Higher Secondary Course

XII

Computer Science

Part 1

Government of Kerala
DEPARTMENT OF EDUCATION
State Council of Educational Research and Training (SCERT); Kerala
2015

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.

In Class XI, we started with the history of computing followed by hardware and software components, computer network and Internet. The major part of the textbook as well as the syllabus established a strong foundation to construct and enhance the problem solving and programming skills of the learner.

The syllabus of Class XII gives thrust on the means of handling complex data involved in problem solving and the development of web applications. The textbook, designed in accordance with the syllabus, begins with some advanced features of C++ programming language like structures and pointers. Web technology is introduced with the theoretical background and proceeds with the design of simple web pages followed by a brief idea about both client-side and server-side scripting. The concept of database and facilities of information retrieval are included. An exclusive section about the advances in computing opens the door to the latest developments in the world of computing discipline. Considering the increase in the use of Internet, an awareness about Cyber laws is also presented to safeguard against Cyber crimes.

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 S. Raveendran Nair
Director
SCERT, Kerala

Textbook Development Team

Joy John
HSST, St. Joseph's HSS
Thiruvananthapuram.

Vinod V.
HSST, NSS HSS, Prakkulam, Kollam.

A. N. Sathian
HSST, GM HSS, Koyilandy,
Kozhikode.

A. S. Ismael
HSST, PJMS GHSS, Kandassankadavu,
Thrissur.

Roy John
HSST, St. Aloysius HSS
Elthuruth, Thrissur

Prasanth P. M.
HSST, St. Joseph's Boys' HSS,
Kozhikode.

Sunil Kariyatan
HSST, Govt. Brennen HSS, Thalassery

Rajamohan C.
HSST, Nava Mukunda HSS, Thirunavaya,
Malappuram.

T. Mohammed Salim
HSST, Oriental HSS, Thirurangadi,
Malappuram.

Experts

Dr Lajish V. L.
Assistant Professor, Dept. of Computer Science, University of Calicut

Madhu V. T.
Director, Computer Centre, University of Calicut

Dr Sushil Kumar R.
Associate Professor, Dept. of English, D.B. College, Sasthamcotta

Vinayakumaran Nair N.
Assistant Commandant, Hi-Tech Cell, Police Head Quarters, Trivandrum

Dr Madhu S. Nair
Assistant Professor, Dept. of Computer Science, University of Kerala

Dr Binu P. Chacko
Associate Professor, Dept. of Computer Science, Prajyoti Niketan College, Pudukad, Thrissur

Dr Deepa L. C.
Assistant Professor, Dept. of English, Govt. Women's College, Trivandrum

Dr Kabeer V.
Asst. Prof & Head, Dept. of Computer Science, Farook College, Kozhikode

Artists

Sudheer Y. **Vineeth V.**

Academic Co-ordinator

Dr Meena S.
Research Officer, SCERT

Contents

1. Structures and Pointers 7

1.1 Structure
1.2 Pointer
1.3 Methods of memory allocation
1.4 Operations on pointers
1.5 Pointer and Array
1.6 Pointer and String
1.7 Pointer and Structure

2. Concepts of Object Oriented Programming 41

2.1 Programming paradigm
2.2 Basic concepts of OOP

3. Data Structures and Operations 59

3.1 Data structure
3.2 Stack
3.3 Queue
3.4 Linked list

4. Web Technology 85

4.1 Communication on the web
4.2 Web server technologies
4.3 Web designing
4.4 Static and dynamic web pages
4.5 Scripts
4.6 Cascading Style Sheet
4.7 Basic concepts of HTML documents
4.8 Creating an HTML document
4.9 Essential HTML tags
4.10 Some common tags
4.11 HTML entities for reserved characters
4.12 Adding comments in HTML document
4.13 Inserting images

5. Web Designing Using HTML 135

5.1 Lists in HTML
5.2 Creating links
5.3 Inserting music and video
5.4 Creating tables in a web page
5.5 Dividing the browser window
5.6 Forms in web pages
5.7 Overview of HTML 5

6. Client Side Scripting Using JavaScript 177

6.1 Getting started with JavaScript
6.2 Creating functions in JavaScript
6.3 Data types in JavaScript
6.4 Variables in JavaScript
6.5 Operators in JavaScript
6.6 Control structures in JavaScript
6.7 Built-in functions
6.8 Accessing values in a text box using JavaScript
6.9 Ways to add scripts to a web page

7. Web Hosting 221

7.1 Web hosting
7.2 Free hosting
7.3 Content Management System
7.4 Responsive web design

Icons used in this textbook

Let us do

Know your progress

Information box

Let us practice

Let us conclude

1 Structures and Pointers

Significant Learning Outcomes

After the completion of this chapter, the learner

- identifies the need of user-defined data types and uses structures to represent grouped data.
- creates structure data types and accesses elements to refer to the data items.
- uses nested structures to represent data consisting of elementary data items and grouped data items.
- develops C++ programs using structure data types for solving real life problems.
- explains the concept of pointer and uses pointer with the operators & and *.
- compares the two types of memory allocations and uses dynamic operators `new` and `delete`.
- illustrates the operations on pointers and predicts the outputs.
- establishes the relationship between pointer and array.
- uses pointers to handle strings.
- explains the concept of self referential structures.

We started writing C++ programs in Class XI for solving problems. Almost all problems are related to the processing of different types of data. Last year we came across only elementary data items such as integers, fractional numbers, characters and strings. We used variables to refer to these data and the variables are declared using basic data types of C++. We know that all data is not of fundamental (basic) types; rather many of them may be composed of elementary data items. No programming language can provide data types for all kinds of data. So programming languages provide facility to define new data types, as desired by the users. In this chapter, we will discuss such a user-defined data type named structure. This chapter also discusses a new kind of variable known as pointer. The concept of pointers is a typical feature of languages like C, C++. It helps to access memory locations by specifying the memory addresses directly, which makes execution faster. A good understanding of this concept will help us to design data structure applications and system level programs.

Last year you might have used GNU Compiler Collection (GCC) with Geany IDE or Turbo C++ IDE for developing C++ programs.

GCC differs from Turbo C++ IDE in the structure of source code, the way of including header files, the size of int and short, etc. In this chapter the concepts are presented based on GCC.

1.1 Structures

Now-a-days students, employees, professionals, etc. wear identity cards issued by institutions or organisations. Figure 1.1 shows the identity card of a student. The first column of Table 1.1 contains some of the data printed on the card. You try to fill up the second column with the appropriate C++ data types discussed in Class XI.

Fig.1.1: ID card of a student

Data	C++ data type
12345 Sneha S. Raj 20/02/1997 O+ve Snehanilayam, Gandhi Nagar, Chemmanavattom, Pin 685 531	

Table 1.1: Data and C++ data types

You may use `short` or `int` for admission number (12345), `char` array for name (Sneha S. Raj), blood group (O +ve) and even address. Sometimes you may not be able to identify the most appropriate data types for date of birth and address. Let us consider the data 20/02/1997 and analyse the composition of this data. It is composed of three data items namely day number (10), month number (02) and year (1997). In some cases, the name of the month may be used instead of month number. Address can also be viewed as a composition of data items such as house number/name, place, district, state and PIN code. Even the entire details on the identity card can be considered as a single unit of data. Such data is known as grouped data (or aggregate data or compound data). C++ provides facility to define new data types by which such aggregate or grouped data can be represented. The data types defined by user to represent data of aggregate nature are generally known as user-defined data types.

Structure is a user-defined data type of C++ to represent a collection of logically related data items, which may be of different types, under a common name. We learnt array in Class XI, to refer to a collection of data of the same type. But structure

can represent a group of different types of data under a common name. Let us discuss how a structure is defined in C++ and elements are referenced.

1.1.1 Structure definition

While solving problems, the data to be processed may be of grouped type as mentioned above. We have to define a suitable structure to represent such grouped data. For that, first we have to identify the elementary data items that constitute the grouped data. Then we have to adopt the following syntax to define the structure.

```
struct  structure_tag
{
    data_type  variable1;
    data_type  variable2;
    ..................;
    ..................;
    data_type  variableN;
};
```

In the above syntax, **`struct`** is the keyword to define a structure, `structure_tag` (or `structure_name`) is an identifier and `variable1`, `variable2`, ..., `variableN` are identifiers to represent the data items constituting the grouped data. The identifier used as the structure tag or structure name is the new user-defined data type. It has a size like any other data types and it can be used to declare variables. This data type can also be used to specify the arguments of functions and as the return type of functions. The variables specified within the pair of braces are known as elements of the structure. The data types preceded by these elements may be basic data types or user-defined data types. These data types determine the size of the structure.

Now let us define a structure to represent dates of the format 20/02/1997 (as seen in the ID card). We can see that this format of date is constituted by three integers which can be represented by `int` data type of C++. The following is the structure definition for this format of date:

```
struct date
{
    int dd;
    int mm;
    int yy;
};
```

Here, `date` is the structure tag (structure name), and `dd`, `mm` and `yy` are the elements of the structure `date`, all of them are of `int` type. If we want to specify the month as a string (like January instead of 1), the definition can be modified as:

```
struct strdate
{
    int day;
    char month[10];      // name of month is a string
    int year;
};
```

While writing programs for solving problems, some of the data involved may be logically related in one way or the other. In such cases, the concept of structure data type can be utilised effectively to combine the data under a common name to represent data compactly. For example, the student details such as admission number, name, group, fee, etc. are logically related and hence a structure can be defined as follows:

```
struct student
{
    int adm_no;
    char name[20];
    char group[10];
    float fee;
};
```

Let us do

Now, try to define separate structures yourself to represent **address** and **blood group**. Blood group consists of group name and rh value.

We know that the details of an employee may consist of employee code, name, gender, designation and salary. Define a suitable structure to represent these details.

We have discussed the way of defining a structure type data. Can we now store data in it? No, it is simply a data type definition. Using this data type we should declare a variable to store data.

1.1.2 Variable declaration and memory allocation

As in the case of basic data items, a variable is required to refer to a group of data. Once we have defined a structure data type, variable is declared using the following syntax:

```
struct structure_tag var1, var2, ..., varN;
```

OR

```
structure_tag var1, var2, ..., varN;
```

In the syntax, `structure_tag` is the name of the structure and `var1`, `var2`, ..., `varN` are the structure variables. Let us declare variables to store some dates using the structure `date` and `fulldate`.

```
date dob, today;          OR    struct date dob, today;
strdate adm_date, join_date;
```

We know that variable declaration statement causes memory allocation as per the size of the data type. What will be the size of a structure data type? Since it is user-defined, the size depends upon the definition of the structure. The definition of `date` shows that its variables require 12 bytes each because it contains three `int` type elements (size of `int` in GCC is 4 bytes). The memory allocation for the variable `join_date` of `strdate` type is shown in Figure 1.2.

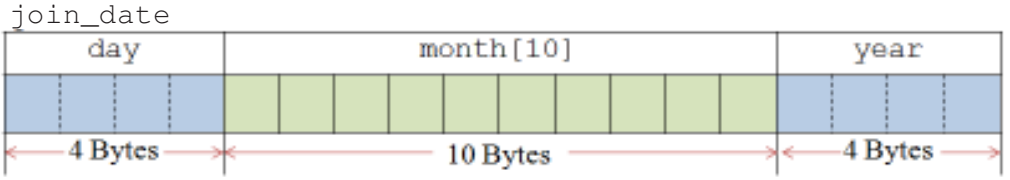


Fig. 1.2: Memory allocation for a structure variable

The size of `int` type in GCC is 4 bytes and in Turbo IDE, it is 2 bytes. In Figure 1.2, the elements day and year are provided with 4 bytes of memory since we follow GCC. Since this much memory is not required for storing values in these elements, it is better to replace int with `short`.

The variable `join_date` consists of three elements `day`, `month` and `year`. These elements require 4 bytes, 10 bytes and 4 bytes, respectively and hence the memory space for `join_date` is 18 bytes.

Let us do

Now you find the size of the structure `student` defined earlier. Also write the C++ statement to declare a variable to refer to the details of a student and draw the layout of memory allocated to this variable.

A structure variable can be declared along with the definition also, as shown below:

```
struct complex
{
    short real;
    short imaginary;
}c1, c2;
```

This structure, named `complex` can represent complex numbers. The identifiers `c1` and `c2` are two structure variables, each of which can be used to refer to a

complex number. If we declare structure variables along with the definition, structure tag (or structure name) can be avoided. The following statement declares structure variables along with the definition.

```
struct
{
    int a, b, c;
}eqn_1, eqn_2;
```

There is a limitation in this type of definition cum declaration. If we want to declare variables, to define functions, or to specify arguments using this structure later in the program, it is not possible since there is no tag to refer. The above structure also shows that if the elements (or members) of the structure are of the same type, they can be specified in a single statement.

Variable initialisation

During the declaration of variables, they can be assigned with some values. This is true in the case of structure variables also. When we declare a structure variable, it can be initialised as follows:

```
structure_tag variable={value1, value2,..., valueN};
```

For example, the details of a student can be stored in a variable during its declaration itself as shown below:

```
student s={3452, "Vaishakh", "Science", 270.00};
```

The values will be assigned to the elements of structure variable `s` in the order of their position in the definition. So, care should be given to the order of appearance of the values. The above statement allocates 38 bytes of memory space for variable `s`, and assigns the values 3452, "Vaishakh", "Science" and 270.00 to the elements `adm_no`, `name`, `group` and `fee` of `s`, respectively.

If we do not provide values for all the elements, the given values will be assigned to the elements on First Come First Served (FCFS) basis. The remaining elements will be assigned with 0 (zero) or '\0' (null character) depending on numeric or string.

A structure variable can be assigned with the values of another structure variable. But both of them should be of the same structure type. The following is a valid assignment:

```
student st = s;
```

This statement initialises the variable `st` with the values available in `s`. Figure 1.3 shows this assignment:

s

adm_no	name	group	fee
3452	Vaishakh	Science	270.00

st

adm_no	name	group	fee
3452	Vaishakh	Science	270.00

Fig. 1.3: Structure assignment

While defining a structure, the elements specified in it cannot be assigned with initial values. Though the elements are specified using the syntax of variable declaration, memory is not allocated for the structure definition statement. Hence, values cannot be assigned to them. The structure definition can be considered as the blue print of a house. It shows the number of rooms, each with a specific name and size. But nothing can be stored in these rooms. The total space of the building will be the sum of the spaces of all rooms in the house. Any number of houses can be constructed based on this plan. All of them will be the same in terms of number of rooms and space, but each house will be given different name. Structure definition is the blue print and structure variables are the realisation of the blue print (similar to the construction of houses based on the blue print). Each variable can store data in its elements (similar to the placing of furniture, house-hold items and residents in rooms).

1.1.3 Accessing elements of structure

We know that array is a collection of elements and these elements are accessed using the subscripts. Structure is also a collection of elements and C++ allows the accessibility to the elements individually. The period symbol (`.`) is provided as the operator for this purpose and it is named ***dot operator***. The dot operator (`.`) connects a structure variable and its element using the following syntax:

```
structure_variable.element_name
```

In programs, the operations on the structure data can be expressed only by referring the elements of the structure. The following are some examples for accessing the elements:

```
today.dd = 10;
strcpy(adm_date.month, "June");
cin >> s1.adm_no;
cout << c1.real + c2.real;
```

But the expression `c1+c2` is not possible, since the operator + can be used with numeric data types only.

Let us discuss an interesting fact about assignment operation on structure variables. Two structures are defined as follows:

```
struct test_1
{
    int a;
    float b;
}t1={3, 2.5};
```

```
struct test_2
{
    int a;
    float b;
}t2;
```

The elements of both the structures are the same in number, name and type. The structure variable `t1` of type `test_1` is initialised with 3 and 2.5 for its `a` and `b`. But the assignment statement: `t2=t1;` is invalid, because `t1` and `t2` are of different types, i.e., `test_1` and `test_2`, respectively. But if we want to copy the values of `t1` into `t2`, the following method can be adopted:

```
t2.a = t1.a;        t2.b = t1.b;
```

It is possible because we are assigning an `int` type data into another `int` type variable.

Now let us write a program to implement the concepts discussed so far. We define a structure `student` to represent register number, name and scores awarded in continuous evaluation (CE), practical evaluation (PE) and term-end evaluation (TE). The details are input and the total score as part of Continuous and Comprehensive Evaluation (CCE) is displayed.

Program 1.1: To find the total score of a student

```
#include <iostream>
#include <cstdio>     //To use gets() function
using namespace std;
struct student  //structure definition begins
{
    int reg_no;  //Register number may exceed 32767, so int
    char name[20];
    short ce;//int takes 4 bytes, but ce score is a small number
    short pe;
    short te;
};                    //end of structure definition
int main()
{
    student s;        //structure variable
    int tot_score;
    cout<<"Enter register number: ";
    cin>>s.reg_no;
```

```
    fflush(stdin);    //To clear the keyboard buffer
    cout<<"Enter name: ";
    gets(s.name);
    cout<<"Enter scores in CE, PE and TE: ";
    cin>>s.ce>>s.pe>>s.te;
    tot_score=s.ce+s.pe+s.te;
    cout<<"\nRegister Number: "<<s.reg_no;
    cout<<"\nName of Student: "<<s.name;
    cout<<"\nCE Score: "<<s.ce<<"\tPE Score: "<<s.pe
     <<"\tTE Score: "<<s.te;
    cout<<"\nTotal Score     : "<<tot_score;
    return 0;
}
```

A sample output window of Program 1.1 is given below:

Output window:

```
Enter register number: 23545
Enter name: Deepika Vijay
Enter scores in CE, PE and TE: 19  38  54

Register Number: 23545
Name of Student: Deepika Vijay
CE Score: 19 PE Score: 38    TE Score: 54
Total Score     : 111
```

In Program 1.1, the structure is defined outside `main()` function. It may be defined inside `main()` also. The position of the definition determines the scope and life of the structure. Recollect the concept of local and global scope of variables and functions that we discussed in Chapter 10 of Class XI. If the definition is inside the `main()`, the structure can be used to declare variables within the `main()` function only. On the other hand, if the definition has a global scope, it allows declaration of structure variables in any function in the program.

Program 1.1 uses `fflush()` function before the `gets()` function. It is required in programs where an input of string facilitated by `gets()` function is followed by any other input. When we press <Enter> key as the delimiter for the former input, the `'\n'` character corresponding to the <Enter> key available in the keyboard buffer will be taken as the input for the string variable. This character will be considered as the delimiter for the string variable and the program control goes to the next statement in the program. In effect, we will not be able to input the actual string. So, we used `fflush()` function before inputting the name.

In Program 1.1, only one structure variable is used and hence the data of only one student can be referenced by the program at a time. If we have to deal with the details of a group of students, we will use an array of structures. So, let us write a program to illustrate the concept of array of structures. Program 1.2 accepts the details of a group of salesmen, each of which includes salesman code, name and amount of sales in 12 months. The program displays the entered details along with the average sales of all the salesmen. We can also see an array of floating point numbers as one of the elements of the structure.

Program 1.2: To find the average sales by salesmen

```
#include <iostream>
#include <cstdio>
#include <iomanip>    //To use setw() function
using namespace std;
struct sales_data
{
    int code;
    char name[15];
    float amt[12]; //To store the amount of sales in 12 months
    float avg;
};
int main()
{
    sales_data s[20]; //array of structure
    short n,i,j; //short is to minimise the amount of memory
    float sum;
    cout<<"Enter the number of salesmen: ";
    cin>>n;
    for(i=0; i<n; i++)
    {
        cout<<"Enter details of Salesman "<<i+1;
        cout<<"\nSalesman Code: ";
        cin>>s[i].code;
        fflush(stdin);
        cout<<"Name: ";
        gets(s[i].name);
        cout<<"Amount of sales in 12 months: ";
        for(sum=0,j=0; j<12; j++)
        {
            cin>>s[i].amt[j];
            sum=sum+s[i].amt[j];
        }
```

```
        s[i].avg=sum/12;
    }
    cout<<"\t\tDetails of Sales\n";
    cout<<"Code\t\tName\t\tAverage Sales\n";
    for(i=0;i<n;i++)
    {
        cout<<setw(4)<<s[i].code<<setw(15)<<s[i].name;
        for (j=0;j<12;j++)
            cout<<setw(4)<<s[i].amt[j];
        cout<<s[i].avg<<'\n';
    }
    return 0;
}
```

You may try out this program in the lab and see the output. In program 1.2, we used a floating point array as one of the elements of the structure. It uses array of structures for handling the details of different salesmen. Note that the variables `n`, `i` and `j` are declared using `short`. It allocates only 2 bytes for each of these variables. If `int` would have been used, 4 bytes would be used.

1.1.4 Nested structure

An element of a structure may itself be another structure. Such a structure is known as ***nested structure***. The concept of nesting enables the building of powerful data structures. If we want to include date of admission as an element in the structure `student`, any of the definitions given in Table 1.2 can cater to the need.

Definition A	Definition B

`struct date`
`{`
`  short day;`
`  short month;`
`  short year;`
`};`
`struct student`
`{`
`  int adm_no;`
`  char name[20];`
`  date dt_adm;`
`  float fee;`
`};`
	`struct student`
	`{`
	`  int adm_no;`
	`  char name[20];`
	`  struct date`
	`  {`
	`  short day;`
	`  short month;`
	`  short year;`
	`  } dt_adm;`
	`  float fee;`
	`};`

Table 1.2: Two styles of nesting

Definition A of Table 1.2 contains the two structures defined separately. The second structure, `student`, contains structure variable `dt_adm` of `date` type as an element. Here we have to make sure that the inner structure is defined before making it nested. But in definition B we can see that structure `date` is defined inside the structure `student`. If this style is followed, the scope of `date` is only within `student` structure and hence a variable of type `date` cannot be declared outside `student`. Since the variable declaration of the inner structure is essential, its tag may be avoided in the definition. The following statements illustrate how a nested structure variable is initialised and the elements are accessed:

```
student s = {4325, "Vishal", {10, 11, 1997}, 575};
cout<<s.adm_no<<s.name;
cout<<s.dt_adm.day<<"/"<<s.dt_adm.month<<"/"<<s.dt_adm.year;
```

Let us do

Define a structure `employee` with the details employee code, name, date of joining, designation and basic pay.
Draw the layout of memory location allocated to a variable of `employee` type and find its size.

Note that the format for accessing the inner structure element is:

```
outer_structure_varaiable.inner_structure_variable.element
```

Array Vs Structure

We discussed arrays and structures as data types to refer to a collection of data under a common name. But they differ in some aspects. Table 1.3 shows a comparison between these two data types.

Arrays	Structures
• It is a derived data type. • A collection of same type of data. • Elements of an array are referenced using the corresponding subscripts. • When an element of an array becomes another array, multi-dimensional array is formed. • Array of structures is possible.	• It is a user-defined data type • A collection of different types of data. • Elements of structure are referenced using dot operator (.) • When an element of a structure becomes another structure, nested structure is formed. • Structure can contain arrays as elements

Table 1.3: Comparison between arrays and structures

Know your progress

1. What is structure?
2. Structure combines different types of data under a single unit. State whether this is true or false.
3. Which of the following is true for accessing an element of a structure?
   a. `struct.element`
   b. `structure_tag.element`
   c. `structure_variable.element`
   d. `structure_tag.structure_variable`
4. What is nested structure? Write an example.
5. As subscript is for array, _____ is associated with structure.

1.2 Pointers

Suppose we have to prepare an assignment paper on 'Advances in Computing'. We may need suitable books for collecting the material. Obviously we may search for the books in the library. We may not be able to locate the book in the library. The librarian or our Computer Science teacher can help us to access the book. Let us think of the role of the librarian or the teacher. He/she is always a reference. He/she can provide us with the actual data (book) that is stored somewhere in the library. Figure 1.4 illustrates this example.

Fig. 1.4: Example for reference

Pointer is something like the librarian or teacher in the above example. It is a kind of reference. Consider the following C++ statement:

```
int num=25;
```

We know that it is a variable initialisation statement, in which `num` is a variable that is assigned with the value 25. Naturally, this statement causes memory allocation as shown in Figure 1.5.

In the figure, we can see that a variable has three attributes - its name, address and data type. Here, the name of the variable is `num` and the content 25 shows the data type. What about the address? Is it 1001, 1002, 1003 or 1004? It is 1001. Variable `num`

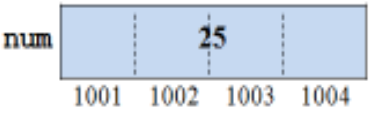


Fig.1.5: Memory allocation

being `int` type, 4 bytes (in GCC) are allocated. We know that each cell in RAM is of one byte size and each cell is identified by its unique address. But, when more than one cell constitute a single storage location (known as *memory word*), the address of the first cell will be the address of that storage location. That is how 1001 becomes the address of `num`. In class XI, we learnt that a variable is associated with two values: L-value and R-value, where ***L-value*** is the address of the variable and ***R-value*** is its content. Figure 1.5 shows that L-value of `num` is 1001 and R-value is 25.

Suppose we want to store the L-value (address) of a variable in another memory location. A variable is needed for this and it is known as pointer variable. Thus we can define ***pointer*** as a variable that can hold the address of a memory location. Pointer is primitive since it contains memory address which is atomic in nature. So we will say that pointer is a variable that points to a memory location (or data).

Harold Lawson (born 1937), a software engineer, computer architect and systems engineer is credited with the 1964 invention of the pointer. In 2000, Lawson was presented the Computer Pioneer Award by the IEEE for his invention.

As you know, computers use their memory for storing the instructions of a program, as well as the values of the variables that are associated with it. The memory is a sequential collection of 'storage cells' as shown in Figure 1.6. Each cell, commonly known as a byte, has a number called address associated with it. Typically, the addresses are numbered consecutively, starting from 0 (zero). The address of the last cell depends on the memory size. A computer memory having 64 K (64 x 1024 = 65536 Bytes) memory will have its last address as 65,535.

Fig.1.6: Memory organisation

Whenever we declare a variable in a program, a location is allocated somewhere in the memory to hold the R-value of the variable. Since every byte has a unique number as its address, this location will have its own address. Nowadays, the size of RAM is in terms of GBs and the address of memory location is expressed in hexadecimal number. It is because hexadecimal system can express larger values with lesser number of digits compared to decimal system.

1.2.1 Declaration of pointer variable

Pointer is a derived data type and hence a variable of pointer type is to be declared prior to its use in the program. The following syntax is used to declare pointer variable:

```
data_type * variable;
```

The `data_type` can be fundamental or user-defined and variable is an identifier. Note that an asterisk (*****) is used in between the data type and the variable. The following are examples of pointer declaration:

```
int *ptr1;
float *ptr2;
struct student *ptr3;
```

As usual memory will be allocated for these pointers. Do you think that the amount of memory for these variables is dependent on the data types used? We know that memory addresses are unsigned integer numbers. But it does not mean that pointers are always declared using `unsigned int`. Then, what is the criterion for determining the data type for a pointer? The data type of a pointer should be the same as that of the data pointed to by it. In the above examples, `ptr1` can contain the address of an integer location, `ptr2` can point to a location containing floating point number, and `ptr3` can hold the address of location whose R-value is `student` type data. So what will be the size of a pointer variable? The memory space for a pointer depends upon the addressing scheme of the computer. Usually, the size of a pointer in C++ is 2 to 4 bytes. As far as a programmer is concerned, there is no need to bother about the size of pointer while solving problems.

1.2.2 The operators & and *

Once a pointer is declared, memory address of a location of the same data type can be stored in it. When a variable is referenced in a C++ statement, actually its R-value is referred to. How can we retrieve its address (L-value)? C++ provides an operator named ***address of operator*** (**&**), to get the address of a variable. If `num` is an integer variable, its address can be stored in pointer `ptr1` by the following statement:

```
ptr1 = &num;
```

The statement, on execution, establishes a link between two memory locations as shown in Figure 1.7.

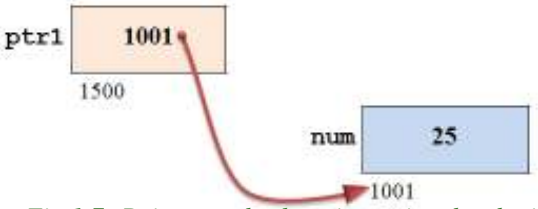


Fig.1.7: Pointer and a location pointed to by it

We have discussed that pointer is a kind of reference. Since a pointer references

a data stored somewhere in the memory, by dereferencing the pointer we get the data. C++ provides this dereferencing facility by an operator named ***indirection*** or ***dereference operator*** (*). The following statement retrieves the value pointed to by the pointer `ptr1` and displays on the screen.

```
cout << *ptr1;
```

It is clear that this statement is equivalent to the statement: `cout << num;`

Since the operator * retrieves the value at the location pointed to by the pointer, the * operator is also known as ***value at operator***.

Note that the operators address of (&) and indirection (*) are unary operators. The & operator can be used with any kind of variable since every variable is associated with a memory address. But, the * operator can be used only with pointers.

Considering the variables used in Figure 1.7, the following statements illustrate the operations performed by these operators:

```
cout<< &num; // 1001 (address of num) will be the output
cout<< ptr1; // 1001 (content of ptr1) will be the output
cout<< num;  // 25 (content of num) will be the output
cout<< *ptr1;/* 25 (value in the location pointed to by
                    ptr1) will be the output */
cout<< &ptr1;// 1500 (address of ptr1) will be the output
cout<< *num; // Error!! num is not a pointer
```

The last statement is invalid. An error will be reported during compilation, because `num` is not a pointer and the content 25 is not a memory address. The indirection operator (*) should be used only with pointers.

1.3 Methods of memory allocation

We know that variable declaration statements initiate memory allocation. The required memory is allocated when the program is loaded in RAM. The execution of the program begins only after this memory allocation. The amount of memory allocated depends upon the number and data type of variables used in the program. This amount is static, i.e., it will not increase or decrease during the program run. The memory allocation that takes place before the execution of the program is known as ***static memory allocation***. It is due to the variable declaration statements in the program. There is another kind of memory allocation, called ***dynamic memory allocation***. In this case, memory is allocated during the execution of the program. It is facilitated by an operator, named **`new`**. As complementary to this operator, C++ provides another operator, named **`delete`** to de-allocate the memory.

1.3.1 Dynamic operators - new and delete

The operator `new` is a keyword in C++ and it triggers the allocation of memory during run-time (execution). It is a unary operator and the required operand is either a fundamental or user-defined data type. Dynamic memory allocation being an operation, the operator `new` and the operand data type constitute an expression. Naturally it returns a value and this value will be the address of a location. The size of this location will be the same as that of the data type used as the operand. The following syntax is used for dynamic memory allocation:

```
pointer_variable = new data_type;
```

Note that a pointer variable is used to hold the address returned by the `new` operator. So, it should be declared earlier with the same data type specified after `new` operator. The following are examples for dynamic memory allocation:

```
short * si_ptr;
float * fl_ptr;
struct complex * cx_ptr;
si_ptr = new short;
fl_ptr = new float;
cx_ptr = new complex;
```

The memory allocations are shown in Figure 1.8.

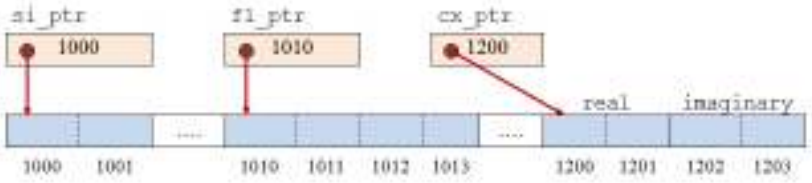


Fig. 1.8: Layout of dynamic memory allocation

Figure 1.8 shows that 2 bytes of location for `short` type data is allocated at the address 1000 and it is stored in `si_ptr`. Similarly 4 bytes from the address 1010 for `float` type data is allocated and this address is stored in `fl_ptr`. Earlier we discussed a structure named `complex` that consists of two `short` type elements. The pointer `cx_ptr` holds the address 1200 that is allocated for a `complex` type data of size 4 bytes (2 bytes each for `short real` and `short imaginary`). Note that, the dynamically allocated memory locations cannot be referred to by ordinary variables. Rather these are accessed using indirection (dereferencing) operator only as shown in the following examples:

```
*si_ptr = 247;
cin >> *fl_ptr;
```

We have a structure pointer `cx_ptr`, but the data pointed to by this pointer cannot be accessed in this format. We will discuss the accessing method later in this chapter.

As in the case of variable initialisation during static memory allocation, dynamically allocated memory locations can also be initialised using the following syntax:

```
pointer_variable = new data_type(value);
```

The following examples show initialisation along with dynamic memory allocation:

```
si_ptr = new short(0);
fl_ptr = new float(3.14);
```

In the case of `cx_ptr`, this kind of initialisation is not possible.

Once memory is allocated dynamically using `new` operator, it should be de-allocated or released before exiting the program. C++ provides `delete` operator for this purpose. In the case of static memory allocation, operating system itself allocates and releases memory depending on the scope and life of the variables. But in the case of dynamic memory allocation, the program should have an explicit statement to release (or free) the memory. For that, `delete` operator is used with the following syntax:

```
delete pointer_variable;
```

The following are valid examples:

```
delete si_ptr;
delete fl_ptr, cx_ptr;
```

1.3.2 Memory leak

If the memory allocated using `new` operator is not freed using `delete`, that memory is said to be an orphaned memory block - a block of memory that is left unused, but not released for further allocation. This memory block is allocated on each execution of the program and the size of the orphaned block is increased. Thus a part of the memory seems to disappear on every run of the program, and eventually the amount of memory consumed has an unfavorable effect. This situation is known as ***memory leak***.

The following are the reasons for memory leak:

- Forgetting to delete the memory that has been allocated dynamically (using `new`).
- Failing to execute the `delete` statement due to poor logic of the program code.
- Assigning the address returned by `new` operator to a pointer that was already pointing to an allocated object.

Remedy for memory leak is to ensure that the memory allocated through `new` is properly de-allocated through `delete`. Memory leak takes place only in the case of dynamic memory allocation. But in case of static memory allocation, the Operating System takes the responsibility of allocation and deallocation without user's instruction. So there is no chance of memory leak in static memory allocation

Let us do

Now let us compare static memory allocation and dynamic memory allocation. Table 1.4 may be used for comparison. Some of the entries are left for you to complete using proper points.

Static memory allocation	Dynamic memory allocation
i. Takes place before the execution of the program.	
ii.	`new` operator is required
iii.	Pointer is essential
iv. Data is referenced using variables	
v. No statement is needed for de-allocation	

Table 1.4: Static Vs Dynamic memory allocation

Know your progress

1. What is pointer?
2. What is the criterion for determining the data type of a pointer?
3. If `mks` is an integer variable, write C++ statements to store its address in a pointer.
4. If `ptr` is an integer pointer, write C++ statement to allocate memory for an integer number and initialise it with 12.
5. Consider the statements: `int *p, a=5; p=&a; cout<<*p+a;` What is the output?

1.4 Operations on pointers

We have discussed that indirection (*) and address of (&) operators can be used with pointers. In Class XI, we used arithmetic, relational and logical operators. In this section, we will have a look at the operators that can be used with pointers and how these operations are performed.

1.4.1 Arithmetic operations on pointers

We have seen that memory address is numeric in nature. Hence some of the arithmetic operations can be performed on pointers. Let us consider the pointers `si_ptr` and `fl_ptr` declared in section 1.3.1 (Refer Figure 1.8). Now, observe the following statements:

```
cout << si_ptr + 1;
cout << fl_ptr + 1;
```

What will be the output? Do you think that it will be 1001 and 1011?

Adding 1 to a pointer is not the same as adding 1 to an `int` or `float` type data. When we add 1 to a `short int` pointer, the expression returns the address of the next location of `short int` type. The cells with addresses 1000 and 1001 constitute a single storage location for an integer data of `short` type. Hence the address of the next addressable short integer location is 1002. So when 1 is added to a `short int` type pointer, actually its size (i.e., 2) is to be added to the address contained in the pointer variable. Similarly, to add 1 to `float` type pointer, its size (i.e., 4) is to be added to the address. So the expression `fl_ptr+1` returns 1014. So, it is clear that the expression `si_ptr+5` returns 1010 (1000+5×2) and `fl_ptr+3` returns 1022 (1010+3×4). Similarly, subtraction operation can also be performed on pointers.

Let us do

Find the values returned by the following arithmetic expressions:

```
si_ptr + 10        fl_ptr + 7
si_ptr - 5         fl_ptr - 10
```

Note that this kind of operation is practically wrong. Because we are trying to access locations that are not allocated for authorised use. These locations might have been used by some other variables. Sometimes these locations might not have been accessible due to the violation of access rights.

No other arithmetic operations are performed on pointers. So we can conclude that pointers are only incremented or decremented. The following statements illustrate various operations on pointers:

```
int *ptr1, *ptr2; // Declaration of two integer pointers
ptr1 = new int(5); /*  Dynamic memory allocation (let the
             address be 1000)and initialisation with 5*/
ptr2 = ptr1 + 1;  /*   ptr2 will point to the very next
             integer location with the address 1004 */
++ptr2;           //   Same as ptr2 = ptr2 + 1
```

```
    cout<< ptr1;        //    Displays 1000
    cout<< *ptr1;       //    Displays 5
    cout<< ptr2;        //    Displays 1004
    cin>> *ptr2;        /*    Reads an integer (say 12) and
                        stores it in location 1004 */
    cout<< *ptr1 + 1; //    Displays 6 (5 + 1)
    cout<< *(ptr1 + 2);//  Displays 12, the value at 1004
    ptr1--;             //    Same as ptr1 = ptr1 - 1
```

Let us write a program to demonstrate the operations on pointers. Program 1.3 gives the average height of a group of students.

Program 1.3: To find the average height of students

```
#include <iostream>
using namespace std;
int main()
{
    int *ht_ptr, n, s=0;
    float avg_ht;
    ht_ptr = new int;       //dynamic memory allocation
    cout<<"Enter the number of students: ";
    cin>>n;
    for (int i=0; i<n; i++)
    {
        cout<<"Enter the height of student "<<i+1<<" - ";
        cin>>*(ht_ptr+i);//to get the address of the next location
        s = s + *(ht_ptr+i);
    }
    avg_ht = (float)s/n;
    cout<<"Average height of students in the class = "<<avg_ht;
    return 0;
}
```

In program 1.3, an integer location is dynamically allocated and the address is stored in the pointer `ht_ptr`. When the body of the loop is executed for the first time, 0 is added to this address and it does not make any change. The input data is stored in this location. During the second execution of the loop-body, 1 is added to this address and the next integer location is referenced for the input. This process is continued for entering the heights of n students. Sum of these heights is calculated along with the input and after the completion of the loop, average is calculated.

Here explicit type conversion is used to get the accurate result. A sample output is shown below:

```
Enter the number of students: 5
Enter the height of student 1 - 170
Enter the height of student 2 - 169
Enter the height of student 3 - 175
Enter the height of student 4 - 165
Enter the height of student 5 - 177
Average height of students in the class = 171.199997
```

Program 1.3 also shows that a collection of the same type of data can be handled by utilising pointer arithmetic. Last year, we used arrays in such a situation. But the size should be specified during the array declaration. This may cause wastage or insufficiency of memory space. Pointer and its arithmetic overcome this drawback.

But there is a problem in this kind of memory usage. It is not sure that Program 1.3 will always run with any value of `n`. GCC may not give any output for the `avg_ht`. Though there is no problem theoretically, unexpected results may occur during execution. As mentioned earlier, pointer `ht_ptr` is initialised with the address of only one location. The memory locations accessed using pointer arithmetic on `ht_ptr` are unauthorised, since these locations are not allocated by the OS. This may lead to unexpected termination of the program or loss of some data that already reside in those locations. We can overcome these issues by the facility of dynamic arrays, which we discuss in Section 1.5 of this chapter.

1.4.2 Relational operations on pointers

Among the six relational operators, only == (equality) and != (non-equality) operators are used with pointers. Memory address is simply a unique number to identify each memory location. If `p` and `q` are two pointers, they may contain the address of the same integer location or different memory locations. This can be verified with the expressions `p==q` or `p!=q`.

Know your progress

1. Dynamic memory allocation operator in C++ is _____.
2. What happens when the following statement is executed?
   ```
   int *p = new int(5);
   ```
3. What is orphaned memory block?
4. If p is an integer pointer, which of the following are invalid?
   a. `cout<<&p;` b. `p=p*5;` c. `p>0`
   d. `p++;` e. `p=1500;` f. `cout<<*p * 2;`

1.5 Pointer and array

We learnt that an array can contain a collection of homogeneous type of data under a common name. This data is stored in contiguous memory locations. Figure 1.9 shows the memory allocation of an array `ar[10]` of `int` type with 10 numbers.

It is assumed that the array begins at location 1000 and each location consists of 4 bytes (as per GCC). We know that any element of this array can be referenced by specifying the subscript along with the array name. For example, `ar[0]` returns 34, `ar[1]` returns 12, and at last `ar[9]` returns 19.

	ar
1000	34
1004	12
1008	8
1012	18
1016	24
1020	38
1024	43
1028	14
1032	7
1036	19

Fig. 1.9: Memory allocation for array `ar`

Let us do

Write C++ statement to display all the 10 elements of this array.

How can we store the address of the first location of this array into a pointer?

If `ptr` is an integer pointer, the address of the first location of array `ar[10]` can be stored in it with the following statement:

```
ptr = &ar[0];
```

Now let us see the output of the expressions used in the following statements:

```
cout<<ptr;     //Displays 1000, the address of ar[0]
cout<<*ptr;    //Displays 34, the value of ar[0]
cout<<(ptr+1);  //Displays 1004, the address of ar[1]
cout<<*(ptr+1); //Displays 12, the value of ar[1]
cout<<(ptr+9);  //Displays 1036, the address of ar[9]
cout<<*(ptr+9); //Displays 19, the value of ar[9]
```

Can you predict the output of the statement: `cout<<ar;`?

The output will be 1000, which is the address of the first location of the array. This address is known as base address of the array. We have seen that a variable that contains the address of a memory location is called pointer. In that sense, array-name `ar` can be considered as a pointer. So the following statements are also valid:

```
cout<<ar;      //Displays 1000, the address of ar[0]
ptr=ar;        //same as ptr=&ar[0];
cout<<*ar;     //Displays 34, and is same as cout<<ar[0];
cout<<(ar+1); //Displays 1004, the address of ar[1];
cout<<*(ar+1);  //Displays 34, and is same as cout<<ar[1];
```

The following C++ statement displays all the elements of this array:

```
for (int i=0; i<10; i++)
    cout<<*(ar+i)<<'\t';
```

There is a difference between an ordinary pointer and an array-name. The statement `ptr++;` is valid and is equivalent to `ptr=ptr+1;`. After the execution of this statement `ptr` will point to the location of `ar[1]`. That is, `ptr` will contain the address of `ar[1]`. But the statement `ar++;` is invalid, because array-name always contains the base address of the array, and it cannot be changed.

Dynamic array

In C++, array helps to handle a collection of same type of data. But, if the number of data items is not known in advance, there is a problem in declaring the array. As we know, size of array is to be specified in the declaration statement, and it should be an integer constant. How can we declare an array to store the percent of pass obtained by the schools in any district in the Higher Secondary examination? Neither `float pass[n];` nor `float pass[];` is valid. We have to mention an integer constant as size of the array and it may cause insufficiency or wastage of memory space. The district, and hence the number of schools are unknown while writing the program. So, the program should provide the facility to allocate the required locations as per the user's input. The solution in such a situation is dynamic array.

Dynamic array is created during run time using the dynamic memory allocation operator `new`. The syntax is:

```
pointer = new data_type[size];
```

Here, the `size` can be a constant, a variable or an integer expression. Program 1.4 illustrates the concept of dynamic array. It can store the percent of pass secured by the schools. The number of schools will be decided by the user only at the time of execution of the program.

Program 1.4: To find the highest percent of pass in schools

```
#include <iostream>
using namespace std;
int main()
{
    float *pass, max;
    int i, n;
    cout<<"Enter the number of schools: ";
    cin>>n;    //To input number of schools
    pass = new float[n];//dynamic array having n elements
```

```
    for (i=0; i<n; i++)
    {
        cout<<"Percent of pass by school "<<i+1<<": ";
        cin>>pass[i]; //Concept of subscripted variable
    }
    max=pass[0];
    for (i=1; i<n; i++)
        if (pass[i]>max) max = *(pass+i);
    /* Elements are accessed using subscript and pointer
    arithmetic operation */
    cout<<"Highest percent is "<<max;
    return 0;
}
```

Output:

```
Enter the number of schools: 5
Percent of pass by school 1: 75.6
Percent of pass by school 2: 66.5
Percent of pass by school 3: 89.3
Percent of pass by school 4: 71
Percent of pass by school 5: 70.6
Highest percent is 89.3
```

Program 1.4 uses dynamic array to store the data. Memory is allocated only during execution and five locations, each with 4 bytes, are reserved for the array `pass`. Elements of this array are accessed using subscript as well as pointer arithmetic operation.

Let us do

Read the following statements and write the difference between them:

```
int *ptr = new int(10);
int *ptr = new int[10];
```

1.6 Pointer and string

In Class XI, we learnt that string data can be referenced by character array and the array-name can be considered as string variable. In the previous section, we saw that array-name contains the base address of the array, and hence it can be considered as a pointer. Let us discuss how these two aspects are combined to refer to strings using pointer. The following statements illustrate how character pointer differs from the other pointers:

```
char str[20];      //character array declaration
char *sp;          //character pointer declaration
cin>>str;          //To input a string, say "Program"
cout<<str;         //Displays the string "Program"
sp=str;   //Content of str is copied into the pointer sp
cout<<sp; //Displays the string "Program"
cout<<&str[0];     //Displays the string "Program"
cout<<sp+1;        //Displays the string "rogram"
cout<<&str+1;      //Displays the string "rogram"
/* The two statements given above display the substring
starting from 2nd character onwards */
cout<<str[0];      //Displays the character 'P'
cout<<*sp;         //Displays the character 'P'
cout<<&str; //Displays the base address of the array str
cout<<&sp;  //Displays the address of the pointer sp
```

A string contained in an array cannot be copied into another character array using assignment operator (=) (*we used* `strcpy()` *function last year*). But, the assignment is possible with character pointers. The statements `sp=str;` and `cout<<sp;` show this fact. It proves that a character pointer can be used to store a string and this pointer can be considered as a string variable. That is, as we use character array name to refer to string data, character pointer can also serve the same.

Another interesting aspect is that, the statement `cout<<&str[0];` also displays the entire string, instead of the address of the first location (base address). That means, if we access the address of a string data, we get the string itself. But `str[0]` and `*sp` gives the first character of the string.

Advantages of character pointer

The use of character pointer for storing string offers the following advantages over character array:

- Since there is no size specification, a string of any number of characters can be stored. There is no wastage or insufficiency of memory space. But it should be done with initialization. (e.g., `char *str = "Program";`)
- Assignment operator (=) can be used to copy strings.
- Any character in the string can be referenced using the concept of pointer arithmetic which makes access faster.
- Array of strings can be managed with optimal use of memory space.

Array of strings

Suppose, we want to store the names of days in a week. A character array or character pointer can be used to store only one name at a time. Here we need to refer to a collection of strings ("Sunday", "Monday", ..., "Saturday"). Obviously we should use an array of character arrays (2D array of `char` type) or an array of character pointers. The following statement declares an array of character pointers to handle this case:

```
char  *name[7];
```

This array can contain a maximum of 7 strings, where each string can contain any number of characters. But we should make sure that the pointer array is initilaised. It may be as follows:

```
char *week[7]={"Sunday", "Monday", "Tuesday", "Wednesday",
               "Thursday", "Friday", "Saturday"};
```

Figure 1.10 shows the optimal use of memory locations. Only the shaded portion will be allocated.

