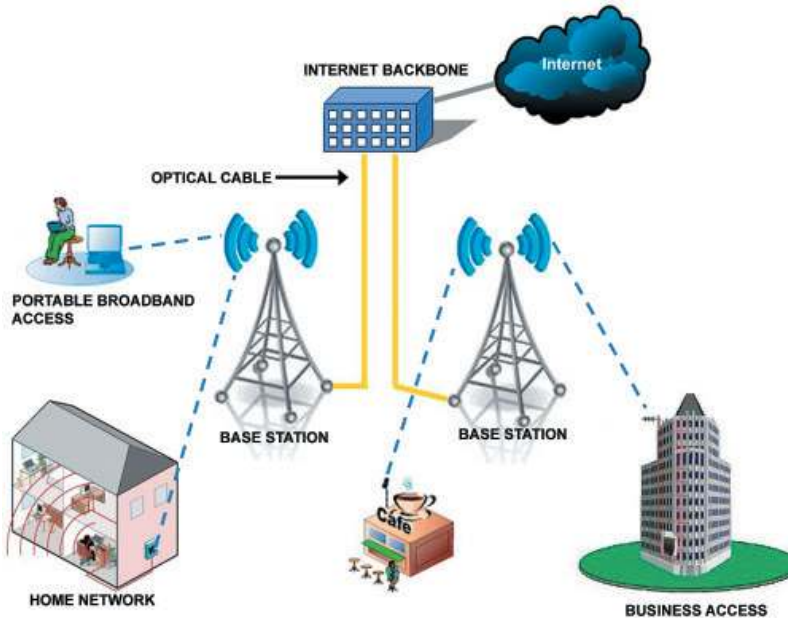
### *Characteristics of Wi-Fi transmission*

- 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*

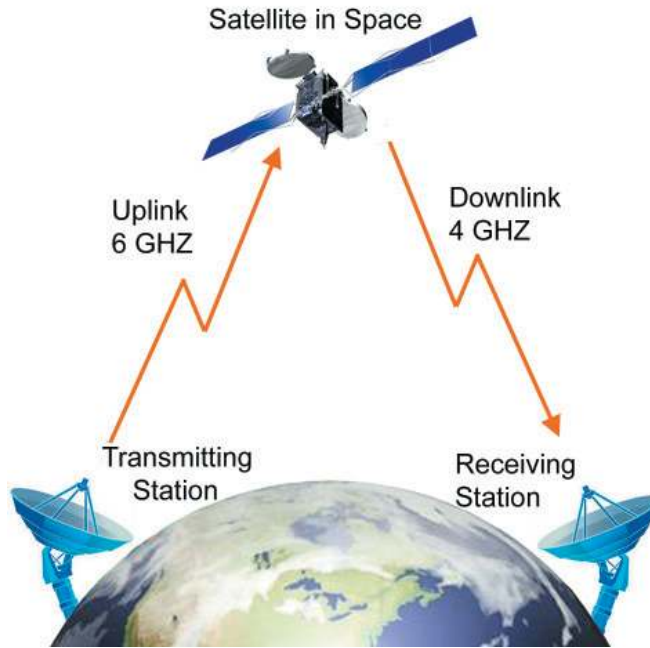
- Hundreds of users can connect to a single station.
- Provides higher speed connection upto 70 Mbps over an area of 45 Kilometres.
- Line of sight between communicating devices is not required.
- Weather conditions like rain, storm, etc. could interrupt the signal.
- Very high power consumption.
- High 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

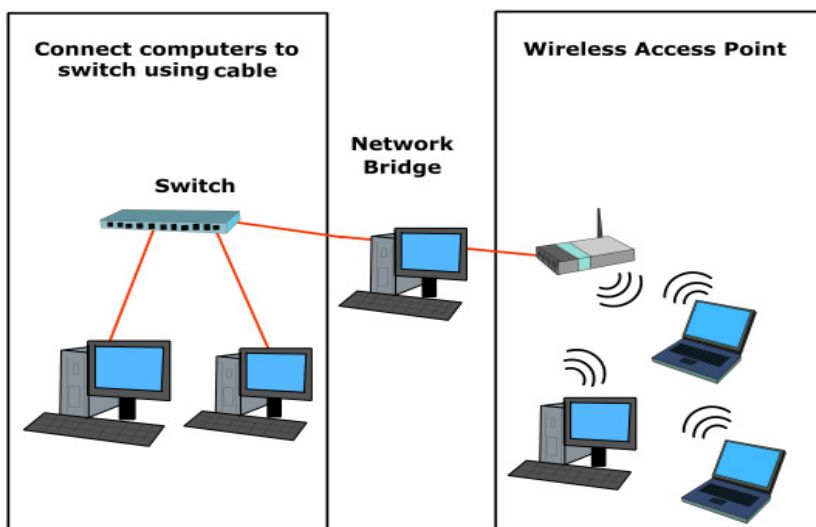
A repeater is a device that regenerates incoming electrical, wireless or optical signals through a communication medium (refer Figure 11.17). Data transmissions through wired or wireless medium can travel only a limited distance as the quality of the signal degrades due to noise. Repeater receives incoming 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.