Let us do

Write C++ statements to sort these names using any of the sorting techniques we discussed in Class XI. Since we use character pointer, strings can be copied using assignment operator. Check the correctness of your code during your lab work.

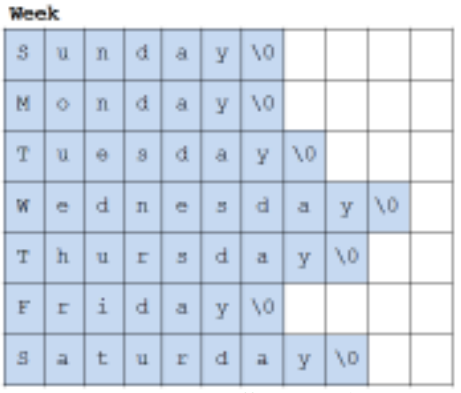


Fig. 1.10: Memory allocation for strings

The following statement illustrates the accessing of these strings:

```
for (i=0; i<7; i++)
    cout<<name[i];
```

An array of strings can be handled using a 2D character array as given below:

```
char name[10][20];
```

This array can contain 10 names, each of which can have a maximum of 19 characters. One byte is reserved for null character (`'\0'`). Each string is referred to by the expression `name[i]`, where the subscript `i` can take values from 0 to 9. In this case `strcpy()` function should be used for copying the strings into variables.

Know your progress

1. What is dynamic array?
2. Address of the first location of an array is known as _____.
3. If `arr` is an integer array, which of the following are invalid?
   a. `cout<<arr;` b. `arr++;` c. `cout<<*(arr+1);`
   d. `cin>>arr;` e. `arr=1500;` f. `cout<<*arr * 2;`
4. Write a declaration statement in C++ to refer to the names of 10 books using pointers.
5. Write a statement to declare a pointer and initialise it with your name.

1.7 Pointer and structure

Earlier in this chapter, we discussed structure data type and its applications. This section discusses how structures are accessed by pointers. A structure is defined to represent the details of employees as follows:

```
struct employee
{
    int ecode;
    char ename[15];
    float salary;
};
```

Now, observe the following declaration statement:

```
employee *eptr;
```

It is clear that `eptr` is a pointer that can hold the address of `employee` type data. The statement:

```
eptr = new employee;
```

allocates 23 bytes of memory and its address is stored in the pointer `eptr`. Figure 1.11 illustrates the effect of this statement.

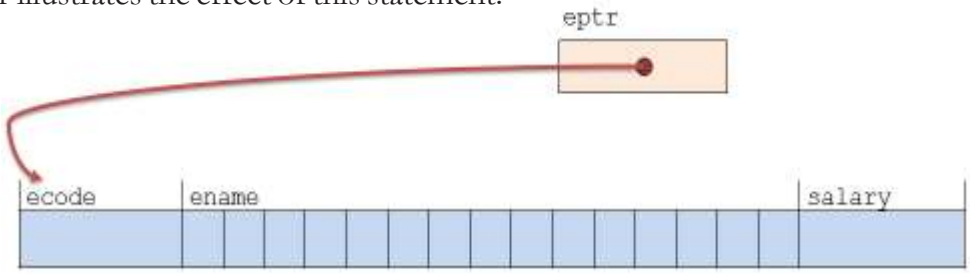


Fig. 1.11: Dynamic memory allocation for `employee` *type data*

We learnt that a structure is accessed in terms of its elements with the following format:

```
structure_variable.element_name
```

Here, we do not have a structure variable to access the elements `ecode`, `ename` and `salary`. So we have to use the pointer `eptr`. The syntax for accessing the elements of a structure is as follows:

```
structure_pointer->element_name
```

Note that structure pointer and an element is connected using arrow operator (**->**). It is constituted by a hyphen (**-**) followed by greater than symbol (**>**). The following statements are examples for accessing the elements of the structure shown in Figure1.11.

```
eptr->ecode = 657346;   //Assigns an employee code
gets(eptr->ename);      //inputs the name of an employee
cin>> eptr->salaray;    //inputs the salary of an employee
cout<< eptr->salary * 0.12; //Displays 12% of the salary
```

Let us do

Earlier in this chapter, in Section 1.3.1, we mentioned a pointer `cx_ptr` of `complex` type structure. Write C++ statements to input a complex number and display in its actual format.

Let us modify the structure `employee` by adding an element as follows:

```
struct employee
{
    int ecode;
    char ename[15];
    float salary;
    int *ip;
};
```

Obviously, the element `ip` can contain the address of an integer location. The following statements illustrate the use of pointer `ip`:

```
eptr->ip = new int(5); /* Dynamic allocation for integer
                          and initialiastion with 5 */
cout << *(eptr->ip);   // Displaying the value 5
int n = eptr->*ip+1;   // Adding 1 to 5 and stores it in n
```

Observe that the value pointed to by `ip` can be referenced in two ways: `*(eptr->ip)` and `eptr->*ip`. A structure can contain pointer of any data type as its element. Even it may be of the same structure data type as follows:

```
struct employee
{
    int ecode;
    char ename[15];
    float salary;
    employee *ep;
};
```

The element ep is a pointer of employee data type

Now, the structure employee is known as self referential structure. Let us discuss more on this type of structures and their applications.

Self referential structure

Self referential structure is a structure in which one of the elements is a pointer to the same structure. A location of this type contains data and the address of another location of the same type. This location can again contain data and address of yet another location of the same structure type. It can be extended as per the requirement. Figure 1.12 shows this concept.

Fig. 1.12: An employee of structure type points to another employee

An employee named "Sunil" points to the next employee whose table number is 12. The employee at table number 12 is "Anil" and he points to the next employee "Nisha" and so on.

Self referential structure is a powerful tool of C and C++ languages that helps to develop dynamic data structures like linked list, tree, etc. Dynamic data structure means a collection of data for which memory will be allocated during run-time. The memory locations are scattered, but there will be a link from one location to another. More about the data structure linked list will be discussed in Chapter 3.

Let us conclude

We have discussed more advanced data types in C++ in this chapter. Structure data type is introduced to represent grouped or aggregate data under single name. Accessing of elements with dot operator (.) is discussed. Pointer is presented as a special type of data. Operations associated with pointers are illustrated with the help of expressions. The concept of dynamic memory allocation and the required

operators, and its advantages are discussed. The relationship between array and pointer is illustrated, and string data are handled using pointer. A good understanding of the concepts dealt with in this chapter will help you to attain the learning outcomes specified in Chapter 3 and equip you for higher studies.

Let us practice

1. Define a structure to represent the details of telephone subscribers which include name of the subscriber and telephone number. Write a menu driven program to store the details of some subscribers with options for searching the name for a given number, and the number for a given name.
2. Define a structure to represent the details of customers in a bank. The details include account number, name, date of opening the account and balance amount. Write a menu driven program to input the details of a customer and provide options to deposit, withdraw and view the details. During deposit and withdrawal, proper update is to be made in the balance amount. A minimum balance of Rs. 1000/- is a must in the account.
3. Write a program to input the TE scores obtained by a group of students in Computer Science and display them in the descending order using pointers.
4. Write a program to input a string and check whether it is palindrome or not using character pointer.
5. Write a program to input the names of students in a class using pointers and create a roll list in which the names are listed in alphabetical order with roll number starting from 1.
6. Define a structure student with the details register number, name and CE marks of six subjects. Using a structure pointer, input the details of a student and display register number, name and total CE score.

Let us assess

1. Compare array and structure in C++.
2. Identify the errors in the following structure definition and write the reason for each:

```
struct
{
    int roll, age;
    float fee=1000;
};
```

3. Read the following structure definition and answer the following questions:

```
struct Book
{
    int book_no;
    char bk_name[20];
    struct
    {
        short dd;
        short mm;
        short yy;
    }dt_of_purchase;
    float price;
};
```

   a. Write a C++ statement to declare a variable to refer to the details of a book. What is the memory requirement of this variable? Justify your answer.

   b. Write a C++ statement to initialise this variable with the details of your Computer Science text book.

   c. Write C++ statement(s) to display the details of the book.

   d. The missing of structure tag in the inner structure does not cause any error. State whether this is true or false. Give reason.

4. "Structure is a user-defined data type". Justify this statement with the help of an example.

5. Read the following statements:

   i. While defining a structure in C++, tag may be omitted.

   ii. The data contained in a structure variable can be copied into another variable only if both of them are declared using the same structure tag.

   iii. Elements of a structure is referenced by structure_name.element

   iv. A structure can contain another structure.

   Now, choose the correct option from the following:

   a. Statements (i) and (ii) are true
   b. Statements (ii) and (iv) are true
   c. Statements (i), (ii) and (iv) are true
   d. Statements (i) and (iii) are true

6. Read the following C++ statements:

```
int * p, a=5;
p=&a;
```

   a. What is the speciality of the variable p?

   b. What will be the content of p after the execution of the second statement?

   c. How do the expressions `*p+1` and `*(p+1)` differ?

7. Identify the errors in the following C++ code segment and give the reason for each.

```
int *p,*q, a=5;
float b=2;
p=&a;
q=&b;
cout<<p<<*p<<*a;
if (p<q)cout<<p;
    cout<<*p * a;
```

8. While writing a program, the concept of dynamic memory allocation is applied. But the program does not contain a statement with `delete` operator and it creates a problem. Explain the problem.

9. Read the C++ statements given below and answer the following questions:

```
int ar[] = {34, 12, 25, 56, 38};
int *p = ar;
```

   a. What will be the content of p?

   b. What is the output of the expression: `*p + *(ar+2)`?

   c. The statement `ar++;` is invalid. Why? How does it differ from `p++;`?

10. Explain the working of the following code segment and predict the output:

```
char *str = "Tobacco Kills";
for (int i=0; str[i]!='\0'; i++)
    if (i>8)
        *(str+i) = toupper(*(str+i));
cout<<str;
```

11. Observe the following C++ statements:

```
int ar[] = {14, 29, 32, 63, 30};
```

One of following expressions cannot be used to access the element 32. Which is that?

a. `ar[2]` b. `ar[*ar%3]` c. `*ar+2` d. `*(ar+2)`

12. Explain the operations performed by the operaors `new` and `delete` with the help of examples.

13. What is meant by memory leak? What are the reasons for it? How can we avoid such a situation?

14. Compare the following two statements.

```
int a=5;
int *a=new int(5);
```

15. Read the structure definition given below and answer the following questions:

```
struct sample
{
    int num;
    char *str;
}   *sptr;
```

a. Write C++ statements to dynamically allocate a location for `sample` type data and store its address in sptr.

b. Write C++ statements to input data into the location pointed to by `sptr`.

c. Modify this structure into a self referential structure.

2 Concepts of Object Oriented Programming

Significant Learning Outcomes

After the completion of this chapter, the learner

- compares various programming paradigms.
- lists the features of procedure-oriented paradigm.
- lists the advantages of object-oriented paradigm.
- explains the concepts of data abstraction and encapsulation, citing examples.
- explains inheritance and polymorphism with the help of real life examples.

We learn a programming language to develop programs which make the computer a more useful machine. The bigger the program is, the more difficult it becomes to manage it. To overcome this difficulty there are a lot of tools like IDE, debugger, compiler, etc. that we can use, and approaches like structured, procedural, modular, object oriented, etc. that we can follow in software development. Implementation of these tools and approaches helps us to address a number of issues faced in software development, like maintainability, reusability, portability, security, integrity, and the user-friendliness of software products.

In the previous chapter, we learnt various programming concepts and developed programs in C++ language for solving problems. Our aim has been to process the inputs and produce the output. As the programs developed were small and less complex, we never thought of giving importance to the approaches we adopted or to the security of data used or in organising the processing steps. But, knowingly or unknowingly, we were following some approach while writing programs to solve problems. This chapter discusses the

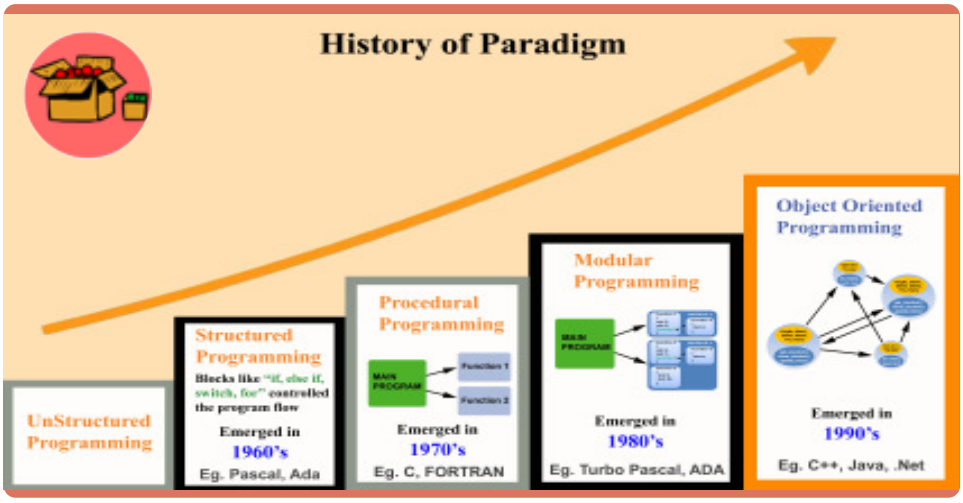
programming approach that we have been following so far and presents new approaches, so that we can choose a suitable approach for program development.

2.1 Programming paradigm

A programming paradigm denotes the way in which a program is organised. If the program is very small, there is no need to follow any organising principle. But, as it becomes larger, some measures have to be taken for managing it and for reducing its complexity.

While some paradigms give more importance to procedures, others give more importance to data. Capabilities and styles of various programming languages are defined by the programming paradigms supported by them. The various approaches that have been tried are modular programming, top-down programming, bottom-up programming and structured programming. Each one of these approaches were used to reduce the complexity of programming and to create reliable and maintainable programs.

Some programming languages are designed to follow only one paradigm, while others support multiple paradigms. C++ is a multiple paradigm language. Using C++, we can implement two of the most important programming paradigms, the procedural paradigm and the object-oriented paradigm. Let us discuss each of these in detail.



2.1.1 Procedure-Oriented Programming paradigm

Procedure-Oriented Programming specifies a series of well-structured steps and procedures to compose a program. It contains a systematic order of statements, functions and commands to complete a computational task or program. The statements may be to accept input, to do arithmetic or logical operations, to display

result, etc. In this approach emphasis is on doing things. In this paradigm, when the program becomes larger and complex, the list of instructions is divided and grouped into **functions**. A function clearly defines the purpose, and interface to other functions in the program, thus reducing the complexity. To further reduce the complexity, the functions associated with a common task are grouped into **modules**. Figure 2.1 shows a procedural paradigm in which a large program is divided into five separate functions, and functions associated with a common task grouped into two modules.

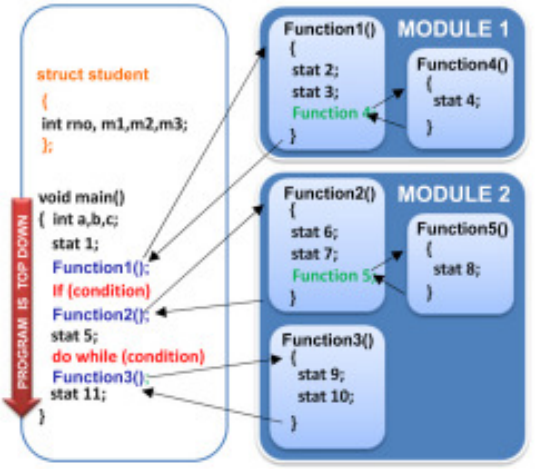


Fig. 2.1: Procedural paradigm

C, Pascal, FORTRAN, BASIC, etc. are procedural languages. In the C++ learning process, the organising principle we followed till now was the procedural paradigm. Procedural programming languages are also known as top-down languages.

But, when the program gets larger and more complex, the procedure-oriented programming approach is found to have several limitations. No matter how well this approach is implemented, large programs become excessively complex. The main reasons for the increasing complexity with procedural languages are:

a. Data is undervalued.
b. Adding new data element may require modifications to all/many functions.
c. Creating new data types is difficult.
d. Provides poor real world modelling.

Let us discuss each of them in detail.

a. Data is undervalued

In procedural language, emphasis is on doing things. Here data is given less importance. Let us explain this with the help of an example. Assume that we have to develop a software for automating the activities at our school. The activities may include admitting a new student, removing a student, recording fee collection details etc. Imagine we implemented the activities into the software using a function for each activity. The data that may be required by most of these functions for their functioning may be the student details stored in an array of structures. The easy way of making this data available to all these functions is to declare the student array as global (see section 10.5 of your Class XI textbook). Now the data is exposed and

any function other than the function that requires this data may also access it and make changes to the data knowingly or unknowingly as we cannot imply any restrictions.

This is like leaving your assignment to be submitted the next day, over the dining table. There are chances of this document getting destroyed, as a small child may tear it or draw pictures on it or a cup of tea placed on the table may spill over it accidentally. This can happen as the assignment is kept exposed in a place where anybody can access it.

The arrangement of local variables, global variables and functions in a procedural programming paradigm is shown in Figure 2.2. The green lines show which functions require a particular data and red lines show illegal access of data by functions that do not require it.

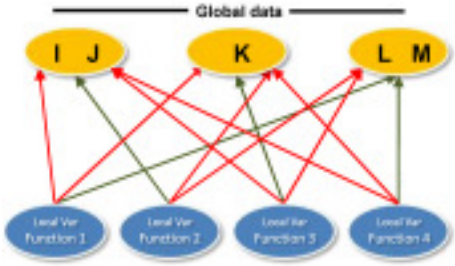


Fig. 2.2: Functions accessing data

b. Adding new data element may require modifications to all/many functions

Since many functions access global data, the way the data is stored is important. The arrangement of data cannot be changed without modifying all the functions that access it. If we add new data items, we will need to modify all the functions that access the data, so that they can also access the new items. It will be hard to find all such functions, and even harder to modify all of them correctly.

For example, assume that we need to add an item named '`age`' to the existing student structure in our school software. For the proper functioning of the software, we will have to find all functions that access the student data and incorporate proper changes.

c. Creating new data types is difficult

Computer languages typically have several built-in data types like integer, float, character and double. Certain programming language allows creation of data types other than the built-in data type, which means they are extensible. The ability of a program to allow significant extension of its capabilities, without major rewriting of code or changes in its basic architecture is called **extensibility.** This feature helps us in reducing the complexity of a program while extending its capabilities. Procedural languages are not extensible.

d. Provides poor real world modelling

In procedural programming paradigm, functions and data are not considered as a single unit and are independent of each other. So neither data nor functions in procedure oriented paradigm, by themselves, cannot model real-world objects effectively.

For example, in our school software, we have student data and functions that access it. Similarly, assume that the software also maintains teacher data and functions accessing this data. In procedural programming we may not be able to club student data and the functions accessing it or teacher data with the functions accessing it.

Every real-world object that we deal with, have both characteristics and behaviour bundled in a single unit. If we take the object 'humans' as an example, its characteristics can be name, gender, nationality, etc. and behaviour can be speaking, laughing, etc. Even though behaviour may be implemented as a function and characteristics may be represented as data in a program, we are not able to club data and functions. Therefore modelling things in the real world is difficult in procedural programming.

In the next section, we will discuss Object-Oriented Paradigm and see how it tries to eliminate the limitations of Procedure Oriented paradigm.

Know your progress

1. State whether the following three statements are true or false:
   a. Global variables cannot be accessed in more than one function.
   b. Procedural programming resembles closeness to real world.
   c. In procedural programming paradigm, data and functions are not bound together.
2. Pick the procedural languages from the following.
   C, C++, Fortran, JAVA, Pascal.

2.1.2 Object-Oriented Programming (OOP) paradigm

Object-oriented paradigm eliminates the problems in the procedural paradigm by clubbing data and functions that operate on that data into a single unit. In OOP, such a unit is called an **object**.

For example, by implementing OOP in our school software, we can create a `Student` object by clubbing student data and its functions as well as `Teacher` object by clubbing teacher data and its functions. Now the functions of one object will not be able to access the data of other object without permission.

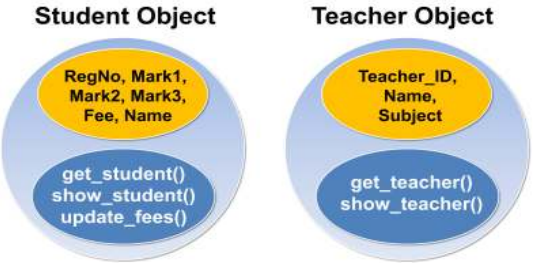


Fig. 2.3: Objects containing data and functions

Advantages of using OOP are:

a. OOP provides a clear modular structure for programs.

b. It is good for defining abstract data types.

c. Implementation details are hidden from other modules and have a clearly defined interface.

d. It is easy to maintain and modify the existing code as new objects can be created without disturbing the existing ones.

e. It can be used to implement real life scenarios.

f. It can define new data types as well as new operations for operators.

2.2 Basic concepts of OOP

Object Oriented Programming simplifies the software development and maintenance by providing some concepts such as objects, classes, data abstraction, data encapsulation, modularity, inheritance, polymorphism. Let us discuss these concepts in detail.

2.2.1 Objects

Anything that we see around us can be treated as an object and all these objects have properties (also called member/data/state) and behaviour (also called methods/member functions). Some examples of objects are listed in Figure 2.4 with their properties and methods.

Fig. 2.4: Real world objects with their properties (state) and methods (behavior)

Let us do

Observe some real world objects around you, identify properties each of them posseses and the behaviours eacht exibits. Write your findings in the following table:

Object Name	Properties	Behaviour

When OOP is to be implemented to solve a programming problem, instead of dividing the problem into functions we will have to think about dividing it into objects. When thinking in terms of objects rather than functions, program designing becomes easier, as there is a close match between objects in the program and objects in the real world.

In OOP an **object** is obtained by combining data and functions acting upon the data into a single unit. After combining, the functions inside an object are called **member functions** and data is called **member** (see Figure 2.6).

2.2.2 Classes

An object is defined via its class which determines everything about an object. A **class** is a prototype/blue print that defines the specification common to all objects of a particular type. This specification contains the details of the data and functions that act upon the data. Objects of the class are called individual **instances** of the class and any number of objects can be created based on a class (see Figure 2.5).

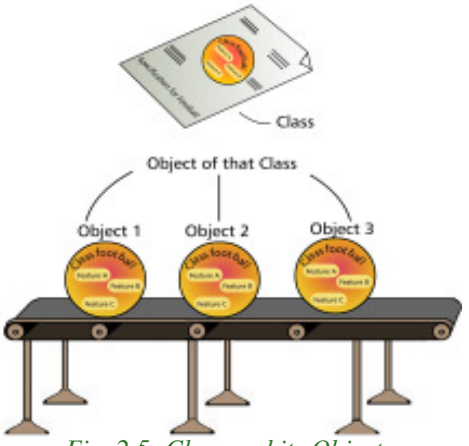


Fig. 2.5: Class and its Objects

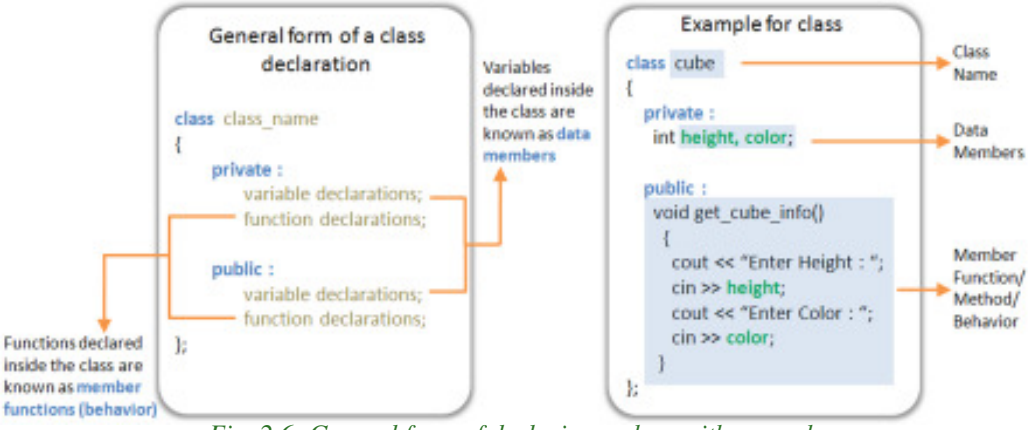


Fig. 2.6: General form of declaring a class with example

The declaration and usage of class is almost similar to that of a structure. A structure includes specifications regarding data, where as a class includes specification regarding both data and functions that use the data (See Figure 2.6). A structure is declared using the keyword '`struct`', where as class is declared using the keyword '`class`'.

If `Student` is the name of a C++ class, to create two objects named '`S1`' and '`S2`', (as in Figure 2.7) declaration will be as follows:

```
Student  S1,S2;
```

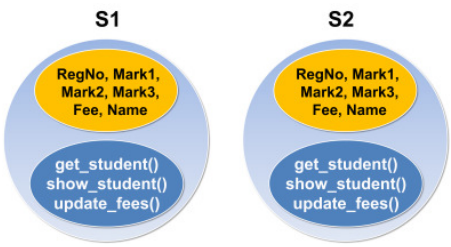


Fig. 2.7: Objects of Student class

Objects can be declared and created from a class using the statement.

```
Cube S;     Or     Cube *C;
                   C = new Cube;
```

where `Cube` is the name of the class and `C` is the name of the object.

Objects can communicate with each other by passing message, which is similar to people passing message with each other. This helps in building systems that simulate real life. In OOP, calling member function of an object from another object is called **passing message**. Message passing involves specifying the name of object, the name of the member function, and the information to be sent.

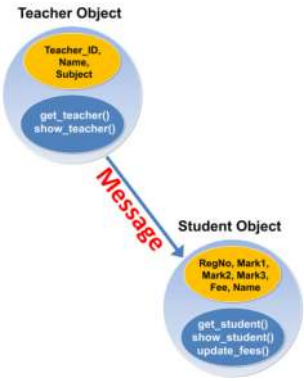


Fig. 2.8: Objects passing message

For example, in our school software, the `Teacher` object to update the fees of a student, will pass a message to `Student` object by calling the `update_fee()` member function (See Figure 2.8) like

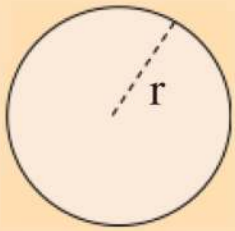
```
S1.update_fees("Rahul",  1000);
```

where `S1` is an object of `Student` object. The `Student` object on reciving the message will update the fees of the student with the data provided by the `Teacher` object.

Know your progress

1. OOP stands for _____.
2. A blueprint for an object in OOP is called a _____.
3. The functions associated with the class are called _____.
4. The variables declared inside the class are known as _____.
5. What is the difference between structure and class?

The following program implements a circle as an object. The class 'Circle' declares the only data required - the radius and the member functions for accepting the radius and displaying the area.




```
#include<iostream>
using namespace std;
class Circle
 {
   private:
      float r;
   public:
   void get_radius()
   {
     cout << "Enter Radius :";
     cin >> r;
   }
   void display_area()
   {
     cout<< "Area:"<< 3.14*r*r;
   }
   };
   int main()
   {
   Circle C1;
   C1.get_radius(); //Sending message
   C1.display_area(); //Sending message
   }
```

Member

Member functions

Class Declaration

Class

Object

Main program

Output :

Enter Radius : 2.0

Area: 12.56

2.2.3 Data Abstraction

Data abstraction refers to showing only the essential features of the application and hiding the details from outside world.

Let us take a real life example of a television which we can turn on and off, change the channel, adjust the volume, and add external components such as speakers, VCRs, and DVD players, but we do not know about its internal details, that is, we do not know how it receives signals over the air or through a cable, how it translates them, and finally displays them on the screen.

Thus, we can say a television clearly separates its internal implementation from its external interface and we can use its interfaces like the power button, channel selector, and volume control without having any knowledge about its internal features.

Like this, C++ classes provide great level of data abstraction. They provide public methods/functions to the outside world to use the functionality of an object and to manipulate object data. These methods helps to manipulate objects from outside without actually knowing how object has been implemented internally.

Data abstraction separates interface and implementation. **Implementation** denotes how variables are declared, how functions are coded, etc. It is through **interface** (function header/function signature) a communication is send to the object as messages.

For example, in our school software, a member function e.g. `show_student()` of `Student` object may be called without actually knowing what algorithm the function uses internally to display the given values. At any time we can change the implementation of `show_student()`, and the function call will still work as long as long there is no change in the interface.

Data abstraction provides two important advantages:

- Class internals are protected from accidental user-level errors, which might corrupt the state of the object.
- Any change in class implementation may be done over time in response to changing requirements, without changing the statements of the class..

2.2.4 Data Encapsulation

All C++ programs are composed of two fundamental elements, functions and data. **Encapsulation** is an OOP concept that binds together the data and functions that manipulate the data, and keeps both data and function safe from outside interference and misuse.

C++ implements encapsulation and data hiding through the declaration of a class. A C++ class can contain `private`, `protected` and `public` members (See Figure 2.6). By default, all items defined in a class are `private`. Members declared under `private` section are not visible outside the class. Members declared as

`protected` are visible to its derived class also (explained in Section 2.2.6) , but not outside the class.

In `Student` class of our school software, the variables `Regno, Name, Mark1, Mark2, Mark3` and `Fee` are to be declared `private`. This means that they can be accessed only by member functions of the `Student` class, and not by any other part of our program. Here data is hidden and encapsulation is achieved.

To make parts of a class accessible to other parts of our program, we must declare them under the `public` section. All variables or functions defined after the `public` access specifier are accessible anywhere in our program. For example, in `Student` class of our school software, all member functions in it that need to be called by other objects are to be declared `public`, in order to make them visible outside the class.

2.2.5 Modularity

When we write a program, we try to solve a problem by decomposing the problem into small sub-problems and then try to solve each sub-problem separately. Solution to each sub-problem is a separate component that includes interface, specification and implementation.

Modularity is a concept through which a program is partitioned into modules that can be considered and written on their own, with no consideration of any other module. These modules are later linked together to build the complete software. These modules communicate with each other by passing messages.

We have already studied in detail about implementing modularity using functions. (See Chapter 10, Functions in class XI text book). In object oriented-programming, modularity is implemented with the help of class. For example, in our school software we can separate everything related to students and teachers into two separate modules (See Figure 2.9) using the concept of class.

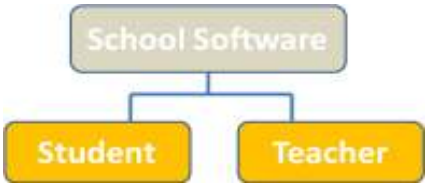


Fig. 2.9: Modularity

2.2.6 Inheritance

Inheritance is the process by which objects of one class acquire the properties and functionalities of another class. Inheritance supports the concept of hierarchical classification and reusability.

Let us take a real life example to explain the scenario. In Figure 2.10, `land vehicle` and `water vehicle` acquires the properties (i.e. data members and member functions) of vehicle. Again `car` and `truck` acquires the properties of `land`

vehicle (i.e. car/truck = vehicle + land vehicle) and boat acquires the properties of water vehicle (i.e. boat = vehicle + water vehicle). In the case of hovercraft which travels both in land and water, it acquires the properties of both land vehicle and water vehicle (i.e. hovercraft = vehicle + land vehicle + water vehicle). A car can have further classification such as hatchback, sedan, SUV etc., which will acquire the properties from car, land vehicle and vehicle, but will still have some specific properties of its own. Thus, the level of hierarchy can be extended to any level.

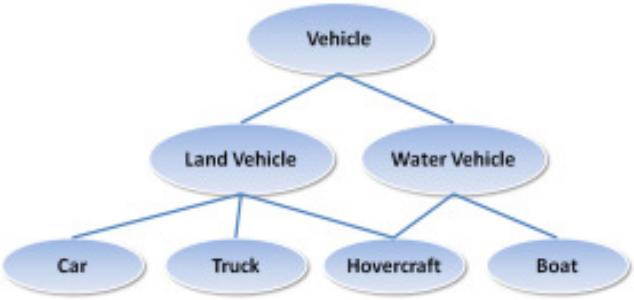


Fig. 2.10: Inheritance in real world

In the above example, if the common properties and functionalities of both land vehicle and water vehicle were not separated and placed in vehicle, then it would have to be repeated in both the classes. This would have increased the size of the program and the time taken for coding and debugging. Keeping it in mind, if we observe from top to bottom of the chart, we can understand how complexity is reduced greatly through the introduction of inheritance.

Once a class is written, created and debugged, if needed, it can be distributed for use in other programs. This is called **reusability**. In OOP, the concept of inheritance provides an important extension to the idea of reusability. Through this we can add additional features to an existing class without modifying it. This is made possible through deriving a new class from the existing one. The new class will inherit the capabilities of the old one, and can add features of its own. The existing class is called the **base class**, and the new class is referred to as the **derived class**. The derived class will have combined features of both the classes. Any number of classes can be derived from an existing class. Figure 2.11 shows the concept of derivation of new classes.

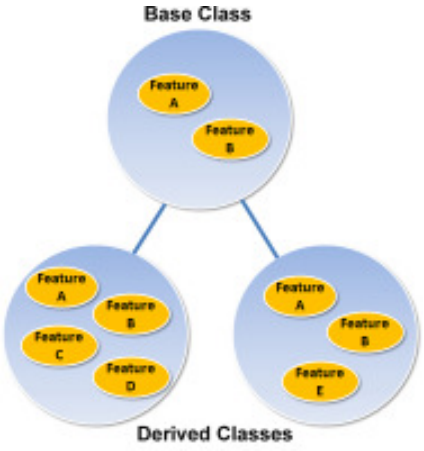
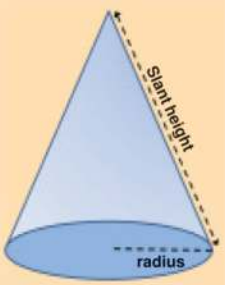


Fig. 2.11: Inheritance in OOP

The following program derives a class 'Cone' from class 'Circle' through inheritance. As radius(r) is already declared in the base class 'Circle', the derived class 'Cone' needs to declare only one data member the slant height(s). Two member functions, one to accept data and other to display the area of cone are also declared. The data member of the class 'Circle' are to be declared 'protected' so that they can be accessed by the derived class 'Cone'.




```
#include <iostream>
using namespace std;
class Circle
{ protected:
    float r;
  public:
    void get_radius(){
      cout << "Enter Radius :";
      cin >> r;
    }
    void display_area(){
      cout<< "Area:"<< 3.14*r*r;
    }
};
class Cone : public Circle
{ private:
    float s;
  public:
    void get_cone_data() {
      get_radius();
      cout << "Enter slant height:";
      cin >> s;
    }
    void display_cone_area(){
    cout << "Area :" << 3.14*r*(s+r);
    }
};
int main()     {
    Cone C1;
    C1.get_cone_data(); //Sending message
    C1.display_cone_area(); //Sending message
}
```

Base Class

New member

New member functions

Derived class

Main function

Output :
```
Enter Radius : 2.0
Enter slant height: 5.0
Area :43.96
```

Different forms of Inheritance are Single Inheritance, Multiple Inheritance, Multilevel Inheritance, Hierarchical Inheritance, and Hybrid Inheritance (see Figure 2.12).

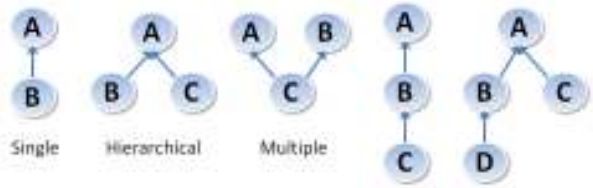


Fig. 2.12: Types of inheritance

Fig.: 2.13 Sample Inheritance

Let us try to implement the concept of inheritance in our school software. Assume that in addition to existing data and function, we need to add new data - age of the student, and standard in which the student is studying and a function to find the examination result, to `Student` class. Instead of modifying the existing `Student` class, we can derive a class named `NewStudent` from the existing `Student` class, so that the `Student` class remains undisturbed. Here `Student` is the base class and `NewStudent` is the derived class (see Figure 2.13).

The syntax for declaring a derived class is as follows:

```
class derived_class: AccessSpecifier base_class
{
   //declaration of members and member functions
};
```

where `derived_class` is the name of the derived class and `base_class` is the name of the class on which it is based. The `AccessSpecifier` may be `public`, `protected` or `private`. This access specifier describes the access level for the members that are inherited from the base class.

2.2.7 Polymorphism

'Poly' means many. 'Morph' means shapes. So polymorphism can be defined as the ability to express different forms. This is demonstrated in Figure 2.14. Here the same command "Now Speak" is issued to all objects, but each object responds differently to the same command.

In object-oriented programming, polymorphism refers to the ability of a programming language to process objects differently depending on their data type or class. More specifically, it is the ability to redefine methods for derived classes.

Fig. 2.14: Demonstration of Polymorphism

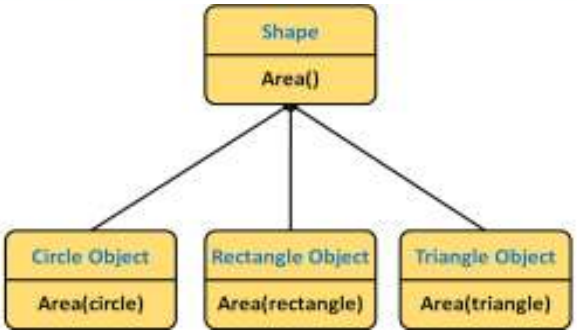


Fig.: 2.15 Example of Polymorphism

There are two types of polymorphism.

a. Compile time (early binding/static) polymorphism

b. Run time (late binding/dynamic) polymorphism

For example, given a base class `Shape`, polymorphism enables the programmer to define different `area` methods for any number of derived classes such as `circle`, `rectangle` and `triangle`. No matter what shape an object has, applying the `area` method to it will return the correct results.

Fig.: 2.16 Classification of Polymorphism

a. Compile time polymorphism

Compile time polymorphism refers to the ability of the compiler to relate or bind a function call with the function definition during compilation itself. Function overloading and operator overloading comes under compile time polymorphism.

Function Overloading: Functions with the same name, but different signatures can act differently. For example `area(int)` can be used to find the area of a square whereas `area(int, int)` can be used to find the area of a rectangle. Thus, the same function `area()` acts in two different ways depending on its signature. Defining multiple functions with the same name and different function signatures is known as function overloading.

Operator overloading: Operator overloading is the concept of giving new meaning to an existing C++ operator (like +, -, =, * etc.). It makes it possible to use the ordinary operator to exhibit different behaviors on different objects of a class, depending on the types of operands it receives. To overload an operator, we need to write a member function for the operator we are overloading.

For example, the + (plus) operator in C++ is already overloaded as it can do integer addition (`4 + 5`) and floating point addition (`3.14 + 2.6`). If needed, we can add additional functionality to it and make it add two objects. For example `T1 = T2 + T3`, where `T1, T2` and `T3` are all objects of a class named '`time`'. Here '+' may be used to add two time sequences represented in HH:MM:SS format.

b. Run time polymorphism

Run time polymorphism refers to the binding of a function definition with the function call during runtime. It uses the concept of pointers and inheritance.

The following program implements function overloading, to find the area of a square and a rectangle. It defines two functions, one to find the area of a square and the other to find the area of a rectangle. Both functions are given the same name 'area' but have different signatures.

Function names are same

Signatures are different

```
#include<ciostream>
using namespace std;
int area(int s){ //To find the area of a square
return s * s;
}
int area(int s1, int s2){ //To find area of a
return (s1 * s2);          //rectangle
}
int main()
{
cout << "Area of Square:" << area(5); << endl;
cout << "Area of Rectangle:"<< area (7,2);
}
```

Output:

```
Area of Square: 25
Area of Rectangle: 14
```

Know your progress

1. The wrapping up of data and functions into a single unit is referred to as _____.
2. Access to data is restricted by the feature known as _____.
3. Objects normally communicate with each other through _____.
4. C++ supports _____ and _____ binding.
5. Late binding is also called _____.
6. Early binding is also called _____.
7. What are the different types of inheritance?

Let us conclude

As software makes the computer a useful machine, the development and maintenance of software requires special consideration. In order to make software development more productive and to reduce maintenance costs, various methods/paradigms have been tried. They are Structured paradigm, Procedural paradigm, Modular paradigm and Object-oriented paradigm (OOP). Most of the latest and widely-used programming languages follow OOP. OOP implements a problem using objects that can communicate with each other. Here, data is given more importance than in previous paradigms. To give maximum protection to data from unauthorised access they are defined using various access specifiers and kept along with functions that operate on the data. OOP also provides effective modularisation and features to improve reusability and extensibility of a code, and can implement static and dynamic polymorphism.

Let us assess

1. Protecting data from unauthorised access is _____.
   a. Polymorphism b. Encapsulation c. Data abstraction d. Inheritance
2. A base class may also be called _____.
   a. Child class b. Subclass c. Derived class d. Parent class
3. Which of the following is not a type of inheritance?
   a. Hybrid b. Multiple c. Multilevel d. Multiclass
4. Subclass is the same as:
   a. Derived class b. Super class c. Base class d. None of these
5. Default access specifier is :
   a. public b. private c. protected d. none
6. Which of the following is not an OOP concept?
   a. Overloading b. Procedural programming
   c. Data abstraction d. Inheritance
7. The ability of a message or data to be processed in more than one form is called
   a. Polymorphism b. Encapsulation c. Data hiding d. Inheritance
8. C++ is a _____ language.
   a. Object based b. Non-procedural c. Object oriented d. Procedural

9. Which of the following is not a characteristic of OOP?
   a. It emphasizes procedure more than data.
   b. It offers good real world modelling.
   c. It wraps up related data items and associated functions in the unit.
   d. None of these.
10. Which among the following is true about OOPs?
   a. It supports data abstraction
   b. It supports polymorphism
   c. It supports structured programming
   d. It supports all of these
11. What do you mean by programming paradigm? Name the programming paradigms.
12. What are the limitations of procedural programming approaches?
13. What is object oriented-programming paradigm? List the basic concepts of OOP.
14. How does OOP implemented in C++?
15. What is encapsulation?
16. Distingush between an object and a class?
17. What is a base class and a sub class? What is the relationship between base class and subclass?
18. Explain the concept of data abstraction. Give an example.
19. Write a short note on inheritance.
20. To operate a car, we use behaviors such as steering, brakes, accelerator etc. All we know is how to use these. We need not know what happens internally when we apply these. Can you connect this with any of the OOP concept? Explain?
21. What do you mean by inheritance? How does this support 'reusability'?
22. What is polymorphism? Give an example to illustrate this feature.
23. Explain the concept of OOP with examples.
24. There is a plug point with a switch. What a switch "does", depends on what is connected to the plug point, and the context in which it is used. Can you connect this with any of the OOP concepts? Explain?
25. Let us assume there is a base class named '`LivingBeings`' from which the subclasses '`Horse`', '`Fish`' and '`Bird`' are derived. Let us also assume that the `LivingBeings` class has a function named '`Move`', which is inherited by all subclasses mentioned. When the `Move` function is called in an object of the `Horse` class, the function might respond by displaying trotting on the screen. On the other hand, when the same function is called in an object of the `Fish` class, swimming might be displayed on the screen. In the case of a `Bird` object, it may be flying. Can you connect this with any of the OOP concept? Explain?

3 Data Structures and Operations

Significant Learning Outcomes

After the completion of this chapter, the learner

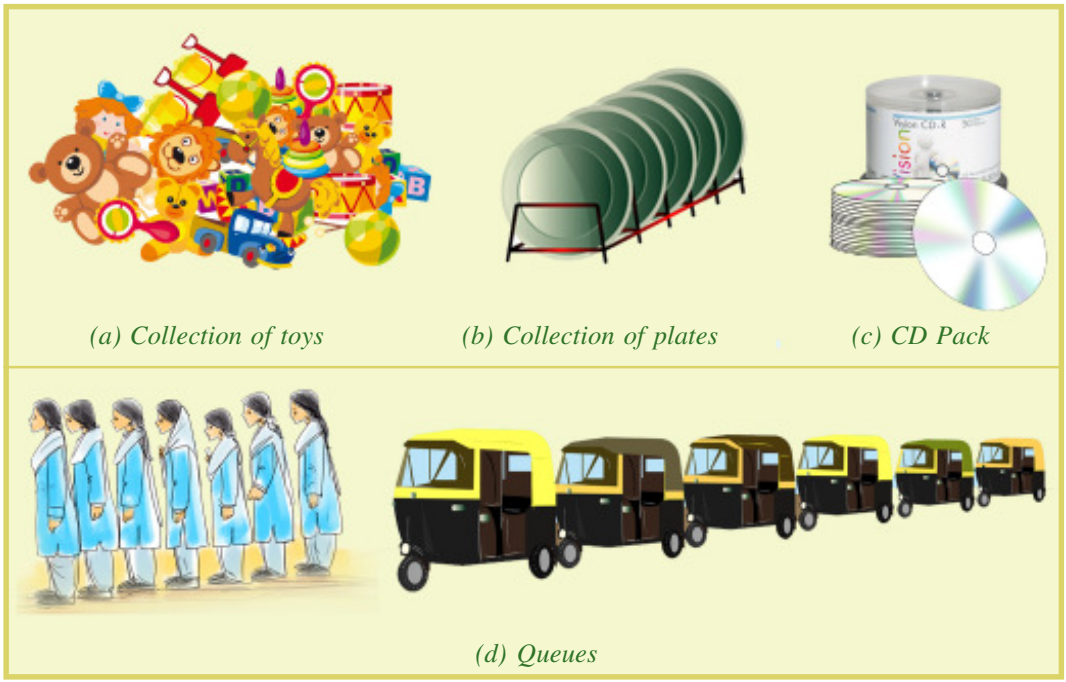
- explains the concept of data structure by citing examples.
- classifies data structures based on different criteria.
- lists different operations on data structures and explains them.
- explains the organisation of stack data structure with the help of examples.
- develops algorithms for push and pop operations in a stack.
- explains the organisation of queue data structure with the help of examples .
- develops algorithms for insertion and deletion operations in a linear queue.
- identifies the advantage of circular queue over linear queue.
- explains the concept of linked list data structure and its advantages over arrays and other static data structures.
- develops procedures to create a linked list and to perform traversal operation.

While solving problems using computers, the data may have to be processed in most of the cases. These data may be of atomic (fundamental) type or aggregate (grouped) type. We know that variables are required to refer to these data. Languages like C, C++, Java, etc. insist on declaration of variables before their use in the program. We learnt that in C++, variables for atomic type data are declared using fundamental data types like `int`, `char`, `float` and `double` or with their type modifiers. We have also seen that grouped data are referred using arrays and structures. Array is a collection of homogeneous type of data, whereas structure can be a collection of different types of data.

This chapter presents the facilities of programming languages to organise data in groups based on different principles and criteria. The amount of data that can be accommodated in a group and the operations performed on them vary depending on the organising principle followed in constituting each group of data.

3.1 Data Structure

Figure 3.1 shows some items in groups. Each group follows some kind of grouping strategy. Can you identify the principle or format followed in arranging the items in each group?



(a) Collection of toys *(b) Collection of plates* *(c) CD Pack*

(d) Queues

Fig. 3.1: Different ways of grouping

Figure 3.1(a) is a collection of toys. The toys are dumped together without any specific order or arrangement. Figure 3.1(b) is a collection of plates in a stand. The plates are placed one after the other. There is a limit for the number of plates that can be placed in a stand. A new plate can be placed at any position in the stand if there is space and any plate can be taken out from it. Figure 3.1(c) is a set of discs in a CD pack. There is a limit to the number of discs in this collection also. A new disc can be added in the collection only at the top. Similarly only the CD at the top can be removed from the collection. Figure 3.1(d) shows queues in which a new person (or a new auto rickshaw) can join the queue only at the rear end. The person (or auto rickshaw) leaves the queue only from the front end. In this collection there may not be a limit for the number of persons in the queue. But in some cases there may be a limit in the size of the queue also.

The concept of data structure is similar to the collections in figures (b), (c) and (d) in Figure 3.1. Figure 3.1(b) is a collection, which is very similar to an array that we learnt in Class XI. So we say that array is a data structure. In Computer Science, a

data structure is a particular way of organising similar or dissimilar logically related data items which can be processed as a single unit. Data structures not only allow the user to combine various types of data but also allow processing of the group as a single unit.

3.1.1 Classification of data structures

Data structures can be generally classified into simple data structures and compound data structures. Figure 3.2 shows the detailed classification.

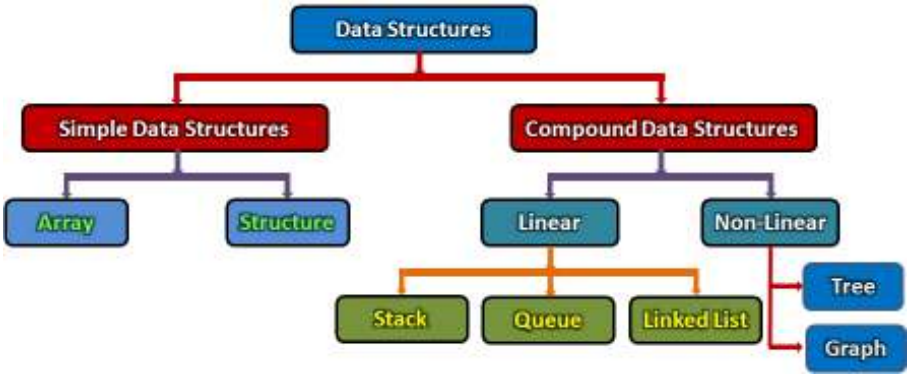


Fig. 3.2: Classification of data structures

We are familiar with the simple data structures such as array and structure. We used them in C++ programs to refer to a collection of elements. These simple data structures are combined in various ways to form compound data structures. As shown in Figure 3.2, compound data structure is further classified into linear and non-linear. A data structure is said to be linear if its elements form a sequence. The elements of a linear data structure can be represented by means of sequential memory locations. A data structure is said to be non-linear if its elements do not form any sequence in memory. In this type of data structure the elements are stored in random memory locations and they need not be accessed in sequential order. Non-linear data structures are more complex and hence you will study them in higher classes. Linear data structures such as stack, queue and linked list will be presented in detail in the coming sections.

Since data structures represent collections of data, these are closely related to computer memory, as it is the storage space for the data. The memory may be primary or secondary. Depending upon memory allocation, data structures may be classified as ***static data structures*** and ***dynamic data structures***. Static data structures are associated only with primary memory. The required memory is allocated before the execution of the program and the memory space will be fixed throughout the execution. That is, the size of the data structure is specified during the design and it cannot be changed later. Data structures designed or implemented

using arrays are static in nature. But for dynamic data structures, memory is allocated during execution. Data structures implemented using linked lists are dynamic in nature. The size of the collection will not be specified in advance; rather it grows or shrinks during run-time as per user's desire. When we consider secondary memory for the storage of data, it will be in the form of files. The size of such data files increases on addition of data and reduces on deletion of data. So we can say that files are also dynamic data structures.

3.1.2 Operations on data structures

The data represented by the data structures are processed by means of certain operations. In fact, a particular data structure that one chooses for a given situation depends largely on the frequency with which specific operations are performed. The operations performed on data structures are traversing, searching, inserting, deleting, sorting and merging. Let us have some basic idea about these operations.

a. Traversing

Traversing is an operation in which each element of a data structure is visited. The travel proceeds from the first element to the last one in the data structure. Processing of the visited element can be done according to the requirements of the problem. Reading all the elements of an array is an example for traversing (Refer Chapter 8 of Computer Science Textbook for Class XI).

b. Searching

Searching, in the literal sense, is the process of finding the location of a particular element in a data structure. Searching may also be a process of finding the locations of all the elements satisfying one or more conditions. In other words, searching implies accessing the values stored in the data structure. We learned two methods for the search operation in an array in Class XI.

c. Inserting

Insertion is the operation in which a new data is added at a particular place in a data structure. In some situation, where the elements in the data structure are in a particular order, the position may need to be identified first and then the insertion is to be done.

d. Deleting

Deletion is the operation in which a particular element is removed from the data structure. The deletion is performed either by mentioning the element itself or by specifying its position.

e. Sorting

We are familiar with the sorting of an array using two methods named bubble sort and selection sort. It is the technique of arranging the elements in a specified order, i.e., either in ascending or descending order. Sorting of elements in a data structure makes searching faster.

f. Merging

Merging usually refers to the process of combining elements of two sorted data structures to form a new one. But the simplest form of merging is the joining of the elements of both the data structures into a third empty data structure. In the case of an array, first, copy all the elements of one array into a third empty array, and then append all the elements of the other array to those in the third array.

Searching, sorting and merging are three related operations which make the job of retrieval of data from the storage devices easier, faster and efficient.

In Class XI, we learned the data structure array and how the operations given above are performed. We also discussed the concept of structures and the way of operations on their elements in Chapter 1 of this book. Now, let us discuss the compound linear data structures stack, queue and linked list.

3.2 Stack

Have a close look at Figure 3.1(c) and also the collections shown in Figure 3.3. The organisation of items in these groups is the same.

Fig. 3.3: Real life examples for stack

The collection is formed by adding each item one over the other. We can say that the items are added at the top position. Similarly, we can remove only that item which is placed at last. This organising principle is known as Last-In-First-Out (LIFO). The data structure that follows LIFO principle is known as ***stack***. It is an ordered list of items in which all insertions and deletions are made at one end, usually called **Top**. Since it follows the LIFO principle, the element added at last will be the first to be removed from the stack.

Stack is a logical concept. There is no exclusive tool or facility in programming languages to create a stack. A stack can be physically implemented by an array. Such

a stack inherits all the properties of arrays. The size is predetermined and hence it is static. Figure 3.4 is a stack of integers implemented by an array **STACK**. This stack can accommodate a maximum of 10 integers. The figure shows that there are six elements at present and the last element is 56 at position 5. The last position is indicated by **TOS** (to denote Top Of Stack). So, the value of **TOS** is 5. The last element in a stack is always referred to by the expression **StackName[TOS]**. In this case **STACK[TOS]** gives 56. Since we utilise arrays of C++ for implementing stacks, the first element will always be referred to by the subscript 0. Here **STACK[0]** is 25.

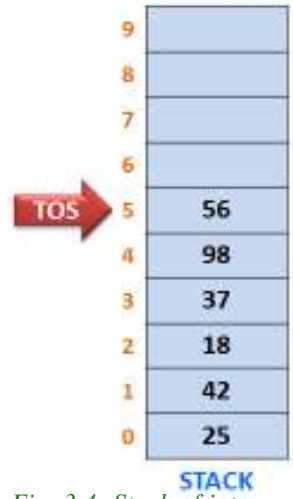


Fig. 3.4: Stack of integers

3.2.1 Implementation of stack

It is mentioned that stack can be implemented using array. In such a case, there is a limit to the number of elements that can be represented by the stack and it depends on the size of the array. Initially, the value of TOS will be set to -1 (minus one) to denote the fact that the stack is empty. Whenever an element is added (or inserted) into the stack, the value of TOS will be incremented by 1 until it reaches the highest subscript of the array. If an array STACK of N elements is used to implement a stack, the values of Top can vary from 0 to (N-1) and the elements in the stack can be referred to by the expressions STACK[0], STACK[1], STACK[2], ..., STACK[N-1].

3.2.2 Operations on stack

Though array is used to implement stack data structure, all the operations applicable to array are not performed in the same fashion. For example in an array, insertion and deletion operations can be performed at any position. But in stack, there is restriction in the position. These operations are performed only at the top position. The insertion and deletion operations on stack are known as ***push*** and ***pop*** operations respectively. Let us discuss the procedure involved in these operations.

a. Push operation

As mentioned above, ***push*** operation is the process of inserting a new data item into the stack. Even the creation of a stack itself is a repeated execution of push operations.

Let us do

Figure 3.5 shows the status of a stack during a sequence of push operations. Let us assume that we have created an array and value of TOS is set with -1. Now observe the figure and write down the procedure to perform the operation.

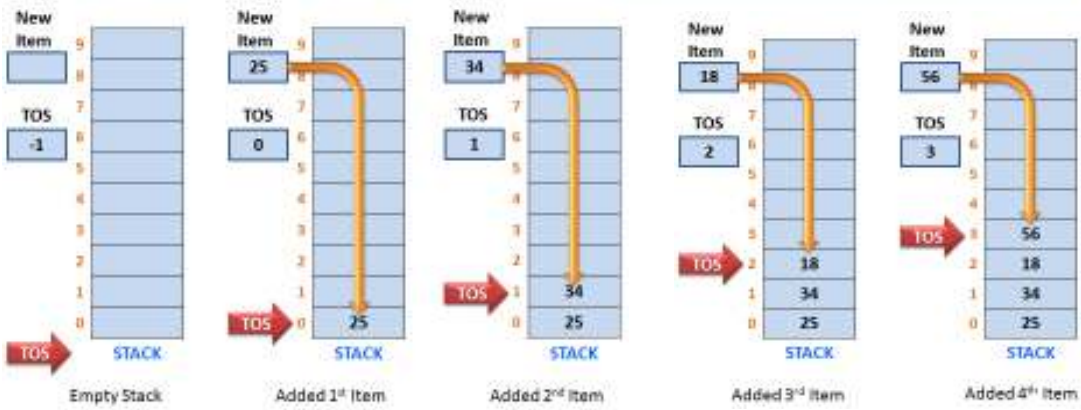


Fig. 3.5: Status of stack after a sequence of push operations

Verify whether you could derive the following steps for push operation on a stack:

Step 1: Accept the value in a variable to insert into the stack.
Step 2: Increment the value of Top by 1.
Step 3: Store the value at the TOS position.

The above steps are valid only if there is free space for insertion. The stack shown in Figure 3.5 cannot accommodate a new item when the value of TOS is 9. Once the stack is full and if we attempt to insert an item, an impossible situation arises. This is known as ***stack overflow***. Now let us write an algorithm for push operation in a stack.

Algorithm: To perform PUSH operation in a stack

Consider an array STACK[N] that implements a stack, where N is the maximum size of the array. A variable TOS keeps track of top position of the stack. A data item available in the variable VAL is to be added into the stack. The steps required for push operation are given between the Start and Stop instructions.

```
Start
    1:   If  (TOS < N) Then              //Space availability checking (Overflow)
    2:        TOS = TOS + 1
    3:        STACK[TOS] = VAL
    4:   Else
    5:        Print "Stack Overflow "
    6:   End of If
Stop
```

a. Pop operation

The process of deleting an element from the top of a stack is called ***Pop*** operation. After every pop operation the value of TOS is decremented by one.

Let us do

Let us try ourselves to derive the steps for deleting the element at the top of a stack. Figure 3.6 is given to you for reference. Observe the figure and write down the steps for pop operation.

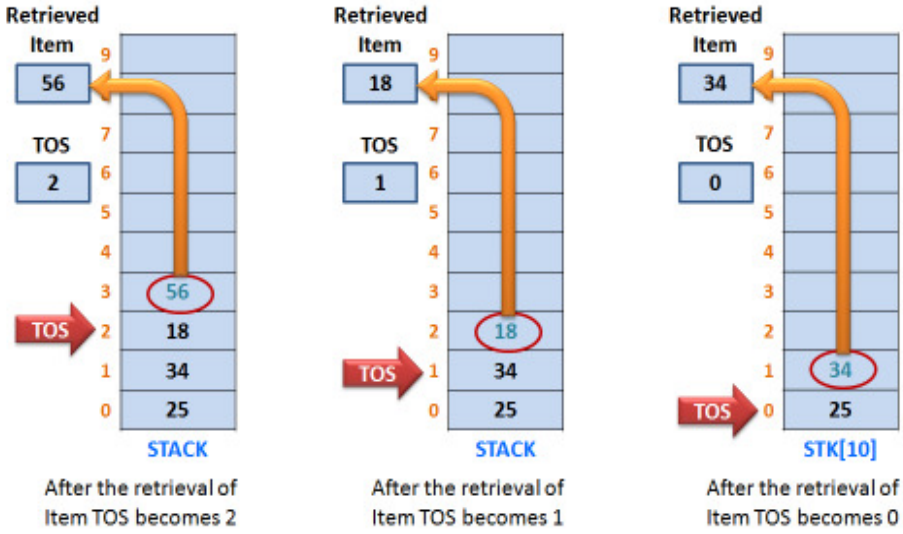


Fig. 3.6: Status of stack after a sequence of pop operations

You might have derived the following steps for pop operation:

Step 1: Retrieve the element at the TOS into a variable.

Step 2: Decrement the value of TOS by 1.

These two steps will work when there are elements in the stack. During the pop operation in an array stack, the element is not physically removed, but the access is restricted by decrementing the value of TOS. The stack shown in Figure 3.6 can be given for pop operation until the element at position 0 is deleted. After the deletion of that last element, the value of TOS becomes -1. Now the stack is empty and if we try to delete an item from the stack, an unfavourable situation arises. This situation is known as ***stack underflow***. Now let us write an algorithm for pop operation in a stack.

Algorithm: To perform POP operation in a stack

Consider an array STACK[N] that implements a stack that can store a maximum of N elements. A variable TOS keeps track of the top position of the stack. The data item retrieved from the stack may be kept in a variable, say VAL. The steps required for pop operation are given between the Start and Stop instructions.

```
Start
    1:   If  (TOS > -1) Then          //Empty status checking (Underflow)
    2:        VAL = STACK[TOS]
    3:        TOS = TOS - 1
    4:   Else
    5:        Print "Stack Underflow "
    3:   End of If
Stop
```

C++ functions for stack operations

The variables `tos` and `n` are assumed as global variables

PUSH operation

```
void push(int stack[],int val)
{
    if (tos < n)
    {
        tos++;
        stack[tos]=val;
    }
    else
        cout<<"Overflow";
}
```

POP operation

```
int pop(int stack[])
{
    int val;
    if (tos > -1)
    {
        val=stack[tos];
        tos--;
    }
    else
        cout<<"Underflow";
    return val;
}
```

Application of Stacks

Since stacks follow the LIFO principle, they are used for applications such as reversing a string, creating polish strings, evaluation of polish strings etc. Reversing a string involves the formation of a string by reversing the characters of the original string. For example, "SAD" is a string and its reversed string is "DAS". Polish string is an arithmetic expression, in which operator is placed before the operands or after the operands. For example, the arithmetic expression A+B can be converted into AB+ or +AB. The expression A+B is familiar to us and is known as infix expression. The expression AB+ or +AB is the required format for Arithmetic Logic Unit (ALU) of the computer and are known as postfix expression and prefix expression, respectively. These conversions are carried out in the computer with the help of push and pop operations. The prefix or postfix version of an arithmetic expression is also evaluated using these stack operations by ALU.

3.3 Queue

In many situations we might have become part of a queue. Figure 3.7 shows a queue in a polling station, where the voter at the front position cast his/her vote first and a new person can join the queue at the rear position. It is clear that, the first voter in the queue will come out first from the queue. This style of oraganising a group is said to follow the First-In-First-Out (FIFO) principle. So, we can say that a data structure that follows the FIFO principle is known as a ***queue***. As we can see in the figure a queue has two end points - front and rear. Inserting a new data item in a queue will be at the rear position and deleting will be at the front position. Queue is also a logical concept. It can be physically implemented by an array and such a queue is static in nature.

Fig. 3.7: Queue in a polling station

3.3.1 Implementation of queue

When a queue is implemented by an array, there is a limit for the number of elements that can be represented by it. Figure 3.8 is a queue of integers implemented by an array QUEUE, which can accommodate a maximum of 10 integers. The figure shows that there are five elements in the queue stored from positions 0 to 4. So, the value of Front is 0 and that of Rear is 4. The first element of this queue is always referred to by QUEUE[Front] and that last element by QUEUE[Rear].

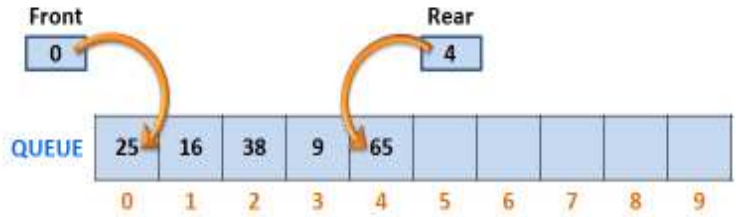


Fig. 3.8: Queue implemented by array

The highest value allowed to Rear is 9 as it is the last subscript of the array. Initially, the value of Front and Rear would have been -1, which indicates that the queue is empty. When the first element is inserted into the queue, the values of these end points will be set to 0. Thereafter, whenever an element is inserted, the value of Rear will be incremented by 1, until it reaches the highest subscript (here it is 9). Similarly on each deletion, the value of Front will be incremented by 1, until it crosses the value of Rear.

3.3.2 Operations on queue

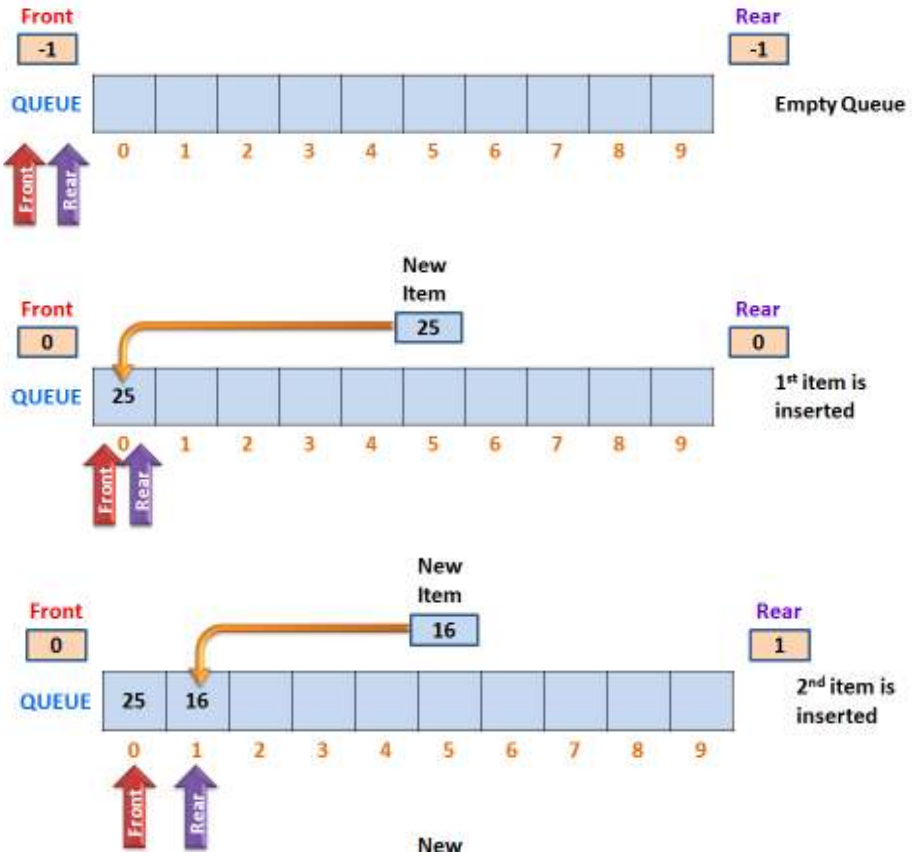
As in the case of stacks, there are some restrictions in the insertion and deletion operations on queues. In an ordinary array, insertion and deletion operations are performed at any position and in stacks these operations are performed only at the top position. But in the case of queue, insertion and deletion operations are done at different end points.

a. Insertion operation

Insertion is the process of adding a new item into a queue at the rear end. The value of Rear is incremented first to point to the next location and the element is inserted at that position. Even the creation of a queue is accomplished by performing the insertion operation repeatedly.

Let us do

Figure 3.9 shows the status of a queue during a sequence of insertion operations. Let us assume that we have created an array and value of Front and Rear are set to -1. Now, observe the figure and write down the procedure to perform the operation.



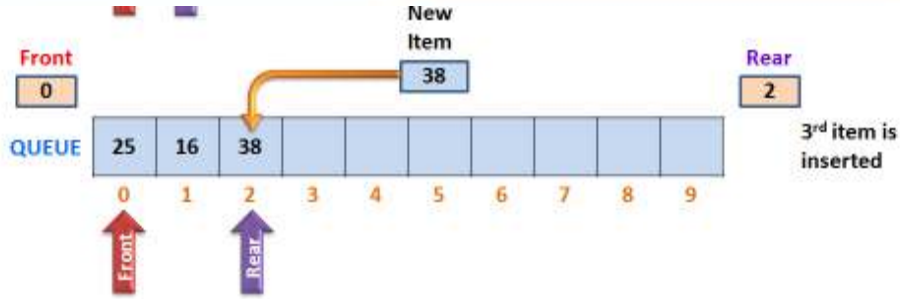


Fig. 3.9: Status of a queue after a sequence of insertion operations

Now check whether you could derive the following steps for the insertion operation

Step 1: Accept the value in a variable for inserting into the queue.
Step 2: Increment the value of Rear by 1.
Step 3: Store the value at the Rear position.

Note that during the first insertion operation in the empty queue, the values of both Front and Rear are incremented, i.e., they are set to 0. Thereafter, only the Rear is incremented. This can be continued until Rear becomes 9 (the last subscript of the array). The attempt of next insertion causes ***queue overflow*** as in the case of stack. Now, let us write an algorithm for insertion operation in a queue.

Algorithm: To perform INSERTION operation in a queue

Consider an array QUEUE[N] that implements a queue, where N is the maximum size of the array. The variables FRONT and REAR keep track of the front and rear positions of the queue. A data item available in the variable VAL is to be inserted into the stack. The steps required for insertion operation are given between the Start and Stop instructions.

```
Start
    1:   If (REAR == –1) Then          //Empty status checking
    2:        FRONT = REAR = 0
    3:        Q[REAR] = VAL
    4:   Else If (REAR < N) Then       //Space availability checking
    5:        REAR = REAR + 1
    6:        Q[REAR] = VAL
    7:   Else
    8:        Print "Queue Overflow "
    9:   End of If
Stop
```

b. Deletion operation

Deletion operation is the removal of the item at the front end. After the deletion, the value of Front is incremented by 1. In the case of array queue, the item is not physically removed, rather its access is denied by incrementing the value of Front.

Let us do

Assume that we have a queue implemented by an array. Figure 3.10 shows the status of this queue during a sequence of deletion operations. Let us try to derive the steps for the operation ourselves.

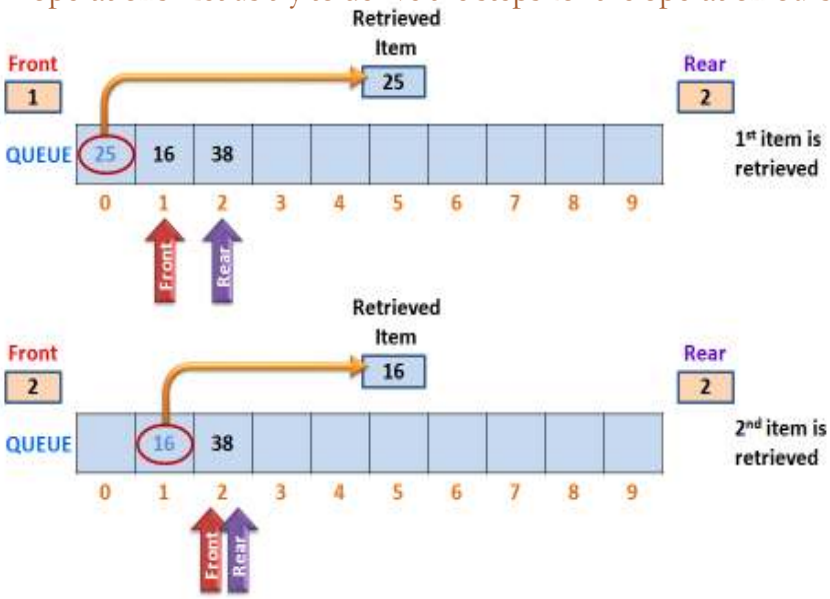


Fig. 3.10: Status of a queue after a sequence of dequeue operations

Could you derive the following steps for deletion operation?

Step 1: Retrieve the element at the Front position into a variable.

Step 2: Increment the value of Front by 1.

According to Figure 3.10, there is no shifting of elements towards the front as you may expect. The queues in real life situations may conflict with this concept. But in queue data structure, deletion operation does not cause a shift of elements; rather the value of Front is incremented by 1. The two steps derived will be enough as long as there is element in the queue. Look at the status of the queue after the deletion of 2nd element. Front and Rear point to the same element, which is at subscript 2. Assume that, one more deletion is carried out in the queue. According to the procedure the value of Front becomes 3, which is greater than that of Rear. We know that it is not reasonable. We can also see that the queue is now empty. Remember that, the value of Front and Rear is -1 when the queue is empty. So we have to include a step to reset the values of Front and Rear as -1 when the last element is deleted from the queue. In this state, further deletion cannot be allowed.

If an attempt is made to delete an item from an empty queue, the situation is called ***queue underflow***. Now let us develop a complete algorithm for deletion operation on a queue.

Algorithm: To perform DELETION operation in a queue

Consider an array QUEUE[N] that implements a queue with a maximum of N elements. The variables FRONT and REAR keep track of the front and rear positions of the queue. The data item removed from the queue will be stored in the variable VAL. The steps required for deletion operation are given between the Start and Stop instructions.

```
Start
    1:   If (FRONT > –1 AND FRONT < REAR) Then    // Empty status checking
    2:       VAL = Q[FRONT]
    3:       FRONT = FRONT + 1
    4:   Else
    5:       Print "Queue Underflow "
    6:   End of If
    7:   If (FRONT > REAR) Then       // Checking the deletion of last element
    8:       FRONT = REAR = -1
    9:   End of If
Stop
```

C++ functions for queue operations

The variables `n`, `front` and `rear` are assumed as global variables

INSERTION operation

```
void ins_q(int queue[],int val)
{
    if (rear == -1)
    {
        front=0;
        rear=0;
        q[rear]=val;
    {
    else (if rear < n)
    {
        rear++;
        q[rear]=val;
    }
    else
        cout<<"Overflow";
}
```

DELETION operation

```
int del_q(int queue[])
{
    int val;
    if (front > -1)
    {
        val=q[front];
        front++;
    }
    else
        cout<<"Underflow";
    if (front > rear)
    {
        front= -1;
        rear= -1
    }
    return val;
}
```

Application of Queues

Mostly the queue concept in computer applications occurs in job scheduling. The computer operating systems use this queue concept in scheduling the memory, processor and the files. Print queue is an example of job scheduling. Since the printer is slow compared to the processor, the print jobs submitted by the user are put in the print buffer. The jobs are organised in the buffer following the FIFO principle and thus the print buffer becomes a queue.

3.3.3 Circular queue

Queues we discussed so far are known as linear queues. The elements are arranged in a row or line. The two ends of such queues never meet at any point. There is a drawback in linear queue. Consider the queue shown in Figure 3.11 in which six deletion operations are done. At present, there are only three elements. Obviously, the value of Front is 6 and that of Rear is 8.

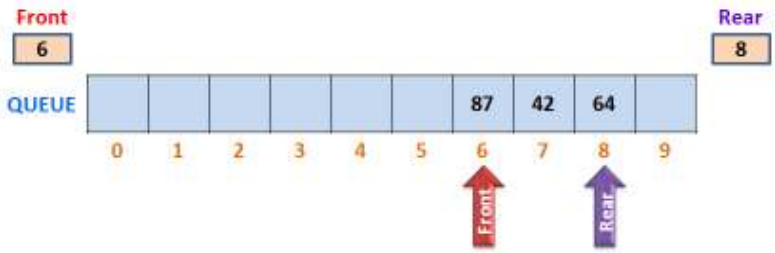


Fig. 3.11: Queue with three elements after a sequence of six deletion operations

Even though the first six locations are free, we can insert only one element according to the insertion algorithm. Imagine the worst situation that there is only one element at the last position of the queue (say at 9). If we try an insertion operation in this queue, there will be overflow situation. This limitation of linear queue can be overcome by circular queue. It is a queue in which the two end points meet as shown in Figure 3.12.

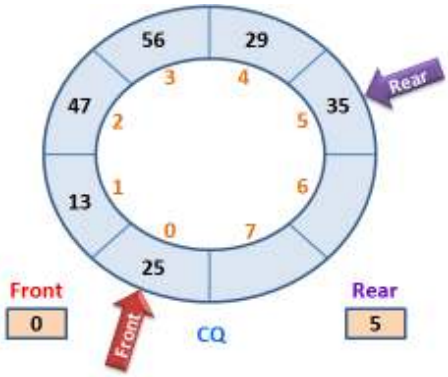


Fig.3.12: Circular queue

Assume that two deletions are performed at this stage. So the value of Front will be 2 and now we have four free locations as shown in Figure 3.13(a). The subscripts of these position are 6, 7 and then 0, 1. If we insert two elements one by one, the value of Rear becomes 7, but still we have two free locations with subscripts 0 and 1,

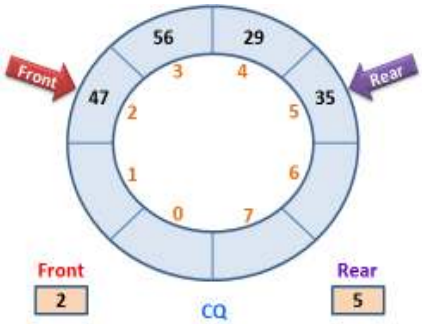


Fig.3.13(a): Circular queue after deletion

as shown in Figure 3.13(b). So insertion operation can be performed again. This time the value of Rear should be set with 0 first and then insertion is to be carried out. Figure 3.13(c) shows this status of the queue.

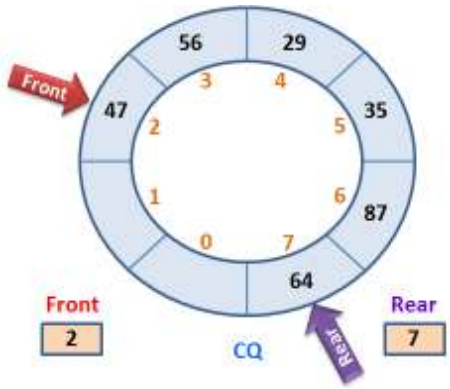


Fig.3.13(b): Rear gets its highest value

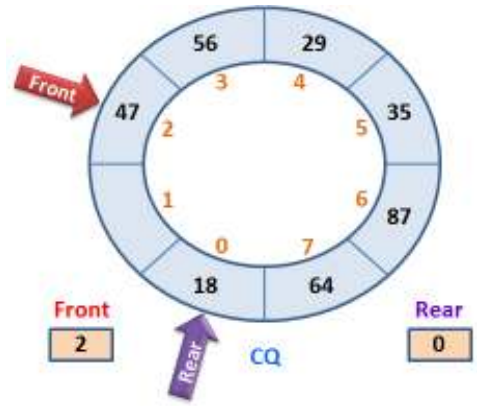


Fig.3.13(c): Rear takes the lower bound value

Know your progress

1. Define data structure.
2. Stack follows _____ principle for oraginsing data.
3. Name the data structure that follows FIFO principle.
4. What is meant by underflow?
5. Which element of stack can be deleted (First or Last)?

3.4 Linked list

Linked list is a collection of data items where there is no limit in the number of items. Stacks and queues discussed in the previous sections are merely logical concepts, which are physically implemented using arrays. So these implementations are static. But linked list is a dynamic data structure. It grows as and when new data items are added in the list and shrinks whenever any data is removed from the list. Memory will not be allocated in advance for the entire list. The memory allocation takes place during the execution time just when a new item is about to insert in the list. That is why it is considered as an example for dynamic data structure. Another difference compared to array based data structures is that the elements in the linked list are scattered in the memory. But they are linked with pointers. As we learnt in Chapter 1, pointer is a variable that contains the address of a memory location. So it is clear that an element in a linked list consists of data and an address. Such an element is known as ***node***. The address contained in the node is known as ***link***.

So, a ***linked list*** is a collection of nodes, where each node consists of a ***data*** and a ***link*** - a pointer to the next node in the list. That is, the first node in the list contains the first data item and the address of the second node; the second node contains the second data and the address of the third node; and so on. The last node at a particular stage contains the last data in the list and a *null pointer*, i.e., a pointer that points to nowhere. Now, a question arises. Where will the address of the first node be? There is a special pointer associated with each linked list, known as Start or Header and it contains the address of the first node. Figure 3.14 shows a linked list of five numbers.

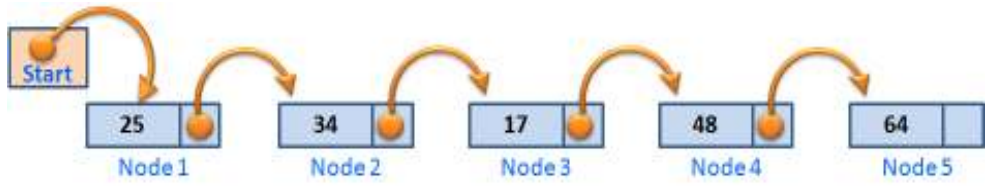


Fig. 3.14: Linked list with five nodes

The figure shows that each node in the linked list consists of a number as its data and a pointer as the link that points to the next node. The size of all the nodes will be the same. That is, memory space allocated for each node will be of the same size.

3.4.1 Implementation of linked list

We have seen that linked list is a collection of nodes and each node is allocated memory space. The memory location for a node consists of at least two types of data, one is the actual data item and the other is a pointer to the memory location for the next node. In Chapter 1, we learnt that structure is a user-defined data type consisting of different types of data. The element of a structure can be a pointer and it may even be a pointer to the same structure. We call such a structure a self referential structure. So, linked list is created with the help of self referential structures. The nodes in the linked list in Figure 3.14 can be designed using the following self referential structure:

```
struct Node
{
    int data;
    Node *link;
};
```

As we can see, the name of the structure is Node and the second element is a pointer to the same structure type **`Node`**. The special pointer **`Start`** that points to the first node can be created using the following statement:

```
struct Node * Start;        or        Node *Start;
```

Figure 3.15 shows a linked list of strings. The data of the nodes will be filled with strings and link with addresses of other nodes. Note that the addresses are only assumed numbers. The following structure can represent these nodes:

```
struct Node
{
    char data[10];
    Node *link;
};
```

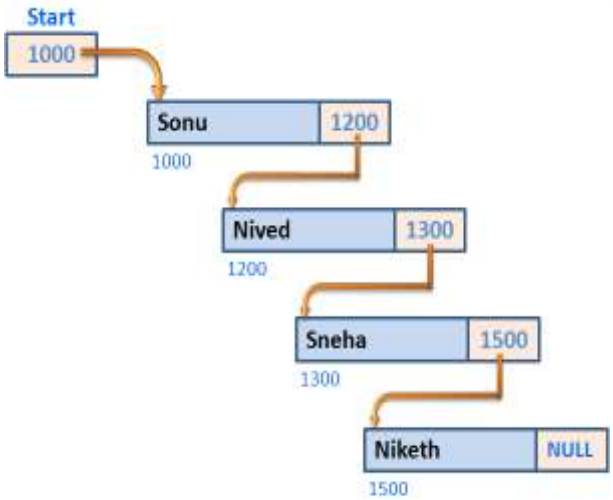


Fig.3.15: Implementation of a linked list

It is assumed that memory location at address 1000 is allocated during run-time for the first node and hence the pointer `Start` contains 1000. The data part of this node is filled with a name "Sonu". The second node is given memory space at location 1200 and its data part is filled with another name "Nived". Being the second node, its address is stored in the link part of the first node. As seen in the figure, the last node contains null pointer.

3.4.2 Operations on linked list

All operations specified in Section 3.1.2 can be performed in linked lists without any restrictions. But we will discuss only the creation, traversal, insertion and deletion operations only. You will learn the remaining operations during higher studies.

a. Creation of linked list

We have to define a suitable self referential structure in the beginning. A pointer variable say Start or Header is then declared and initialised with NULL value. Now we start creating a linked list by dynamically allocating memory for the nodes according to the requirements. In Chapter 1, we learnt that when memory is dynamically allocated, an address is be returned. This address is stored in a pointer variable and using this variable the location can be accessed.

Let us do

Figure 3.16 shows the status of a linked list during its creation. Let us assume that we have defined a self referential structure named Node and initialised a Node type pointer Start with NULL. Now observe the figure and write down the procedure to create a linked list.

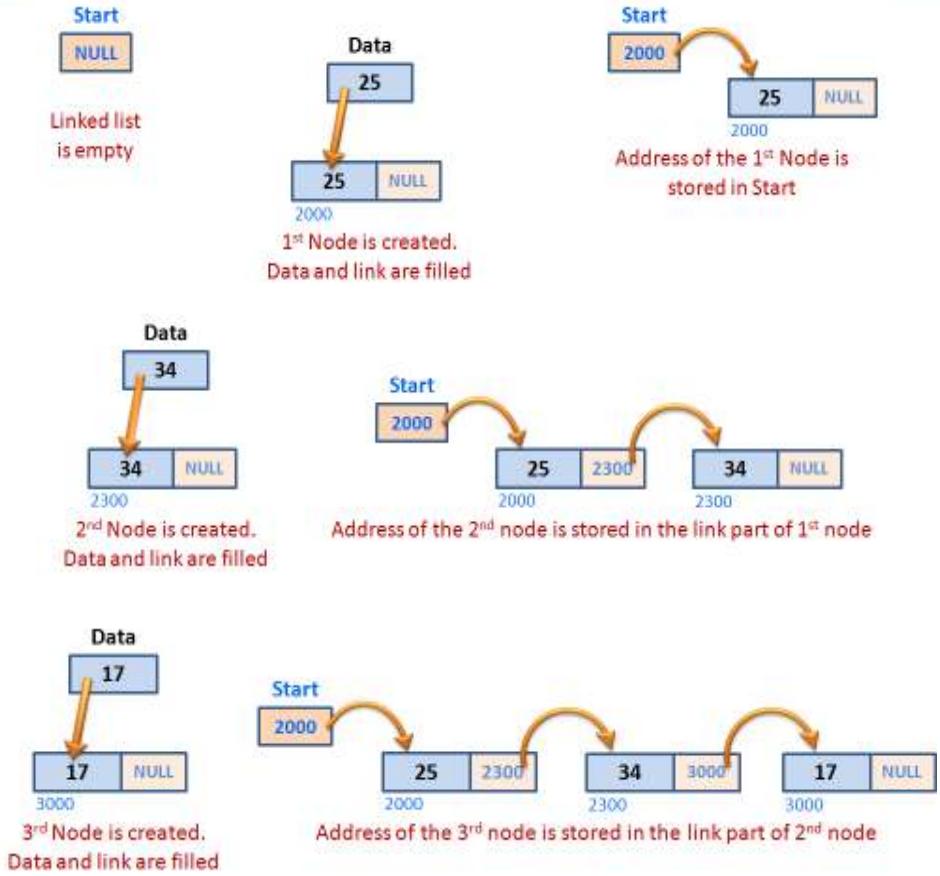


Fig. 3.16: Sequence of operations to create a linked list

Could you develop the following steps?

Step 1: Create a node and obtain its address.
Step 2: Store data and NULL in the node.
Step 3: If it is the first node, store its address in Start.
Step 4: If it is not the first node, store its address in the link part of the previous node.
Step 5: Repeat the steps 1 to 4 as long as the user wants.

Actually the creation of linked list can be viewed as repeated insertion operations at the end of a linked list. Insertion of the second node onwards requires a traversal operation in the list to get the address of the last node in the current list. So let us discuss the traversal operation.

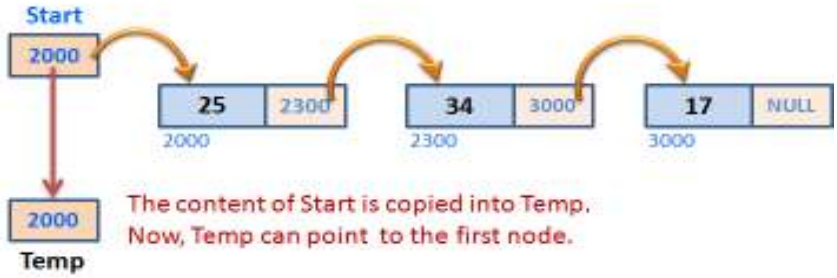
b. Traversing a linked list

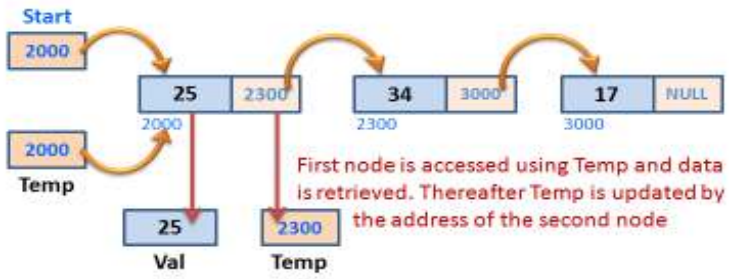
We know that traversal means visiting all the elements in a data structure. In the case of a linked list, we begin traversal from the first node. The pointer Start gives

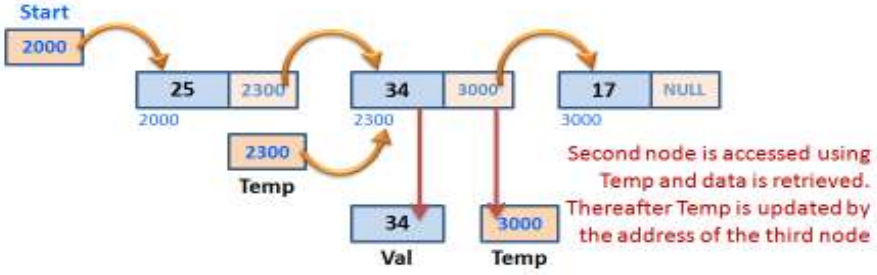
the address of the first node, so that we can access the data part using *arrow* (**–>**) *operator*. Then we access the link part of the first node, which is the address of the second node. Using this address we can access the data and link of the second node. This process is continued until we found NULL pointer in the link of a node.

Let us do

Let us observe Figure 3.17 and derive the steps required for traversal operation in a linked list. It is assumed that Temp is a pointer of Node type and Val is a variable to store the data read from a node.







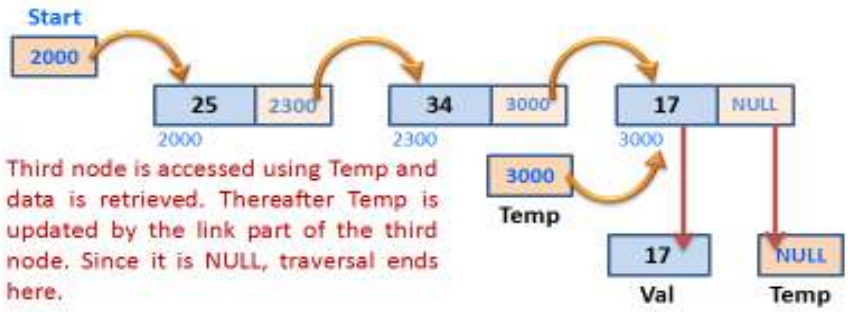


Fig. 3.17: Traversal operation in a linked list

Check whether you could derive the following steps:

Step 1: Get the address of the first node from Start and store it in Temp.

Step 2: Using the address in Temp, get the data of the first node and store in Val.

Step 3: Also get the content of the link part of this node (i.e., the address of the next node) and store it in Temp.

Step 4: If the content of Temp is not NULL, go to step 2; otherwise stop.

The above steps are required in the creation of a linked list in case Start is not NULL. In such a case, we have to find the address of the last node for storing the address of the new node in its link. The traversal operation begins from the first node by procuring its address from Start. This address will be copied into a temporary pointer variable (Temp in the figure) and it will be updated by copying the content of the link of the node pointed to by Temp. The visit will be continued until the link of a node pointed by Temp shows NULL value.

c. Insertion in a linked list

Insertion of an item in a linked list is the process of placing the node containing the item in a particular position. As in the case of arrays, a node can be inserted anywhere in a linked list - at the beginning, at the end or in between any two nodes. Once a new node is created, it can be inserted at the beginning by copying the content of Start into the link part of the new node and the address of the new node into Start. Similarly to insert a node at the end, we have to copy the address of the new node into the link part of the last node. Let us try to inert a node at a specified position.

Let us do

Figure 3.18 shows a series of operations to be performed to insert a node at the 3rd position in a linked list that has three nodes initially. Observe the figure and let us derive the steps required for insertion operation in a linked list. It is assumed that Temp, PreNode and PostNode are pointers of Node type structure. Let POS be a variable that contains the position value where the node is to be inserted.