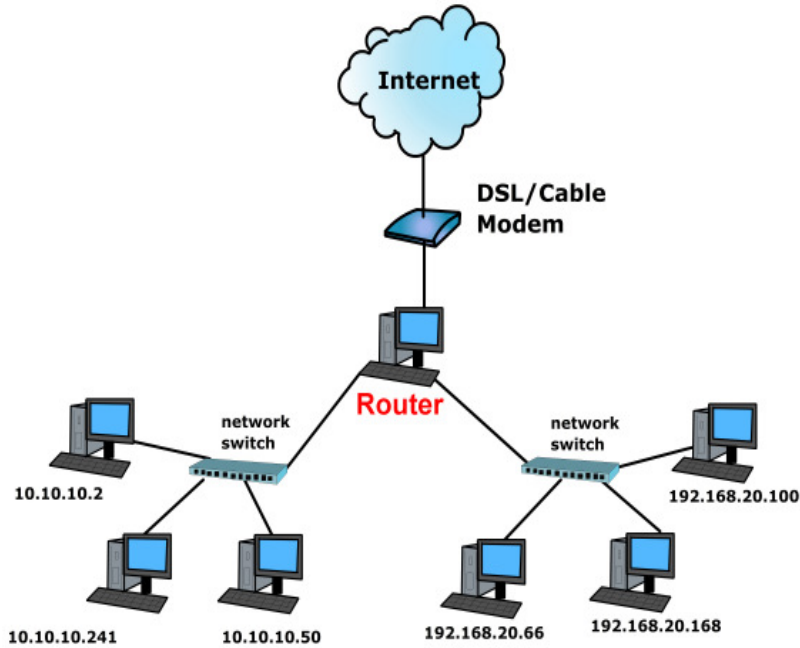


Fig. 11.19 : Router

### 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

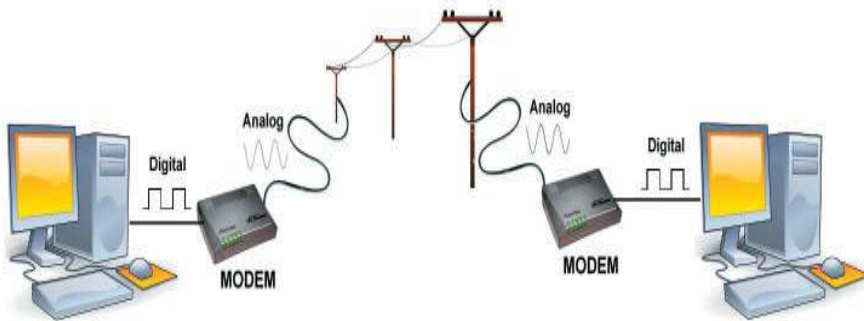
best path for packets to reach the destination. Further, while interconnecting two networks with different protocols, there must be some mutual understanding between the networks. A gateway is capable of understanding the address structure used in different networks and seamlessly translate the data 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.

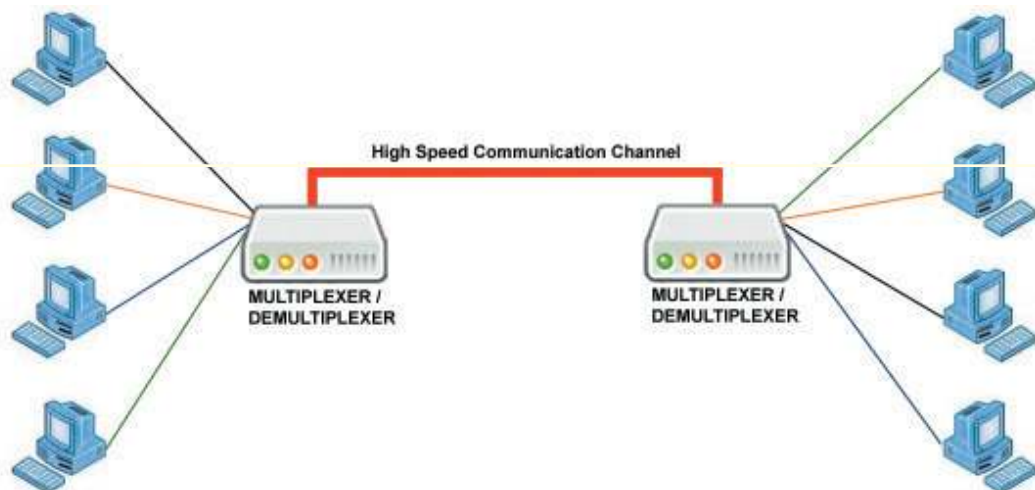


Fig. 11.22 : Multiplexer/De-multiplexer



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 \_\_\_\_\_.

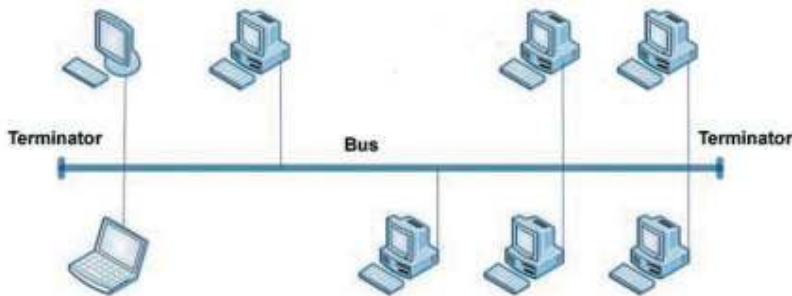
## 11.6 Network topologies

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.

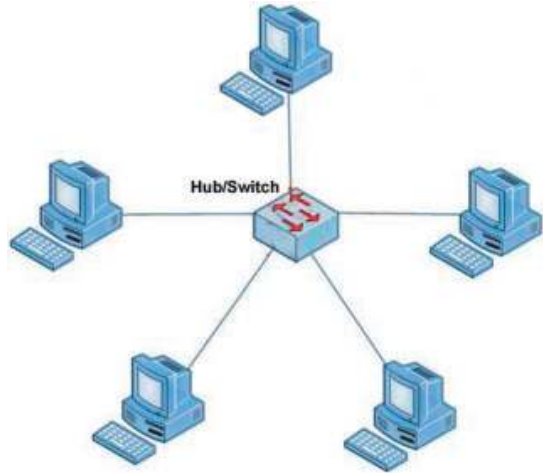
### 11.6.2 Star topology

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***

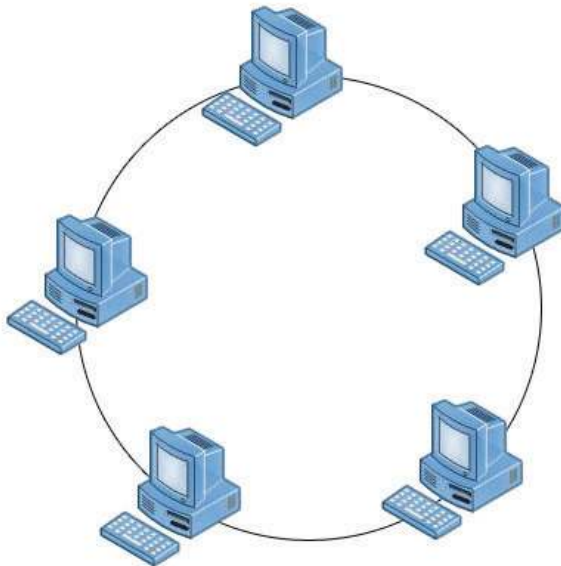
- More efficient compared to bus topology.
- Easy to install.
- Easy to diagnose faults.
- Easy to expand depending on the specifications of central hub/switch.
- Failure of hub/switch leads to failure of entire network.
- Requires more cable length compared to bus topology.



*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*

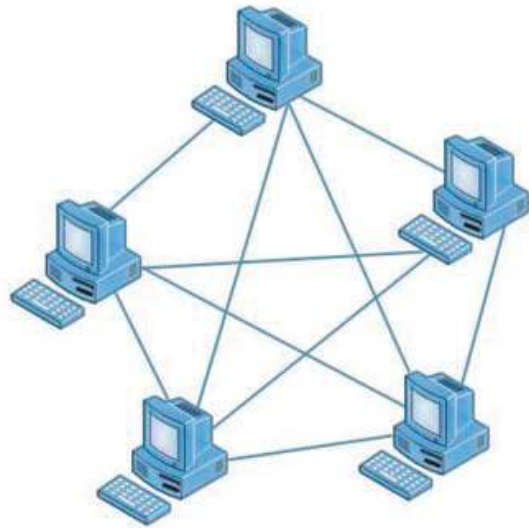


### *Characteristics of ring topology*

- 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).

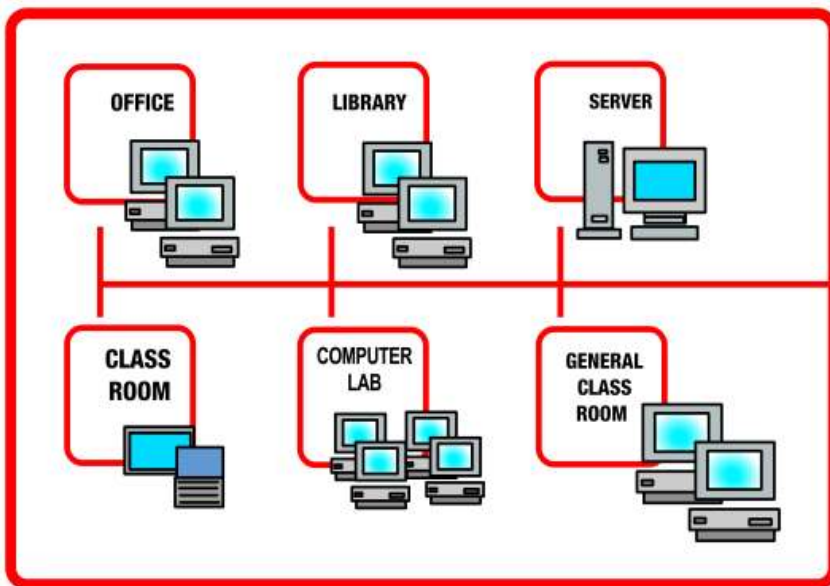
When we transfer songs from one cell phone to another or from a PC to an MP3 player, a PAN is set up between the two. PAN can be set up using guided media (USB) or unguided media (Bluetooth, infrared).



*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*

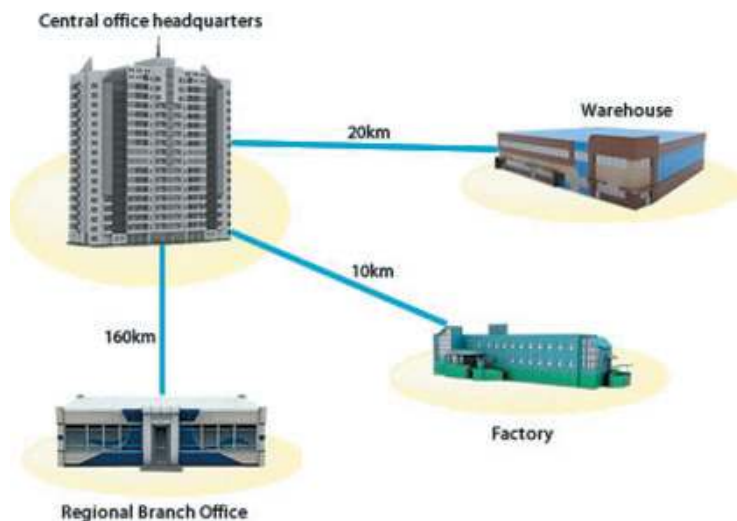


In addition to operating in a limited space, a LAN is owned, controlled and managed by a single person or an organisation.