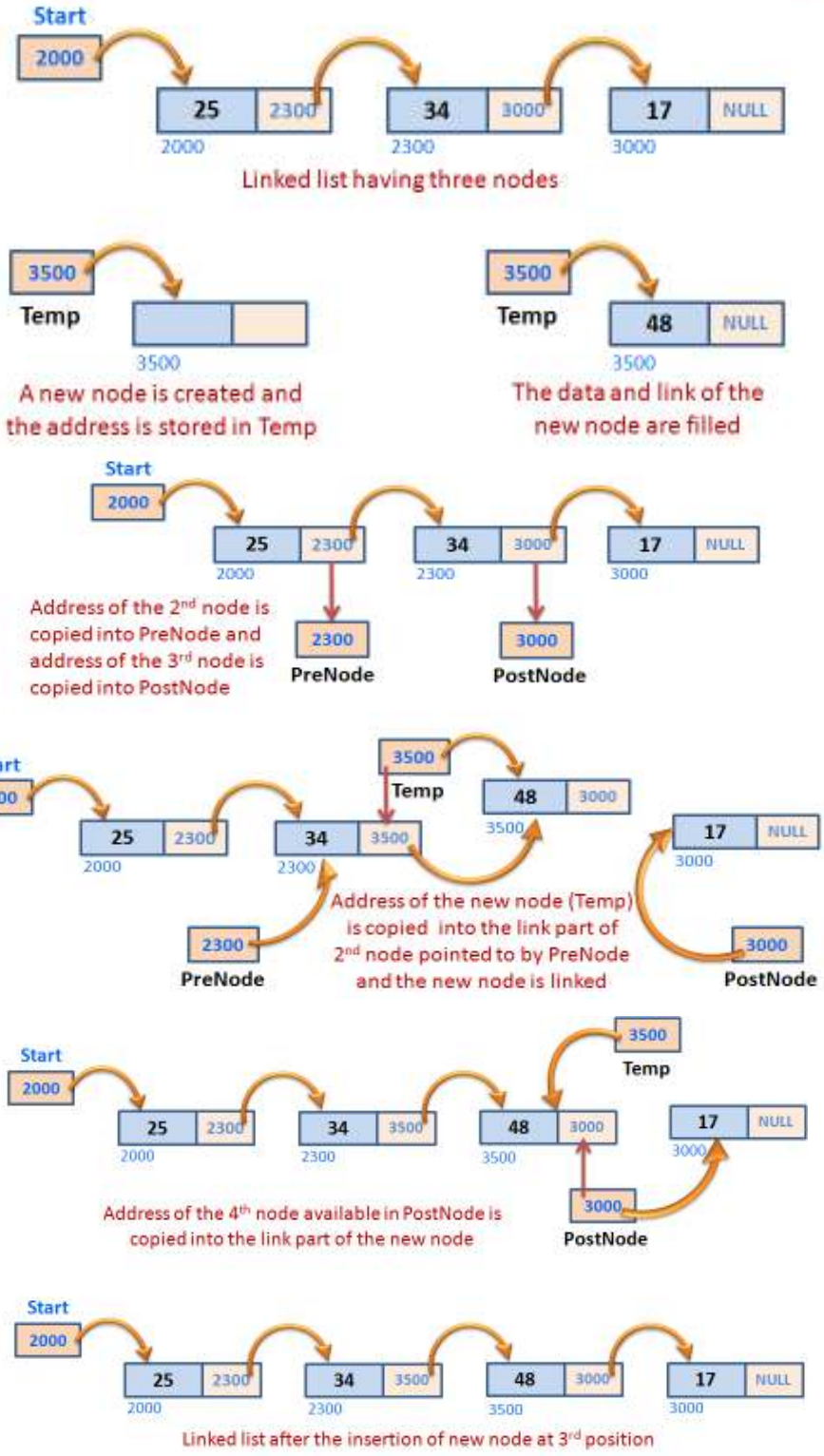


Fig. 3.18: Insertion operation in a linked list

The following steps can be developed for the insertion operation:

Step 1: Create a node and store its address in Temp.

Step 2: Store the data and link part of this node using Temp.

Step 3: Obtain the addresses of the nodes at positions (POS-1) and (POS+1) in the pointers PreNode and PostNode respectively, with the help of a traversal operation.

Step 4: Copy the content of Temp (address of the new node) into the link part of node at position (POS-1), which can be accessed using PreNode.

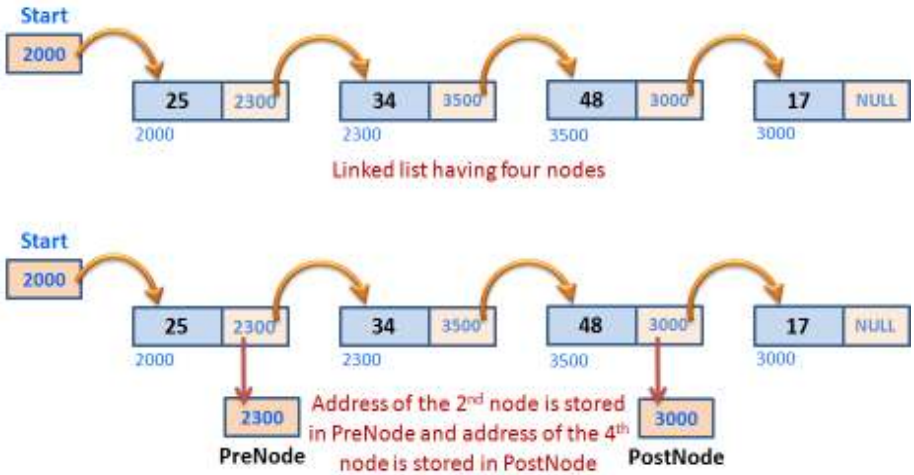
Step 5: Copy the content of PostNode (address of the node at position POS+1) into the link part of the new node that is pointed to by Temp.

d. Deletion from a linked list

Deletion of an item from a linked list is the process of removing a node from the list. The position of the node to be removed will be given. Instead of this, if the data item is given, the node containing that item is to be searched and its position is to be noted. Then we apply the steps for removal operation. Any node can be removed from a linked list. To remove the first node, we have to copy the content in the link part of the first node into Start. The last node can be removed by assigning the NULL value to the link part of the second last node. Let us discuss the procedure to remove a node from a given position.

Let us do

Figure 3.19 illustrates the procedure for the removal of 3rd node from the linked list having four nodes initially. Observe the figure and try to develop the steps required for deletion operation. It is assumed that PreNode and PostNode are pointers of Node type structure. Let POS be a variable that contains the position of the node to be removed.



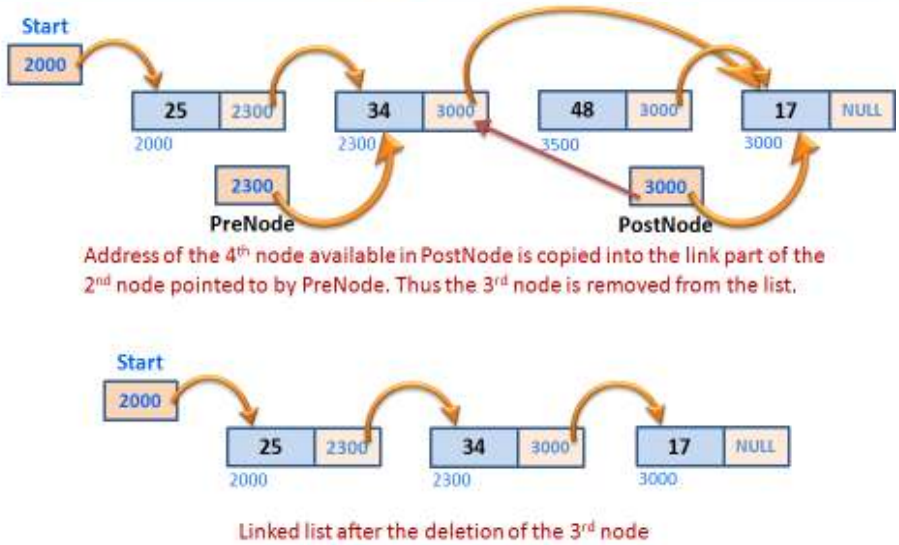


Fig. 3.19: Deletion operation in a linked list

The figure may help you derive the following steps for the deletion operation:

Step 1: Obtain the addresses of the nodes at positions (POS-1) and (POS+1) in the pointers PreNode and PostNode respectively, with the help of a traversal operation.

Step 2: Copy the content of PostNode (address of the node at position POS+1) into the link part of the node at position (POS-1), which can be accessed using PreNode.

Step 3: Free the node at position POS.

If we apply the first two steps in the linked list illustrated in Figure 3.19, even after the delinking of the 3rd node from the 2nd one, the presence of the 3rd node will be still there in the memory, pointing to the 4th node. So it should be freed using memory de-allocation facility provided by the programming language.

While implementing operations in linked lists, the temporary pointers like TEMP, PreNode, PostNode, etc. should also be freed after the operations.

Know your progress

1. Name an example for dynamic data structure.
2. What is linked list?
3. A node of a linked list consists of _____ and _____.
4. Which is the facility of programming language used to define the node of a linked list?
5. What is the content of Start or Header in a linked list?

Stacks and queues can be implemented using linked list too, which results into dynamic stacks and queues. The linked lists we have discussed are singly linked list, in the sense that a node can point to the next node only. But there are doubly linked lists, in which each node points to the next node as well as the previous node. It is made possible by including two pointers in the self referential structure. Complex data structures like tree are created using doubly linked lists.

Let us conclude

We are familiarised with the concept of data structures and the operations performed on them. There is a variety of data structures, so that any kind of data can be represented using a suitable one. The operations also vary depending on the oraganising principle followed in grouping the elements. Though some of the data structures are logical concepts, we have discussed their physical implementations. The difference between the array and linked list implementations has been mentioned during the discussion. The concepts covered in this chapter are the essential and important basics for your higher studies in Computer Science.

Let us assess

1. Read the following statements:
   (i) A collection of data processed as a single unit.
   (ii) All data structures are implemented using arrays.
   (iii) Stacks and queues are logical concepts and implemented using arrays and linked lists.
   (iv) Overflow occurs in the case of static data structures.

   Which of the above statements are true? Choose the correct options from the following:

   a. Statements (i) and (iii) only
   b. Statements (i), (ii) and (iii) only
   c. Statements (i), (iii) and (iv) only
   d. Statements (i), (ii) and (iv) only

2. Data structures may be static or dynamic.
   a. Give two examples for static data structures.
   b. Static data structures may face overflow situation. Why?
   c. Linked list is a dynamic data structure. Justify this statement.
3. Write an algorithm to insert an element into a stack.
4. What is meant by push and pop operations?

5. Write an algorithm to delete an element from a stack.
6. Write algorithms to perform insertion and deletion operations in linear queues.
7. How does circular queue overcome the limitation of linear queue?
8. Some of the operations performed on data structures are given below:

   i. Insertion ii. Deletion iii. Searching iv. Sorting

   a. Which of these operations may face underflow situation?

   b. Explain this situation in the context of an appropriate data structure.
9. Match the following:

A	B	C
a. Array	i. Start	1. Insertion and deletion at different ends
b. Stack	ii. Subscript	2. Insertion and deletion at the same end
c. Queue	iii. Rear	3. Self referential structure is utilized
d. Linked list	iv. Top	4. Elements are accessed by specifying its position

10. Explain why linked lists do not face overflow situation as in the case of array based data structures.

4 Web Technology

Significant Learning Outcomes

After the completion of this chapter, the learner

- explains how secure communication is brought about on the web.
- describes the use of a web server and the concept of web hosting.
- differentiates static and dynamic pages.
- identifies the differences between programming languages and scripts.
- compares the different types of scripting languages.
- explains the need for cascading style sheet.
- identifies the basic HTML elements to create web pages.
- lists fundamental HTML tags and their important attributes.
- classifies the HTML tags.
- uses formatting tags appropriately in an HTML document to make the web page attractive.
- identifies the similarities and differences among the formatting tags.
- observes the use of the tags <PRE> and <DIV>.
- chooses the tag for moving objects/ contents in a document.
- uses the <FONT> tag to format the text content effectively.
- utilises the facility of comments in an HTML document.
- inserts images into HTML document using <IMG> tag.

We are all living in an era of the Internet and might have accessed it for seeking information. You might have searched for your Class XI results and viewed it from a website. People rely on different websites for various purposes like submitting online applications, viewing web contents, watching movies, banking transactions, purchase of goods through online transaction, and so on. We know that a website is a collection of web pages. A web page may contain texts, graphics, sounds, animations, and movies. A website consisting of several web pages are designed to give information about an organisation, product, service, etc. How is this website made available on the Internet? These web pages are to be stored in web servers connected to the Internet, to be made available to others. This chapter presents an overview of communication over the Internet and the role of web servers in it. The different tools and technologies that are available for developing websites are introduced here. The concept of dynamic web pages and an overview of the scripting languages used to make web pages dynamic are also discussed. Web pages are developed with the help of a language called Hyper Text Markup Language

(HTML). HTML is also known as the language of the Internet. HTML tells the browser how to display the contents on a browser window. In this chapter, we will also familiarise ourselves with the fundamentals of creating web pages using HTML.

4.1 Communication on the web

We have learned the actions that occur when a web browser (client) tries to access a website from the Internet in Chapter 12 of Class XI. First, the URL (Uniform Resource Locator) is sent to a Domain Name System (DNS) server to obtain its corresponding IP (Internet Protocol) address and then the browser connects to this server using the IP address. The web server software installed in the server computer processes this request and the web page to be displayed is then sent to the client. The client browser formats and displays this web page.

In order to communicate on the web, computers/devices need to understand each other. This is made possible by making all devices follow the same protocol, namely the Transmission Control Protocol/Internet Protocol (TCP/IP). We have discussed TCP/IP protocol and its working in detail in Chapter 11, Computer Networks of Class XI. The data to be sent is broken down into small data packets along with the address of the recipient computer by the TCP protocol. The devices called routers, route and transport these data packets to their destination computers using Internet Protocol. Figure 4.1 shows the route of a data packet from a sender to a recipient.

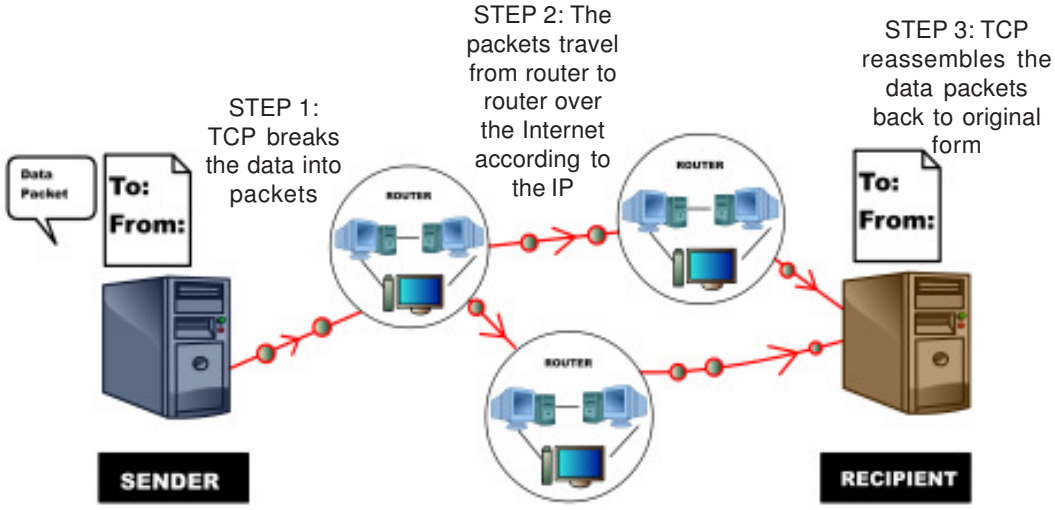


Fig. 4.1: Routing of a data packet from a sender to a recipient

In the Internet, there are several types of communication like, accessing websites, sending e-mails, etc. We have already learned in Chapter 12 of Class XI that websites are accessed using HTTP (Hyper Text Transfer Protocol) and e-mail communication

happens using SMTP (Simple Mail Transfer Protocol). Both these protocols work on top of lower level protocol called Internet Protocol. Internet Protocol provides the basics of communication over Internet. The use of a single protocol - Internet Protocol - for all communication over the Internet has several advantages. The routers need not be programmed separately to handle different types of data. They do not need to know about the data they are transporting and are concerned only about the address to which the packet is to be delivered. This openness of the data part of the packet gives the freedom to design new protocols for the Internet. It is this openness and flexibility of TCP/IP that led to the development of protocols for social media websites to handle messages, content websites to handle video and audio, banking websites to handle money transfer securely, etc. This turned out to be a deciding factor for the economic success of the Internet.

Communication on the web can be categorised as (i) client (browser) to web server and (ii) web server to web server communication. Authentication and security are essential for communication over the Internet. Authentication on the web is the process of determining whether a computer/server is the computer that it claims to be. Security should be provided to communication over Internet so that the messages are not intercepted and modified by hackers.

4.1.1 Client to web server communication

Client to web server communication does not usually require authentication. But in the case of web based banking applications/e-mail services, user names and passwords are required to be sent to the server. This information cannot be sent to the server in plain text due to security reasons. The hackers may steal usernames and passwords, if it is communicated and shared as plain text. In such cases we use HTTPS (Hyper Text Transfer Protocol Secure) technology to encrypt the username and password, and then send it to the server. HTTPS works using Secure Sockets Layer (SSL) which provides a standard security technology for establishing an encrypted connection between computers on Internet. SSL provides security capabilities to HTTP. The SSL protocol not only ensures privacy, but also ensures that no other website can impersonate the user's login account nor alter the information sent.

When the browser requests for a secure web page, the server first returns its SSL certificate. The browser then verifies whether this certificate is valid by checking it with the corresponding certification authority. A certification authority certifies whether an SSL certificate given to it is a valid one. This is an assurance that the particular website is of the organisation it claims to be. Verisign Inc. is a certification authority. If an SSL certificate is valid, the browser starts an encrypted session.

During this session, the browser and the server encrypts all the transmitted data. This process is displayed in Figure 4.2(a). In India, Information Technology Act makes it mandatory for websites that provide banking transactions to use HTTPS to accept confidential information from clients. You can click on the lock symbol in the address bar and view the certificate as shown in Figure 4.2(b).

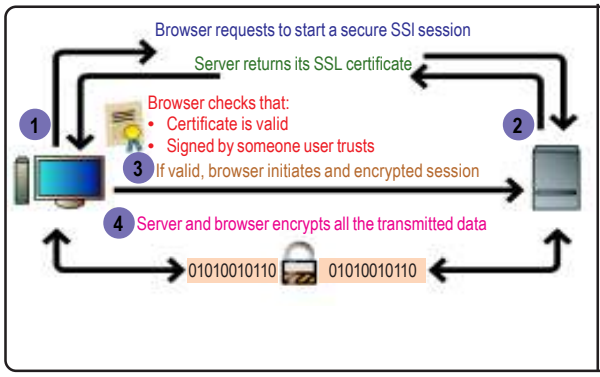


Fig. 4.2(a) : Client - server authentication process

Fig. 4.2(b): https authentication of SBI website

4.1.2 Web server to web server communication

Web server to web server communication also may be required in certain web applications. For example, the web server of an online shopping website (seller/ merchant on Internet) needs to send confidential information to a bank web server and vice versa. In such cases the web servers of the merchant and the bank are to be authenticated. Digital certificates help to verify whether the data received is from

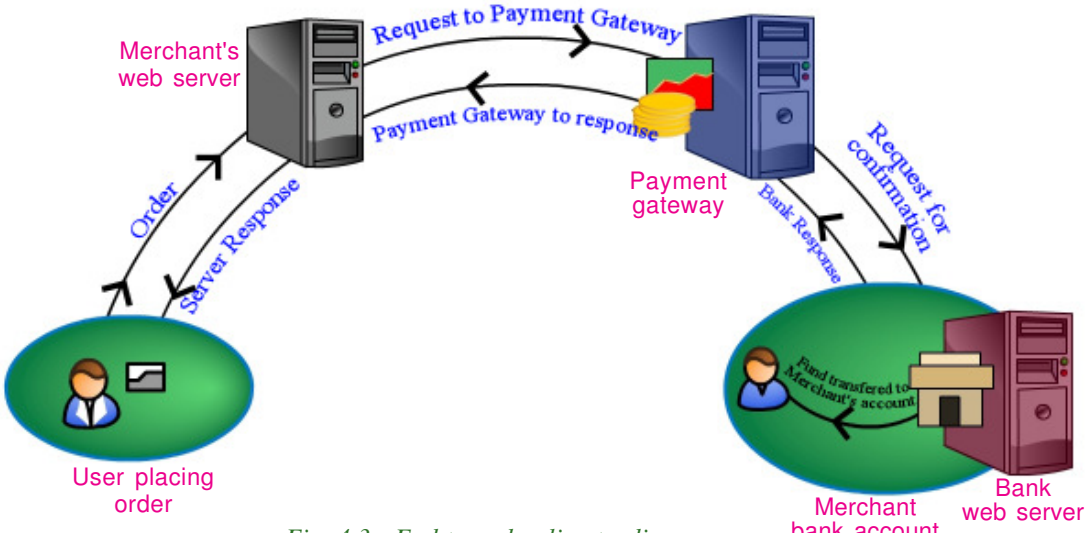


Fig. 4.3 : End to end online trading process

the actual server itself. Once the servers are authenticated, the servers communicate using encrypted data. Payment gateway is a server that acts as a bridge between merchant server and bank server and transfers money in an encrypted format whenever an online payment/money transfer is made. This process is shown in Figure 4.3.

4.2 Web server technologies

Let us see what happens internally when we visit the website www.dhsekerala.gov.in, the official website of the Directorate of Higher Secondary Education (DHSE). At first, the home page (the first page displayed when a website will be accessed) of the website will be transferred to our computer (client) from the web server of DHSE. The client is usually a computer or a mobile device which has a browser software that requests for web pages. The web server uses the web server software to store the pages of the websites and delivers it on request from the client. In the following sections, we will discuss the features of a web server and the requirements for setting up a web server.

4.2.1 Web server

The term web server is often used for referring to the server computer that hosts websites. It is also used to refer to a web server software that is installed in a server computer in order to make it a web server. In this section we will discuss the features of a web server computer and a web server software.

A web server enables us to deliver web pages or services like e-mail, blog, etc. to users on the Internet. A web server consists of a server computer that runs a server operating system and a web server software installed on it for providing services like www, e-mail, etc. over the Internet.

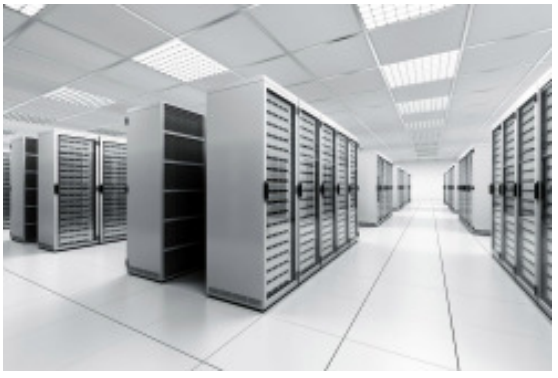


Fig. 4.4 : A data center

A web server is a powerful computer which is always switched on and connected to a high bandwidth Internet connection. This will facilitate Internet users around the world to access the websites and services hosted by it at any point of time. Depending on the requirements, a web server can have single or multiple processors, fast access RAM, high performance hard disks, Ethernet cards that supports fast communication, etc. To ensure faster Internet connectivity and redundant power supply, a web server is usually installed in a data center.

A data center is a dedicated physical location where organisations house their servers and networking systems. Data centers are used for storing, processing and serving large amounts of mission-critical data to their clients. A data center requires extensive backup power supply systems, cooling systems, high speed networking connections and security systems. A typical data center facility is shown in Figure 4.4. In a data center several servers may be mounted into special racks that offer good ventilation and easy maintenance as shown in Figure 4.5.

Fig. 4.5 : Rack mounted servers

Popular server operating systems include various Linux distributions (Redhat, openSUSE, Debian, Ubuntu, etc.), Microsoft Windows Server, FreeBSD, Oracle Solaris, etc.

After setting up a server, a web server software has to be installed in it and configured according to the given operating system. The web server software is a program that uses the client-server model and the Hypertext Transfer Protocol (HTTP) to ensure timely distribution of files stored in it to the users. These files are sent as webpages to the user on request using the web server software and can be accessed using a web browser. Some of the preferred web server packages are Apache Server, Microsoft Internet Information Server (IIS), Google Web Server (GWS) and nginx (pronounced as engine-x).

After the installation and configuration of a web server software, software packages like FTP (File Transfer Protocol), e-mail, DNS, database, etc. can be added as well. The main features provided by the web server software are the provisions to create multiple websites, configure website security, etc.

4.2.2 Software ports

We have discussed hardware ports in Chapter 3, Components of the Computer System of Class XI. Hardware ports are used to connect external devices to the computer. These devices communicate with the computer using these ports, i.e., VGA ports are used to connect monitors, PS/2 ports for keyboard/mouse, etc. Similarly, a software port is used to connect a client computer to a server to access its services like HTTP, FTP, SMTP, etc. To distinguish the ports, the software ports are given unique numbers. The purpose of software ports is to identify different services like e-mail, file transfer, etc. running on a single server computer. Each service available on the server can be set and accessed using a different port number.

Port number is a 16-bit number which when added to a computer's IP address/ URL, can be used for communicating with a particular service available on that server. A service in a website can be accessed in the format given below:

http://google.co.in:80

Here http is the protocol, google.co.in is the domain name and 80 is the port number. Some well-known ports and the associated services are listed in Table 4.1.

Default Port No.	Service
20 & 21	File Transfer Protocol (FTP)
22	Secure Shell (SSH)
25	Simple Mail Transfer Protocol (SMTP)
53	Domain Name System (DNS) service
80	Hypertext Transfer Protocol (HTTP)
110	Post Office Protocol (POP3)
443	HTTP Secure (HTTPS)

Table 4.1: Ports and their services

4.2.3 DNS servers

In Chapter 11, Computer Networks of Class XI, we learned about Domain Name System (DNS). A DNS server runs a special purpose networking software that contains a database of domain names and their IP addresses. The Domain Name System (DNS) runs on a DNS server and returns the IP address of a domain name requested by the client computer.

A Domain Name System contains several DNS server computers. The DNS servers are arranged in a hierarchy. At the top level there are 13 root servers that contain name server information for all the generic top-level domains such as .com and .org as well as country-specific domain addresses such as .in or .uk. Several copies of these root servers are placed in different locations around the globe as shown in Figure 4.6. All other DNS servers are installed in the lower level of hierarchy.

Fig. 4.6 : Root servers around the globe

Let us see how the DNS searches and locates the IP address of a domain name. Suppose we are visiting the website of the Police department of the Government of Kerala. Let us type the domain name for the police department, namely www.keralapolice.org in our browser. The following steps illustrate how the DNS resolves the IP address. See Figure 4.7.

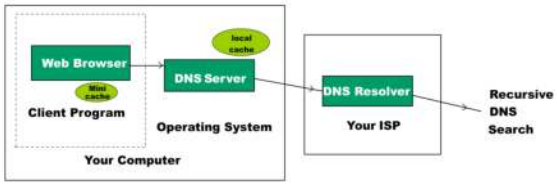


Fig. 4.7 : DNS search

1. All browsers store the IP addresses of the recently visited websites in its cache. Therefore, the browser first searches its local memory (mini cache) to see whether it has the IP address of this domain. If found, the browser uses it.
2. If it is not found in the browser cache, it checks the operating system's local cache for the IP address.
3. If it is not found there, it searches the DNS server of the local ISP.
4. In the absence of the domain name in the ISP's DNS server, the ISP's DNS server initiates a recursive search starting from the root server till it receives the IP address.
5. The ISP's DNS server returns this IP address to the browser.
6. The browser connects to the web server using the IP address of www.keralapolice.org and the webpage is displayed in the browser window. If the IP address is not found, it returns the message 'Server not found' in the browser window.

The 13 root servers around the globe manage the entire database of domain names and their corresponding IP addresses. Each of these root servers are a network of servers installed in many countries around the world. They are named as A, B, C, D, E, F, G, H, I, J, K, L and M. These servers are maintained by various agencies like ICANN, NASA (National Aeronautics and Space Administration), University of Maryland, VeriSign Inc., etc. ICANN's (Internet Corporation for Assigned Names and Numbers) Root Server System Advisory Council is comprised of the organisations that manage the root servers. They are responsible for advising ICANN on matters relating to the operation, administration, security and integrity of the Internet's Root Server System. In India, NIXI (National Internet Exchange of India) has sponsored three root servers at Mumbai (I Root), Delhi (K Root) and Chennai (F Root).

In large organisations like educational institutions, government departments, software firms, etc. where there are hundreds of computers or devices connected to the Internet, a local DNS server is hosted inside the intranet of the organisation.

This local DNS server contains a list of domain names and their IP addresses which the users in the organisation regularly access. This list can be updated periodically to include new domain names. Whenever a computer in the intranet tries to access a website in the Internet, it first searches for the domain name in the local DNS server and finds the IP address. This speeds up the Internet access in the organisation. Only if the domain name is not found in the local DNS server, it searches the DNS of ISP.

Instead of directing the computer to search for IP address in the ISP's DNS server, we can direct it to search the public DNS operated by Google. Google Public DNS is a free Domain Name System (DNS) resolution service that can be used as an alternative to our current DNS provider. The IP addresses for Google's public DNS are 8.8.8.8 and 8.8.4.4. We can configure our network settings to direct to Google's public DNS using either of the IP addresses.

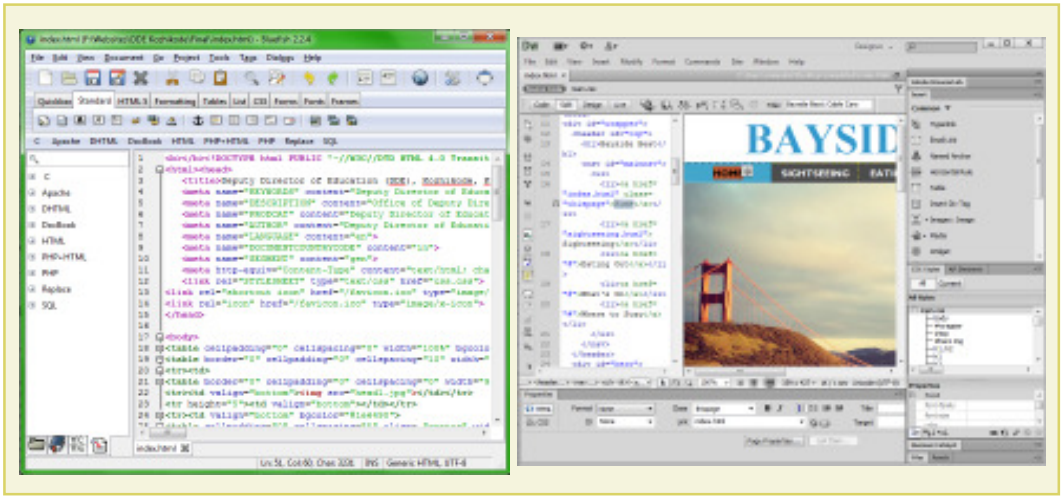
Know your progress

1. Name a protocol that works on top of the Internet Protocol (IP).
2. Expand HTTPS.
3. What are the advantages of installing web servers in data centers?
4. State whether true or false.
   a. The web server software works based on a client-server model.
   b. The web server consists of server operating system and web server software.
5. The number of bits in a software port number is _____.
   a. 8 b. 16 c. 32 d. 64
6. A Domain Name System returns the _____ of a domain name.

4.3 Web designing

The first step in setting up a website is planning the web pages for the website. Suppose we are going to develop a website for our school. After deciding the pages and the links, we need to design these web pages. Any text editor can be used to create a home page, a page for displaying the courses in the school, facilities available, contact address, etc. and link them using a menu to form an attractive website.

There are many web designing softwares available. You can also employ the features available in any of the web designing tools that helps you to create a web page with the ease of preparing a document in a word processor. These software also provide features to design web pages and link them together to form a website. Facilities to transfer files to the server using FTP protocol is also built into such software. Popular web designing softwares are Bluefish, Bootstrap, Adobe Dreamweaver, Microsoft Expression Web, etc. Figure 4.8 shows the Integrated Development Environment (IDE) of popular web designing softwares.



Bluefish *Dreamweaver*

Fig. 4.8 : IDE of web designing software

You have already learned the basics of creating web pages in high school classes. HTML consists of tags and its attributes used to format web pages. In this chapter we will also get familiar with the different HTML tags and attributes.

4.4 Static and dynamic web pages

You might have noticed that the web pages in the website of some small business/ organisations, school website, etc. remain the same whenever you visit it. These websites are called static websites. There are some other websites like the website that displays your mark in SSLC or Higher Secondary Examination (HSE), that changes when different register numbers are given to them. They are called dynamic web pages.

Static web pages are web pages that remain the same all the time until their code is changed manually. Initially, web pages were created with HTML only and such web pages are static web pages. Later, the emergence of scripting languages like JavaScript, VBScript, etc. brought animations into the web page. The colour and style of a portion of the web page changes when the mouse is kept over it, images are loaded

and displayed one after another in a sequence - all these are designed using scripting languages. The web pages that contain all these features are also classified as static web pages.

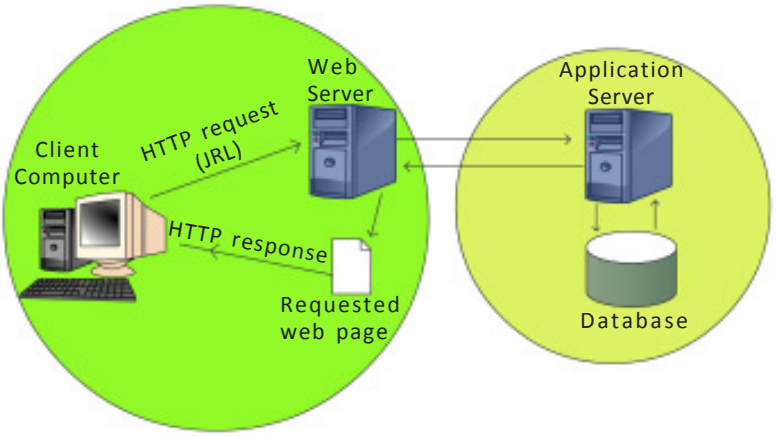


Fig. 4.9 : Static and dynamic web page requests

The web pages that contain server side code which creates a web page each time it is accessed are called dynamic web pages. Program code that runs on the server is called server side code. Dynamic web pages are executed using a server side application program that is installed on a web server. The script in the web page is executed on the web server and the resulting HTML page is sent to the client browser. In most cases data is accessed from databases to create web pages. The web pages that display SSLC, HSE results, bus, train and flight booking, banking/financial transactions, etc. display dynamic content, and are considered as dynamic web pages. Technologies like PHP, ASP, JSP, etc. are used to develop dynamic web pages. The working of static and dynamic web pages is represented using Figure 4.9. A comparison of static and dynamic web pages is given in Table 4.2.

Static web page	Dynamic web page
The content and layout of a web page is fixed.	The content and layout may change during run time.
Static web pages never use databases.	Database is used to generate dynamic content through queries.
Static web pages directly run on the browser and do not require any server side application program.	Dynamic web page runs on the server side application program and displays the results.
Static web pages are easy to develop.	Dynamic web page development requires programming skills.

Table 4.2: Comparison of static and dynamic web pages

4.5 Scripts

Scripts are program codes written inside HTML pages. They are written using a text editor like notepad. Scripting languages like JavaScript, VB script, PHP, Perl, etc. are used to create dynamic web pages.

Traditional programming languages are set of instructions carried out by the computer hardware with the help of an operating system, whereas scripting languages are interpreted by a web browser or by a web server software. Today most of the standalone programs are being replaced by web based programs. Earlier, the software used in banks were installed in the branches of the bank itself. Almost all banks have their banking software available on the bank's web server and the bank employees access them using the Internet. Web based software like banking software, higher secondary admission software, etc. use scripting languages for their development.

In an HTML page, a script is written inside **<SCRIPT>** and **</SCRIPT>** tags. `<SCRIPT>` is used to embed or refer to an executable script within an HTML document. It has the attributes **`Type`** and **`Src`**. `Type` attribute is used to identify the scripting language code embedded within the script tag. We will see more about HTML tags and attributes later in this chapter.

`<SCRIPT Type="text/javascript">` is used to insert JavaScript code in the HTML code.

Scripts can be written as an external file and can be linked to an HTML file. `Src` attribute is used to specify the URL of an external file containing scripting code to be linked to a web page.

`<SCRIPT Type="text/javascript" Src="scriptcode.js">` is used to insert JavaScript code inside the file `scriptcode.js` into an HTML file.

4.5.1 Types of scripting languages

Let us consider a login page of a website in the Internet. Usually the page will prompt the user to enter the user name and password. User will click the 'Login' button after entering the user name and password. What will happen if the user clicks the 'Login' button without entering the user name? Naturally, the computer will tell the user that he/she has not entered the user name. Remember that the web page we are viewing is controlled by two computers - the client computer where the web page is viewed and the server computer where the web page comes from.

Where will we check whether the user has entered a user name or not - in the client computer or in the server computer? The server computer is thousand times busier

than the client computer, because a large number of people may be visiting the same web site and the server is the only one computer to handle all those requests. Therefore, all the tasks that can be done at the client side must be done at the client side itself. This will reduce the workload of the server.

It should also be noted that, if this type of checking is done at the server, when the user clicks the 'Login' button, the entered data has to be sent to the server from the client through the Internet. The data has to travel a long distance through the Internet to reach the server. When the data reaches the server, it will be placed in a queue because a large number of clients may be sending the same request to the server. The sent data will wait in the queue till it gets the chance to be processed by the server. The server will check whether any user name or password is entered by the client or not. If not, it will send a message back to the client specifying that the user name is not given as displayed in Figure 4.10. Again, the message has to travel a long distance back from the server to the client through the Internet. In short, if the user clicks the submit button, without entering the user name, he/she has to wait a few seconds till he/she gets the message that he/she has not given the user name. Besides this, the data has to travel from client to server and back from server to client, which unnecessarily makes network traffic busy.

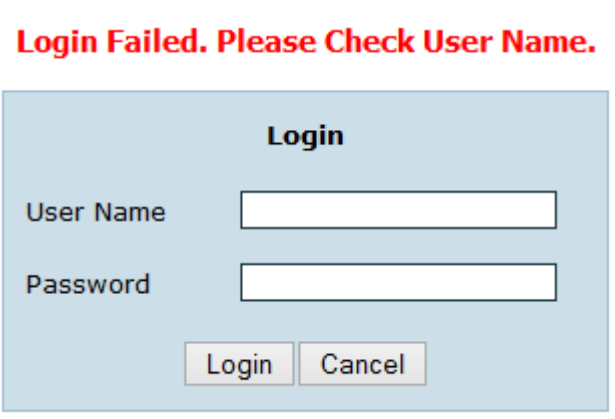


Fig. 4.10 : Response from the server

If this checking is done at the client side itself, when the user clicks the submit button, the script code in the client side can check whether the user has given some entry as the user name and password. If not, a message can be displayed. During this process, the data does not travel across the Internet to the server, nor does it disturb the server for this simple task. When the user clicks the submit button, the user gets the message that he has not given the user name, within a second as shown in Figure 4.11. It also does not engage network resources unnecessarily.

Now let us consider another situation. Suppose a user enters a wrong user name and password. The client computer may be able to check whether there is any entry with such a user name and password. However it cannot check whether the user name and password are correct or not. This is because only the server computer has the details of all user names and corresponding passwords. Therefore, whether

the user name and password match each other, can be checked at the server side only. This is a situation where you have to use the server side scripting for the validation. When you need to perform some kind of validation of data that makes use of information from the server, it must be done at the server side itself.

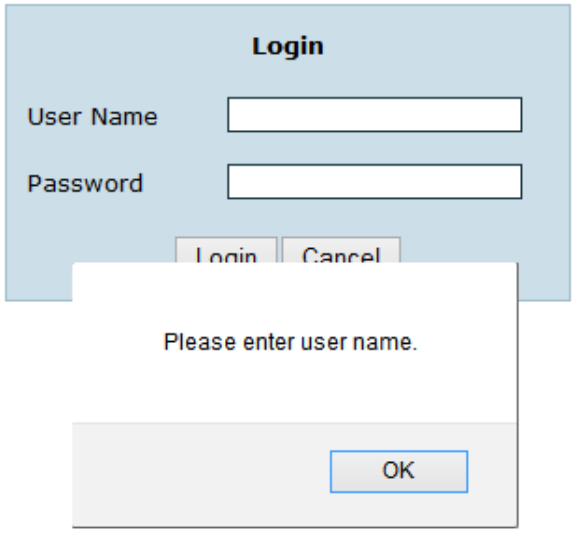


Fig. 4.11 : Response from client browser

Isn't it now clear that scripting languages are classified into client side scripts and server side scripts? Client side scripting is used to perform any task at the client side and is executed in the browser. Scripts that are executed in the server are called server side scripts. The output produced after execution of the server side scripts is in the form of an HTML page which is sent to the client.

A. Client side scripting

In client side scripting, the script code for validation is downloaded along with the HTML code to our browser. When we click the submit/save button this client side script is executed in our browser. If there is an error, it displays the message. It will send the data to the web server only if the validations are correct.

Since the script code is executed on the client browser, it provides a quicker response to users. This type of scripting will allow the client browser to share some of the burden on the web server while running a web application. The disadvantage of client side scripting is that there are browsers that do not support scripts. In some cases users may turn off the execution of scripts in the browser. In such cases client side scripting may not work.

Popular client side scripting technologies are JavaScript and VB script. Client side scripting is mostly used for validations and also for performing simple calculations at the client side before sending the data to the web server.

B. Server side scripting

We discussed dynamic web pages in the previous section. Server side scripts are used to create dynamic web pages. Let us discuss another example for server side scripting. Consider the website that displays the results of SSLC examination. When we enter the SSLC register number of a student, the website displays the scores of

that particular student. This is the same for each student who has appeared for the SSLC examination. We know that it is not practical to design a web page for each of the several lakhs of students who have written the SSLC examination. If so, how is this done? The results of these several lakhs of students are stored in a database in the web server. Server side scripts are used to access the result of a particular student from the database whose register number is entered by the user. The server side script then uses this result to create a simple HTML web page. This web page is then sent to the client browser and the browser displays it. This way the server side script creates a web page for each student who has appeared for SSLC examination as shown in Figure 4.12. The rapid growth of web based applications has increased the use of server side scripting.

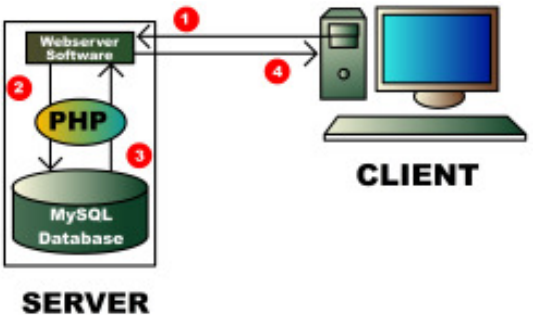


Fig. 4.12 : Working of server side scripts

Server side scripting is a technology in which the web page containing server side scripts requested by the user is executed in the server and the result, which is an HTML code, is sent to the client browser. Server side scripting creates web pages dynamically. Since the scripts are executed at the server, the type and version of the browser or operating system on the client's computer does not affect its execution. Since the scripts are executed on the server, it consumes server resources. Popular server side scripting languages are Perl, PHP, ASP, JSP, etc.

A comparison of the classifications of scripting languages is given in Table 4.3.

Client side scripting	Server side scripting
Script is copied to the client browser	Script remains in the web server
Script is executed in the client browser	Script is executed in the web server and the web page produced is returned to the client browser
Client side scripts are mainly used for validation of data at the client.	Server side scripts are usually used to connect to databases and return data from the web server
Users can block client side scripting	Server side scripting cannot be blocked by a user
The type and version of the web browser affects the working of a client side script	The features of the web browser does not affect the working of server side script

Table 4.3 : Comparison of client side and server side scripting

We have seen that the client side scripts are mainly used for validations at the user's browser and that it reduces the load on the server and network traffic. Therefore, the entire validation checking scripts are moved to the client side. Now the data sent to the web server is free from all errors and can be directly saved to the database. But if the client's browser does not support scripts or the user has turned off the scripting in the browser for security reasons, the data will be sent to the server without validation. This causes invalid data to be stored in the database. In order to protect the validity of the data saved in the database, it is better to have a validation check at the server side also.

4.5.2 Scripting languages

We have learned the different classifications of scripting languages. Let us now discuss the features of some of the popular scripting languages.

A. JavaScript

JavaScript is a client side scripting language used to make web pages interactive. JavaScript was developed by Brendan Eich (Figure 4.13) while he was working for Netscape Communications Corporation. On the client side, JavaScript is implemented as an interpreted language. Any text editor such as Geany IDE, Notepad, etc. can be used to write JavaScript code. It is popular as a client side scripting tool and works in almost every web browser. JavaScript can be inserted inside HTML code or can be written as an external file and then included inside HTML file. If a JavaScript code is written as an external file, it is common to use the extension .js. This identifies the file as a JavaScript file. JavaScript is popular as a tool for validation of forms in the client side. It can also be used for performing simple calculations and also for bringing animations to a web page. We will discuss JavaScript in detail in Chapter 6.

Fig. 4.13: Brendan Eich (1961 -)

The popularity of JavaScript has led to more developments in client side scripting. While applying online for Higher Secondary Plus One admissions, immediately after entering your SSLC register number, you might have noticed that your name, date of birth and other details appear in the text boxes below. This is done without reloading the entire web page. The data is taken from the server and filled in the text boxes without refreshing the web page. Ajax is the technology used here. Ajax improves the interactivity of the browsers. Ajax is Asynchronous JavaScript and Extensible Markup Language (XML). XML is a markup language which helps users to create new tags. After implementing Ajax on a website, it does not require the entire page to be reloaded for displaying dynamic content on web pages. Ajax

allows web pages to be updated by exchanging small amounts of data between the client and the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the entire web page. However, since Ajax relies more on JavaScript, if the browser is not capable of handling JavaScript or the user has turned off JavaScript functionality in the browser, the Ajax application will not work.

B. VB Script

VBScript is a scripting language developed by Microsoft Corporation based on the popular programming language Visual Basic. VBScript was developed to use either as a client side scripting language for the Microsoft Internet Explorer or as a server side scripting language with the Microsoft Internet Information Server (IIS). Unfortunately, browsers other than Internet Explorer may not be able to correctly interpret and display the VBScript code. Therefore, it is less popular as a client side scripting tool. Since Windows operating system is popular as a server based operating system, VBScript is popular for server side scripting. With the introduction of .NET framework - a library of usable program code, Microsoft has taken the decision to incorporate VBScript as a part of ASP.NET in .NET framework.

C. PHP

PHP stands for 'PHP: Hypertext Preprocessor'. PHP is an open source general-purpose scripting language that is suited for web development and can be embedded into HTML code. It is a server side scripting tool and its code is similar to Java, C and Perl. The main objective of PHP is to develop dynamic web pages at ease. PHP was originally created by Rasmus Lerdorf (Figure 4.14) in 1994 but it is now developed by The PHP Group. The web page files that contain PHP code have the extension .php.

Fig. 4.14: Rasmus Lerdorf (1968 -)

PHP code is inserted inside HTML code and when the user requests for a PHP web page, it is interpreted and executed on the web server. To process the PHP code on the web server, a PHP interpreter has to be installed on the web server. After execution of the PHP code in the web server, an HTML page is created, which is sent to the client browser. One of the strongest and most significant features in PHP is its support for database programming. The most common database used with PHP is MySQL. PHP interpreter is available for all operating systems. Linux platforms commonly use LAMP (Linux, Apache, MySQL and PHP) server software which is freely downloadable. LAMP uses Linux as the server operating system, Apache as the web server, MySQL as the database and PHP for server side scripting. Windows operating systems use WAMP server software which is also available for free download. More about PHP will be discussed in Chapter 10.

D. Active Server Pages

Microsoft Active Server Pages (ASP) is a server-side scripting environment that can be used to create and run interactive web applications. ASP contains HTML and a scripting language code. The scripting language can be VBScript or JavaScript. ASP files have the extension .asp. These files are compiled using a feature built in Microsoft's web server software, Internet Information Server (IIS). ASP files are executed only on Windows operating systems. After execution at the server, the resultant HTML web page is sent to the client browser. It is a very powerful scripting language that provides support to a variety of databases. The introduction of ASP.NET stopped Microsoft from releasing further versions of ASP. ASP.NET provides features like reduced coding, availability of a variety of buttons, text boxes etc. which help in creating web based applications.