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.

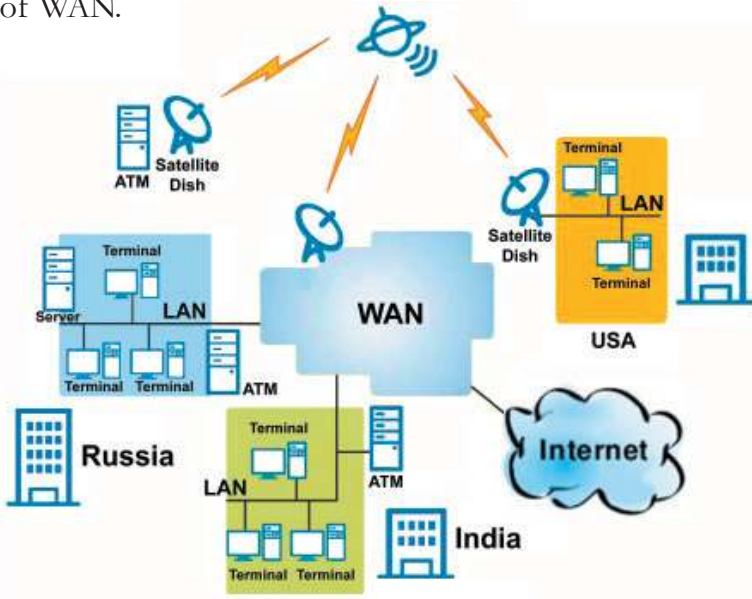


Fig. 11.30 : Wide Area Network

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 speed | High speed                        | High speed                                                  | Moderate speed                                       | Low speed                                 |
| Networking cost    | Negligible                        | Inexpensive                                                 | Moderately 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.

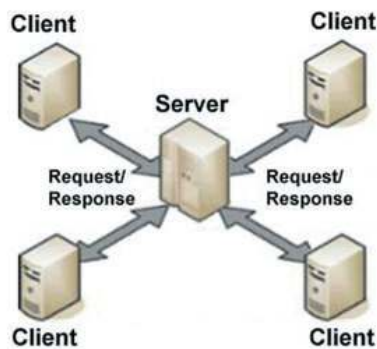


Fig. 11.31 : Client - Server

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.

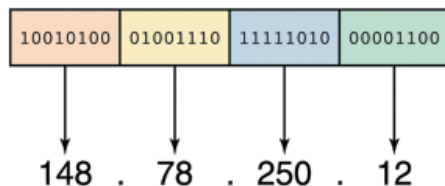


Fig. 11.32 : IP address

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.



Let us do

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).

| 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.

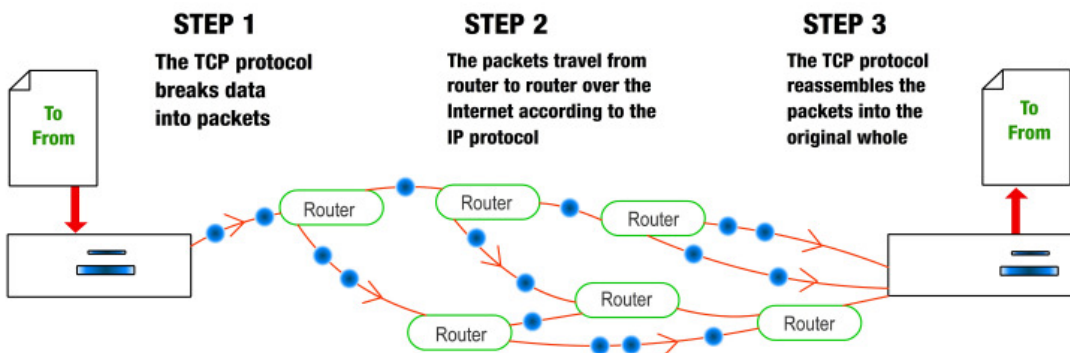
Several computer network protocols have been developed for specific purposes and environments. Some commonly used protocols are TCP/IP, SPx/IPx, etc.

### 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.

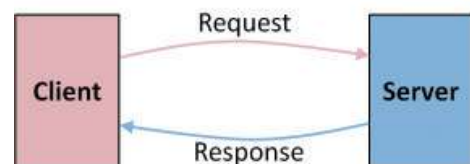


*Fig: 11.33 : How TCP/IP works*

### 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**

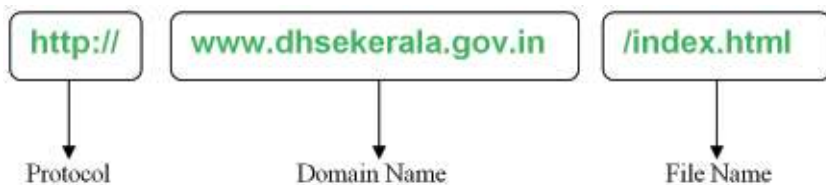
## **11.11 Uniform Resource Locator (URL)**

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`, `keralareresults.nic.in`, `google.com`, `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
- a. URL
  - b. an IP address
  - c. website
  - d. all of the above

### Short answer type

1. Define bandwidth.
2. Switch and Hub are two devices associated with networking. Differentiate them.
3. What is an IP address? Give an example.
4. What is TCP/IP? What is its importance?
5. Define a computer network.
6. What is Bluetooth?
7. What is a Modem?
8. Distinguish between router and gateway.
9. Explain the need for establishing computer networks.
10. What are the uses of computer networks?
11. What is the limitation of microwave transmission? How is it eliminated?
12. Briefly describe the characteristics of Wi-Fi.
13. 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.
14. What is NIC? What is its importance in networking?
15. 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.
16. You need to transfer a biodata file stored in your computer to your friend's computer that is 10 kms away using telephone network
  - a. Name the device used for this at both ends.
  - b. Explain how the file is send and received inside the device, once a connection between two computers is established.
17. When is a repeater used in a computer network?
18. Compare infrared and Bluetooth transmission.
19. Identify and explain the device used for connecting a computer to a telephone network.
20. 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.

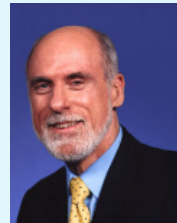


## 12.1 History of the Internet

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.

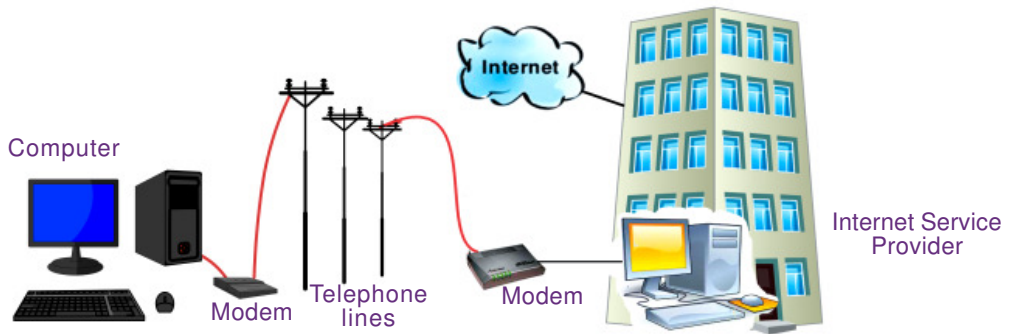
Note that data transfer rate is measured in decimal units and memory is measured in binary.

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

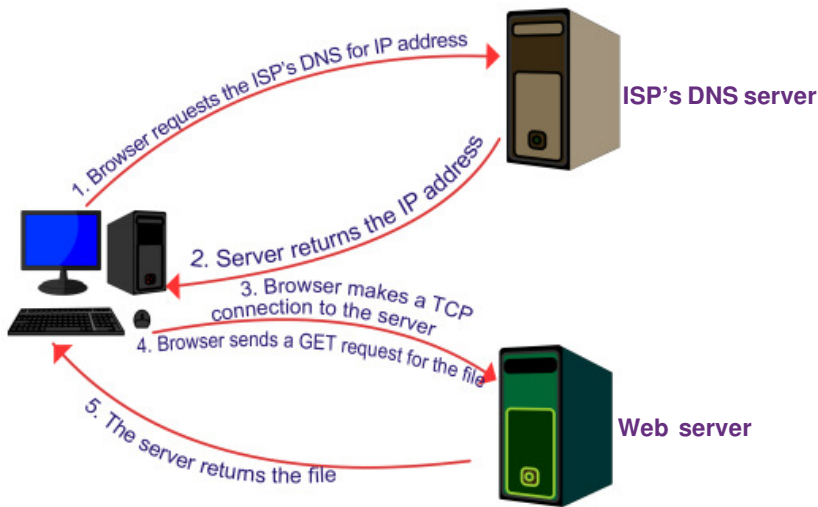
A web browser is a software that we use to retrieve or present information and to navigate through web pages in the World Wide Web. The document to be displayed is identified using a URL. A URL consists of its DNS name and the file name to be retrieved. It also specifies the protocol for transferring 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.



*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 web sites 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