E. Java Server Pages

Java Server Pages (JSP) technology provides a simple and fast way to create dynamic web content. It is a server side scripting language that was developed by Sun Microsystems in 1999. JSP is similar to PHP, but uses Java as programming language. JSP files have the extension .jsp. To run JSP, Apache Tomcat web server is required. The JSP code consisting of HTML and Java is executed on the web server and the resulting HTML code is sent to the browser. JSP is an integral part of Java 2 Platform Enterprise Edition (J2EE) which is used for developing and running large scale web based softwares.

4.6 Cascading Style Sheet

Cascading Style Sheets (CSS) is a style sheet language used for describing the formatting of a document written in HTML. Using CSS, we can control the colour of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, borders and its colours, what background images or colours are used, as well as a variety of other effects in a web page. A CSS file allows us to separate HTML content from its style. CSS can be implemented in three different ways - inline, embedded and linked.

- In inline style, the CSS style is applied to each tag separately using the style attribute in the body part of the web page.
- Embedded CSS codes are placed within the **<HEAD>** part of the web page.
- Linked CSS implementation is done using an external file with the file extension .css that contains only CSS code and is linked with the web page.

The advantage of CSS is that we can reuse the same code for all the pages. If the CSS styles for a website is implemented as a linked external file, then the modification

of a style, modifies the way the tags are presented in all the web pages of the website. Since CSS styles are written in a common place, it separates CSS and HTML, which makes it easy for maintenance. Moreover, the tags in web pages are well organised with the style specifications and therefore it is easy to understand. This also reduces the size of the web page thereby providing faster downloads for web pages.

CSS allows adapting the presentation of a web page to devices with different screen sizes such as desktop monitors, tablets or mobiles. It is considered that CSS along with JavaScript will be used in the next version of HTML called HTML5, to bring animations and interaction to web pages.

Know your progress

1. The web pages that remain the same until their code is changed manually are called _____.
2. Name two technologies that can be used to develop dynamic web pages.
3. The tag used to embed scripts is _____.
4. Write any one of the uses of client side scripting.
5. A JavaScript file has the extension _____.
6. What is the advantage of using Ajax?
7. Classify the following scripting languages into client side and server side.

   Javascript, PHP, ASP, VBScript
8. .asp files are compiled using the web server software _____.
9. List the different ways of implementing CSS.

4.7 Basic concepts of HTML documents

HTML is the most widely used language to write web pages. Every web page is actually an HTML file. Each HTML file is a plain text that defines a set of commands for creating hypertext documents. These commands are known as ***HTML tags***. While using these tags, some keywords may be attached to them, which make the instruction more specific. These words are known as attributes. Therefore, an ***HTML document*** is made up of tags and attributes which work together to decide how the contents of the web page have to be displayed on the browser. Actually, the study of HTML means the study of tags and their attributes. Before going into the details of tags and attributes, let us have a look at the basic structure of an HTML document.

4.7.1 Basic structure of an HTML document

The basic structure of an HTML document is shown in Example 4.1.

Example 4.1: A sample HTML document to illustrate the structure of a web page

```
<HTML>
    <HEAD>
      <TITLE> This is the title of web page </TITLE>
    </HEAD>
    <BODY>
      Hello, Welcome to the world of web pages!
    </BODY>
</HTML>
```

You can see some of the words in the upper case within a pair of angle brackets **<** and **>**. These are HTML tags. It is not necessary that tags be written in the upper case. HTML is not case sensitive. We can use either the upper or lower case or even a mix of the two. In this book, we follow the style of using the upper case for HTML tags and sentence case (i.e., the first letter is capital) for attributes to distinguish them from other words or text.

As shown in Example 4.1, all HTML pages begin with the tag **<HTML>** and end with tag **</HTML>**. There are mainly two sections in an HTML document namely head section and body section. The **<HEAD>** tag is used to define the head section. The head section contains the information about the document, including the title of the web page. The **<TITLE>** tag is used to define the title of the page, which will be displayed on the title bar of the browser window. The **<BODY>** tag is used to define the body section. The body section contains the contents to be displayed in the web page. If we open this document in a web browser, it will appear as shown in Figure 4.15.

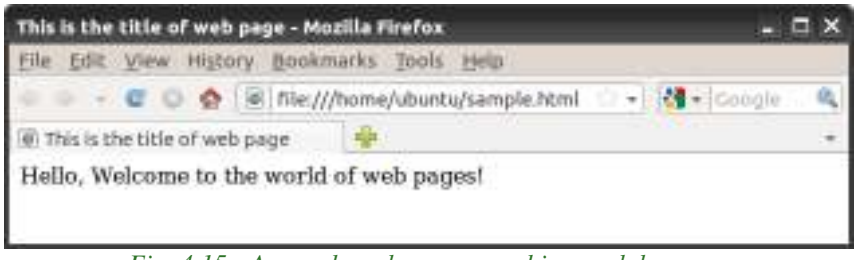


Fig. 4.15 : A sample web page opend in a web browser

Versions in HTML: HTML was created by Tim Berners Lee in late 1991 but "HTML 2.0" was the first standard HTML specification which was published in 1995. HTML 4.01 was a major version of HTML and it was published in late 1999. The latest version, HTML 5 was released in 2012. But it is still being modified for adding more multimedia integration.

4.7.2 Tags in HTML document

As mentioned earlier, tags are the commands used in the HTML document that tell web browsers how to format and organise our web pages to show the contents. Every tag consists of a tag name enclosed between the angle brackets '**<**' and '**>**'. HTML tags are not case sensitive. Therefore the tags `<HTML>`, `<html>`, `<Html>`, `<HtmL>`, etc. have the same meaning.

4.7.3 Container tags and empty tags

Most tags are used in pairs - an opening tag and a closing tag. For example `<HTML>` is the opening tag and `</HTML>` is the closing tag. Note that the closing tag has the same text as the opening tag, but has an additional forward slash (**/**) character after the first angle bracket. Tags that require opening tag as well as closing tag are known as container tags. Container tags are applicable for a section. The opening tag is given at the beginning of a section and closing tag is placed at the end of the section. For example, `<HTML>` and `</HTML>` forms the opening and closing tag pairs for an HTML document.

Some tags are given an exemption to this rule, and these tags do not require closing tag. Such tags are known as empty tags. This kind of tag does not span over a section. The tags `<BR>`, `<HR>`, `<IMG>`, etc. are examples of empty tags. We will see these tags in the forthcoming section of this chapter.

4.7.4 Attributes of tags

Certain parameters are frequently included within the opening tag to provide additional information such as colour, measurement, location, alignment or other appearances to the web browser. These parameters are called attributes. Most of the attributes require a value. In HTML, the value can be given in single quotes or double quotes (i.e., `attribute='value'` or `attribute="value"`). Each tag has a standard set of attributes and we can use them as per the requirement. If an attribute is used, normally it appears after the tag name separated by a space. If more than one attribute is used, their order of appearance is not important.

For example, to change the background colour of the web page to yellow, we can write the code as `<BODY Bgcolor = "Yellow">`. Here, Bgcolor is the attribute and Yellow is the value of this attribute. The other attributes of `<BODY>` tag and other tags will be discussed in the forthcoming section.

4.7.5 HTML Elements

A pair of tags and the content enclosed between these tags are known as an element. Figure 4.16 shows that the Body element contains the opening tag `<BODY>` with

attributes if any, the closing tag </BODY>, and the contents in between these two tags.

The basic structure of an HTML document contains four sets of HTML tags.

Fig. 4.16 : An HTML element

They are:

```
<HTML>    </HTML>
<HEAD>    </HEAD>
<TITLE>   </TITLE>
<BODY>    </BODY>
```

4.8 Creating an HTML document

Now, let us create a simple web page using the HTML code given in Example 4.1. Text editors like Geany, Gedit, TextPad, Notepad, Notepad++, etc. can be used to create HTML documents. The file is to be saved with a name with an extension **.html** or **.htm** (for example, sample.html). Figure 4.17 shows an HTML code in Geany editor, saved as **sample.html**.

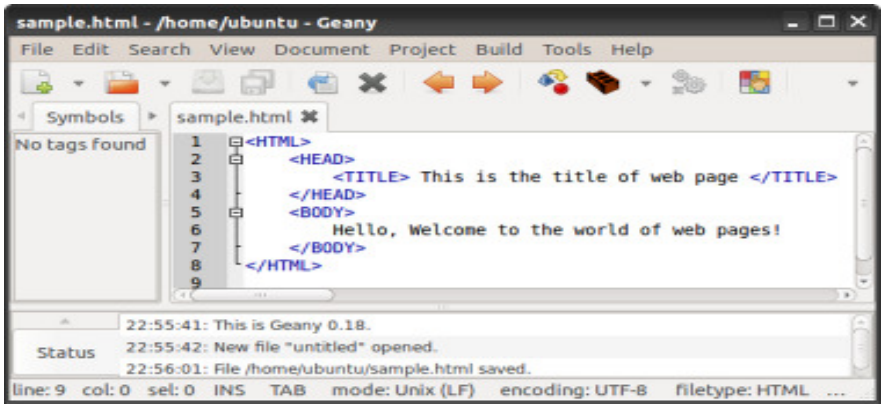


Fig. 4.17: HTML code in Geany Editor

Viewing an HTML document in a Browser

Once the HTML document is prepared, it can be viewed in a browser by opening the document with the browser. There are many browsers like Mozilla Firefox, Google Chrome, Internet Explorer, Netscape navigator, etc. Screen shots given in this book, are obtained by opening the web page using Mozilla Firefox. The output of the HTML code created using Geany can also be obtained by clicking the Execute button in the tool bar of Geany.

4.9 Essential HTML tags

Let us have a detailed discussion on the essential tags required to create web pages. The tags, their use, associated attributes and their values, and their appearance in the browser window will be illustrated in this section.

4.9.1 <HTML> - Starting an HTML page

The entire HTML document is bounded by a pair of <HTML> and </HTML> tags. The <HTML> tag identifies the document as an HTML document. In general <HTML> is always the first tag in an HTML page and the </HTML> is the last tag. Everything else in the web page is in between these two tags. That is, the Head section and the Body section lie inside the <HTML> and </HTML> tags. It is a container tag pair. The main attributes of the <HTML> tag are **Dir** and **Lang**.

Dir

The Dir attribute of <HTML> tag specifies the direction of the text to be displayed on the web page. This attribute can have values either **ltr** (left-to-right) or **rtl** (right-to-left). By default, the value of this attribute is ltr. The value rtl is used when languages like Arabic, Hebrew, Chinese etc. are used for content presentation. For example, <HTML Dir = "rtl"> specifies that the document is to be read from right-to-left.

Lang

The **Lang** attribute of <HTML> tag specifies the language we have generally used within the document. The value "en" is used for English language and "it" is used to specify Italian language.

Sl. No.	Code	Language
1	En	English
2	Fr	French
3	De	German
4	It	Italian
5	El	Greak
6	Es	Spanish
7	Ar	Arabic
8	Ja	Japanese
9	Hi	Hindi
10	Ru	Russian

Table 4.3 : Some common language codes

For example, the code <HTML Lang = "ar"> specifies that the language used in the HTML document is Arabic language. Some common language codes used with Lang attribute are given in Table 4.3.

4.9.2 <HEAD> - Creating head

It contains the head of an HTML document, which holds information about the document such as its title, scripts used, style definitions, etc. The tag pair <HEAD> and </HEAD> declares the head section. It is also a container tag pair.

4.9.3 <TITLE> - Creating a title

It is a container tag pair that contains the title of the HTML document, which will appear in the web browser's title bar. The search engine uses the Title to identify the page. The tag pair <TITLE> and </TITLE> is used inside the tag pair <HEAD> and </HEAD> to mention the document title.

4.9.4 <BODY> - Creating a body

The body tag pair <BODY> and </BODY> specifies the document body section. This section contains the content to be displayed in the browser window. Hence, all other tags, which define the document content are given in the body section. Before discussing these tags, let us discuss various attributes of <BODY> tag.

Background

This attribute sets an image as background for the documents body. This attribute of <BODY> tag makes the page more attractive. The general format is:

```
<BODY Background = "URL of the picture">
```

The HTML code given in Example 4.2 shows the sky as the background image of a web page.

Example 4.2: To set an image as background for a web page

```
<HTML>
<HEAD>
   <TITLE> Background Image </TITLE>
</HEAD >
<BODY Background = "Sky.jpg">
   Hello, Welcome to the world of Web Pages!........
</BODY>
</HTML>
```

Here, the HTML code in Example 4.1 is modified by prioviding an attribute Background with the value "Sky.jpg" in the <BODY> tag as <BODY Background = "Sky.jpg">. Before opening the page, we have to place the image file in the current working directory. The web page is displayed as shown in Figure 4.18.

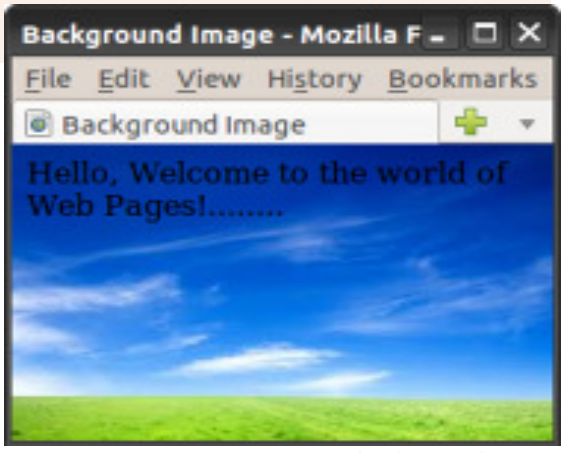


Fig. 4.18 : An image as background

Bgcolor

This attribute specifies a colour for the background of the document body. For example, `<BODY Bgcolor = "grey">` will display the background in grey colour.

The value of `Bgcolor` attribute can be given in two ways.

- ***Color_name*** - specifies the background colour with a colour name (like "red", "grey" etc.)
- ***Hex_number*** - specifies the background colour with a hexadecimal code (like "#ff6080", "#303030" etc.). Each hexadecimal code will be preceded with a hash sign #.

  The six digit number and letter combinations represent colours by giving their RGB (Red, Green, Blue) value. Of the six digits, the first two digits represent the amount of red, the second two digits represent the amount of green, and the last two digits represent the amount of blue as a hexadecimal value in the range 00 - FF. For example, #000000 is black, #FF0000 is bright red, #00FF00 is bright green, and #FFFFFF is white (fully saturated with all the three colours). We can try various colour combinations according to our choice of hex number. Table 4.4 shows a few colours with their Names and Hex values.

Colour	Colour Name	Colour HEX
	Black	#000000
	Red	#FF0000
	Green	#00FF00
	Blue	#0000FF
	Yellow	#FFFF00
	Aqua	#00FFFF
	Grey	#C0C0C0
	White	#FFFFFF

Table 4.4: List of colours with their Name and Hexadecimal value

Text

This attribute specifies the colour of the text content in the page. By default the browser displays the text in black colour on a white/grey background. We have already discussed how to change the background colour using `Bgcolor` attribute. Simillarly the colour of the text can be changed using the attribute `Text`. For example, `<BODY Text = "yellow">` shows the text in yellow colour. Like `Bgcolor`, the value of `Text` attribute can be given as colour name or hexadecimal code. For example, `Text = "Blue"` or `Text = "#00FFDD"` etc.

Link, Alink and Vlink

A hyperlink is an element, a text or an image that we can click on, and jump into another document or another section of the same document. A hyperlink points to a whole document or to a specific element within a document. We will discuss hyperlink in detail later in this chapter.

Link: This attribute specifies the colour of the hyperlinks that are not visited by the viewer. The default colour for `Link` attribute is blue.

Alink: It specifies the colour of the active hyperlink. The link remains active only for the moment the mouse is clicked on it. Hence at the time of selection the colour of the link will be changed to `Alink` value. The default `Alink` colour is green.

Vlink: It specifies the colour of the hyperlink which is already visited by the viewer. The default colour for `Vlink` is purple.

Leftmargin and Topmargin

The margin refers to the blank area left from the edge of the page. `Leftmargin` attribute is used to leave some blank area on the left side of the document and `Topmargin` refers to the blank area at the top edge of the document window. The value is specified in pixels.

For example, `<BODY Leftmargin = "60" Topmargin = "70">` will make the body text indent 60 pixels away from left edge of the page and 70 pixels away from the top edge of the page.

The code in Example 4.3 is an illustration to the attributes `Bgcolor`, `Text`, `Topmargin`, and `Leftmargin` of `<BODY>` tag. Figure 4.19 shows the corresponding web page.

Fig. 4.19: Use of attributes with BODY tag

Example 4.3: To set a colour in the background of a web page

```
<HTML>
<HEAD>
     <TITLE> This is the title of web page </TITLE>
</HEAD>
<BODY Bgcolor= "cyan" Text= " Blue"
```

```
      Topmargin= "70" Leftmargin= "60">
      Hello, Welcome to the world of Web Pages!........
</BODY>
</HTML>
```

Know your progress

1. HTML stands for _____.
2. What is a container tag?
3. The type of tag that requires only a starting tag but not an ending tag is called _____.
4. State true or false
   a. Tags are case sensitive.
   b. `Bgcolor` is an attribute of `<BODY>` tag.
   c. `<TITLE>` tag is an empty tag.
   d. `Dir` is an attribute of `<HEAD>` tag.
5. Name the attributes of `<HTML>` tag.
6. What is the use of attributes in a tag?
7. List the different attributes of `<BODY>` tag.

Let us do

Create an HTML document to display the name of your school with an image of the school as background of the web page. Then modify the page by changing the colour of the text and background.

4.10 Some common tags

We have discussed the basic tags and their attributes needed for an HTML document. There are several other tags, with which web page contents can be made more attractive. Some of these tags are used for formatting the text contents in the body section of the HTML document and therefore they are called formatting tags. Now let us see some of them, which are essential for the layout of the body content.

4.10.1 <H1>, <H2>, <H3>, <H4>, <H5> and <H6> - Heading tags

A heading is a word, phrase, or sentence given at the beginning of a written passage that explains what it is about. Headings are typically displayed in larger and/or bolder fonts than normal body texts. HTML has six levels of headings from **<H1>**

to **<H6>**. Here <H1> creates the biggest text and <H6> the smallest. While displaying any heading, browser adds one line before and one line after that heading. The main attribute of this tag is **Align** and the possible values are,

Left : Text is aligned to the left margin.

Right : Text is aligned to the right margin.

Center : Text is aligned to the centre of the page.

Example 4.4 shows different heading types and alignments and Figure 4.20 shows the corresponding web page.

Example 4.4: To illustrate different heading styles

```
<HTML>
<HEAD>
   <TITLE> Heading Tags </TITLE>
</HEAD >
<BODY Bgcolor= "#FFEFD5">
   <H1 Align= "left"> This is a Heading type 1 </H1>
   <H2 Align= "center"> This is a Heading type 2 </H2>
     <H3 Align= "right"> This is a Heading type 3 </H3>
     <H4> This is a Heading type 4 </H4>
     <H5> This is a Heading type 5 </H5>
     <H6> This is a Heading type 6 </H6>
</BODY>
</HTML>
```

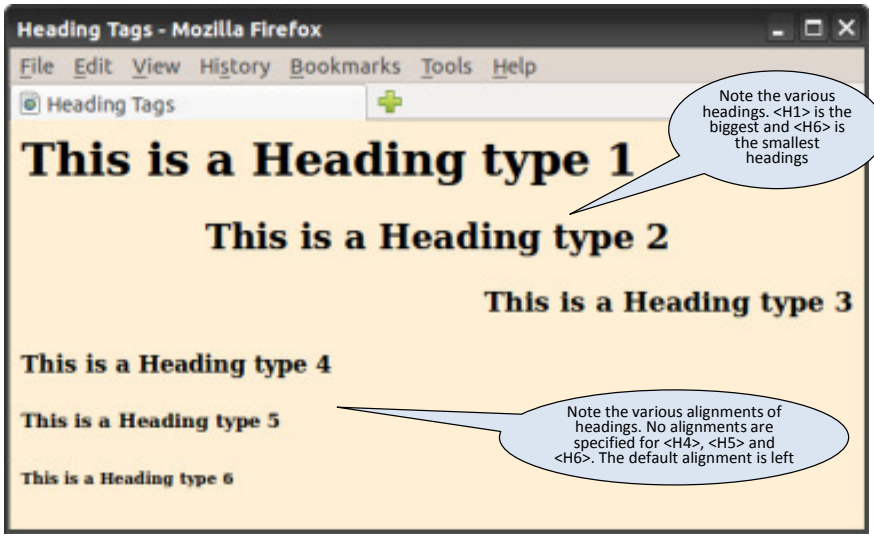


Fig. 4.20 : Setting different heading tags with different alignments

4.10.2 <P> tag - Creating paragraphs

The **<P>** tag enables us to organise the text within the <BODY> tag into paragraphs. It indicates a new paragraph and instructs the browser to add a blank line before the paragraph. Paragraphs in HTML acts much like the paragraphs in any word processor. The paragraph element begins with the <P> tag and ends with </P> tag. The **Align** attribute sets the alignment of the text in the paragraph with the values `left`, `right`, `center` or `justify`.

The code given in Example 4.5 shows how paragraphs are designed with different alignments. Figure 4.21 shows the resultant web page.

Example 4.5: To design paragraphs with different alignments

```
<HTML>
<HEAD>
      <TITLE> Paragraph Tags </TITLE>
</HEAD>
<BODY>
      <P>
      This paragraph          contains
      a lot of lines in the source    code,
            but the browser ignores it.
      </P>
      <P Align= "right">
      This paragraph       contains a lot of spaces
      in the source code,but the browser ignores it.
      </P>
      <P Align= "justify">
      The number of lines in a paragraph depends on
      the size of the browser window.
      If you resize the browser window, the number of lines
            in this paragraph will change.
      When we align a paragraph as justify, the space between
      the words in a line will be adjusted
      so that the lines will be both left and right aligned
      at the same time.
    </P>
</BODY>
</HTML>
```

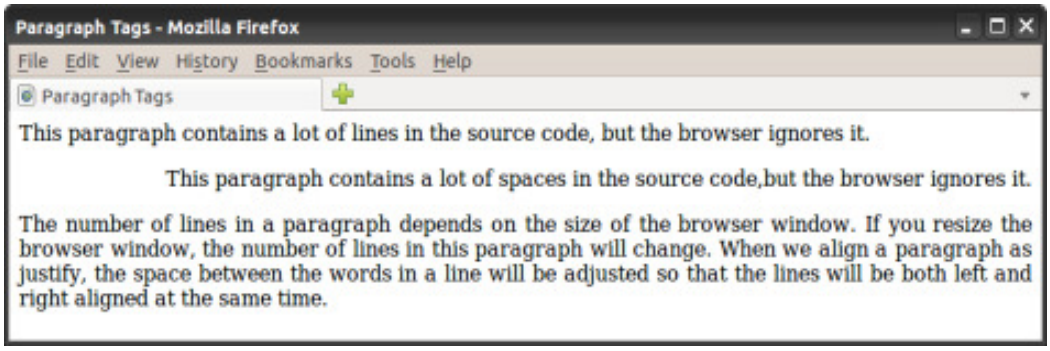


Fig. 4.21: Use of <P> tag with Align attribute

In Example 4.5, we can see that the source code contains three lines in the first paragraph and two lines in the second paragraph with extra spaces. But the web page of this code shown in Figure 4.21 gives a single line for the first two paragraphs. That is, the browser will remove extra spaces and extra lines when the page is displayed. Note that the second paragraph is right aligned and the third paragraph is aligned as justified. Any number of spaces and any number of lines count as only one space. Therefore with HTML, we cannot change the output by adding extra spaces or extra lines in the HTML code. But this is possible in HTML. Now let us discuss how an extra line can be added to the text content using **
** tag. Note that the web pages obtained in large or small screens and resized windows may not match with the one given in Figure 4.21.

4.10.3 <BR> tag - Inserting line break

The purpose of BR element is that it creates a line break within a block of text in a web page. The **
** tag is used to break the current line of text and continue from the beginning of the next line. The <BR> tag is an empty tag, which means that it has no ending (closing) tag.

The HTML code in Example 4.6 displays our National Pledge and Figure 4.22 shows the resultant page in which we can see the effect of <BR> tag and the difference between <P> tag and <BR> tag.

Example 4.6: To show the National Pledge with line breaks

```
<HTML>
<HEAD>
   <TITLE> Line Breaks </TITLE>
</HEAD >
<BODY Bgcolor = "#FFEFC5">
   <H1 Align = "center"> Our National Pledge </H1>
   <P>India is my country and all Indians
```

```
      are my brothers and sisters.<BR>
      I love my country and I am proud of its
      rich and varied heritage.<BR>
      I shall always strive to be worthy of it.<BR>
      I shall give respect to my parents, teachers and all
      elders and treat everyone with courtesy.<BR>
      To my country and my people, I pledge my
      devotion. <BR>In theirwell-being and prosperity alone
      lies my happiness.
    </P>
</BODY>
</HTML>
```

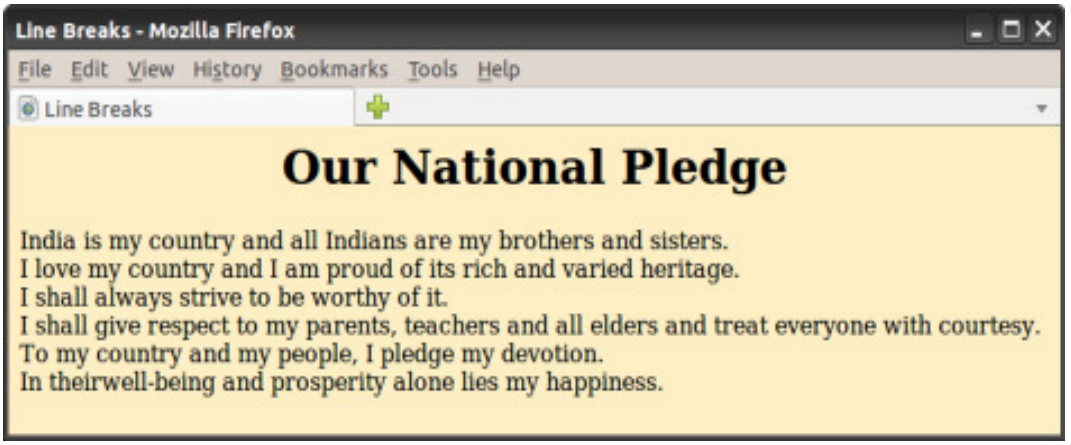


*Fig. 4.22 : Output showing the use of
tag*

Let us do

Referring to Example 4.6 and Figure 4.24, fill in Table 4.5 with appropriate points to distinguish between <P> tag and <BR> tag.

<P> tag	 tag
	Breaks the current line and continues to the next line.
Container tag	

Table 4.5: <P> tag Vs <BR> tag

4.10.4 <HR> tag - creating horizontal line

The **<HR>** tag produces a horizontal line (rule) spread across the width of the browser window. We can change the size (thickness) and width (length) of the line using its attributes **Size** and **Width**. The value of Size is given in pixels; and the value of

Width attribute is given in pixels or in percentage of total width of the document window (72 pixels = 1 Inch). The other important attributes of <HR> tag are Noshade and Color. The Noshade attribute has no value. The Color attribute sets the colour of the line (rule). Remember that <HR> is an empty tag. Obviously Align is another attribute which makes the alignment of the ruler left, center or right.

The code given in Example 4.7 creates a web page as shown in Figure 4.23 to show the effect of <HR> tag and its attributes.

Example 4.7: To draw various types of lines

```
<HTML>
<HEAD>
   <TITLE> Horizontal Rules </TITLE>
</HEAD >
<BODY Bgcolor= "#BDB76B">
<H1 Align= "center"> Horizontal Rules </H1>
Line 1<HR Width= "50%" Align= "center"> <BR>
Line 2<HR Width= "40%" Align= "center" Noshade> <BR>
Line 3<HR Size= "10" Width= "30%" Align= "center"> <BR>
Line 4<HR Size= "10" Width= "30%" Align= "center" Noshade><BR>
Line 5<HR Size= "10" Width= "20%" Align= "center" Noshade
          Color= "gold"> <BR>
Line 6<HR Size="10" Width="20%" Align="center" Color="Aqua">
</BODY>
</HTML>
```

Examining this HTML document and its output, can you identify the similarities and the differences in the shapes of lines? Here, the first two lines are drawn without the Size attribute and the first line hasn't any Noshade atribute. Line 3 and Line 4 are of size 10 but the third one does not have any Noshade attribute. The last two lines have an additional attribute Color. Note the different shapes of these lines in each case. You may modify this code with all possible values for different attributes and see the changes in the web page, during your lab activity.

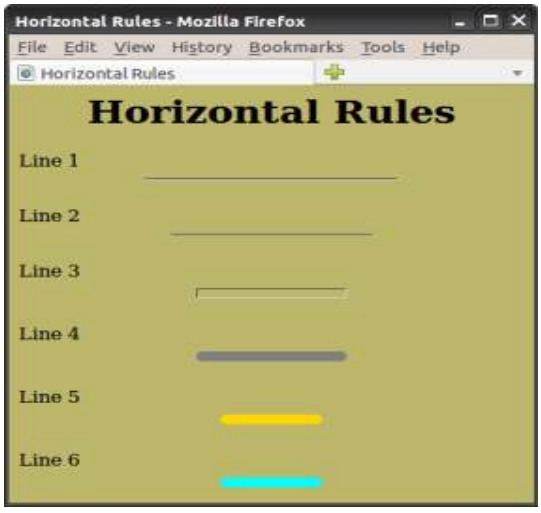


Fig. 4.23 : Different types of horizontal rules

4.10.5 <CENTER> tag - Centering the content

The **<CENTER>** tag brings the content to the centre of a web page horizontally. The content may be text, image, table, etc. This is a container tag and the content enclosed between <CENTER> and </CENTER> tag pair will be centred in the browser window. There is no attribute for this tag.

4.10.6 Text formatting tags

The text in the web page can be formatted, as we often do in word processors. The importance of a text is usually specified with features such as bold, italics, underline, etc. Let us discuss the HTML tags available for this purpose.

<B> - Making text bold

This tag sets the text style to bold. The <B> element displays the content enclosed in bold typeface.

<I> - Italicising the text

It sets the text style to italics. The content enclosed between <I> and </I> tags become italicised.

<U> - Underlining the text

The <U> tag is used to underline a text in HTML. The contents enclosed between <U> and </U> tags become underlined. The formatting tags <U>, <B> and <I> can be combined, so that the content will become bold, italicized and underlined.

<S> and <STRIKE> - Striking through the text

The <S> and <STRIKE> tags are used for the same effect. They display the text in strike through style. For example, ~~Thank you all~~ is a strike through text.

<BIG> - Making the text bigsized

The <BIG> tag is used to make the content bigger in size than the normal text size. It is often used to emphasise the word or lines in the document. Normally the font size of <BIG> tag is one size bigger than the current font size.

<SMALL> - Making the text smallsized

The <SMALL> tag is used to make the size of the text smaller than the current size. Normally the font size of <SMALL> tag is one size smaller than the current font size.

<STRONG> - Making bold text

The <STRONG> tag is a phrase tag. It defines an important text. The <STRONG> text is usually rendered in bold face. It is just the same as <B> tag. The strong element is used to emphasize a phrase of text content.

The code given in Example 4.8 illustrates the application of these tags. For instance, let us quote the words of Mahatma Gandhi. Figure 4.24 shows the resultant web page of this code.

Example 4.8: To illustrate the text formatting tags

```
<HTML>
<HEAD>
   <TITLE> Formatting Tags </TITLE>
</HEAD>
<BODY>
   <P>
   <CENTER><B>Mahatma Gandhi </B>is the <I> <U> Father of
               our Nation.</U> </I> </CENTER>
   Every student must learn the inspiring words (quotes)
   of Gandhiji. It will inspire everyone. Let us have a
   look at it.
   </P><BR>
   <SMALL> The weak can never forgive </SMALL>.
   <BIG> Forgiveness is an attribute of the strong</BIG>
   <BR> Live as if you were to die tomorrow.
   <STRONG> Learn as if you were to live forever.</STRONG>
</BODY>
</HTML>
```

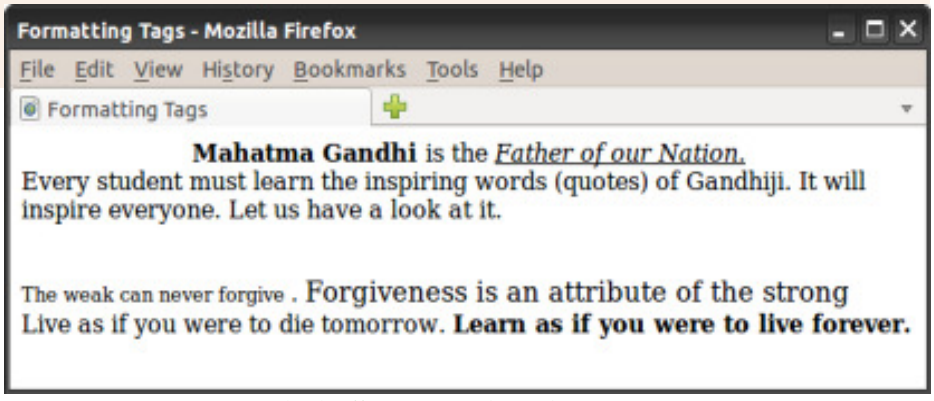


Fig. 4.24: Illustration of text formatting tags

Let us see some more tags which give the text some special appearance.

<EM> - Emphasising the text

The <EM> tag is used to emphasise the text. In practice, the element is usually rendered in italics. The effect of using <EM> tag is the same as that of <I> tag.

<SUB> and <SUP> tags- Creating subscripts and superscripts

We have studied the molecular formula of water, sulfuric acid, etc. in high school classes. Obviously they are H_2O and H_2SO_4 respectively. How can we represent

such notations in HTML? We can see that the figures are written in subscript form. The <SUB> tag is used to create subscripts in a web page. We can display the text H_2O with the code H₂O.

Similarly, the superscripts as in algebraic expressions like $(a+b)^2 = a^2 + 2ab + b^2$ can be represented by the tag <SUP>. The above expression can be written as

```
(a+b)<SUP>2</SUP> = a<SUP>2</SUP> + 2ab + b<SUP>2</SUP>
```

<BLOCKQUOTE> and <Q> tags - Indenting a quotation

The <BLOCKQUOTE> tag is used to indent the content enclosed in these tags. The HTML <Q> tag (Quote tag) is used to indicate the text enclosed in double quotation marks with an indent. This tag is intended for short quotations that do not require paragraph breaks, whereas <BLOCKQUOTE> is used for long quotations.

The illustration of the tags mentioned above is done in Example 4.9. Let us remember that World Environment Day is celebrated on June 5. Let us create thoughts for this day and create a web page using these tags. The corresponding web page is shown in Figure 4.25.

Example 4.9: To illustrate <SUP>, <BLOCKQUOTE> and <Q> tags

```
<HTML>
<HEAD>
   <TITLE> BlockQuote and Q tags  </TITLE>
</HEAD>
<BODY Bgcolor= "#98FB98" Text= "#008000">
   Every year we celebrate World Environment Day on 5<SUP>
   th</SUP> June. Let us have a message to all on this
   occassion.
   <BLOCKQUOTE> <B>June 5<SUP>th</SUP> is World Environment
     Day. </B>
     Mother nature too needs care and protection. Show her
     your care by caring for her trees. Love trees and love
     nature. And work for a greener environment because
     generations have to come... The future depends on us...
   </BLOCKQUOTE>
   <Q>Keep your world clean and green. Save trees, Save the
      environment!!
   </Q>
</BODY>
</HTML>
```

From Figure 4.25, it can be seen that the first paragraph is a normal paragraph, which starts from the left edge of the browser window. The superscripting is also applied in it. The second paragraph starts with an indent due to the use of <BLOCKQUOTE> tag. Finally, the third paragraph is within double quotes. It is the content enclosed by the <Q> tag pair. Observe the alignment of these paragraphs also. Note that, the <BLOCKQUOTE> tag may include more than one paragraph.

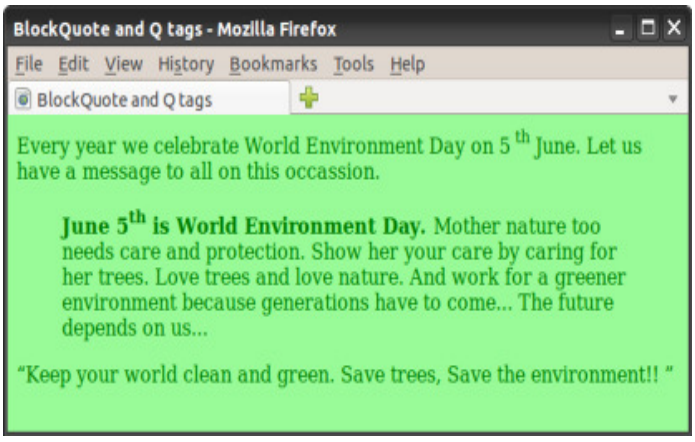


Fig. 4.25: Output of Example 4.9

Know your progress

1. Name some of the text formatting tags.
2. List the different attributes of <HR> tag.
3. How many levels of heading tags are available in HTML?
4. Write an HTML code segment to display x^3+y^3.
5. State True or False.
   a.
 tag is an empty tag.
   b. <EM> and <I> tags have same usage in HTML document.
   c. <U> and <I> tags are not allowed to be used together.
6. What is the use of <STRONG> tag?
7. Which tag performs the same function as that of <STRONG> tag?
8. Pick the odd one out from the following:
   a. HTML b. ALIGN c. HEAD d. CENTER

4.10.7 <PRE> - Displaying preformatted text

Suppose we want to display the content as we entered in the text editor. The **<PRE>** tag can facilitate this purpose. Normally the browser delimitted the white spaces, new line characters, the tab spaces, etc. Therfore, we can turn off the automatic formatting applied by the browser with the help of <PRE> tag. This tag tells the browser that the enclosed text is preformatted and should not be reformatted again; i.e., it tells the browser to display the text exactly in its original form.

Example 4.10 and Figure 4.26 give us an idea of this tag. Let us create a web page that contains some slogans on World Environment Day.

Example 4.10: To illustrate <PRE> tag

```
<HTML>
<HEAD>
   <TITLE> Pre Formatting tags </TITLE>
</HEAD>
<BODY Bgcolor = "#eee8aa" Text = "#b22222">
   <PRE>
     Don't Pollute Water,
            Don't Pollute Air,
     Don't Pollute Environment,
            And Don't Pollute Yourself,
     Celebrate World Environment Day ...
   </PRE>
</BODY>
</HTML>
```

The web page in Figure 4.26 shows that anything written within <PRE> and </PRE> tags will be displayed as it is in the HTML document.

Fig. 4.26: Illustration of <PRE> tag

4.10.8 <ADDRESS> - Displaying the address

The **<ADDRESS>** tag defines the contact information for the author/owner of a document or an article. The content of this tag can include name, phone numbers, PIN numbers, e-mail addresses, etc. Most of the browsers display the texts in italics.

The code given in Example 4.11 illustrates the <ADDRESS> tag. The appearance of this page is shown in Figure 4.27.

Example 4.11: To illustrate <ADDRESS> tag

```
<HTML>
<HEAD>
   <TITLE> Address tag </TITLE>
</HEAD>
   <BODY Bgcolor= "#DDA0DD">
```

```
      The contact details of the "SCERT" is the following:
    <ADDRESS>
      State Council of Educational Research and Training
      (SCERT),<BR>
      Poojappura,<BR>
      Thiruvananthapuram,<BR>
      PIN: 695012, KERALA.<BR>
      Tel : 0471 - 2341883
    </ADDRESS>
</BODY>
</HTML>
```

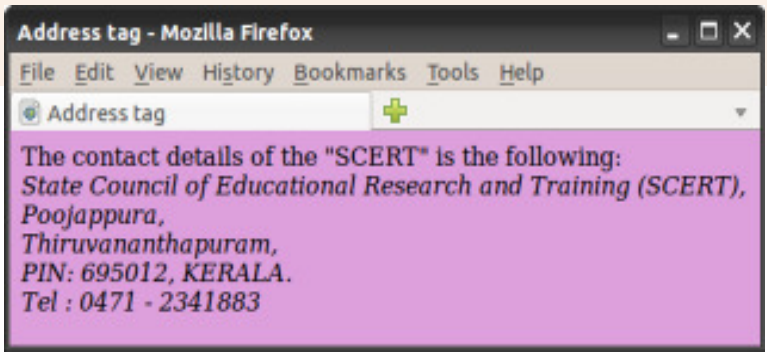


Fig. 4.27 : Web page containing <ADDRESS> Tag

The <ADDRESS> tag is usually used to describe a postal address, when it is considered as a part of the contact information. Although the address element displays the text with the same default styling as that of <I> or <EM> elements, it is suitable for use while dealing with contact information. Typically an <ADDRESS> element can be placed inside the footer which contains information about the author, copyright, etc. of the current section, if any.

4.10.9 <MARQUEE> - Displaying text in a scrolling Marquee

So far we discussed the tags in HTML that just display the contents in the browser. But there is a tag called **<MARQUEE>**, which displays a piece of text or image scrolling horizontally or vertically in the web page.

Given below is the list of important attributes that are used with <MARQUEE> tag.

- **Height**: Sets the height of the marquee in pixels or in percentage of browser window height.
- **Width**: This specifies the width of the marquee in pixels or in percentage of browser window's width value.
- **Direction**: This specifies the direction in which marquee should scroll. This can have a value like `up`, `down`, `left` or `right`.
- **Behaviour**: This specifies the type of scrolling of the marquee. This can have a value like `scroll`, `slide` and `alternate`.

- **Scrolldelay**: This specifies time delay between each jump. This will have value in seconds like 10, 15, etc.
- **Scrollamount**: This specifies the speed of the marquee text.
- **Loop**: This specifies how many times the marquee element should scroll on the screen. The default value is `Infinte`, which means that the marquee scrolls endlessly.
- **Bgcolor**: This specifies background colour in terms of colour name or colour hex value.
- **Hspace**: This specifies horizontal space around the marquee. This can be a value in pixels or percentage value.
- **Vspace**: This specifies vertical space around the marquee. This can be a value in pixels or percentage value.

The code given in Example 4.12 illustrates the use of `<MARQUEE>` tag and Figure 4.28 shows the corresponding web page.

Example 4.12: To illustrate <MARQUEE> tag

```
<HTML>
<HEAD>
    <TITLE> HTML marquee Tag </TITLE>
</HEAD>
<BODY>
    <MARQUEE Width= "50%"> This  will take only 50% width of
      Browser  Window</MARQUEE>
    <MARQUEE Height= "100" Hspace= "100" Bgcolor= "#44BB22"
           Direction= "up"> Scrolling up </MARQUEE>
    <MARQUEE Height= "20" Vspace= "30" Bgcolor= "#FFBB00"
      Direction= "right"> This will scroll from left to right
    </MARQUEE>
</BODY>
</HTML>
```

The first marquee starts from the middle and scrolls towards the left of the window since the width is 50%. The second marquee is in the green coloured portion and 100 pixels height is provided for scrolling. The

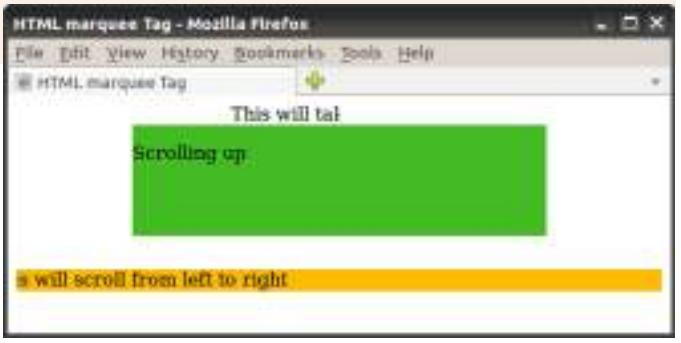


Fig. 4.28 : Marquee tag with different attributes

scroll area is set 100 pixels horizontally away from the left margin. The third marquee is filled with the background colour "#FFBB00" and placed 30 pixels vertically below the previous marquee. The scroll window has a height of 40 pixels and it scrolls from left to right. Instead of words or phrases, we can also use images as the marquee content.

4.10.10 <DIV> - Formatting a block of text

The **<DIV>** tag is used for defining a section or a block in the document. With the <DIV> tag, we can group large sections of HTML documents together and format them. This section may contain paragraphs, tables, etc. Most browsers place a line break before and after <DIV> elements. The attributes of <DIV> tag are:

Align : sets the horizontal alignment with values `left`, `right`, `center`, and `justify`.

Id : assigns a unique identifier for the tag.

Style : indicates how to render the content in terms of colour, font, etc.

In Example 4.13, we use <DIV> tag to apply `Align` and `Style`. Figure 4.29 shows the resultant web page.

Example 4.13: To illustrate <DIV> tag

```
<HTML>
<HEAD>
   <TITLE> DIV Tag </TITLE>
</HEAD >
<BODY Bgcolor= "#ddffff">
   <H2 Align= "center"> Success Story </H2>
   <DIV Align= "Center" Style= "Color:#0000FF"> One day a
   partially deaf four - year old child came home with a
   note in his pocket from his teacher, "Your Tommy is too
   stupid to learn, get him out of the school."
   <P>His mother read the note and answered, "My Tommy is
      not too stupid to learn, I will teach him myself." And
      that Tommy grew up to be the great Thomas Alwa Edison.
   </P> He had only three months of formal schooling.
   </DIV>
   All success stories are stories of great failures.
   You learn from your failure and move forward.
</BODY>
</HTML>
```

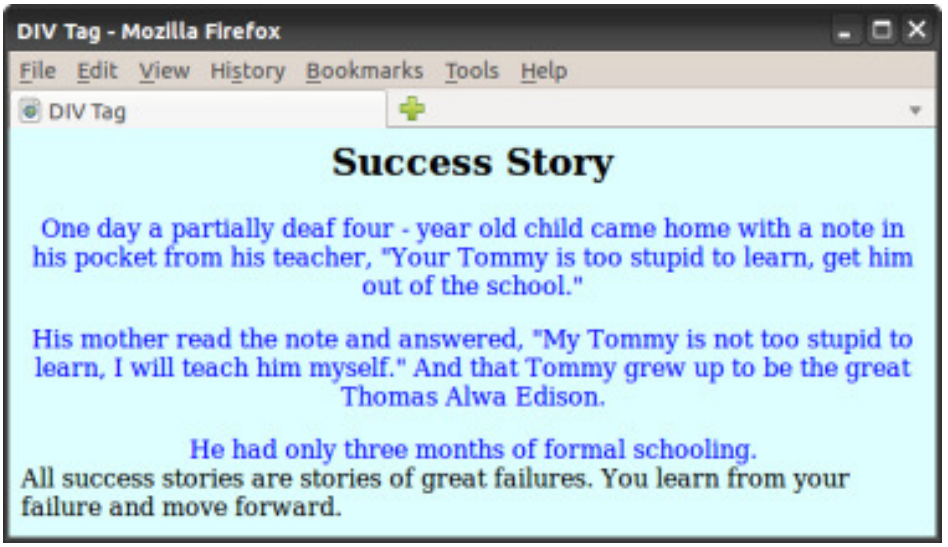


Fig. 4.29 : Application of <DIV> tag

4.10.11 <FONT> - Specifying font characteristics

The **<FONT>** tag allows us to change the size, style and colour of the text enclosed within <FONT> and </FONT> tags. It is generally used for changing the appearance of a short segment of the document. The attributes of <FONT> tag are:

Color : This attribute sets the text colour using either a ColorName or a colour in the Hexadecimal format.