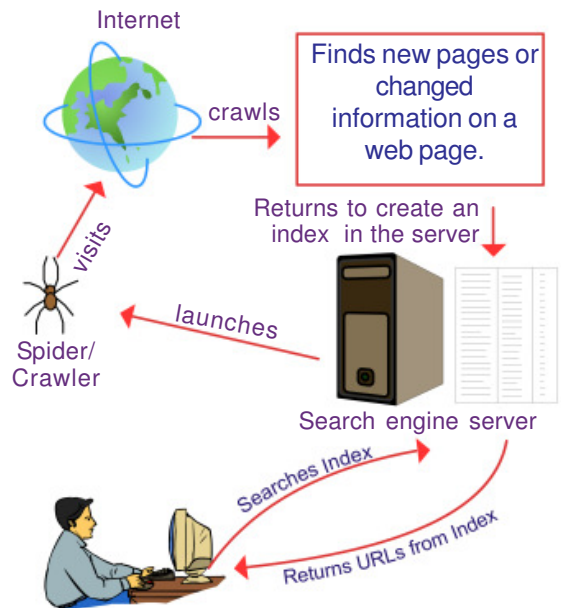


Fig. 12.8 : Working of a search engine

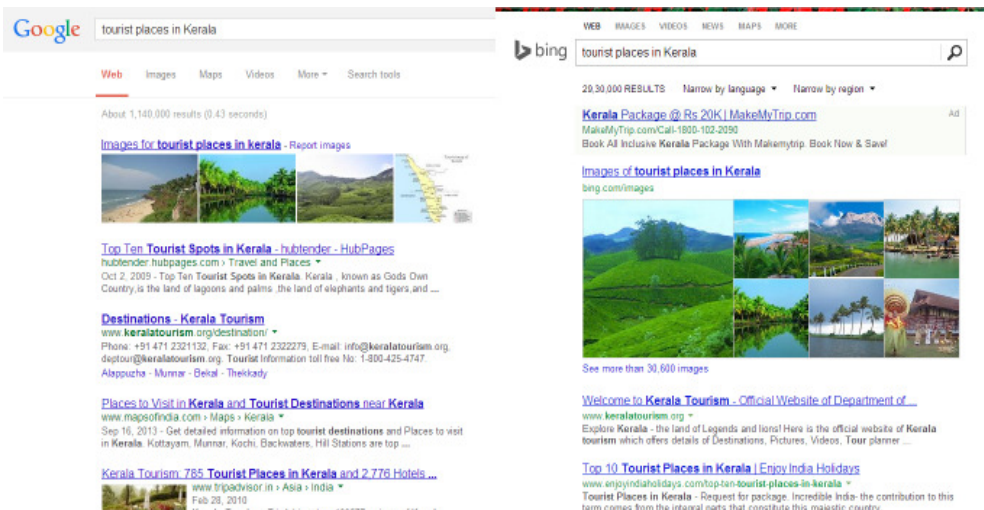


Fig. 12.9 : Search results of different search engines

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.

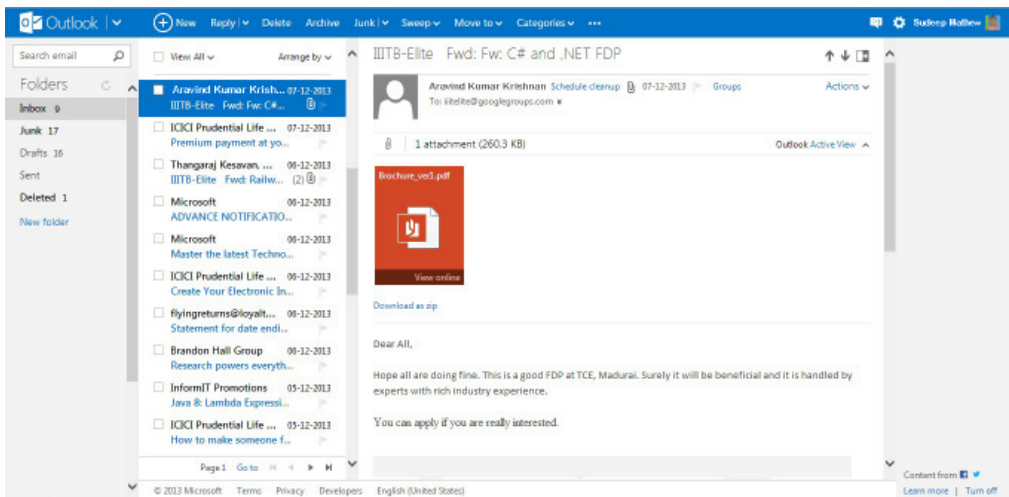


Fig. 12.10 – A sample e-mail message

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.

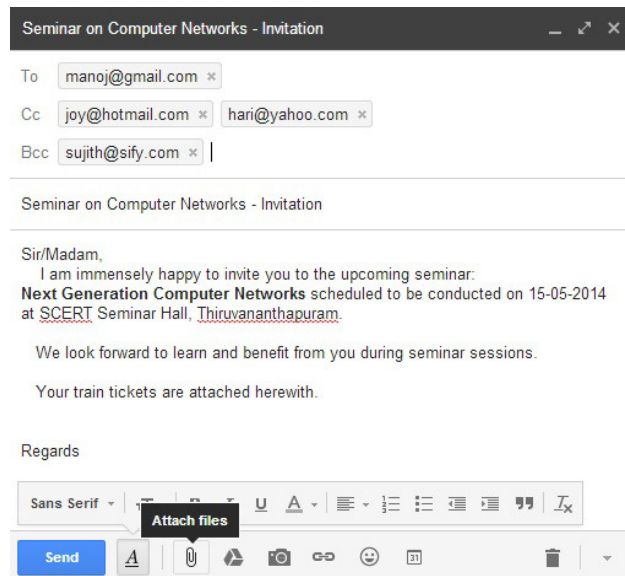


Fig. 12.11 : Composing an e-mail

**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.

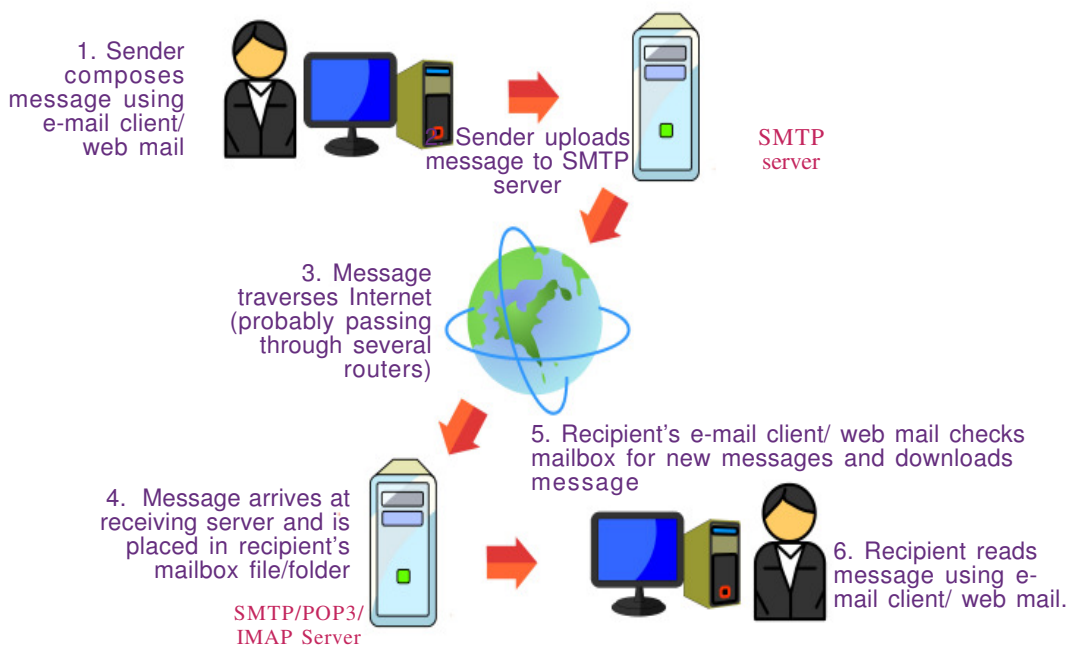


Fig. 12.12: Working of email

### 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 sent 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**

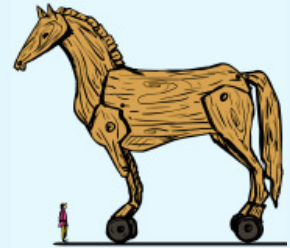
E-mail users received a message that offered a free upgrade to Internet Explorer that contained an executable file Ie0199.exe as attachment. This e-mail instructed the user to download and install this program for the upgrade. The users who followed these instructions got their files infected.





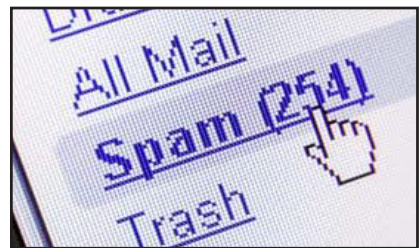
### Trojan War

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.

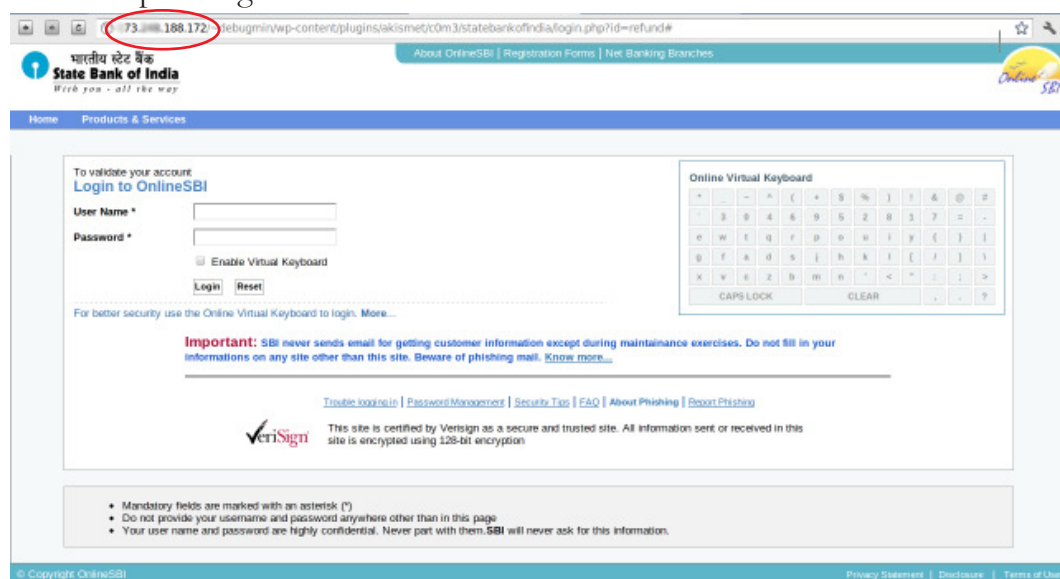
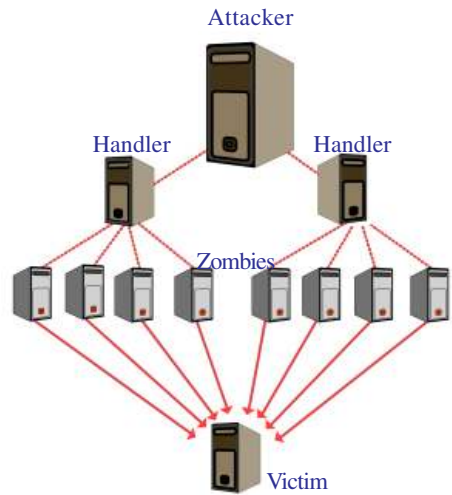