Face : This attribute specifies the font face. If no face attribute is mentioned, the document text in the default style is used in the first font face that the browser supports.

Size : This attribute specifies the font size whose value ranges from 1 to 7, with default value 3.

The code given in Example 4.14 illustrates the usage of <FONT> tag and Figure 4.30 shows the corresponding web page.

Example 4.14: To illustrate <FONT> tag

```
<HTML>
<HEAD> <TITLE> Font tags </TITLE> </HEAD>
<BODY Bgcolor= "#eee8aa">
   Every success story is also a story of great failure.
   The only difference is that every time they failed,
   they bounced back. <BR><BR>
```

```
    <FONT Size="6" Face="Courier New" Color="#B22222">
    Successful people don&apos;t do great things,
    they only do small things in a great way.
    </FONT>
</BODY>
</HTML>
```

In the above code, we marked a portion `'` in green colour. What do you mean by this? From the output, we can understand that this is for getting the single quote in the word **don't**. There are several other characters like this. They are discussed in the following section.

Fig. 4.30: Illustration of <FONT> tag

4.11 HTML entities for reserved characters

In HTML, the symbols like **<**, **>**, **&**, etc. have special meaning and cannot be used in the HTML documents as part of the text content. The browser treats these symbols as the punctuation marks of HTML words like tags and entities. For example, the angle brackets **<** and **>** are used to express tags. So, when we want to display these symbols as part of the text in the web page, we must use HTML entities. Table 4.6 shows a list of a few special characters and their equivalent entities.

In order to display the text A < B & A > C in a web document, we have to specify it in the HTML document as: **`A < B & A > C`**.

Character	Entity	Description
		Non Breaking Space
"	"	Double quotation mark
'	'	Single quotation mark
&	&	Ampersand
<	<	Less than
>	>	Greater than
©	©	Copyright Symbol
™	™	Trademark Symbol
®	®	Registered Symbol

Table 4.6 : List of entities and their description

4.12 Adding comments in HTML document

It is a good practice to add comments in HTML documents, especially in complex documents. Comments help us to understand the code and it increases the code readability. HTML provides comment tag to insert comments in the source code.

Comments are not displayed in the browser window. HTML comments are placed within **<!-- -->** tag. So any content placed within <!-- --> tag will be treated as a comment and will be completely ignored by the browser. In Geany editor the comments are displayed in red colour. The code in Example 4.15 illustrates the use of comments and Figure 4.31 shows the resultant message.

Example 4.15: To illustrate the use of comment tag

```
<HTML>
<HEAD> <!-- Document Header Starts -->
    <TITLE> Comment Tags </TITLE>
</HEAD>
<BODY Bgcolor= "#D8D8D8">
    <!-- This is a comment -->
    <p>The paragraph starts here. Comment statements are
    not displayed in the browser window</p>
    <!-- Comments are not displayed in the browser -->
</BODY>
</HTML>
```

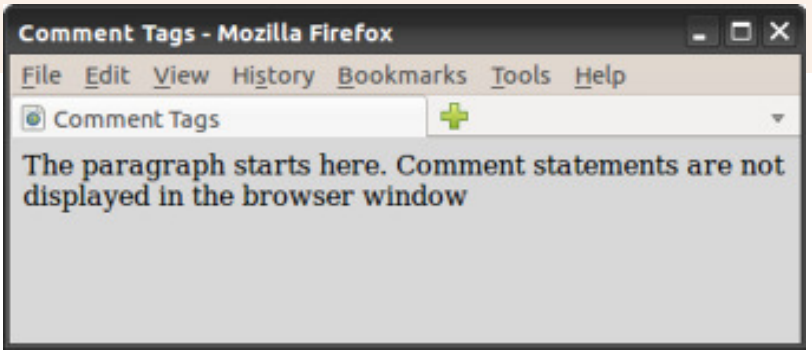


Fig. 4.31: Comments in HTML

Know your progress

1. How are special characters represented in HTML?
2. `Face` attribute is used with _____ tag.
3. List the attributes of `<FONT>` tag.
4. What is the use of `<PRE>` tag?
5. For scrolling a text, we use _____ tag.
6. What are the main attributes of `<MARQUEE>` tag?
7. What is the use of `<ADDRESS>` tag?
8. What is the normal font size in `<FONT>` tag?
9. Name the main attributes of `<DIV>` tag.

4.13 Inserting images

Images always make content presentation more attractive and communicative. Now a days, most of the websites are rich in visuals. New versions of HTML come with many web development features, but the code required for adding images is simple. HTML provides a tag **<IMG>** to insert images in HTML pages. The following is the simple way of using this tag:

```
<IMG Src = "picture1.jpg">
```

The <IMG> tag is an empty tag and it has many attributes. Src is the main attribute and it specifies the file name of the image to be inserted. We can use JPEG, PNG or GIF image files based on our needs, but make sure that the correct filename with the extension is specified using Src attribute. If the image file is not in the current working directory, we have to specify the path of the file or the URL where the file is available.

Setting space for the image

We can set the space in the web page for the image by specifying the values for the **Width** and **Height** attributes. The values are given in terms of either pixels or percentage of its actual size. If these attributes are not specified, the browser will display the image in its actual size.

Now let us discuss how to set space between images. We know that there are two types of space between the images when they are placed in a window - vertical space and horizontal space. HTML offers two attributes **Vspace** and **Hspace**, for providing vertical spacing and horizontal spacing between the images in the web page.

The HTML code in Example 4.16 demonstrates the two types of spacing discussed above and Figure 4.32 shows the web page obtained from the code.

Example 4.16: To provide different types of spacing for images

```
<HTML>
<HEAD>
   <TITLE> Inserting Images </TITLE>
</HEAD>
<BODY Bgcolor="#E0FFFF">
   <H2 Align="center">Inserting vertical and horizontal
                     spacing between images</H2>
   Here the images are placed with <B><I><U> Vspace and
   Hspace </U></I></B> attributes <BR>
```

```
    <IMG Src= "book3.jpg" Height= "50" Width= "70"
         Vspace= "10" Hspace= "10">
    <IMG Src= "book3.jpg" Height= "50" Width= "70"
         Vspace= "10" Hspace= "10"> <BR>
    <IMG Src= "book3.jpg" Height= "50" Width= "70"
         Vspace= "10" Hspace= "10">
    <IMG Src= "book3.jpg" Height= "50" Width= "70"
         Vspace= "10" Hspace= "10"> <BR>
</BODY>
</HTML>
```

Figure 4.32 shows that the images are placed within the given width and height, and the distance between the images is according to the specified horizontal and vertical spacing.

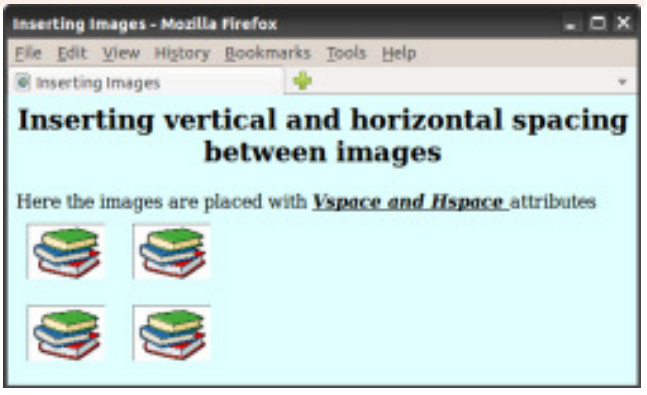


Fig. 4.32: Images within the specified size and with horizontal and vertical spaces between them.

Now let us discuss another important attribute, **Align** for <IMG> tag, which aligns the image with respect to the base line of the text. The possible values for this attribute are the following:

Bottom : Aligns the bottom of the image with the baseline of the text and this is the default setting.

Middle : Aligns the middle of the image (vertically) with the baseline of the text.

Top : Aligns the image with the top of the text.

Let us see the effect of these values for the attribute Align. Example 4.17 and Figure 4.33 demonstrate this.

Example 4.17: To provide different alignments for an image

```
<HTML>
<HEAD>
    <TITLE> Alignment of Images </TITLE>
</HEAD>
<BODY Bgcolor= "#E0FFFF">
    <H2 Align= "center">Alignment of Images</H2>
    This Image is <I><U>aligned at the bottom </U></I>
```

```
    <IMG Src= "book3.jpg" Height= "40" Width= "50"
         Align= "Bottom"> <BR> <BR>
        This Image is <I><U>aligned at the Middle </U></I>
    <IMG Src= "book3.jpg" Height= "40" Width= "50"
         Align= "Middle"> <BR> <BR>
         This Image is <I><U>aligned at the Top </U></I>
    <IMG Src= "book3.jpg" Height= "40" Width= "50" Align="Top">
</BODY>
</HTML>
```

There are some more values for the Align attribute of <IMG> tag. They are left and right, which align the image the left and right sides of the browser window respectively.

Fig. 4.33: Different alignment of images with the text

Fig. 4.34: Images aligned left and right of browser window

Let us do

Observe Figure 4.34, in which the left and right alignments of images in the browser window are shown. Write the HTML document for this web page.

Setting border around an image

Suppose we want to give a border to an image inserted in a web page. It is possible with the Border attribute of <IMG> tag. The thickness of the border can be set by giving appropriate value to this attribute. The HTML code in Example 4.18 and the corresponding web page shown in Figure 4.35 give the effect of border attribute.

Example 4.18: To give a border to an image

```
<HTML>
<HEAD>
    <TITLE> Inserting Images </TITLE>
</HEAD>
<BODY Bgcolor= "#E0FFFF">
    <H2 Align= "center">Inserting Border to Images</H2>
    Here is an image <B><I><U> with Border </U></I></B>
    attribute
```

```
    <IMG Src= "book3.jpg" Height= "50" Width= "70" Border="5">
    <BR>Here is an image<B><I><U> without Border</U></I></B>
    attribute
    <IMG Src= "book3.jpg" Height= "50" Width= "70">
</BODY>
</HTML>
```

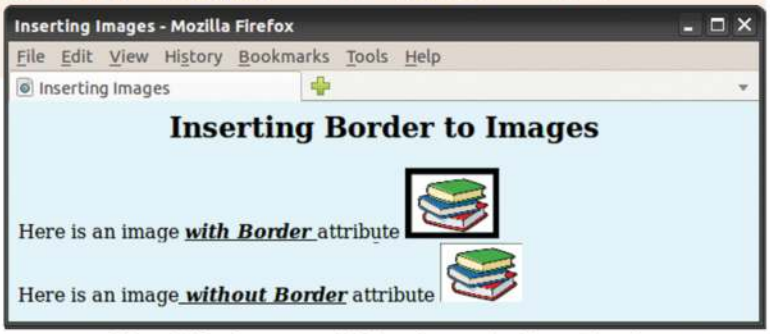


Fig. 4.35: Images with border and without border

We have learnt various attributes of <IMG> tag and their effects on the image. If the image file specified with the Src attribute is not found in the given path, how will the web page look? The specified place in the web page for the image will be left blank. Due to some other reason also, the web browser may not be able to display the image. In such a situation, we can provide an alternative text in the browser in the absence of image. HTML provides the attribute **Alt** to specify an alternate text for an image, if the browser cannot display the image. The code in Example 4.19 illustrates the effect of Alt attribute and Figure 4.36 shows the resultant web page.

Example 4.19: To illustrate the effect of Alt attribute of <IMG> tag

```
<HTML>
<HEAD> <TITLE> Inserting Images </TITLE> </HEAD >
<BODY Bgcolor= "#E0FFFF">
    <H2 Align= "center">Inserting Images</H2>
    If the browser cannot display the image, then the text
    entered in the <B>Alt</B> attribute will be displayed.<BR>
    <IMG Src= "book5.jpg" Height= "20%" Width= "20%"
         Alt= "Image of an opened book"> <BR>
</BODY>
</HTML>
```

Fig. 4.36 : Effect of Alt attribute in <IMG> tag

Know your progress

1. To insert images in an HTML document _____ tag is used.
2. _____ is the main attribute of <IMG> tag.
3. What is the use of Alt attribute of <IMG> tag?
4. Name the attributes that are used to display an image in a particular size.
5. Which are the attributes that provides horizontal and vertical spaces between two images?

Let us conclude

The security of communication over the Internet is a determining factor in the success of the Internet. The security of transactions over the Internet is implemented using HTTPS and digital certificates. Internet infrastructure consists of technologies like web server, software ports and DNS servers used to store and communicate data. Web servers consist of a single/several websites. Websites consists of web pages. We can design web pages either by writing HTML code or using web designing softwares. Webpages can be classified as static and dynamic. Dynamic web pages can be developed using scripts. We can perform client side validations using client side scripting languages like, JavaScript, VBScript, etc. Server side scripts like PHP, ASP, etc. are used to create pages dynamically in the web server. Cascading Style Sheet (CSS) can be used to provide a uniform style to the entire website. The basic concepts of HTML language and web page designing are discussed here. We are now familiar with different tags and their important attributes. We know that some of the tags are container tags, but some others are empty tags. We can make a web page with neatly formatted text using a variety of formatting tags. Different kinds of listing are discussed to make the text more presentable. The beauty of the web pages can be enhanced by including marquees, images, audio and video. We have also discussed the importance of hyper linking and have been familiarised with various kinds of hyperlinks. We should have a sound knowledge of the concepts discussed in this chapter and an excellent practical experience in creating HTML documents, so that we can easily internalise the concepts in the following chapters. Also, we will be able to design beautiful web sites and develop web applications ourselves.

Let us practice

1. Wrtie an HTML code for a web page of Kerala with the following details and features:
   - A heading followed by a paragraph of 5 sentences about Kerala using text formatting tags and attributes.
   - Background of the page needs an image of a scenary.
2. Wrtie an HTML code for a web page for your school with the following details and features:
   - A heading followed by a paragraph of 3 sentences about the district using text formatting tags and attributes.
   - Provide a colour to the background of the page.
   - Include an image of the school.
3. Wrtie an HTML code for a web page for your school with the following details and features:
   - A heading followed by a paragraph of 3 sentences about the district using text formatting tags and attributes.
   - Give the postal address of the school.
   - Include a marquee that "Admission for the new accademic year commences on 10th May"
4. Wrtie an HTML code for a web page to show the lyrics of our National Anthem with the following details and features:
   - A heading with a different font characteristics.
   - An image of our national flag

Let us assess

1. What is the role of routers in transporting data over the Internet?
2. The social media websites developed their own protocol for communicating over the Internet. How is this possible when Internet uses TCP/IP protocol?
3. The user name and password of an e-mail account has to be sent securely over the Internet.
   a. Name the technology used to send the data to the server.
   b. How does this technology support secure data communication?
4. What is the role of payment gateway in online purchases?
5. ABC Engineering College has about 1000 computers connected to the Internet, in its campus. What is the advantage of having a local DNS server in the college's intranet?

6. Write an example of a web server operating system and a web server package.
7. What is the use of software ports in a web server?
8. The port used for HTTP is _____.
9. Suppose you are browsing the website www.prdkerala.org. Explain how the DNS resolves the IP address.
10. What are scripts? Explain the different scripting languages.
11. Consider the home page of your school website and the web page that displays the results of class XI examinations.
    a. Compare the difference between the two web pages based on their creation.
    b. Write the technologies that can be used for developing these web pages.
12. a. The file name extension for a JavaScript file is _____.
    b. Write two popular uses of JavaScript in a web page.
13. What is Ajax? What is its use?
14. Your friend Ravi wishes to create a website that displays the marks of the students in your class in each examination.
    a. Suggest a technology to implement this.
    b. Justify your suggestion.
15. Consider that Manoj is developing a website using PHP that uses a database in MySQL. What are the requirements to implement this if he is using Linux web hosting?
16. "Almost all websites today use CSS for its development." What are the advantages of using CSS in web sites?
17. Who developed HTML?
18. In HTML, there are mainly two sections. Can you name them?
19. If you analyse web pages you can see different colours for links, visited links, background etc. Expalin how this can be done in HTML, with examples.
20. Compare container and empty tags in HTML with examples.
21. The default colour of the attribute `Alink` is _____.
22. The default color of the attribute `Vlink` is _____.
23. Classify the following HTML related words:
    `BR`, `IMG`, `ALIGN`, `FONT`, `FACE`
24. Name the tag which has `Noshade` attribute.
25. Write the main attribute of `<IMG>` tag to insert an image file in the webpage.
26. Mention the purpose of `Alt` atribute in `<IMG>` tag.
27. The default alignment of an image obtained by using `<IMG>` tag is _____.
28. List the main attributes of `<FONT>` tag.

5 Web Designing using HTML

Significant Learning Outcomes

After the completion of this chapter, the learner

- uses different list tags to present the content effectively in web pages.
- identifies the relevance of hyper linking and uses <A> tag for different types of linking.
- provides audio and video in web pages with the help of <EMBED> tag.
- produces inline sounds and videos in web pages.
- lists and explains the tag and attributes for creating a table.
- uses the tags associated with <TABLE> tag to design tables with different characteristics.
- constructs different tables using the tags and attributes.
- identifies the importance of frames in the web page.
- creates frames using appropriate tags to display different web pages in the same browser window.
- identifies the concept of Forms in the web page.
- explains various components in a Form and creates them using proper tags and attributes.
- designs web pages with tables, frames and forms.

In the previous chapter, we studied the basic tags of HTML. We have also learnt to create some simple web pages using those tags and their attributes. But we are familiar with websites that provide much more facilities and utilities. There are websites that contain different types of lists. Linking between web pages is the backbone of World Wide Web. The different types of linking are discussed in this chapter. You might have come across certain information in tabular form. Sometimes you see more web pages in the same browser window. We are also familiar with websites through which we submit the register number to obtain the mark list of examinations, apply for admission and scholarships, pay the bills of electricity and water consumption etc. How can we create tables in web pages to present information? Can we place more than one web page in a single browser window? If yes, how? How are web pages created to accept data from the user to provide information? HTML provides all these facilities for web developers. In this chapter, we discuss the HTML tags required to answer all these questions.

5.1 Lists in HTML

While presenting information, the facility of listing can make it more communicative. Lists are of different types. We are familiar with numbered lists and bulleted lists. HTML offers several mechanisms for specifying lists of information. All lists must contain one or more list elements. There are three kinds of lists in HTML - unordered lists, ordered lists and definition lists.

5.1.1 Unordered lists

Unordered lists or bulleted lists display a bullet or other graphic in front of each item in the list. We can create an unordered list with the tag pair **<UL>** and **</UL>**. Each item in the list is presented by using the tag pair **<LI>** and **</LI>**. Unordered lists are used when a set of items can be placed in any order.

The code in Example 5.1 presents some hardware components of a computer in bulleted list. The corresponding web page is shown in Figure5.1.

Example 5.1: To create an unordered list

```
<HTML>
<HEAD>
   <TITLE> Unordered Lists </TITLE>
</HEAD>
<BODY Bgcolor= "#DEB887">
   <CENTER> <H2> Unordered List </H2> </CENTER>
   While buying a computer, we have to consider many items.
   Here are some important items to consider.
   <UL>
     <LI> RAM </LI>
     <LI> Hard Disk </LI>
     <LI> Mother Board </LI>
     <LI> Processor </LI>
   </UL>
</BODY>
</HTML>
```

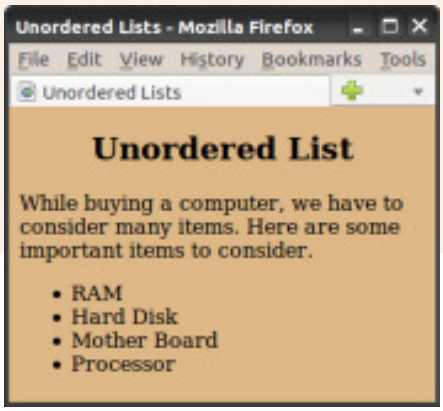


Fig.5.1: Web page containing unordered list

We can customise unordered lists by setting the **Type** attribute to three different values: `Disc` (default value), `Square` and `Circle`, which set the type of bullet that appears before each list item. The following code creates a list as shown in Figure 5.2.

```
<UL Type= "Square">
    <LI> RAM </LI>
    <LI> Hard Disk </LI>
    <LI> Mother Board </LI>
    <LI> Processor </LI>
</UL>
```

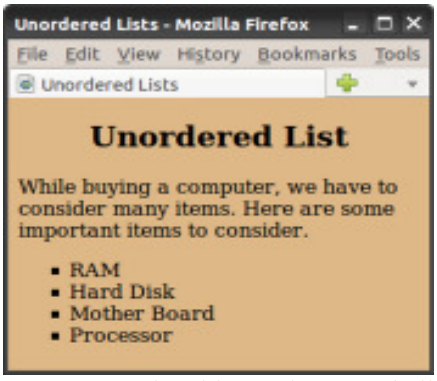


Fig.5.2: Unordered list with square bullet

5.1.2 Ordered lists

Ordered lists present the items in some numerical or alphabetical order. HTML provides the tag pair **<OL>** and **</OL>** to create an ordered list. The items in the ordered list are presented by <LI> tag in <OL> element. The ordered list is also called numbered list. Ordered lists or numbered lists are used to display a list of items that need to be placed in a specific order.

Example 5.2 is a code that presents an ordered list of items. The corresponding web page is shown in Figure 5.3.

Example 5.2: To create an ordered list

```
<HTML>
<HEAD>
    <TITLE> Ordered Lists </TITLE>
</HEAD>
    <BODY Bgcolor= "#DDA0DD">
    <H2 Align= "center"> Ordered List </H2>
    Consider the memory devices of a computer.
    Then according to the speed of data processing,
    we can arrange the memory devices as follows.
    <OL>
      <LI> Registers </LI>
      <LI> Cache </LI>
      <LI> RAM </LI>
      <LI> Hard Disk </LI>
    </OL>
</BODY>
</HTML>
```

Fig.5.3: Web page containing ordered list

We can see that the list in Figure 5.3 is numbered from one through four. There are other numbering styles for presenting the list

items. We can customise the numbering system used in ordered list by using the **Type** attribute, which can set with the values as detailed below:

1 Default numbering scheme (1, 2, 3, ...)

A Upper case letters (A, B, C, ...)

a Lower case letters (a, b, c, ...)

I Large roman numerals (I, II, III, ...)

i Small roman numerals (i, ii, iii, ...)

An ordered list, by default, starts with the first number in the series used in the list. That is, the starting number will be any one from 1, A, a, I and i. If we want to start with any other number in the series, then the **Start** attribute of <OL> tag enables us to change the beginning value. To start numbering a list at 5, for example, we may write: **<OL Start= "5">**. Thus, the numbering starts from 5 and then proceeds with 6, 7, 8, ... and so on.

The Start attribute sets the starting value of the item (it must be an integer) and the Type attribute sets the numbering style. For example, the following ordered list starts numbering from V and continues with VI, VII, ... and so on. The output of this code is shown in Figure 5.4.

```
<BODY Bgcolor ="#DDA0DD">
<H4 Align="center">Ordered List with Type attribute</H4>
<OL Type= "I" Start= "5">
  <LI> Registers </LI>
  <LI> Cache </LI>
  <LI> RAM </LI>
  <LI> Hard Disk </LI>
</OL>
</BODY>
```

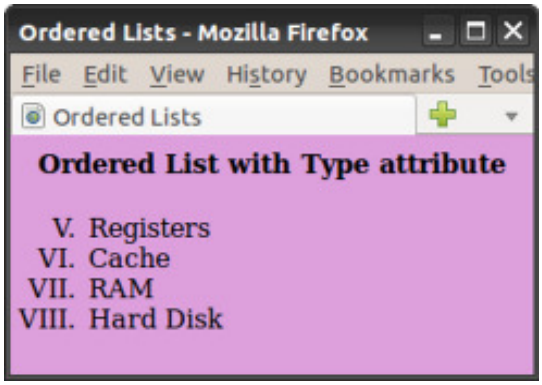


Fig.5.4: Ordered list with different numbering style and starting value

5.1.3 Definition lists

A definition list is a list of terms and the corresponding definitions. The definition text is typically indented with respect to the term. No bullet symbol or number is provided for the list items. The tag pair **<DL>** and **</DL>** enclose the definition lists. Each term in the list is created using the **<DT>** tag and the **<DD>** tag supplies the definition of the term.

The code in Example 5.3 creates a web page to present the definitions of some terms related to security aspects of Internet. Figure 5.5 shows the resultant web page.

Example 5.3: To create a definition list

```
<HTML>
<HEAD> <TITLE> Definition List </TITLE> </HEAD >
<BODY Bgcolor= "#FFE4C4" Leftmargin= "100" Rightmargin= "150">
    <H2 Align= "center"> Definition List </H2>
      Today cyber security has an immense role in the
      field of Internet. The following are some of
      the threats that affect a computer network.
    <DL>
      <DT>Spam :</DT>
      <DD> Spam is the unsolicited e-mail sent in the
           hope of increasing the sales of some product, or
           just for annoying people.</DD>
      <DT>Phishing :</DT>
      <DD> Phishing is an attempt to acquire information
           such as usernames, passwords and credit card details
           by posting as the original website, mostly that
           of banks and other financial institutions. </DD>
    </DL>
</BODY>
</HTML>
```

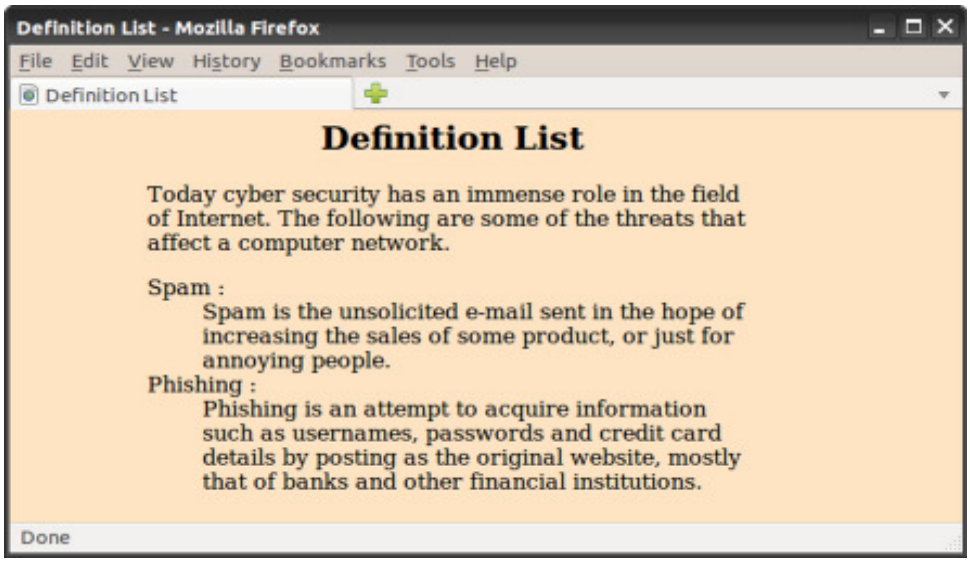


Fig. 5.5: Web page containing a definition list

In Figure 5.5, we can see that each annotation is indented under the corresponding term. Also note the left and right margins in the window.

5.1.4 Nested lists

A list of items can be given under each item of another list. It is known as nesting of lists. This is possible in many ways. For example, we can insert an unordered list into another unordered list, an unordered list into an ordered list, an ordered list inside an unordered list, etc. The code given in Example 5.4 demonstrates the concept of nested list and Figure 5.6 shows the resultant web page.

Example 5.4: To create a nested list

```
<HTML>
<HEAD>
    <TITLE> Nested Lists </TITLE>
</HEAD>
<BODY Bgcolor= "#E0FFFF">
    <H2 Align= "center"> Nested List </H2>
    Consider the devices of a computer.
    We can list some of them as follows.
    <OL>
        <LI> Input Devices </LI>
        <UL>
            <LI>Keyboard</LI>
            <LI>Mouse</LI>
            <LI>Scanner</LI>
            <LI>MICR</LI>
        </UL>
        <LI> Output Devices </LI>
        <UL Type= "Square">
            <LI>Printers</LI>
            <LI>Monitors</LI>
            <LI>Speakers</LI>
        </UL>
        <LI> Memory Devices </LI>
        <UL Type= "Circle">
            <LI>Hard Disc</LI>
            <LI>CD Rom</LI>
            <LI>Flash Drive</LI>
        </UL>
    </OL>
</BODY>
</HTML>
```

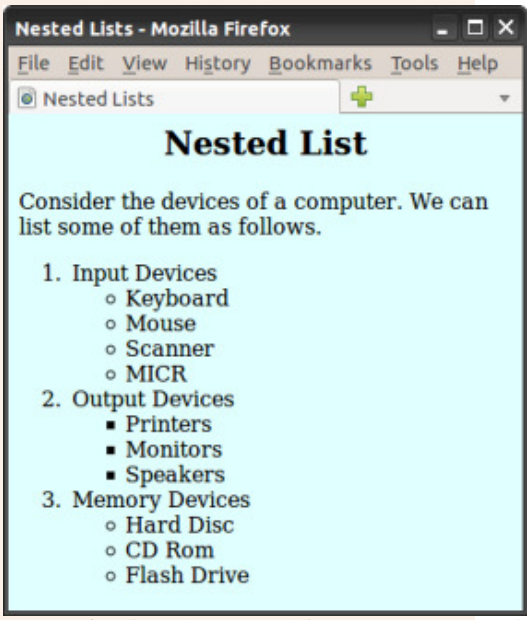


Fig.5.6: Classification of devices using nested list

In Example 5.4, three sets of unordered lists are nested into an ordered list.

Know your progress

1. What are the different types of lists in HTML?
2. Suppose your teacher asks you to display the list of students in your class using HTML document. Which type of list will you prefer? Why?
3. What are the common attributes of <UL> and <OL> tags?
4. What is the difference between <UL> tag and <OL> tag?
5. Name the tags used in the definition list.

5.2 Creating links

A hyperlink is an element, a text, or an image in a web page that we can click on, and move to another document or another section of the same document. Hyperlinks allow visitors to navigate between websites by clicking on words, phrases and images. HTML provides the ability to hyperlink text, image etc. to another document or section of a document. Hyperlink is often referred to as links. In HTML, the **<A>** tag provides the facility to give hyperlinks. This tag is called anchor tag and anything given between the tag pair <A> and </A> becomes part of the link and a user can click that part to reach the linked document. **Href** is the main attribute of <A> tag and it means hyper reference. The value of this attribute is the URL of the document (address of the web page/site) to which hyperlink is provided.

For example, consider the following code segment:

```
<A Href= "http://www.dhsekerala.gov.in">Higher Secondary</A>
```

This creates the target of the hyperlink to the website http:// www.dhsekerala.gov.in. At the time the user clicks the link, the browser opens the home page of this URL. The text inside the tag pair <A> and </A> will appear underlined and in different colour.

The HTML code in Example 5.5 and Figure 5.7 show how hyperlinks are created in web pages.

Example 5.5: To create a hyperlink in a web page

```
<HTML>
<HEAD>
    <TITLE> Anchor Tag </TITLE>
</HEAD>
<BODY Bgcolor= "#FFFFFF">
    <H2 Align= "center"> Hyperlinks </H2>
```

```
    <P>Now this will create a hyperlink to the website of
       Higher Secondary Department.<BR>
       Kindly click on the words
    <A Href= "http://www.dhsekerala.gov.in">Higher Secondary
       Education</A>.
</BODY>
</HTML>
```

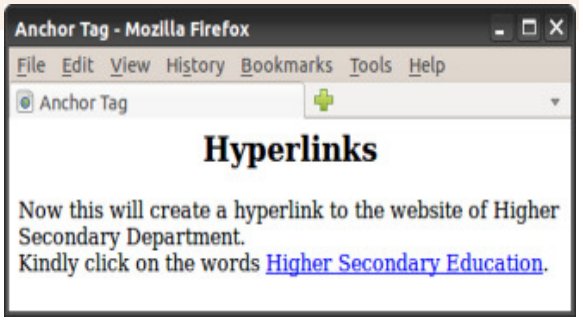


Fig.5.7: Web page containing hyperlink

As shown in Figure 5.7, the hyperlinked text is in underlined format (Higher Secondary Education) and in a different colour.

Hyperlinks are considered either "internal" or "external" depending on their target.

5.2.1 Internal linking

A link to a particular section of the same document is known as internal linking. The attribute **Name** of <A> tag is required for this. We have to give a name to identify the section to be opened by the browser. This name is used with the Name attribute to establish the link.

For example, suppose we have three paragraphs on the subject "Environment Pollution". Let it be the Introduction section, Air pollution section, and Water pollution section. We refer to these sections by giving different values to the Name attribute of <A> tag.

```
<A Name= "Introduction"> INTRODUCTION </A>
<A Name= "Air"> Air Pollution </A>
```

Now, we can refer to these sections by giving the values of Href attribute as #Introduction, #Air (the # symbol is essential) from another section of the document as follows:

```
<A Href = "#Introduction"> Go to Introduction </A>
<A Href = "#Air"> Air pollution </A>
```

Let us create a web page incorporating the concepts of internal linking. Example 5.6 is an HTML code that has two internal links. Figure 5.8 shows the resultant page of this code.

Example 5.6: To create a web page containing internal links

```
<HTML>
<HEAD> <TITLE> Internal Linking </TITLE> </HEAD>
<BODY Bgcolor= "f8f8f8">
    <H2 Align= "center">ENVIRONMENTAL POLLUTION</H2>
    <A Name= "Introduction"><B>INTRODUCTION</B></A>
    <P><FONT Size= "15">E</FONT>nvironment pollution is a
    wide-reaching problem and it is likely to affect the
    health of human population.Here we discuss the environment
    pollution in the perspective of <A HREF= "#Air">air
    pollution </A>,
    <A Href= "#Water"> water pollution </A>and land/soil
    waste pollution. Studies find that these kinds of
    pollutions are not only seriously affecting humans
    but also animals and plants.
    </P>
    <A Name= "Air"><B> Air Pollution</B></A>
    <P>The air we breathe is an essential ingredient for our
    health and wellbeing. Unfortunately polluted air is common
    throughout the world, especially in developed countries.
    </P>
    <A Name= "Water"><B> Water pollution</B></A>
    <P>The water we drink is an essential ingredient for our
    health and wellbeing. Unfortunately polluted water and
    air are common throughout the world.Water pollution is
    caused by the discharge of industrial effluents, sewage
    water and agricultural or household waste.
    </P>
    <A Href= "#Introduction">Go to Introduction </A>
</BODY>
</HTML>
```

On clicking the hyperlink air pollution and water pollution on the web page shown in Figure 5.8, we get that particular section of the document as shown in Figure 5.9.

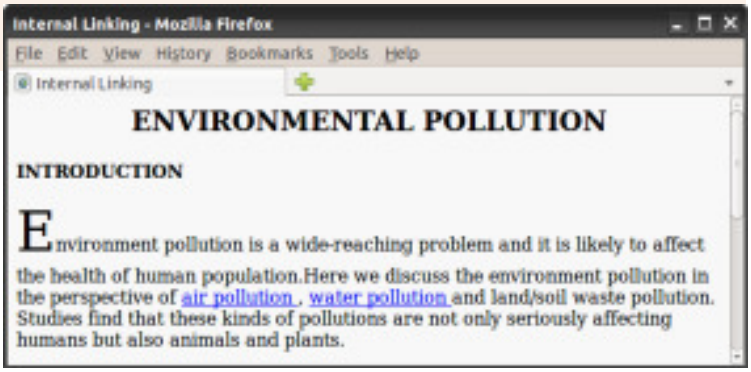


Fig. 5.8: Web page containing internal hyperlinks

Similarly, when we click on the link Go to Introduction at the bottom of the page shown in Figure 5.9, the introduction section will be displayed on the window.

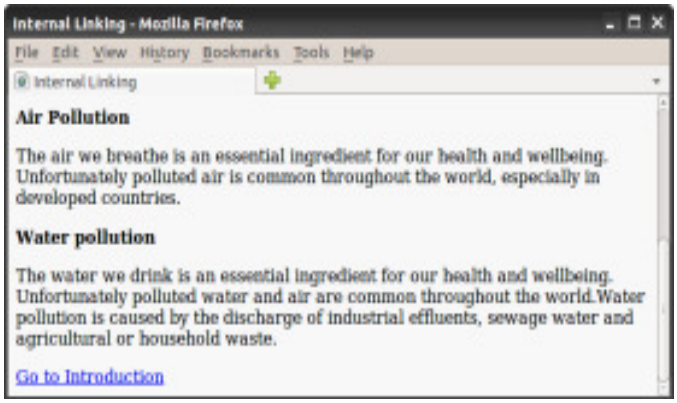


Fig. 5.9: Hyperlinked section of a page in the browser

5.2.2 External linking

The link from one web page to another web page is known as external linking. It is made possible by providing the URL of the external file in the Href attribute of **<A>** tag of the current page. The procedure for external linking is illustrated in the beginning of Section 5.2.

5.2.3 Concept of URL

URL stands for Uniform Resource Locator, and it means the web address. In fact, there are two types of URLs - relative URL and absolute URL.

The referencing `<A Href= "http://www.scertkerala.gov.in">` is an absolute URL, because it is referring to a specific URL. On the other hand, `<A Href = "image.html">` is relative reference. Here we give the file name, '*image.html*' as the URL, and it is a relative URL, relative to the current document. If the URL of the current web page document is *D:\HTML\hyperlink.html*, the web browser knows that the hyperlink `<A Href = "image.html">` points to the file *image.html* in the directory *D:\HTML* itself.

5.2.4 Creating graphical hyperlinks

In the previous sections, we gave hyperlinks to texts. We can make hyperlinks to images also using **<IMG>** tag inside the **<A>** tag. The HTML code given in Example 5.7 and Figure 5.10 show a web page containing graphical hyperlink.

Example 5.7: To create a web page containing graphical hyperlink

```
<HTML>
<HEAD> <TITLE> Graphical Hyperlink </TITLE> </HEAD>
<BODY Bgcolor = "#E0FFFF">
    <H2 Align= "center">Graphical Hyperlink</H2>
    Here is the image with <I>Graphical hyperlink </I>
    <A Href= "https://www.wikipedia.org">
```

```
    <IMG Src= "wiki.jpg" Alt= "Image of Wiki"
         Height= "30" Width= "40" Border= "1"> </A>.
         We can click over this image and the home page of
         linked site, wikipedia.org will open.
</BODY>
</HTML>
```

Suppose you move the mouse pointer upon the image of the wikipedia logo in the web page, the mouse pointer will change to the hand symbol. This indicates that, this image is a hyperlink. When we click on this image, then the corresponding web page www.wikipedia.org will be opened.

Fig. 5.10: Web page containing graphical hyperlink

5.2.5 Creating e-mail linking

We can create an e-mail hyperlink to a web page using the hyperlink protocol **mailto:**. Let us see the code in Example 5.8 and the web page shown in Figure 5.11 to understand the concept of e-mail hyper linking.

Example 5.8: To create a web page containing e-mail hyperlink

```
<HTML>
<HEAD> <TITLE> e-mail Linking </TITLE> </HEAD >
<BODY Bgcolor= "#E0FFFF">
    <H2 Align= "center">e-mail linking</H2>
    Now we can create an <B><I>e-mail hyperlink </I></B> to
    SCERT in the following way. Kindly click on the word
    <A Href= mailto: "scertkerala@gmail.com"> SCERT</A> Kerala.
</BODY>
</HTML>
```

As shown in Figure 5.11, the web contains a link SCERT. If we click on it, e-mail program will be opened with an empty message box addressed to the mail address scertkerala@gmail.com.

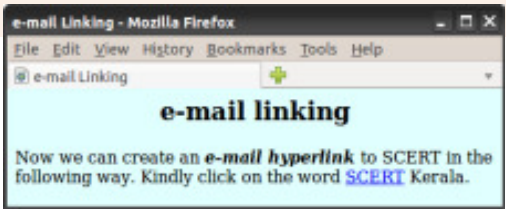


Fig. 5.11: Web page containing e-mail hyperlink

5.3 Inserting music and video

Nowadays, web pages are enriched not only with text and images, but also with plenty of music, video and other multimedia resources. Let us have a basic idea about the inclusion of such resources in web pages. There are two ways of handling multimedia in a web browser. One is as inline and the other is as external data. The inline refers to files and data that are handled as part of the page. These files play the music or video when the page is visible in the browser window. We can also link a web page to external multimedia files with extensions .jpg, .gif, .avi, .png, .tiff, .mp3, .mp4, etc.

The easiest way to add music or video to the web page is with a special HTML tag, called **<EMBED>**. This tag includes the controls of the multimedia automatically in the browser window. In case, the browser does not support the `<EMBED>` tag, then we can use **<NOEMBED>** tag. The content within this tag pair will be displayed, if the `<EMBED>` tag is not supported by the browser.

The main attribute of the `<EMBED>` tag is **Src**, which specifies the URL of the music or video files to be included. The other attributes are **Height**, **Width**, **Align**, **Alt**, etc. The values of these attributes are familiar to us. One more important attribute of `<EMBED>` tag is **Hidden** which indicates whether the embedded component should be made visible or not. It is done by setting the value to `True` (by default) or `False`.

The code in Example 5.9 creates a web page that includes an audio link. Figure 5.12 shows the corresponding web page.

Example 5.9: To create a web page containing audio link

```
<HTML>
<HEAD>
      <TITLE> Embed Tag </TITLE>
</HEAD>
<BODY Bgcolor = "#DDFFFF">
    <H2 Align= "center"> Adding Music and Videos </H2>
    Here is a tag <B><I>EMBED </I></B> with the Multimedia
    features.<BR>
    <EMBED Src= "song1.mp3" Width= "300" Height= "60">
    </EMBED>
</BODY>
</HTML>
```

When the HTML document in Example 5.9 is opened, the webpage will play the music that we embedded along with the document. The audio controls like play/pause, volume, etc. will also be displayed on the web page. Similarly we can add movies in a webpage by using the tag <EMBED> and its attribute **Src**.

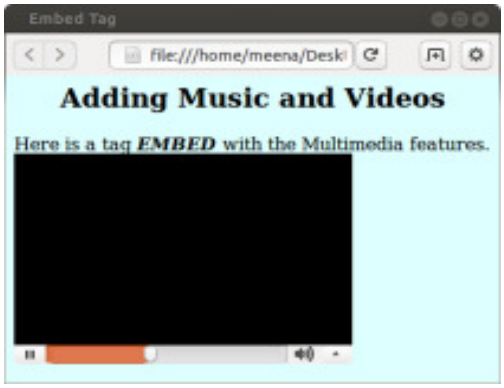


Fig. 5.12: Web page containing audio link

The code in Example 5.10 and the resultant web page shown in Figure 5.13 illustrate this linking.

Example 5.10: To create a web page containing video link

```
<HTML>
<HEAD>
    <TITLE> Embed Tag </TITLE>
</HEAD >
<BODY Bgcolor = "#DDFFFF">
    <H2 Align="center">Adding Music and Videos</H2>
    Here is a tag <B><I>EMBED </I></B>with the Multimedia
    features.
    <EMBED Src= "alan.mp4" Width= "300" Height= "150">
    </EMBED>
    <NOEMBED><IMG Src= "book2.jpg"
          Alt= "Alternative Media">
    </NOEMBED>
</BODY>
</HTML>
```

When we open this page, the video starts to play. Here again, we can control the different properties of the audio and video, such as volume, pause, full screen mode, etc. We can also include audios and videos from other web pages or sites by linking them to our pages.

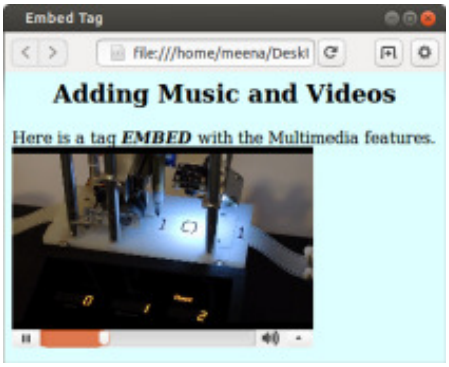


Fig. 5.13: Web page with video

As mentioned earlier, music can be played in the background while the page is viewed. This kind of audio inline is achieved by the tag **<BGSOUND>**. The code in Example 5.11 illustrate the use of this tag.

Example 5.11: To illustrate the use of <BGSOUND> tag

```
<HTML>
<HEAD> <TITLE> Background Music </TITLE> </HEAD >
    <BODY Bgcolor = "#DDFFFF">
    <H2 Align= "center">Adding Background Music </H2>
    Here is a tag <B><I> BGSOUND </I></B> which helps
    us to play background music in our web page.
    <BGSOUND Src= "Song2.mp3" Loop= "Infinite">
</BODY>
</HTML>
```

We can adjust the volume control by using the `Volume` attribute of `<BGSOUND>` to make the music softer or harder. Note that while creating HTML documents that include external files in `Src` and `Href` attributes, the correct path should be provided. If you try out the codes given in the examples, you have to make necessary modifications in these attributes. The attribute `Loop` determines the duration of play. The value `Infinite` causes the music play as long as the page is in view.

5.4 Creating tables in a web page

Sometimes, presenting a lot of information in a structured way is essential, especially in websites. Suppose we need to compare the data collected from a hospital for the period from 2012 to 2014, which states the number of newly diagnosed cancer patients among regular smokers, pan users and alcohol consumers. It can be presented in a tabular form as shown in Table 5.1.

NEWLY DIAGNOSED CANCER PATIENTS IN THE HOSPITAL FOR THE LAST 3 YEARS			
Year	**2012**	**2013**	**2014**
Smokers	129	140	143
Pan users	54	56	49
Alcohol users	74	68	77
Other cases	95	93	92

Table 5.1: Cancer patients diagnosed in a hospital

As we know, a table is a structured element that consists of rows and columns of cells. We can place any kind of content like text, image, and even other tables within these cells.In HTML, **<TABLE>** tag is used to create tables. This tag needs the support of some other tags like **<TH>**, **<TR>** and **<TD>** for constructing tables. Let us discuss these tags in the following sections.

5.4.1 <TABLE> tag

<TABLE> tag is a container tag. The whole content of the table should be enclosed within the tag pair <TABLE> and </TABLE>. This tag has many attributes to improve the general layout of the table. Some of the main attributes are listed below:

1. **Border**: This attribute specifies the thickness of the border line around the table. To draw the border of the table we have to specify a non-zero value to Border attribute in pixels. A border value of zero produces a table without border.
2. **Bordercolor**: It is used to assign colour to the table border.
3. **Align**: The position of the table inside the browser window is determined by Align attribute. The possible values are left (by default), right or center.
4. **Bgcolor**: We can change the background colour of the table using this attribute.
5. **Background:** It is used to assign a background image for a table. To set the background, specify the pathname of the image as value of this attribute. For example, <TABLE Background = "images/flower.gif"> sets the background of a table with the image in the file *flower.jpg* stored in the folder *images* in the current drive. When both Bgcolor and Background are specified, the Background will override Bgcolor.
6. **Cellspacing**: Table cells have space in between them. We can either increase or decrease the space between cells. The Cellspacing attribute determines the space to be left between cells. The value is given in number of pixels.
7. **Cellpadding**: There is a space between the content and cell border. We can increase or decrease the space between cell border and content. The Cellpadding attribute specifies the space in between the cell border and cell content. The value is given in number of pixels.
8. **Width and Height**: We can set a table width and height using Width and Height attributes. We can specify table width or height in terms of pixels or in terms of percentage of browser window.
9. **Frame**: This attribute with one of the values given in Table 5.2 indicates how table borders are displayed.