Fig. 12.15: A phishing website

### 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.

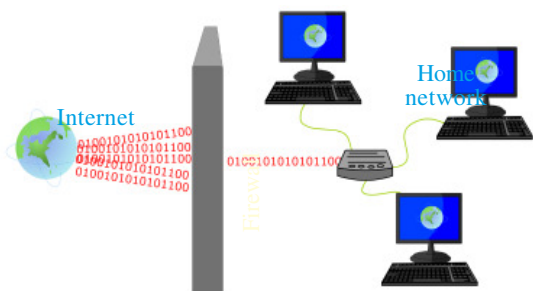


Fig. 12.17: Firewall

### 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.

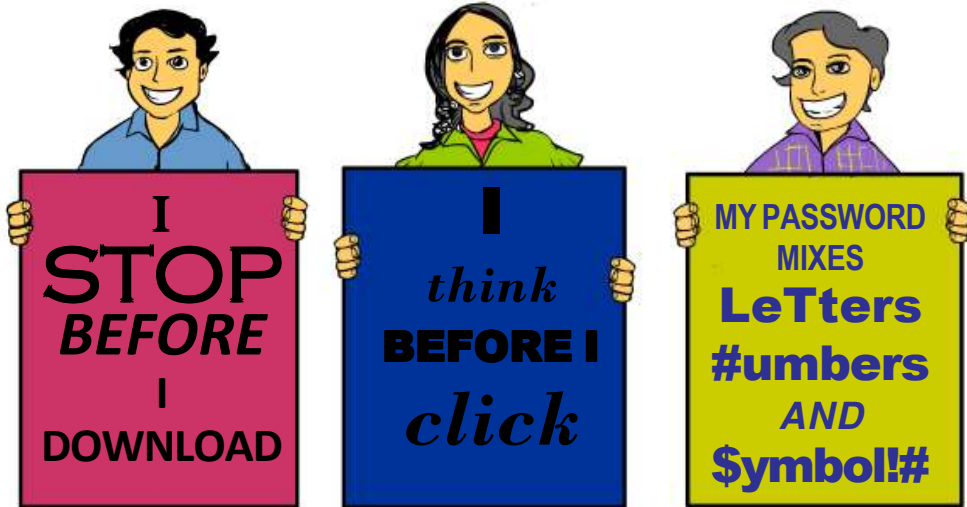
### 12.6.3 Cookies

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.

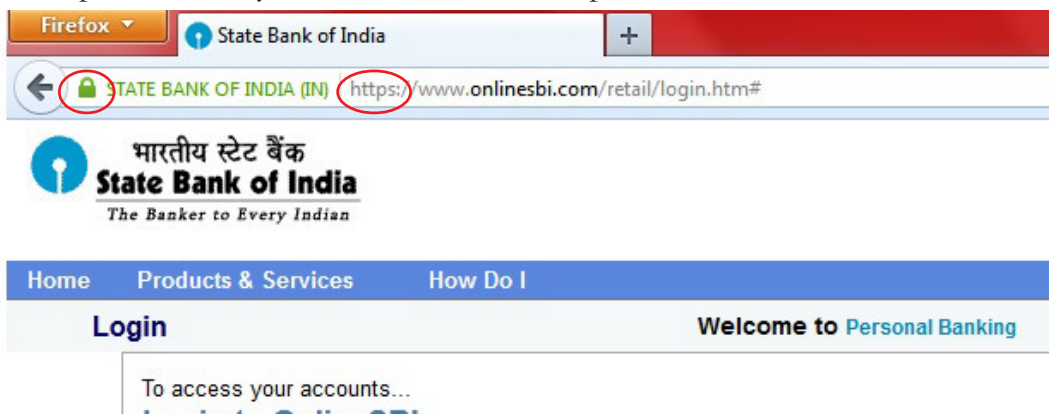


Fig 12.18: Secure banking - lock icon and https



- 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.

### Let us do

- 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'.

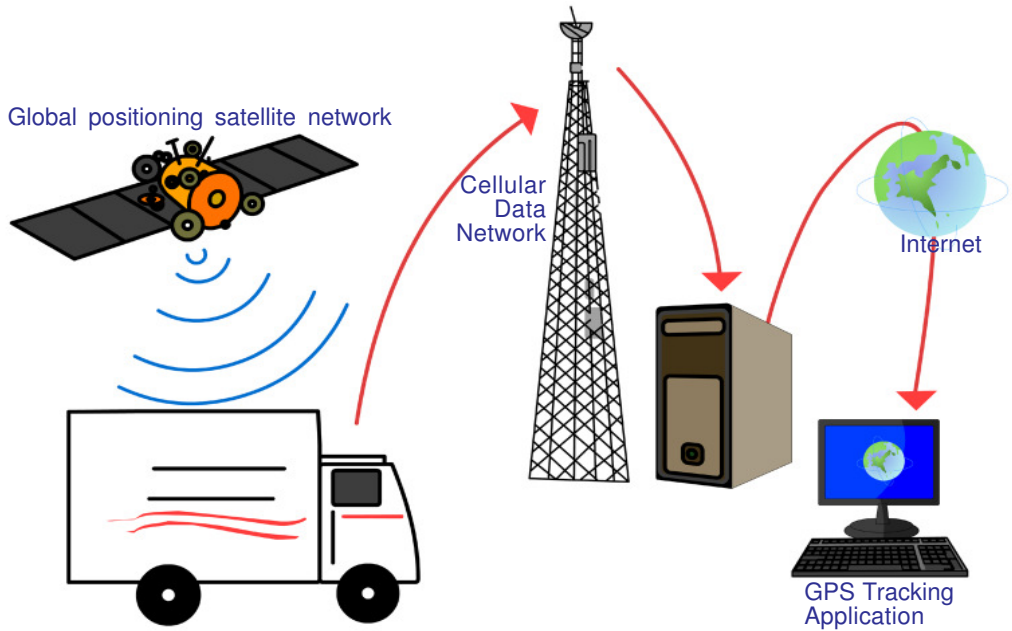
### b. Multimedia Messaging Service (MMS)

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