Value	Description
Void	Display no borders
Above	Display a border on the top of the table only
Below	Display a border on the bottom of the table only

Value	Description
Hsides	Display borders on the horizontal sides (top and bottom) only
lhs or rhs	Display the border only the left side or the right side
Vsides	Display borders on the vertical sides (right and left) only
box or border	Display borders on all sides of the table (It is the default value)

Table 5.2: Values of Frame attribute

10. **Rules**: We can use the Rules attribute to control what rules (borders between cells) are displayed in a table. Table 5.3 shows the values of Rules attribute.

Value	Description
none	Display no rules
cols	Display rules between columns only
rows	Display rules between rows only
groups	Display rules between row groups and column groups only
all	Rules will appear between all rows and columns

Table 5.3:Values of Rules attribute

Now let us discuss some other tags associated with <TABLE> tag.

5.4.2 <TR> tag

The rows in a table are created using <TR> tag. It is a container tag. The whole row is enclosed within the tag pair <TR> and </TR>. A <TR> tag always comes inside the <TABLE> tag. A row itself is a collection of cells. A cell is the smallest component of a table. There are two types of cells - heading cells and data cells. As seen in Table 5.1, the values given in red colour are of heading type. The values given in blue are just data cells.

5.4.3 <TH> tag

<TH> tag is used to define heading cells. It is also a container tag. The heading data should be enclosed between <TH> and </TH> tags. The heading cells are displayed in bold face and in centred form. <TH> tag always comes inside the <TR> tag.

5.4.4 <TD> tag

<TD> tag is similar to <TH> tag and is used to display data cells. It is also a container tag. The data is given in between <TD> and </TD> tags. Similar to <TH> tag,<TD> tag is also placed within the <TR> tag.

The code given in Example 5.12 creates a simple table and Figure 5.14 shows the resultant web page.

Example 5.12: To create a web page containing a simple table

```
<HTML>
<HEAD> <TITLE> Html Tables </TITLE>
</HEAD>
<BODY>
    <TABLE Border="1">
        <TR>
            <TH>Roll No</TH>
            <TH>Name</TH>
        </TR>
        <TR>
            <TD>1</TD>
            <TD>Aliya</TD>
        </TR>
        <TR>
            <TD>2</TD>
            <TD>Arun</TD>
        </TR>
    </TABLE>
</BODY>
</HTML>
```

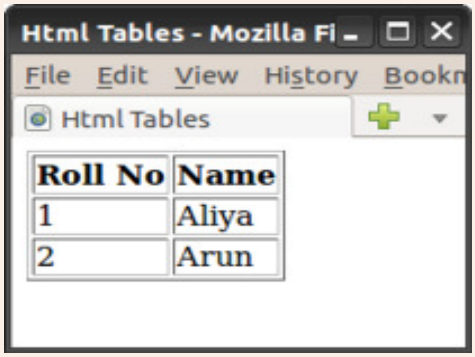


Fig. 5.14: Web page containing a simple table with two columns

Now, let us create the following table with some of the attributes discussed above.

Year	2012 - 14
Smokers	412
Pan users	159
Alcohol users	219
Other cases	280

The code given in Example 5.13 can create a web page as shown in Figure 5.15.

Example 5.13: To create a web page containing a table with border and colour

```
<HTML>
<HEAD> <TITLE> Hospital Table </TITLE> </HEAD>
<BODY>
    <TABLE Border= "3" Bordercolor= "RED" Bgcolor= "#4EB0AF"
           Align= "left" Cellspacing= "16" Cellpadding= "5"
           Width= "50%">
        <TR>
            <TH> Year </TH>
            <TH> 2012-14 </TH>
        </TR>
```

```
        <TR>
            <TH> Smokers </TH>
            <TD> 412 </TD>
        </TR>
        <TR>
            <TH> Pan users </TH>
            <TD> 159 </TD>
        </TR>
        <TR>
            <TH> Alcohol users </TH>
            <TD> 219 </TD>
        </TR>
        <TR>
            <TH> Other cases </TH>
            <TD> 280 </TD>
        </TR>
    <TABLE>
<BODY>
<HTML>
```

Fig. 5.15: Table using Cellspacing and Cellpadding

In Example 5.13, we used <TH> tags within all <TR> tag pairs, which make the first column of the table similar to header column as shown in Figure 5.15.

Attributes of <TR> tag

The characteristics of a row can be changed using the attributes of <TR> tag.

1. **Align**: This attribute specifies the horizontal alignment of the text in a cell in that particular row. This can take the values `left`, `right` or `center`. The default is `left` for data and `center` for headings (see Figure 5.15).
2. **Valign**: We can specify the vertical alignment of the content in a cell of any row using `Valign`. The possible values are `top`, `middle`, `bottom` or `baseline`. Baseline vertical alignment aligns the baseline of the text across the cells in the row.
3. **Bgcolor:** This attribute gives background colour to a particular row. Usually this helps in highlighting a row.

The following code segment is the modified part of the code given in Example 5.13. It modifies the third row of the table shown in Figure 5.15 with a background colour, and horizontal and vertical alignments.

```
<TR Bgcolor= "yellow" Align= "right" Valign= "middle">
    <TH> Pan users </TH>
    <TD> 159 </TD>
</TR>
```

The modified web page is shown in Figure 5.16.

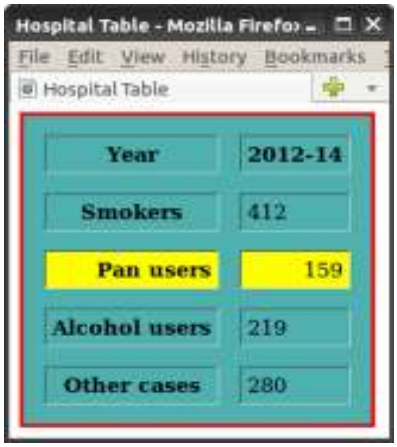


Fig. 5.16: Table with modified row using Bgcolor, Align and Valign

Attributes of <TH> and <TD> tags

Most of the attributes of <TH> and <TD> tags are common, as both these tags are used for creating table cells. Let us discuss some of the main attributes.

1. **Align**: It specifies the horizontal alignment of the content within a cell. It can take the values left, right or center. The default value is center for <TH> and left for <TD>.
2. **Valign**: It specifies the vertical alignment of the content within a cell. It can take a value top, bottom, middle or baseline.
3. **Bgcolor**: We can set background colour for any cell using this attribute. The tags <TABLE>, <TR>, <TD>/<TH> all have Bgcolor attribute. If Bgcolor attribute is set for all these tags, the cell will take the colour assigned to this attribute in <TH>/<TD> tag.
4. **Colspan**: Usually a cell spans over a single column, but sometimes we may need columns occupying more than one cell. Colspan value defines the number of columns occupied by that particular cell. For example, <TH Colspan="3"> makes the cell span over three columns.
5. **Rowspan:** Similar to Colspan, Rowspan attribute specifies the number of rows to be spanned by the cell. The default cell span is single row, but we can make it span multiple rows by setting Rowspan attribute a numeric value greater than one. For example, <TD Rowspan="4"> makes the cell span over four rows.

Let us create a web page with a simple table to illustrate the effect of the attributes discussed above. The code given in Example 5.14 shows the use of these attributes and Figure 5.17 shows the corresponding web page.

Example 5.14: To create a table to demonstrate Colspan and Rowspan

```
<HTML>
<HEAD> <TITLE> Students Registration </TITLE>
</HEAD>
```

```
<BODY>
   <TABLE Border= "1" Cellspacing= "3" Cellpadding= "5">
      <TR>
         <TH Colspan= "3"> No. of Registered Students </TH>
      </TR>
      <TR>
         <TH Rowspan= "2"> Year </TH>
         <TD> 2014 </TD> <TD> 75 </TD>
      </TR>
      <TR>
         <TD> 2015 </TD> <TD> 88 </TD>
   </TR>
   </TABLE>
<BODY>
<HTML>
```

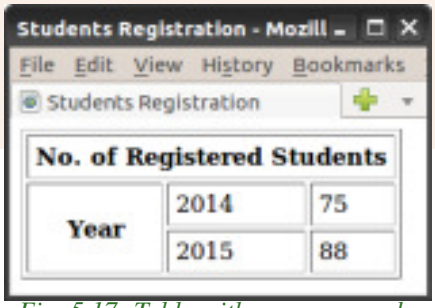


Fig. 5.17: Table with row span and column span

Now, let us create a web page to display the details of cancer patients as shown in Table 5.1. The code given in Example 5.15 can design a table as shown in Figure 5.18.

Example 5.15: To create a table to show the data of cancer patients

```
<HTML>
<HEAD> <TITLE> CompleteTable </TITLE> </HEAD>
<BODY Bgcolor= "silver">
   <TABLE Border= "3" Bordercolor= "red" Bgcolor= "#4EB0AF"
          Align= "left" Cellspacing= "2" Cellpadding= "2"
          Width= "50%">
      <TR>
         <TH Colspan= "5"> Number of cancer patients reported
                           at the hospital </TH>
      </TR>
      <TR Align= "center">
         <TH Colspan= "2"> Year </TH>
         <TH> 2012 </TH>
         <TH> 2013 </TH>
         <TH>  2014 </TH>
      </TR>
      <TR Align= "center">
         <TH Rowspan= "4"> Cancer Origin </TH>
         <TH> Smokers </TH>
         <TD> 129 </TD>
```

```
            <TD> 140 </TD>
            <TD> 143 </TD>
        </TR>
        <TR Align= "center">
            <TH> Pan users </TH>
            <TD> 54 </TD>
            <TD> 56 </TD>
            <TD> 59 </TD>
        </TR>
        <TR Align= "center">
            <TH> Alcohol users </TH>
            <TD> 74 </TD>
            <TD> 68 </TD>
            <TD> 77 </TD>
        </TR>
        <TR Align= "center">
            <TH> Other cases </TH>
            <TD> 95 </TD>
            <TD> 93 </TD>
            <TD> 92 </TD>
        </TR>
        <TR Align= "center">
            <TH Colspan= "2"> TOTAL Patients </TH>
            <TD> 352 </TD>
            <TD> 357 </TD>
            <TD> 371 </TD>
        </TR>
    </TABLE>
</BODY>
</HTML>
```

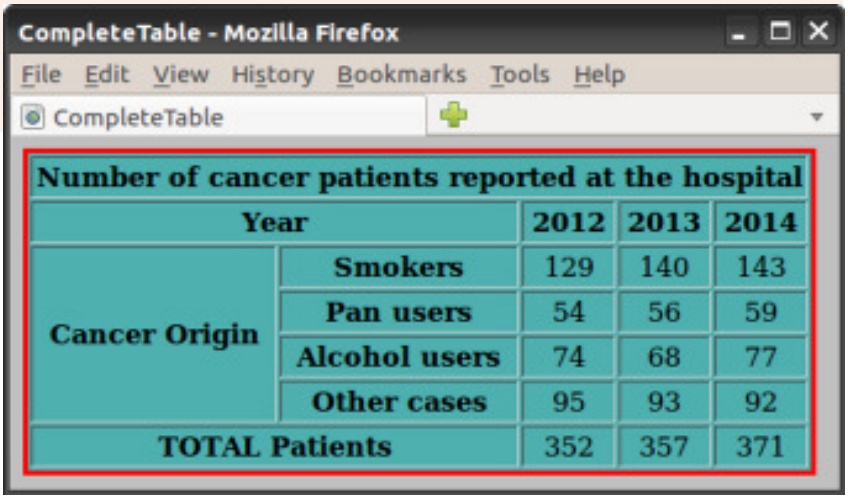


Fig. 5.18: Web page containing a table of cancer patients Bgcolor, Rowspan, Colspan, Align and Valign

Let us do

Try to create a table containing the number of students studying different second languages available in your school. The table should provide details of each class of your higher secondary section.

5.4.5 Table caption with <CAPTION> tag

We can provide a heading to a table using the **<CAPTION>** tag. It provides an easy method to add descriptive text to a table as its caption. Suppose we want to modify the table shown in Figure 5.19 with a caption. We can modify the code section before the first **<TR>** tag in Example 5.15 with following code segment:

```
<TABLE Border= "3" Bordercolor= "red" Bgcolor= "skyblue"
       Align= "left" Cellspacing= "2" Cellpadding= "2"
       Width= "50%">
<CAPTION>  Number of new cancer patients reported at the
           hospital during 2012-14
</CAPTION>
```

The modified HTML code will produce a table as shown in Figure 5.19.

CompleteTable - Mozilla Firefox

File Edit View History Bookmarks Tools Help

CompleteTable

Number of new cancer patients reported at the hospital during 2012-14

Number of cancer patients reported at the hospital				
Year		2012	2013	2014
Cancer Origin	Smokers	129	140	143
	Pan users	54	56	59
	Alcohol users	74	68	77
	Other cases	95	93	92
TOTAL Patients		352	357	371

Fig. 5.19: Modified table with a caption

Know your progress

1. Name any two associated tags of <TABLE> tag.
2. Choose the odd one out:
   a. TABLE b. TR c. TH d. COLSPAN
3. Differentiate between <TD> and <TH>.
4. <TABLE> tag is an empty tag. State whether this statement is true or false.
5. Give any two attributes of <TR> tag.

5.5 Dividing the browser window

Sometimes we need to include more than one webpage inside a single browser window. The browser window can be divided into two or more panes to accommodate different pages simultaneously. HTML provides a facility called frameset to partition the browser window into different sections. Each section accommodates different HTML pages. Each individual section created by a frameset is called a frame. Figure 5.20 shows a simple frameset consisting of three frames. To create a frameset, we need **<FRAMESET>** and **<FRAME>** tags.

Fig. 5.20: A simple frameset with three frames

5.5.1 <FRAMESET> tag

<FRAMESET> is a container tag for partitioning the browser window into different frame sections. The frames are defined within the tag pair `<FRAMESET>` and `</FRAMESET>`. The main attributes are:

1. **Cols**: This attribute determines the number of vertical frames in the frameset page and its dimensions. The column width can be given either in percentage of total width or in number of pixels. For example, `<FRAMESET Cols = "30%,500,*">` will create three vertical frames. The first frame will occupy 30% of the total window width, the next will take 500 pixels space, and the asterisk(*) symbol given at the end denotes the remaining space which is given to the third frame.
2. **Rows**: Similar to `Cols`, `Rows` attribute defines the number and dimension of horizontal frames.
3. **Border:** We can specify the thickness of border for the frames by using `Border` attribute. Here the value is given in pixels.
4. **Bordercolor:** We can specify border colour by assigning a colour to this attribute.

5.5.2 <FRAME> tag

<FRAME> tag is an empty tag and it defines the frames inside the `<FRAMESET>`. For each division inside the `<FRAMESET>` tag, we should have a corresponding `<FRAME>` tag. `<FRAME>` tag always comes with **Src** attribute which specifies the HTML page to be loaded in the frame. The main attributes related to `<FRAME>` tag are:

1. **Src**: As we have already discussed, Src specifies the URL of the document to be loaded in the frame. For example, `<FRAME Src = "school.html">` loads the page *school.html* in a particular frame.
2. **Scrolling**: When the content inside a frame exceeds the frame size, a scrollbar appears depending upon the value of the `Scrolling` attribute. It can take values `Yes`, `No` or `Auto`. `Auto` is the default value and it displays scroll bars if the frame content exceeds the normal display area.
3. **Noresize**: It is used to prevent users from resizing the border of a specific frame by dragging on it. Just giving `Noresize` prohibits resizing the frame window. For example, `<FRAME Src= "school.html" Noresize>`.
4. **Marginwidth** and **Marginheight**: We can set horizontal and vertical margins to the frame by using `Marginwidth` and `Marginheight`, respectively. The values are given in pixels.
5. **Name**: It gives a name to a frame. This particular frame can be referenced using this name later in the code.

The following code can create the frameset shown in Figure 5.20. It is assumed that the three HTML pages *sampleframe1.html*, *sampleframe2.html* and *sampleframe3.html* are already created as in Figure 5.20.

```
<HTML>
<HEAD> <TITLE> A simple Frameset </TITLE> </HEAD>
    <FRAMESET Rows= "20%, 30%, 50%">
        <FRAME Src= "sampleframe1.html">
        <FRAME Src= "sampleframe2.html">
        <FRAME Src= "sampleframe3.html">
    </FRAMESET>
</HTML>
```

5.5.3 Targeting frames

In a frameset page, we can provide hyperlink in any frame and the linked page can be targeted to any other frame. When we click on the link in a frame, the corresponding page is opened within another frame in the frameset. For this, first we have to allot a name for the destination frame using its `Name` attribute. Then we can refer the named frame using `Target` attribute of the link to be displayed.

Let us do

Create three HTML files *bio.html*, *poem.html* and *fiction.html* to show a list of books belonging to biography, poetry and fiction, respectively.

Now let us create a web page with two frames - one for showing the links for these files and the other for opening the respective web pages. When a user clicks on any one of the three links available in one frame, the associated file will open in the other frame.

Once you have created the three files mentioned above, the two HTML codes given in Example 5.16 can satisfy our need.

Example 5.16: To illustrate the concept of targeting frames

Save the following code in a file ***main.html***

```
<HTML>
<HEAD> <TITLE> Left Frame </TITLE> </HEAD>
<BODY Bgcolor= "#00AFFF" Text= "#282D2F">
    <H2> Select the option </H2>
    <FONT Size= "5">
    <A Href= "bio.html" Target="right_frame">Biography</A><BR>
    <A Href= "poem.html" Target="right_frame">Poetry</A><BR>
    <A Href= "fiction.html" Target="right_frame">Fiction</A>
    </FONT>
</BODY>
</HTML>
```

Now create a file to store the following code:

```
<HTML>
<HEAD> <TITLE> Targeting Frames </TITLE> </HEAD>
<FRAMESET Cols= "200, *">
    <FRAME Src= "main.html" Name= "left_frame">
    <FRAME Name= "right_frame">
</FRAMESET>
</HTML>
```

When the above document is executed, a web page with two frames will be opened. The first column of the browser window will show the *main.html* and the second column will be blank. Note that **Src** attribute is not specified for that frame, but a name `right_frame` is assigned. If the first link (biography) is selected, the file *bio.html* will be opened in the second frame as shown in Figure 5.21. It is

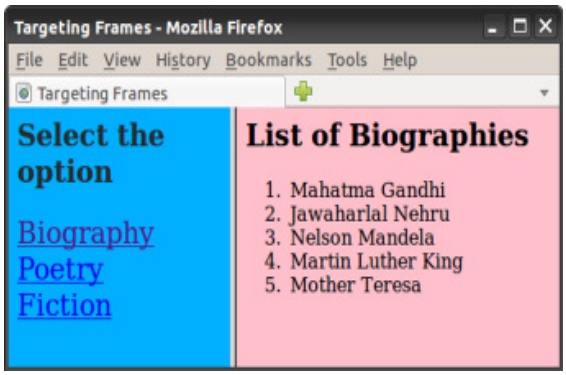


Fig. 5.21 : Web page with target frame

assumed that the file *bio.html* contains the details as shown in the figure. Note the difference in colour in the selected link.

Note that, the HTML document main.htm contains three pairs of <A> tag to link the three HTML files. A tag named **Target** is used in <A> tags to specify the frame in which the linked file is to be opened. Here comes the relevance of names given to the frames.

<FRAMESET> tag is not supported in HTML 5 version.

There are many arguments against the use of frameset. Some of them are given below.

- The browser's back button will not work naturally inside a frameset.
- It is difficult to refer a specific document within a frameset.
- Any attempt to reload a specific frame may reload the entire frameset. As a result it will reset the frame contents to their default sources.
- Search engines can struggle when navigating through framed documents.
- It is difficult to print contents of a frameset.
- Bookmarking contents of a frameset is difficult.

5.5.4 Nesting of framesets

In Figures 5.20 and 5.21, we can see frames in the browser window. The first figure shows horizontal division of the browser window into three frames. But the second figure shows vertical division into two frames.

Suppose we want to divide the browser window as shown in Figure 5.22. It is possible with nested frameset. It is the process of inserting a frameset within another frameset.

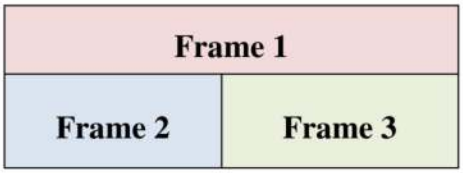


Fig. 5.22: Nesting of frameset

The following steps illustrate the nesting of frameset for the division shown in Figure 5.22.

1. Create the initial frameset, defining two rows as shown below:

```
<FRAMESET Rows= "85, *">
    <FRAME Src= "sampleframe1.html">
</FRAMESET>
```

This code horizontally divides the window into two frames and the first row is reserved for the file *sampleframe1.html*.

2. The second row is left free. Now we will divide it into two frames vertically. So, instead of a second <FRAME> tag, insert another opening <FRAMESET> tag as shown below:

```
<FRAMESET Rows= "85, *">
    <FRAME Src= "sampleframe1.html">
    <FRAMESET Cols= "220, *">
    </FRAMESET>
</FRAMESET>
```

This code will divide the second row into two columns. Now, we can add two <FRAME> tags within the inner frameset and complete the HTML code as given in Example 5.17. The resultant web page is shown in Figure 5.23. It is assumed that the three HTML pages are already created as in Figure 5.23.

Example 5.17: To implement the concept of nested frameset

```
<HTML>
    <HEAD>
        <TITLE> nesting frames </TITLE>
    </HEAD>
    <FRAMESET Rows= "85, *">
        <FRAME Src= "sampleframe1.html">
        <FRAMESET Cols= "200, *">
            <FRAME Src= "sampleframe2.html">
            <FRAME Src= "sampleframe3.html">
        </FRAMESET>
    </FRAMESET>
</HTML>
```

5.5.5 <NOFRAMES> tag

Earlier browsers did not support frames. In such a situation, the browser is expected to respond to the user in some way or the other. The tag pair **<NOFRAMES>** and **</NOFRAMES>** is used to display some text content in the window if the browser is unable to support frames. The following code illustrates the use of <NOFRAMES> tag.

Fig. 5.23: Nested framesets

```
<HTML>
    <HEAD> <TITLE> A simple Frameset </TITLE>
    </HEAD>
```

```
    <FRAMESET Rows= "20%, 30%, 50%">
        <FRAME Src= "sampleframe1.html">
        <FRAME Src= "sampleframe2.html">
        <FRAME Src= "sampleframe3.html">
    </FRAMESET>
    <NOFRAMES>
        <P> Your browser doesnot support frames.<BR>
            Click <A Href="index.htm">here... </A></P>
    </NOFRAMES>
</HTML>
```

Here, if the browser does not support <FRAMESET> then a message "Click here…" will be displayed with an alternate link to the file *index.html*.

Know your progress

1. <FRAMESET Rows="100,*"> divides the frame into ____ sections.
2. List any three attributes of <FRAME> tag.
3. What is nested frameset?
4. What is the use of <NOFRAME> tag?
5. No <BODY> section is needed for frameset page. Say TRUE or FALSE.

5.6 Forms in web pages

HTML Forms are required when we have to collect some data from the webpage viewer for processing. For example, we use www.hscap.kerala.gov.in for the admission to Class XI in Kerala. We enter information such as name, SSLC Register Number, grades, choice of course and school, etc. in this web page. It is an implementation of HTML Form.

A Form will take input from the web page viewer and then it will post the data to a back-end application such as Common Gateway Interface (CGI), Active Server Pages (ASP), PHP, etc. for processing. We will study about these technologies in the next chapter. A form consists of two elements: a **<FORM>** container and any number of Form controls within that container. There are various Form controls like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.

5.6.1 <FORM> tag

A **<FORM>** tag provides a container for creating a Form. An HTML Form starts with <FORM> and ends with </FORM> tag. A web browser can only gather information through Forms. We must provide some back-end application to handle (save, modify, etc.) the data collected. We use Form handlers like CGI, JavaScript or PHP script for that purpose. All of the input elements associated with a single Form are processed by the same Form handler. A Form handler is a program on the web server that manages the data sent through the Form. The form handler is specified as the value for the Action attribute of <FORM> tag. The concept of server and other associated technologies will be discussed in the next chapter.

Most frequently used attributes of <FORM> are:

1. **Action**: It specifies the URL of the Form handler, which is ready to process the received data.
2. **Method**: It mentions the method used to upload data. The most frequently used Methods are Get and Post.
3. **Target**: It specifies the target window or frame where the result of the script will be displayed. It takes values like _blank, _self, _parent, etc. The main values of Target attribute are given in Table 5.4.

Value	Description
_blank	Opens the linked document in a new browser window
_self	Opens the linked document in the same frame as the link
_parent	Opens the linked document in the parent frameset
_top	Opens the linked document in the main browser window, replacing any frame present
name	Opens the linked document in the window with the specified name

Table 5.4: Values for Target attribute

A common use for JavaScript (or any client side scripting language) is to verify that users have filled in all the required fields in a Form and/or to check whether the input data is valid or not, before the browser actually submits the Form to the form handler on the web server.

Form controls

There are different types of Form controls that we can use to collect data using HTML Form. These controls include Text box, Password, Check Box, Radio Button,

Text Area, Select Box, Submit and Reset Button.We can create most of these controls in any HTML Form using **<INPUT>** tag.

5.6.2 <INPUT> tag

The visible part of a Form is a set of controls to accept the input from the viewers. Different types of input can be selected based on the nature of input. The **<INPUT>** tag is an empty tag that can be used to make different types of controls such as Text Box, Radio Button, Submit Button etc. The **Type** attribute determines the type of control created by this tag.

Attributes of <INPUT> tag

1. **Type**: This attribute determines the control type created by the <INPUT> tag. The main values of Type are given in Table 5.5.

Value	Description
text	creates a text box
password	same as text box. But here characters are represented by coded symbols such as asterisk.
checkbox	creates a checkbox where user can enter Yes or No values (check or uncheck).
radio	similar to checkbox but is used to select a single value from a group of values. When multiple radio buttons are assigned with the same value for Name attribute, users can select only one button at a time. When the user changes the selection, the one that is already selected becomes deselected.
reset	a special button used to clear all the entries made in the form and to bring it to the initial state.
submit	another special button used to submit all the entries made in the form to the server.
button	creates a standard graphical button on the form. We can call functions on clicking this button.

Table 5.5: Values of Type attribute

2. **Name**: It is used to give a name to the input control. When the Form is submitted, the data values are passed to the server along with the corresponding name of the control.
3. **Value**: It can be used to provide an initial (default) value inside the control.
4. **Size**: This attribute sets the width of the input text in terms of characters. It is applicable only to the input types text and password.

5. **Maxlength**: It limits the number of characters that the user can type into the field. It is also applicable only to the text and the password.

A password field helps in hiding the content visually. It doesn't encrypt or scramble the information in any way.

The HTML code given in Example 5.18 creates an HTML Form to input a name and password. Figure 5.24 shows the resultant web page.

Example 5.18: To create an HTML Form

```
<HTML>
<HEAD> <TITLE> Login </TITLE> </HEAD>
<BODY Bgcolor= "Pink">
    <FORM Action= "login.php" Method= "post">
        <P> Name:    
        <INPUT Type= "text" Name= "firstname" Size= "30"
               Maxlength= "25"> </P>
        <P>Password:
        <INPUT Type= "password" Name= "Psswd" Size= "30"
               Maxlength= "25"> </P> <BR>
                       
         
        <INPUT Type= "reset" Value= "Clear">
        <INPUT Type= "submit" Value= "Send">
    </FORM>
</BODY>
</HTML>
```

In Example 5.18, a file *login.php* is assigned to the `Action` attribute of the `<FORM>` tag. It means that the data entered through this Form will be captured and processed by the application stored in the file *login.php*. Also observe the use of HTML entity ` ` to provide non break spaces by referring to the code and Figure 5.24.

Fig. 5.24: HTML Form with text and password

Let us design an HTML Form to input data with the help of radio button (or option button) and check boxes. You might have heard that radio buttons are used to specify only one option from a set of alternatives, and that check boxes are used to mark one or more items in a given list.

The code given in Example 5.19 creates a Form to specify the gender and hobbies of a person with the help of radio buttons and check boxes. Figure 5.25 shows the corresponding web page.

Example 5.19: To create an HTML Form with radio buttons and check boxes

```
<HTML>
<HEAD> <TITLE>Checkbox Radio Button Control</TITLE> </HEAD>
<BODY Bgcolor= "#E9BEE5">
    <FORM> <BR>Sex:       
        <INPUT Type= "radio" Name= "sex" Value= "male"> Male
        <INPUT Type= "radio" Name= "sex" Value= "female"> Female
        <BR> <BR>Hobbies:
        <INPUT Type= "checkbox" Name= "Hobby" Value= "Games">
        Playing Games
        <INPUT Type= "checkbox" Name= "Hobby"
               Value="WatchingTV"> Watching TV
        <INPUT Type= "checkbox" Name= "Hobby" Value= "Reading">
        Reading
    </FORM>
</BODY>
</HTML>
```

Fig 5.25:Form with checkbox and radio button

5.6.3 <TEXTAREA> tag

We have seen single line text entry facility in a Form with the statement `<INPUT Type = "text">` (refer Example 5.18 and Figure 5.24). But in practice, there are occasions where we need to enter multiple lines of text in a Form. Postal address is an example. The container tag **<TEXTAREA>** can be used for this purpose. The area

between the tag pair gives space for multi line text depending on the values given to the attributes. The main attributes of <TEXTAREA> tag are:

1. **Name**: It is used to give a name to the control.
2. **Rows**: It specifies the number of rows in a text area control.
3. **Cols**: It indicates the number of columns in a text area, i.e., number of characters in a line.

Consider the following code segment and observe the usage of <TEXTAREA> tag:

```
<FORM Action= "guestbook.php" Method= "post">
    <TEXTAREA Rows= "10" Cols= "30" Name= "address">
        Enter your address.
    </TEXTAREA>
</FORM>
```

On completing the code and opening it, we can see a page as shown in Figure 5.26. Any text we include between tag pair <TEXTAREA> and </TEXTAREA> appears in a text box. When the user enters any data in the text box, it overrides default text.

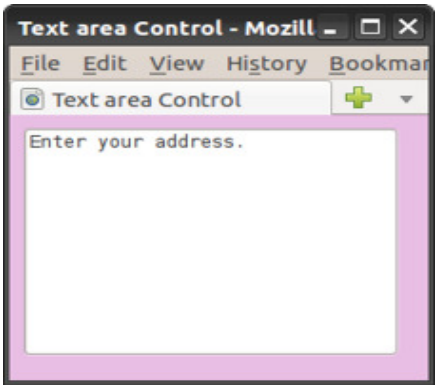


Fig. 5.26: HTML Form with text area

5.6.4 <SELECT> tag

A select box, also called dropdown box, provides a list of various options in the form of a dropdown list, from where a user can select one or more options. Select box is helpful when a number of options are to be displayed in a limited space. The options in the list are specified using **<OPTION>** tag, which will be contained in the **<SELECT>** tag pair.

The container tag pair <SELECT> and </SELECT> encloses a select box. The main attributes of <SELECT> tag are:

1. **Name**: It gives a name to the control, which is sent to the server to be recognized and to get the value.
2. **Size**: This can be used to present a scrolling list box. Its value will decide whether the select box should be a drop down list or a list box. If the value is 1, we get a dropdown list (or combo box).
3. **Multiple**: It allows users to select multiple items from the menu.

Now let us discuss the role of <OPTION> tag in select boxes. It is an empty tag placed inside the container tag <SELECT> and </SELECT>. It lists out the options provided in the select box. The main attributes of <OPTION> tag are:

1. **Selected**: This attribute is used to indicate default selection.
2. **Value**: It is used here to allow the submission of a value that differs from the content of the <OPTION> tag. If it is not present, the content is used as the value.

The HTML code given in Example 5.20 creates a web page that contains a drop down list. Note that the value of the Size attribute of <SELECT> tag is 1 and there are four options inside the select box.

Example 5.20: To create an HTML Form with a drop down list

```
<HTML>
<HEAD> <TITLE> Drop down list </TITLE> </HEAD>
<BODY Bgcolor= "#E9BEE5">
    <FORM Action= "guestbook.php" Method= "post">
        <P> Nationality:
        <SELECT Name= "Nationality" Size= "1">
            <OPTION Value= "Indian" selected> Indian
            <OPTION Value= "British"> British
            <OPTION Value= "German"> German
            <OPTION Value= "Srilankan"> Srilankan
        </SELECT>
    </FORM>
</BODY>
</HTML>
```

Figure 5.27 shows the dropdown list obtained on clicking the combo button.

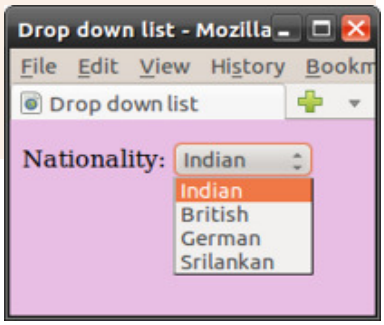


Fig. 5.27: Select box in a Form

Let us do

Modify the code given in Example 5.20 with different values for the Size attribute and predict the changes in the select box.

5.6.5 Grouping Form data with <FIELDSET> tag

Sometimes grouping related controls in a Form become a necessity. The **<FIELDSET>** element helps in grouping related data in a Form. By using <FIELDSET> we can divide a Form into different subsections, each subsection containing related elements. To identify each <FIELDSET>, we can use the **<LEGEND>** tag. The <LEGEND> tag defines a caption for the <FIELDSET> element. It is a container tag.

Now let us create a Form with all the elements discussed so far, to submit the details such as name, age, sex, address, hobbies etc. of a student. The code given in Example 5.21 can be used for this so that an HTML Form as shown in Figure 5.28 will be obtained.

Example 5.21: To create an HTML Form to submit the details of a student

```
<HTML>
<HEAD> <TITLE> FormResume </TITLE> </HEAD>
<BODY Bgcolor= "#E9BEE5">
    <CENTER ><H3>Enter your details</H3></CENTER>
    <FORM Action= "guestbook.php" Method= "get">
        Name: 
        <INPUT Type= "text" Name= "first_name" Size= "20"
               Maxlength= "20" Value= "First Name Here"><BR><BR>
        Age:
        <INPUT Type="text" Name="age" Size="3" Maxlength="3"><BR>
        Sex:     
        <INPUT Type="radio" Name="sex" Value="male"> Male
        <INPUT Type="radio" Name="sex" Value="female"> Female
        <FIELDSET>
            <LEGEND>Nationality</LEGEND>
            <SELECT Name= "Nationality" Size= "4">
                <OPTION Value= "Indian" Selected> Indian
                <OPTION Value= "British"> British
                <OPTION Value= "German"> German
                <OPTION Value= "Srilankan"> Srilankan
            </SELECT><BR><BR>
            Nativity:
            <INPUT Type= "text" Name="State" Size="15"><BR><BR>
            District: 
            <INPUT Type= "text" Name= "District" Size= "15">
        </FIELDSET><BR>
        Hobbies:
        <INPUT Type= "checkbox" Name= "Hobby" Value= "games">
               Playing Games
        <INPUT Type= "checkbox" Name= "Hobby"
                Value= "WatchingTV"> Watching TV
        <INPUT Type= "checkbox" Name= "Hobby" Value= "Reading">
               Reading<BR><BR>
```

```
        <TEXTAREA Rows= "5" Cols= "25" Name= "address">Address
        </TEXTAREA><BR><BR>
        <INPUT Type= "submit" Value= "submit">
        <INPUT Type= "Reset" Value= "reset">
    </FORM>
</BODY>
</HTML>
```

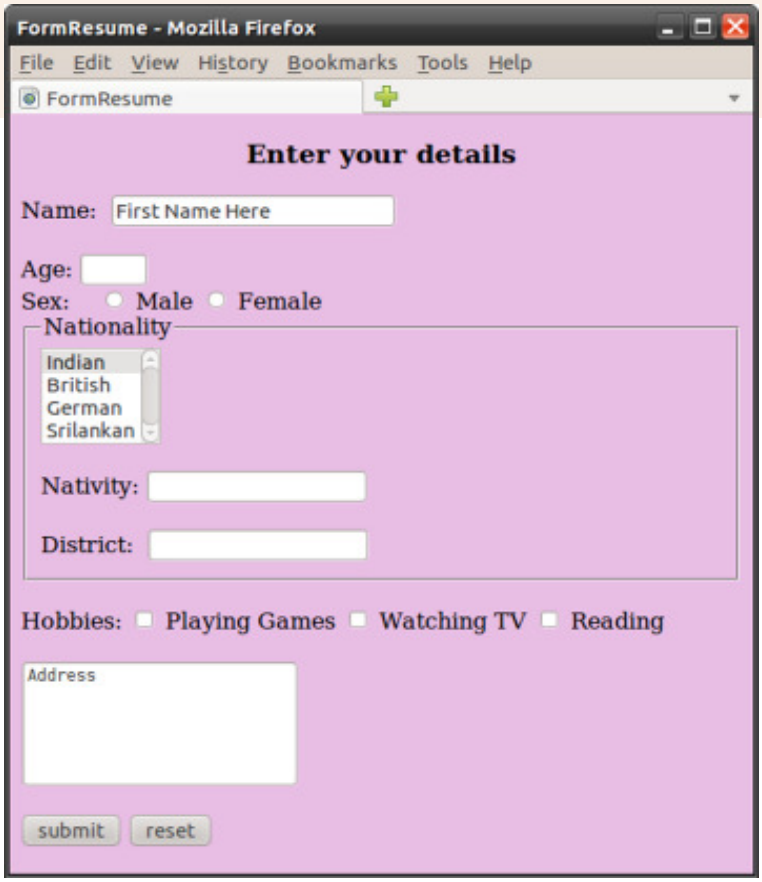


Fig 5.28:An HTML Form to submit the details of a student

5.6.6 Form submission

When we click the submit button on the Form, it will send the collected input data to the server, a computer capable of processing the received data. In a web server, there are special server side programs for processing Form data coming from the browser of a client (a computer, a tab or a mobile phone through which we submit the data).

As mentioned earlier, the `Action` attribute defines the action to be performed when the Form is submitted. Normally it assigns the URL of the server side program to process the Form data. The common way to submit a Form to a server is by using a **submit** button. In Example 5.21, a server-side script *guestbook.php* is specified to handle the submitted form in the `<FORM>` tag.

The Method attribute of the <FORM> tag specifies the HTTP method (get or post) to be used when submitting the forms.

Know your progress

1. HTML provides _____ to input data through web pages.
2. Name the two tags used within <FORM> to enter text data.
3. How do radio button control and check box control differ?
4. Which tag is used to group data within the Form?
5. Name the tag used within <FORM> to input data.

5.7 Overview of HTML 5

HTML 5 is the major revision of the HTML standard after HTML 4.01. HTML5 was developed jointly by World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG). The new standard incorporates features like video playback, drag-and-drop etc.

The latest versions of all leading browsers including Google Chrome, Mozilla Firefox, Apple Safari, Opera and Internet Explorer support HTML5. Mobile web browsers that are pre-installed in iPhones, iPads, Android phones, etc. also support HTML5. HTML5 is designed, as much as possible, to be backward compatible with existing web browsers. The logo of HTML5 is given in Figure 5.29.

Fig. 5.29: HTML5 logo

Some new tags introduced in HTML5 are given below:

1. **<VIDEO> and <AUDIO>**: These tags offer the facility to easily embed media into HTML documents.
2. **<CANVAS>**: This tag gives us an easy and powerful way to draw graphics, build charts and graphs and customise graphics.
3. **<HEADER> and <FOOTER>**: The <HEADER> element specifies a header for a document or section. The <FOOTER> element specifies footer for a document or section. A footer typically contains the author of the document, copyright information, contact information, etc. We can have several <HEADER>, <FOOTER> elements in one document.
4. **<ARTICLE> and <SECTION>**: These tags are used to create articles and sections within a webpage. An article is an independent, stand-alone piece of discrete content like a blog post, or a news item. Article represents a full block of content. Section is used as a way of sectioning a page into different subject areas, or sectioning an article. <ARTICLE> and <SECTION> tags, if properly used will increase search engine visibility of the webpage.

5. **<OUTPUT>**: This tag represents the result of a calculation usually performed by a script.
6. **<DETAILS>**: Sometimes website needs to have an expanding/collapsing block of text. With <DETAILS> tag it is easier to make a simple block that expands and collapses the content when the header is clicked.
7. **<FIGURE>** and **<FIGCAPTION>**: <FIGURE> is a container for content like images, and <FIGCAPTION> element represents a caption or legend for a figure. It comes inside the <FIGURE> tag.
8. **<PROGRESS>** and **<METER>**: <PROGRESS> and <METER> are similar. The <PROGRESS> tag represents the progress of a task. It is useful for things like displaying the progress of a file upload. <METER> tag is used to display a scalar measurement within a known range, like showing hard drive usage.

Let us conclude

In this chapter, we have gone through some of the advanced features of HTML. Different kinds of lists are discussed to make the text more presentable. The beauty of the web pages can be enhanced by including marquees, images, audio and video. We have also discussed the importance of hyper linking and familiarised with various kinds of hyperlinks. Creating a table with the tags <TABLE>, <TR>, <TH>, and <TD>, and their attributes were discussed. The importance of Rowspan and Colspan attributes are identified. We can divide the browser window into different frames using framesets so that different pages can be opened and viewed at the same time. We have also seen that division of browser window can be done in different ways with the concept of nested framesets. We have discussed how Target attribute is used to create link between different frames of the browser window. We have also identified the utility of Form in the web page to submit data to the server using online facility. We have seen various elements such as text box, password, radio button, text area, select box, etc. that facilitate the input of data in different ways. We have just mentioned the concept of client side programs and server side programs. So, this chapter is a stepping stone to the next chapters in which we will discuss how the input data in the Form is verified and how the data is stored or processed in the server.

Let us practice

1. Wrtie an HTML code for a web page of a district in Kerala with the following details and features:
   - A heading followed by a paragraph of 3 sentences about the district using text formatting tags and attributes.
   - A list of the tourist places in the district.
2. Wrtie an HTML code for a web page for your school with the following details and features:
   - A heading followed by a paragraph of 3 sentences about the district using text formatting tags and attributes.
   - Include a list of five co-curricular activites like NCC, NSS, Clubs, etc.
3. Wrtie an HTML code for a web page to show the following details:

Components of a Computer

- **Hardware**
  1. I/O Devices
  2. RAM
  3. Hard Disk & DVD Drive
- **Software**
  1. Operating System
  2. Application Programs

4. Write an HTML code to present some details about Onam - the festival of Kerala, in a web page with the following features:
   - A heading with attractive font characteristics.
   - An image of "Vallam Kali" (boat race) in the background of the page.
   - Internal links to any two of the traditional events like "Pookkalam", "Thumpi Thullal", "Thiruvathira", "Onavillu", "Vallam kali", "Kummatti", "Pulikali", etc.
5. Write HTML codes to create two web pages to show some information about the higher secondary and high school sections of your school. Create another web page to divide the browser window horizontally into two. In the first frame, a brief introduction of the school and two links are to be provided - one for HSS and the other for HS. On clicking these links the respective web page is to be opened in the second frame.

6. Write an HTML code to show the following table in a web page and also provide an external link to the website of Kerala Police given below the table:

Road Accidents in Kerala during 2012 - 2014

Year	Total Number of		
	Cases	Persons Killed	Persons Injured
2012	36174	4286	41915
2013	35215	4258	40346
2014	36282	4049	41096

Data Source: www.keralapolice.org

7. Write an HTML code to display an application form as shown below:

APPLICATION FOR THE BEST STUDENT AWARD

Name: Sex: Male ○ Female ○

Class & Division: Select

Total Grade Point in Class XI:

Average Grade Point in Termly Exams in Class XII:

Cocurricular Activities:

NCC ☐ NSS ☐ Sports ☐ Arts ☐ Literary ☐

Other Achievements:

Let us assess

1. Find the errors in the following and correct it.

a.
```
<UL Type = "i" Start = 3>
```

b.
```
<IMG Src = "Myschool.jpg" Size  = "50" >
```

c.
```
<HTML>
      <HEAD><TITLE><HEAD></TITLE>
      <BODY> this is the body of the HTML
        document</BODY>
</HTML>
```

2. Rohit created a table in HTML but a border was not visible. What could be the reason?
3. _____ attribute is used with <A> tag to display the linked page in a specified frame.
4. Your computer teacher asked you to prepare a list of your best friends in a webpage. Which tag will you prefer? Write HTML code segment for this list.
5. Name the possible values of type attribute of <OL> tag.
6. Write the attributes of <UL> tag.
7. How do you create a list using upper case letters for numbering?
8. Suppose you want to create a list using lower case letters for numbering. How can this be made possible?
9. How can we create a list starting from 6 onwards?
10. Write the HTML code for creating the following webpage:

ABC Pvt. Ltd.
Kerala

1. Health Care
2. Baby Products
   - Toys
   - Dress
3. Ladies Wear
   - Kurthas
   - Jeans

11. Varun is creating a web page. He wants to create a link on the word 'sample' to a file named s*ample.html* that is stored in a subdirectory *Exam* in *D drive*. Write the HTML command for this purpose.
12. Name the tag which has Noshade attribute.
13. Sunil developed a personal website, in which he has to create an e-mail link. Can you suggest the protocol used to achieve this link.
14. Shahir wants to connect his webpage to www.gmail.com. Write the tag and attribute required for this.
15. Mention the characteristics of two types of hyperlinks available in HTML.
16. Differentiate between Cellspacing and Cellpadding.
17. Differentiate between Text control and Textarea control used in Form.
18. Action and _____ are the main attributes of the <FORM> tag.

19. Say true or false:
    a. The default value of `Align` attribute of `<TABLE>` tag is `center`.
    b. `<FRAME>` is a container tag.
    c. Scrolling prevents users from resizing the border of a specific frame.
20. Which tag does allow partitioning of the browser window into different sections?
21. List down the main attributes of `<FRAME>` tag.
22. Create a frameset dividing the page into two sections vertically. Give names of your favorite football players in left frame. The profile of selected players should appear in the right Frame.
23. Predict the output of the following code:

```
<HTML>
<HEAD><TITLE>A simple table</TITLE></HEAD>
<BODY>
<TABLE border="1" Cellspacing= "1" Cellpadding= "10">
  <TR><TD> 1 </TD><TD> 2 </TD><TD> 3 </TD></TR>
  <TR><TD> 4 </TD><TD> 5 </TD><TD> 6 </TD></TR>
  <TR><TD> 7 </TD><TD> 8 </TD><TD> 9 </TD></TR>
</TABLE>
</BODY>
</HTML>
```

24. Write an HTML code to create the given table:

1	2	3
4	5	6
7		

25. Write an HTML code to accept your e-mail, address, phone number and password.
26. Write an HTML code to create a list of 3 bikes of your choice in a frame. Link each one to display the description with a picture in another frame.
27. Raju created a web page as follows. But he is unable to view any tabular format in the web page, when it is displayed in the browser. Find out the reason for it and correct it.

```
<HTML>
<HEAD><TITLE> My Page </TITLE></HEAD>
<BODY>
<TABLE><TR><TH>Roll No. </TH><TH> Name </TH></TR>
<TR><TD>1 </TD><TD> Huda </TD></TR>
<TR><TD>2 </TD><TD>Bincy</TD></TR>
</TABLE>
</BODY>
</HTML>
```

6 Client Side Scripting Using JavaScript

Significant Learning Outcomes

After the completion of this chapter, the learner

- distinguishes the use of client side and server side scripting language.
- explains the need of client side scripting language.
- identifies the importance of JavaScript as the client side scripting language.
- uses JavaScript functions in a web page.
- explains different data types in JavaScript.
- uses correct variables in JavaScript.
- uses appropriate control structures in program codes.
- uses appropriate built-in functions in JavaScript.
- explains the method to access Document Elements using javaScript.
- creates JavaScript functions that handle values in text boxes and combo boxes.

In the previous chapters we learned to create different types of web pages containing texts and graphics. Since this is the world of the Internet, most of us might have visited various websites in the Internet for different purposes. These web pages contain features that we have not learned. Creating such types of web pages requires the knowledge of scripting languages. Different scripting languages are used at the client side and server side. Even though JavaScript and VB Script are the two client side scripting languages, JavaScript is the most commonly used scripting language at the client side. The main reason for this is that the JavaScript is supported by all browsers, but VB Script is not. Since web pages are to be viewed by a large number of people over the Internet, we cannot expect that the user will use a specific browser itself. Hence, a web page should be made browser independent as much as possible. In this chapter, we will learn how JavaScript is used in a web page. Since all of us are familiar with C++, it is easy to understand JavaScript because the JavaScript follows the same syntax of C++.

JavaScript was developed by Brendan Eich for the Netscape Browser. The original name of JavaScript was Mocha. In 1995, when it was deployed in the Netscape browser version 2.0, the name was changed to JavaScript. In early days, only Netscape browser supported JavaScript. But due to the wide popularity of JavaScript, Internet Explorer provided support to JavaScript in 1996. Now, almost all browsers in the world support JavaScript.

Brendan Eich

6.1 Getting started with JavaScript

In Chapter 4 Web Technology, we discussed the use of client side and server side scripting languages. Client side scripting languages are used for validations of data at the client side itself. This reduces network traffic and workload on the server. Server side scripting languages are executed at the server and the web page produced is returned to the client browser. A server may store a large volume of data in the form of a database. So a server side scripting language may have to interact with this large volume of data for different purposes, but a client need not. So, the languages and commands used at the client side and server side are different.

Let us learn the basics of JavaScript as a client side scripting language in this chapter. Using JavaScript, we can embed program segments in different places in an HTML page. `<SCRIPT>` tag is used to include scripts in HTML page.

<SCRIPT> Tag

`<SCRIPT>` tag is used to include scripting code in an HTML page. The **Language** attribute of `<SCRIPT>` tag is used to specify the name of the scripting language used. Here we use JavaScript as the client side scripting language. Therefore, we will give `JavaScript` as the value for `Language` attribute.

The `<SCRIPT>` tag can be used in an HTML page as follows.

```
<SCRIPT Language= "JavaScript">
.....................
.....................
</SCRIPT>
```

The identifiers in JavaScript are case sensitive. It is common in JavaScript, to use camelCase names for identifiers. Examples are firstName, checkData, etc. CamelCase is a naming convention which is used when a single word is formed using multiple words. When the first letter of each word is capitalised, it is called UpperCamelCase and when the first letter of each word except the first word is capitalised, it is called lowerCamelCase. CamelCase improves readability of the word.

Now let us consider the following HTML code given in Example 6.1.

Example 6.1: To create a web page using JavaScript

```
<HTML>
<HEAD> <TITLE>Javascript - Welcome</TITLE> </HEAD>
<BODY>
    <SCRIPT Language= "JavaScript">
        document.write("Welcome to JavaScript.");
    </SCRIPT>
</BODY>
</HTML>
```

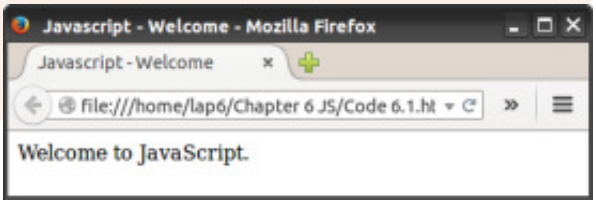


Fig. 6.1: A web page using JavaScript

The above code can be typed in any text editor. We use Geany editor that we used for creating HTML pages in the previous chapters. Save the file as 'Code 6.1.html'. Note that even if we use JavaScript in an HTML page, it is saved with **.html** extension. Then, open the file in any browser. We will get the web page as shown in Figure 6.1. Note that the statement `document.write` is written in lowercase letters. This is because JavaScript is a case sensitive scripting language. The keywords in JavaScript are all in the lowercase.

In the above HTML file, **`document.write()`** is a JavaScript command that includes a text in the body section of the HTML page. That is, the above HTML file has the same effect of the following HTML code.

Example 6.2: To create a web page using HTML

```
<HTML>
<HEAD> <TITLE>Javascript - Welcome</TITLE> </HEAD>
<BODY>
    Welcome to JavaScript.
</BODY>
</HTML>
```

Compare the above two examples (Example 6.1 and Example 6.2). In the second HTML code, the text `Welcome to JavaScript` is directly placed inside the body section, whereas in the first one, "`Welcome to JavaScript`" is included in the body section using the JavaScript method `document.write()`. Actually, `document` represents the body section of the web page. Therefore, `document.write()` is a JavaScript function that will include a text in the body

section of the web page. It is very important to note that like C++, every statement in JavaScript also ends in a semicolon.

<SCRIPT Language= "JavaScript"> tells the browser that the code that follows is a JavaScript code. Now let us see how a browser handles the JavaScript code. The script code is interpreted at runtime by the JavaScript engine. Every browser has a JavaScript engine. JavaScript engine is a virtual machine for executing JavaScript code. When the browser sees a JavaScript code, it is passed to the script engine for processing. The script engine executes the code. If an HTML page does not contain any JavaScript code, the browser alone is able to render the HTML page. But if there is a JavaScript code, the browser takes the help of script engine also to render the HTML page. Hence, an HTML file without JavaScript is always rendered faster than that with JavaScript code.

Let us consider the code given in Example 6.3, that mixes JavaScript codes with HTML tags. The output of the HTML page is given in Figure 6.2.

Example 6.3: To create a web page containing heading tags

```
<HTML>
<HEAD> <TITLE>Javascript - Welcome</TITLE> </HEAD>
<BODY>
    <H1>
    <SCRIPT Language= "JavaScript">
        document.write("This is in H1 Head");
    </SCRIPT>
    </H1>
    <BR>
    <H2>
    <SCRIPT Language= "JavaScript">
        document.write("This is in H2 Head");
    </SCRIPT>
    </H2>
</BODY>
</HTML>
```

Fig. 6.2: Web page containing heading tags

In the above code, we used scripting codes in between the HTML tags more than once. Thus you can use the script codes any number of times in between the HTML tags. Wherever we use script codes, do remember to place them in between <SCRIPT> and </SCRIPT> tags.

All web browsers allow users to enable or disable the execution of JavaScript in a web page. By disabling the JavaScript, we are actually disabling the JavaScript engine in that browser. In Mozilla, the JavaScript can be enabled or disabled by choosing Tools -> Options -> Content -> Enabled JavaScript. In Google Chrome, it can be done by selecting the option 'Do not allow any site to run JavaScript' which is available in the Content Settings window. This window can be accessed from the menu Tools -> Settings -> Show Advanced Settings -> Content Settings.

If JavaScript is disabled in a web browser, the browser will not execute the script code at all. So, the browser will simply ignore the content written inside `<SCRIPT> .... </SCRIPT>` tags.

Performance of a browser mainly depends on the performance of its script engine. One can hardly see a dynamic web page that does not use any JavaScript. All broswers are competing with each other for the development of better, fast and powerful JavaScript engines.

Know your progress

1. Name an attribute of `<SCRIPT>` tag.
2. In JavaScript _____ function is used to print a text in the body section of an HTML page.
3. _____ is the value given to the language attribute of `<SCRIPT>` tag to specify that the code follows is JavaScript code.
4. What is the use of JavaScript engine?
5. State whether the following statements are true or false.
   a. Java Script is the only client side scripting language.
   b. `<SCRIPT>` tag is used to include client side scripting language in an HTML page.
   c. An HTML file can contain only one `<SCRIPT>` tag in it.
   d. We can mix JavaScript code with HTML code.
   e. The JavaScript code must be placed within `<SCRIPT>` and `</SCRIPT>` tags.
   f. Every JavaScript statement ends in a semicolon.

Let us do

Write an HTML code to create the following web page by using JavaScript inside the body section of the page. That is, the body section should be as shown below.

```
<BODY>
<SCRIPT Language= "JavaScript">
.................
.................
</SCRIPT>
</BODY>
```

6.2 Creating functions in JavaScript

We have already learned about functions in C++. In JavaScript also functions are defined and called almost in the same way as in C++. A function is a group of instructions with a name. JavaScript has a lot of built-in functions that can be used for different purposes. We will cover some of these functions later in this chapter. Besides, these built-in functions, we can define our own functions also. The biggest advantage of using a function is that if we need to execute a piece of program code more than once in a web page, the code has to be written only once within the function. We can call the function any number of times to execute it. Look at the following code:

```
function print()
{
    document.write("Welcome to JavaScript.");
}
```

Here, the keyword `function` is used to define a function. The word `print` that follows, is the name of the function. The function name can be any valid identifier. JavaScript uses the same rules for naming identifiers as in C++. Here the `print()` function contains only one statement. We can include any number of statements inside a function.

Defining a function does not execute the function automatically. It must be called for its execution. This means that even if we define a function in a web page and do not call the function, the function will not be executed at all. The function can be called by using the function name as follows:

```
print();
```

Note the semicolon after the function name. Now let us make use of the above `print()` function in an HTML page.

Example 6.4: To create a web page containing print() function

```
<HTML>
<HEAD> <TITLE>Javascript - Functions</TITLE>
    <SCRIPT Language= "JavaScript">
        function print()
        {
            document.write("Welcome to JavaScript.");
        }
    </SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Note that nothing is written inside the body section of the HTML page. This code will not display 'Welcome to JavaScript' in the browser window. This is because even though the function is defined, it is not at all called from any place in the page. Therefore, the function will not be executed at all. Hence nothing will be displayed on the screen. The body section of the HTML page in Example 6.4 should be modified as follows to get the output as shown in Figure 6.3.

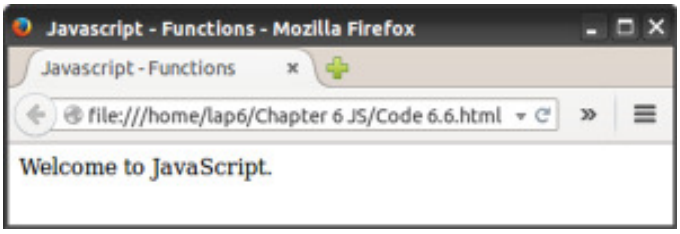


Fig. 6.3: Web page using functions

```
<BODY>
<SCRIPT Language= "JavaScript">
    print();
</SCRIPT>
</BODY>
```

Now let us see the syntax of a JavaScript function.

```
function function_name()
{
    statements;
}
```

Here the line `function function_name()` is called the **function header** and the code within the braces `{ and }` is called **function body**. The main difference from C++ is that, in JavaScript there is no return type, whereas in C++ the function has a return type. In JavaScript also, we can return some value from a function as in C++. Since this chapter is meant to give only a basic idea about the use of JavaScript, we do not discuss them for the time being. Another difference is that, C++ does not use the keyword `function` to define a function, but JavaScript does.

We might have noted that the function is defined within the head section of the HTML page. It is not necessary to define the function within the head section itself. We can define a function in the body section also as given below:

```
<BODY>
    <SCRIPT Language= "JavaScript">
        function print()
        {
            document.write("Welcome to JavaScript.");
        }
        print();
    </SCRIPT>
</BODY>
```

The above code also produces the same output as in Figure 6.3. Note that even if the function is defined within the body section, it must be called for its execution. For example, the following code does not display anything on the screen:

```
<BODY>
<SCRIPT Language= "JavaScript">
function print()
{
    document.write("Welcome to JavaScript.");
}
</SCRIPT>
</BODY>
```

Even though a function can be defined anywhere in an HTML page, it is always better to include the function definition within the head section. Now consider the following code in which the `print()` function is called twice. It will give the output as shown in Figure 6.4.

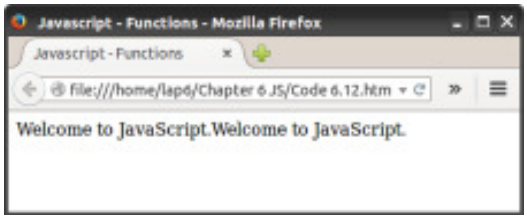


Fig. 6.4: Web page using two print functions without break

```
<BODY>
    <SCRIPT Language= "JavaScript">
        print();
        print();
    </SCRIPT>
</BODY>
```

When we call the function twice, it just places the content 'Welcome to JavaScript' twice in the body section, where the function is called. Hence, the body section in the above code has the same effect as the following code.

```
<BODY>
    Welcome to JavaScript.Welcome to JavaScript.
</BODY>
```

If the message should be displayed in the two different lines, we must use <BR> tag.

```
<BODY>
    Welcome to JavaScript.<BR>Welcome to JavaScript.
</BODY>
```

In order to create the same effect using JavaScript, the function can be modified as shown below.

```
<SCRIPT Language= "JavaScript">
    function print()
    {
        document.write("Welcome to JavaScript.<BR>");
    }
</SCRIPT>
```

This HTML code will give the output as shown in Figure 6.5.

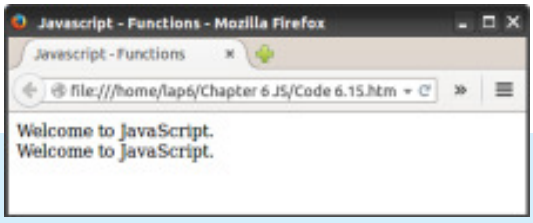


Fig. 6.5: Web page using two print functions

Know your progress

1. Say whether the following statements are true or false:
   a. A function is automatically executed when the browser opens the web page.
   b. A function is usually placed in the head section of a web page.
   d. A function can be called any number of times.
   e. Even though a function is defined within the body section, it will not be executed, if it is not called.
2. List the advantages of using functions in JavaScript.

Let us do

1. Consider the following two HTML codes (Code A and Code B) and try to predict the output.

Code A

```
<HTML>
<HEAD>
<SCRIPT Language= "JavaScript">
    function print()
    {
        document.write("Welcome to JavaScript");
    }
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT Language= "JavaScript">
    print();
    document.write("<BR>");
    print();
</SCRIPT>
</BODY>
</HTML>
```

Code B

```
<HTML>
<HEAD>
<SCRIPT Language= "JavaScript">
function print()
{
      document.write("Welcome to JavaScript");
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT Language= "JavaScript">
      print();
</SCRIPT>
<BR>
<SCRIPT Language= "JavaScript">
      document.write("<BR>");
      print();
</SCRIPT>
</BODY>
</HTML>
```

2. Given below is an HTML code to get the output as shown in the figure below. The code contains four functions namely `startGreen()`, `stopGreen()`, `startRed()` and `stopRed()`. These functions are called from different places from the body section of the HTML page. You can see that the body part in each function definition is blank except for the function `stopGreen()`. You have to complete the definition of all other functions so as to get the output as shown in figure.

```
<HTML>
<HEAD>
<SCRIPT Language= "JavaScript">
      function startGreen()
      {
            ..............
            ..............
      }
      function stopGreen()
      {
            document.write("</FONT>");
      }
      function startRed()
      {
            ..............
            ..............
      }
      function stopRed()
      {
            ..............
            ..............
      }
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT Language= "JavaScript">
      startGreen();
</SCRIPT>
This is in Green colour with size 5
<SCRIPT Language= "JavaScript">
      stopGreen();
</SCRIPT>
```

```
<SCRIPT Language= "JavaScript">
      startRed();
</SCRIPT>
This is in Red colour with size 3
<SCRIPT Language= "JavaScript">
      stopRed();
</SCRIPT>
<SCRIPT Language= "JavaScript">
      startGreen();
</SCRIPT>
This is in Green colour with size 5
<SCRIPT Language= "JavaScript">
      stopGreen();
</SCRIPT>
</BODY>
</HTML>
```

6.3 Data types in JavaScript

All programing languages classify data items into different categories for the effective utilisation of memory. We have learned that the basic data types in C++ are `int`, `char`, `float`, `double` and `void`. Besides these data types, there are type modifiers also in C++. JavaScript reduces this complexity by limiting the number of basic data types into 3. The following are the three basic data types in JavaScript.

Number

All numbers fall into this category. All positive and negative numbers, all integer and float values (fractional numbers) are treated as the data type number. Thus, 27, -300, 1.89 and -0.0082 are examples of number type data in JavaScript.

String

Any combination of characters, numbers or any other symbols, enclosed within double quotes, are treated as a string in JavaScript. That is, "Kerala", "Welcome", "SCHOOL", "1234", "Mark20", "abc$" and "sanil@123" are examples of string type data.

Boolean

Only two values fall in boolean data type. They are the values `true` and `false`. Note that the values are not in double quotes. If they are put inside double quotes, they will be considered as of string type. Not only that, like C++, JavaScript is also case sensitive. So we cannot use TRUE and FALSE to represent boolean values.

6.4 Variables in JavaScript

As we know, variables are used for storing values. In JavaScript, variables can be declared by using the keyword **var** as given below:

```
var  x;
```

Here x is the name of the variable. Like C++, a variable can be given any name, as you like, with only one restriction, that it must be a valid identifier. We have used the keywords `int`, `float`, `char`, etc. to declare different types of variables in C++. But, in JavaScript, the same keyword, **var** is used to declare all type of variables.

In JavaScript, merely declaring a variable does not define the variable. The variable definition is complete only when it is assigned a value. JavaScript understands the type of variable only when a value is assigned to that variable. Consider the following declaration.

```
var x, y;
x = 25;
y = "INDIA";
```

In the above example, the variable x is of number type and y is of string type. Even though we say that x is of number type and y is of string type, note that we do not specify them in code segments. The following example explains how JavaScript defines variables.

Example 6.5: To illustrate the use of variables

```
<HTML>
<HEAD> <TITLE>Javascript - Variables</TITLE> </HEAD>
<BODY>
<SCRIPT Language= "JavaScript">
   var a, b, c, d, e, f;
   a = 25;
   b = 18.5;
   c = "INDIA";
   d = true;
   e = "true";
   document.write("a is of type : ");
   document.write(typeof(a));
   document.write("<BR>b is of type : ");
   document.write(typeof(b));
   document.write("<BR>c is of type : ");
   document.write(typeof(c));
```

```
    document.write("<BR>d is of type : ");
    document.write(typeof(d));
    document.write("<BR>e is of type : ");
    document.write(typeof(e));
    document.write("<BR>f is of type : ");
    document.write(typeof(f));
</SCRIPT>
</BODY>
</HTML>
```

Here, the code uses the function `typeof()`, which is not familiar to you. As the name indicates, this function is used to find the type of a variable. We will study this function in detail later. The output of the above example is given in Figure 6.6.

Fig. 6.6: Web page displaying the use of variables

Note that the variable `f` is declared, but it is not given a value. Therefore, the script engine is unable to understand its type and so it is declared as `undefined`. In JavasScript, `undefined` is a special datatype to represent variables that are not defined using `var`.

Any number of variables can be declared by using a single `var` keyword. The variables should be separated by comma (`,`). Let us consider a Javascript function that makes use of variables.

Example 6.6: To create a web page to find the sum of two numbers

```
<HTML>
<HEAD><TITLE>Javascript - Variables</TITLE>
<SCRIPT Language= "JavaScript">
function add()
{
    var m, n, sum;
    m = 20;
    n = 10;
    sum = m + n;
    document.write("Sum = ");
    document.write(sum);
}
```

```
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT Language= "JavaScript">
    add();
</SCRIPT>
</BODY>
</HTML>
```

In the above example, the body section contains only one JavaScript statement - a call to the `add()` function. After executing the function, it will display the output as shown in Figure 6.7.

Fig. 6.7: Web page to find the sum of two numbers

6.5 Operators in JavaScript

Almost all operators in JavaScript are exactly similar to those in C++. Let us have a quick look at all of these operators.

6.5.1 Arithmetic operators

Table 6.1 shows the arithmetic operators used in JavaScript along with examples.

Operator	Description	Example	Value of y	Result (x)
+	Addition	x = y + 10	15	25
–	Subtraction	x = y – 10	15	5
*	Multiplication	x = y * 3	15	45
/	Division	x = y / 2	15	7.5
%	Modulus (division remainder)	x = y % 2	15	1
++	Increment	x = ++y x = y++	15 15	16 15
––	Decrement	x = --y x = y--	15 15	14 15

Table 6.1: Arithmetic operators

From the above table, we can see that all arithmetic operators work exactly as in C++.

6.5.2 Assignment operators

Table 6.2 displays the usage of various assignment operators in JavaScript.

Operator	Description	Example	Value of a	Value of b	Result (a)
=	Assignment	a = b	10	3	3
+=	Add and assignment	a+=b	10	3	13
-=	Minus and assignment	a-=b	10	3	7
*=	Multiply and assignment	a*=b	10	3	30
/=	Divide and assignment	a/=b	10	3	3.33
%=	Modulus and assignment	a%=b	10	3	1

Table 6.2: Assignment operators

It is very easy to understand the working of each operator from the above table. The 'Result' column gives the result after executing the statement in the 'Example' column with the given values of `a` and `b`.

6.5.3 Relational operators (Comparison operators)

Table 6.3 shows the various relational operators used in JavaScript, along with examples.

Operator	Description	Example	Value of a	Value of b	Result
==	Equal to	a==b	10	3	false
!=	Not equal to	a!=b	10	3	true
<	Less than	a<b	10	3	false
<=	Less than or equal to	a<=b	10	3	false
>	Greater than	a>b	10	3	true
>=	Greater than or equal to	a>=b	10	3	true

Table 6.3: Relational operators

From the table, it is clear that the result of a relational operation is either `true` or `false`. These operators compare the values on the two sides of the operator and give the result accordingly.

6.5.4 Logical operators

Table 6.4 shows the different logical operators in JavaScript along with examples.

Operator	Description	Example	Value of a	Value of b	Result
&&	AND	a && b	true	false	false
||	OR	a || b	true	false	true
!	NOT	!a	true		false

Table 6.4: Logical operators

From the above tables, it is clear that all these operators are exactly similar to C++. JavaScript provides a number of other operators also for different purposes. Since our discussion of JavaScript is limited only to this chapter, the discussion of those operators are beyond the scope of this book. Besides, the operators discussed above are enough to perform almost all the operations for a beginner. However, the following string operator will be of use to us in various situations. This operator is not available in C++.

6.5.5 String addition operator (+)

We have already seen that the operator + is used to add two numbers. The same operator + is used to add two strings also. Adding two strings means concatenating two strings.

Look at the following example code.

```
var x, y;
x = "A good beginning ";
y = "makes a good ending.";
z = x + y;
```

The + operator will add the two strings. Therefore the variable z will have the value `A good beginning makes a good ending`. The same + operator behaves differently based on the type of operands. If the operands are numbers, it will add the numbers. If the operands are strings it will concatenate the strings. Now, predict the value of z in the following code:

```
var  x, y;
x = "23";
y = 5;
z = x + y;
```

The answer is 235. If + operator sees any one operand as string, it will treat both the operands as string type and concatenate the strings to get the result 235. Suppose,

we want to add x and y in the form of numbers, we can rewrite the last statement as follows:

```
z = Number(x) + y;
```

After executing the above statement, the variable z will have the value 28. `Number()` is a function in JavaScript, that converts a string type data containing numbers to number type. We will make use of this function in some examples later in this chapter.

Know your progress

1. number, string and _____ are the basic data types in JavaScript.
2. `true` is a _____ type data.
3. `false` is a _____ type data.
4. The keyword used to declare a variable in JavaScript is _____.
5. The function used to know the type of data in JavaScript is _____.
6. What is the use of % operator?
7. List the logical operators.

6.6 Control structures in JavaScript

Control structures are used to change the sequential flow of execution in a program. All the control structures that we have learned in C++ can also be used in JavaScript without any difference. Let us discuss some of the frequently used control structures with an example each.

6.6.1 if

This is the most frequently used control structure in every programming language. It is used to execute a statement or a group of statements based on some condition. It can be used in two ways as given in Table 6.5.

Syntax of simple if	Syntax of if with else part

`if (test_expression)`
`{`
`  statements;`
`}`
	`if (test_expression)`
	`{`
	`  statements;`
	`}`
	`else`
	`{`
	`  statements;`
	`}`

Table 6.5: Syntax of if and if with else statement

The syntax in the left column of Table 6.5 is simple `if` statement and that in the right column is of `if – else` statement. Now, let us consider an example that makes use of the **`if`** statement.

Example 6.7: To create a web page that checks whether a student has passed or not

```
<HTML>
<HEAD> <TITLE>Javascript – if</TITLE> </HEAD>
<BODY>
<SCRIPT Language= "JavaScript">
    var score;
    score = 35;
    if (score < 30)
    {
        document.write("The student is failed.");
    }
    else
    {
        document.write("The student is passed.");
    }
</SCRIPT>
</BODY>
</HTML>
```

Fig. 6.8: Web page to illustrate `if` withe else statement

The above program code makes use of the `if` statement with `else` part. The output of the program is given in Figure 6.8. We can change the value of the score as a number below 30 and view the output to see the changes.

6.6.2 switch

`switch` is a multi-branching statement. Using this, different program codes can be selected for execution based on the value of an expression. Its syntax is:

```
switch (expression)
{
    case value1:
            statements;
            break;
    case value2:
            statements;
            break;
    ..................
```

```
        ..................
        default:
                    statements;
    }
```

The appropriate case is executed based on the value of the expression. Here the expression can be the name of a variable also. The following is a web page that prints the day corresponding to a given number.

Example 6.8: To create a web page to print the day of a week

```
<HTML>
<HEAD> <TITLE>Javascript - switch</TITLE> </HEAD>
<BODY>
<SCRIPT Language= "JavaScript">
    var d;
    d = 3;
    switch(d)
    {
        case 1:
            document.write("Sunday");
            break;
        case 2:
            document.write("Monday");
            break;
        case 3:
            document.write("Tuesday");
            break;
        case 4:
            document.write("Wednesday");
            break;
        case 5:
            document.write("Thursday");
            break;
        case 6:
            document.write("Friday");
            break;
        case 7:
            document.write("Saturday");
            break;
        default:
            document.write("Invalid Day");
    }
</SCRIPT>
</BODY>
</HTML>
```

The output of the above page is given in Figure 6.9.

Fig. 6.9: Web page that illustrates the use of `switch` statement

6.6.3 `for` loop

`for` loop is used to execute a group of instructions repeatedly. `for` loop uses a loop variable to control the number of iterations. The syntax of `for` loop is:

```
for(initialisation; test_expression; update_statement)
{
    statements;
}
```

Here initialisation is used to initialise loop variables. `test_expression` checks the condition and update statement is used to increment or decrement loop variables. The following example explains the use of `for` loop.

Example 6.9: To create a web page to display the squares of first 10 numbers

```
<HTML>
<HEAD> <TITLE>Javascript - for</TITLE> </HEAD>
<BODY>
<SCRIPT Language= "JavaScript">
    var i, s;
    for (i=1; i<=10; i++)
    {
        s = i*i;
        document.write(s);
        document.write("<BR>");
    }
</SCRIPT>
</BODY>
</HTML>
```

Fig. 6.10 : Web page to illustrate the use of `for` loop

The output of the code is given in Figure 6.10.

Now let us modify the above code to get the output as 'Square of 1 is 1' and so on.

```
for (i=1; i<=10; i++)
{
    s = i*i;
    document.write("Square of " + i + " is " + s);
    document.write("<BR>");
}
```

The code uses the string addition operator + to generate the output.

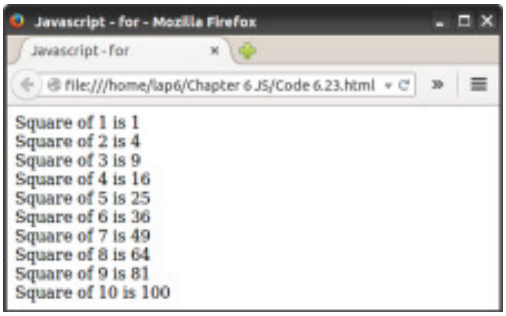


Fig. 6.11: Web page to display the square of numbers

6.6.4 while loop

`while` loop is a simple loop that repeatedly execute a group of statements based on a condition. The syntax is

```
while (test_expression)
{
    statements;
}
```

Here the `test_expression` is a condition. The statements inside the loop will be executed as long as the condition remains true. The following example displays all even numbers up to 10.

Example 6.10: To create a web page that displays even numbers upto 10

```
<HTML>
<HEAD> <TITLE>Javascript - while</TITLE> </HEAD>
<BODY>
<SCRIPT Language= "JavaScript">
    var i;
    i = 2;
    while (i<=10)
    {
        document.write(i);
        document.write("<BR>");
        i += 2;
    }
</SCRIPT>
</BODY>
</HTML>
```

Fig. 6.12: Web page that illustrates the use of while statement

The output of the above code is given in Figure 6.12.

In JavaScript, we can use `do while` loop also exactly as we do in C++. But since almost all the tasks can be done with `for` loops and `while` loops, we do not discuss the other loops here in this chapter.

Let us do

Write the output, if the statement `document.write("<BR>");` is omitted from the Example 6.10?

Know your progress

1. To select a group of statements in the program code for execution _____ or _____ control structures are used .
2. Give examples of looping statements in JavaScript.
3. _____ is a multi branching statement.
4. State whether the following statements are true or false
   a. `break` statement is used within the `switch` block.
   b. If we use `switch`, we will be using `break` within it, at least once.
   c. All the program codes written in `if-else` can be replaced by using `switch`.
   d. All the program codes written in `switch` can be replaced by using `if-else`.
5. What is the difference between `for` loop and `while` loop?

6.7 Built-in functions

JavaScript provides a large number of built-in functions. The functions are also called methods. Here we will discuss only the most commonly used functions.

a. `alert()` function

This function is used to display a message on the screen. For example, the statement `alert("Welcome to JavaScript");` will display the following message window on the browser window as shown in Figure 6.13. This function is used to display a message to the user at the time of data validation.

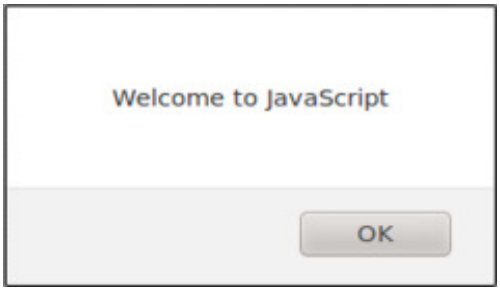


Fig. 6.13: `alert()` *window*

b. `isNaN()` function

This function is used to check whether a value is a number or not. In this function, `NaN` stands for Not a Number. The function returns `true` if the given value is not a number. For example, the following statements return the value `true`.

```
1. isNaN("welcome");
2. isNaN("A123");
3. isNaN("Score50");
4. isNaN("A");
```

The following statements return the value `false`.

```
1. isNaN("13");
2. isNaN(13);
3. isNaN("13.5");
4. isNaN("0.123");
```

The following statement gives the message as shown in Figure 6.14 on the browser window:

```
alert(isNaN("A"));
```

This function is very useful for data validation. For example, suppose the web page contains a text box to enter the age of a student. By mistake, the user may enter a character in the age box, instead of a number. The above function can check whether the entered value is a number or not. If it is not a number, a message will be displayed by using the `alert()` function.

Fig. 6.14: Result of `isNaN()`

c. `toUpperCase()` function

This function returns the upper case form of the given string. Look at the following example code:

```
var x, y;
x = "JavaScript";
y = x.toUpperCase();
alert(y);
```

The output of the above code segment is shown in Figure 6.15. Here you can see how the `toUpperCase()` function is called. It is called along with the name of the string variable `x`. That is, `x.toUpperCase()` returns the upper case form of the string in the variable `x`. JavaScript is a case sensitive

Fig. 6.15 : Result of `toUpperCase()`

language. Hence you have to use the function exactly in the same case form as given in the code.

d. toLowerCase() function

It returns the lower case form of the given string.

Look at the following example code.

```
var x, y;
x = "JavaScript";
y = x.toLowerCase();
alert(y);
```

The output of the above code segment is shown in Figure 6.16. If all the characters in the string are already in the lower case, the `toLowerCase()` returns the same string.

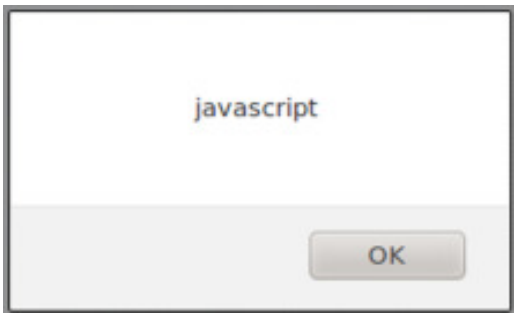


Fig. 6.16: Result of `toLowerCase()`

e. charAt() function

It returns the character at a particular position. `charAt(0)` returns the first character in the string. `charAt(1)` returns the second character in the string and so on. Look at the following example code.

```
var x;
x = "JavaScript";
y = x.charAt(4);
alert(y);
```

The output is given in Figure 6.17. Since the fifth character in the variable `x` is `'S'`, the letter is displayed in the browser window.

Fig. 6.17: Result of `charAt()`

f. length property

Besides functions, a string variable also provides some properties which are of use to programmers. `length` property returns the length of the string. length means, the number of characters in the string. For example,

```
var x, n;
x = "JavaScript";
n = x.length;
alert(n);
```

Here, we can see how the `length` property is called. The property is called along with the variable as `x.length`. Figure 6.18 shows the output of the above code.

The difference between a function and a property is that the function has parentheses (and) with parameters after the function name, but the property does not.

Fig. 6.18: Result of `length` *property*

Know your progress

Write the value of the variable y in the following:

1.
```
x = "welcome";
y = x.length;
```
2.
```
x = "WELCOME";
y = x.toLowerCase();
```
3.
```
x = "Welcome";
y = x.toUpperCase();
```
4.
```
x = "welcome";
y = x.toLowerCase();
```
5.
```
x = "welcome";
y = isNaN(x);
```
6.
```
x = "welcome";
y = charAt(3);
```

6.8 Accessing values in a textbox using JavaScript

In Chapter 5, we learned how to place various controls like textbox, checkbox, radio button, submit button, etc. in a web page. Now we will discuss how to access these web page elements using JavaScript. Actually all the program codes that we have already discussed in this chapter did not accept any value from the user for processing. The necessary data for processing was given directly in the program code itself. After learning this section, we will be able to write a truely interactive web page. That is, the user can enter some values in a text box, and some processing can be done on this value and some result can be displayed in another text box. Go through the following HTML code carefully.

Example 6.11 : To create a web page that displays a web form

```
<HTML>
<HEAD> <TITLE>Javascript - Text box</TITLE> </HEAD>
<BODY>
    <FORM Name= "frmSquare">
```

```
        <CENTER>
            Enter a Number
            <INPUT Type= "text" Name= "txtNum">
            <BR><BR>
            Square is
            <INPUT Type= "text" Name= "txtSqr">
            <BR><BR>
            <INPUT Type= "button" Value= "Show">
        </CENTER>
    </FORM>
</BODY>
</HTML>
```

The output is given in Figure 6.19. Note that we have given the name `frmSquare` for the Form, `txtNum` and `txtSquare` for the two textboxes. Giving names to them is very much important to access them using JavaScript. If we do not give any name to a web page element, the JavaScript cannot access these elements. We can also note that we have not given a name to the submit button. This is because, this button need not be referred from the JavaScript.

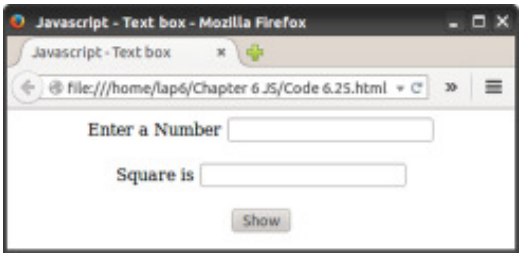


Fig. 6.19 : A form web page

Now let us make a slight modification in the above program code as given in Example 6.12.

Example 6.12: To create a web page that displays the square of a number

```
<HTML>
<HEAD> <TITLE>Javascript - Text box</TITLE>
    <SCRIPT Language= "JavaScript">
        function showSquare()
        {
            var num, ans;
            num = document.frmSquare.txtNum.value;
            ans = num * num;
            document.frmSquare.txtSqr.value = ans;
        }
    </SCRIPT>
</HEAD>
```

```
<BODY>
    <FORM Name= "frmSquare">
    <CENTER>
        Enter a Number
        <INPUT Type= "text" Name= "txtNum">
        <BR><BR>
        Square is
        <INPUT Type= "text" Name= "txtSqr">
        <BR><BR>
        <INPUT Type= "button" Value= "Show"
                              onClick= "showSquare()">
    </CENTER>
    </FORM>
</BODY>
</HTML>
```

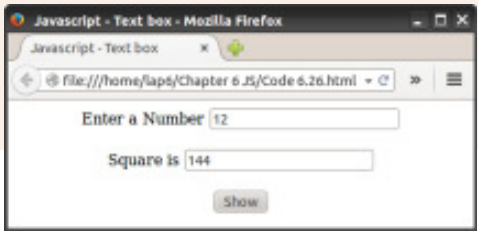


Fig. 6.20: Web page to find the square of a number

Go through the above code carefully. Note the additions made to the code in Example 6.11. A function with the name `showSquare()` is defined in the head section of the web page. This function is called using the following line of code:

```
<INPUT Type= "button" Value= "Show" onClick= "showSquare()">
```

`onclick= "showSquare()"` written within the button means that when the user clicks this button, the function with the name `showSquare()` is called.

Now go through the function definition. Look at the following line:

```
num = document.frmSquare.txtNum.value;
```

Here `document` refers the body section of the web page. `frmSquare` is the name of the Form we have given inside the body section. `txtNum` is the name of the text box within the `frmSquare` and `value` refers to the content inside that text box. That is `document.frmSquare.txtNum.value` means the `document`'s `frmSquare`'s `txtNum`'s `value`. Therefore the above line assigns the value of the first text box in a variable `num`.

Now, you may be able to understand the meaning of the following line.

```
document.frmSquare.txtSqr.value = ans;
```

The above line assigns the value of the variable `ans` in the second text box. Thus the above web page displays the square of the given number in the second text box when the button is clicked. User can type any number in the first text box and click the submit button to see its square. The screen shot of the web page while execution is given in Figure 6.20.

What will happen if the line

```
<INPUT Type= "button" Value= "Show" onClick= "showSquare()">
```

is replaced by

```
<INPUT Type= "button" Value= "Show"
                        onMouseEnter= "showSquare()">
```

The function will be called for execution when you move the mouse over the button. That means, you need not click the button for its execution. You may have noted that when you move the mouse over some buttons in a web page, the colour of the button may keep changing. This can be made possible by writing a JavaScript function to change the colour of the button and call that function on the event `onMouseEnter`. `onClick`, `onMouseEnter`, `onMouseLeave`, `onKeyDown`, `onKeyUp` are some of the commonly used events where we can call a function for its execution. Table 6.6 shows common JavaScript events and their descriptions.

Event	Description
onClick	Occurs when the user clicks on an object
onMouseEnter	Occurs when the mouse pointer is moved onto an object
onMouseLeave	Occurs when the mouse pointer is moved out of an object
onKeyDown	Occurs when the user is pressing a key on the keyboard
onKeyUp	Occurs when the user releases a key on the keyboard

Table 6.6: Common JavaScript events

Now let us dicuss another example (Example 6.13) that allows the user to enter two numbers in two different text boxes and display the sum of these numbers in a third text box, when a button is clicked. The output of this code is given in Figure 6.21.

Example 6.13: To create a web page that displays the sum of two numbers

```
<HTML>
<HEAD> <TITLE>Javascript - Sum</TITLE>
    <SCRIPT Language= "JavaScript">
        function showSum()
        {
            var num1, num2, ans;
            num1 = document.frmSum.txtNum1.value;
            num2 = document.frmSum.txtNum2.value;
            ans = num1 + num2;
            document.frmSum.txtSum.value = ans;
        }
    </SCRIPT>
```

```
</HEAD>
<BODY>
<FORM Name= "frmSum">
    <CENTER>
    Enter a Number 1
    <INPUT Type= "text" Name= "txtNum1">
    <BR><BR>
    Enter a Number 2
    <INPUT Type= "text" Name= "txtNum2">
    <BR><BR>
    The sum is
    <INPUT Type= "text" Name= "txtSum">
    <BR><BR>
    <INPUT Type= "button" Value= "Show" onClick= "showSum()">
    </CENTER>
</FORM>
</BODY>
</HTML>
```

This program displays 1020 as the result as shown in Figure 6.21. This is because the + operator is used to add strings also. By default, the content of the text box is always treated as of string type. Therefore, even though the content in the text box is a number, when we assign this to a variable, it will be treated as string type only. When we add the two strings "10"+"20" the answer is "1020". The function showSum() can be modified as follows to get the sum of two numbers.

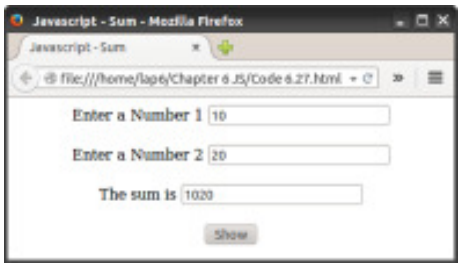


Fig. 6.21: Web page to illustrate the use of + operator

```
function showSum()
{
    var num1, num2, ans;
    num1 = Number(document.frmSum.txtNum1.value);
    num2 = Number(document.frmSum.txtNum2.value);
    ans = num1 + num2;
    document.frmSum.txtSum.value = ans;
}
```

We discussed the Number() function in the previous section of this chapter. The Number() function converts the data into number type and assigns that number into the variable num1. In the above function, num1 and num2 are treated as number

type data and hence the correct result is obtained after adding. The output of the web page is given in Figure 6.22 when the function in the Example 6.13 is replaced by the above function.

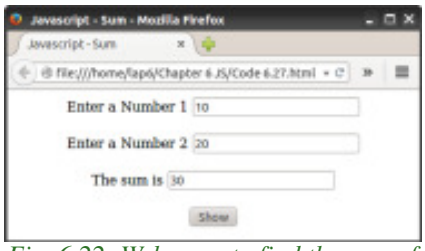


Fig. 6.22: Web page to find the sum of two numbers using + operator

Let us consider another web page which displays the sum of numbers upto a given limit. The user can enter the limit in a text box.

Example 6.14: To create a web page that displays sum of numbers upto a given limit

```
<HTML>
<HEAD> <TITLE>Javascript - Sum</TITLE>
    <SCRIPT Language= "JavaScript">
        function sumLimit()
        {
            var sum = 0, i, limit;
            limit = Number(document.frmSum.txtLimit.value);
            for(i = 1; i <= limit; i++)
                sum += i;
            document.frmSum.txtSum.value = sum;
        }
    </SCRIPT>
</HEAD>
<BODY>
    <FORM Name= "frmSum">
        <CENTER>
            Enter the limit
            <INPUT Type= "text" Name= "txtLimit">
            <BR><BR>
            Sum of Numbers
            <INPUT Type= "text" Name= "txtSum">
            <BR><BR>
            <INPUT Type= "button" Value= "Show"
                                  onClick= "sumLimit()">
        </CENTER>
    </FORM>
</BODY>
</HTML>
```

The output of the above code is given in Figure 6.23. In the above web page, on clicking the **show** button, the sum of numbers up to the given limit is shown in the second text box.

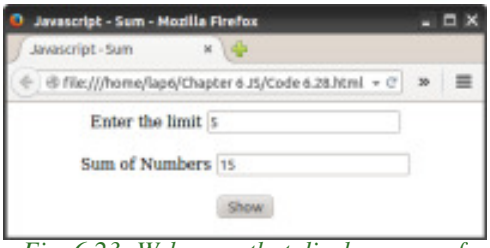


Fig. 6.23: Web page that displays sum of numbers upto a limit

If the **show** button in the Form is clicked without entering the limit, it does not display any message. That is, in this web page, if we click the **show** button without entering the limit, it will give 0 as the sum. If we do not enter any value in the text box, `document.frmSum.txtLimit.value` is empty. Hence `Number()` function will convert this empty value to the value 0. i.e., the limit variable in the function will have the value 0. Hence the loop will not be executed at all. Therefore, the variable `sum` will have the initial value 0 itself, which is displayed in the second text box.

Now, let us provide a check to the `sumLimit()` function as given below:

```
function sumLimit()
{
    var sum = 0, i, limit;
    if (document.frmSum.txtLimit.value == "")
    {
        alert("Please enter the limit!");
        return;
    }
    limit = Number(document.frmSum.txtLimit.value);
    for(i=1; i<=limit; i++)
        sum += i;
    document.frmSum.txtSum.value = sum;
}
```

JavaScript never gives an error. When the script engine is unable to execute an instruction, it will ignore that line as well as the rest of the lines in the function.If we click the **show** button without entering the limit, this code will show a message, reminding us to enter the limit. The `return` statement is used to exit from the function ignoring the rest of the lines. The `return` statement is similar to the `return` statement used in C++.

Let us now check whether a number or an alpbabet is entered in the text box for entering limit. For this, `isNaN()` function can be used. This helps to check whether the user has given the correct data for processing.

```
function sumLimit()
{
    var sum = 0, i, limit;
```

```
        if (document.frmSum.txtLimit.value == "")
        {
            alert("Please enter the limit!");
            return;
        }
        if (isNaN(document.frmSum.txtLimit.value))
        {
            alert("Please enter a number as the limit!");
            return;
        }
        limit = Number(document.frmSum.txtLimit.value);
        for(i = 1; i <= limit; i++)
            sum += i;
        document.frmSum.txtSum.value = sum;
    }
```

Java Script is mostly used for client side validation. When the user clicks the submit button after entering the data, JavaScript can be used to check whether the user has given all the necessary data, check whether the data is in the correct format or not, etc. If they are not in the correct format, a message can be shown reminding the user to enter the correct data.

The following example makes use of a drop-down list in JavaScript. This web page allows the user to select a state from a drop-down list. On clicking the show button, capital of the selected State is displayed in a text box. The output is given in Figure 6.24.

Example 6.15: To create a web page that displays the capital of a State

```
<HTML>
<HEAD> <TITLE>Javascript - switch</TITLE>
<SCRIPT Language= "JavaScript">
function capital()
{
    var n, answer;
    n = document.frmCapital.cboState.selectedIndex;
    switch (n)
    {
    case 0:
        answer = "Thiruvananthapuram";
        break;
    case 1:
        answer = "Bengaluru";
        break;
```

```
    case 2:
        answer = "Chennai";
        break;
    case 3:
        answer = "Mumbai";
        break;
    }
    document.frmCapital.txtCapital.value = answer;
}
</SCRIPT>
</HEAD>
<BODY>
    <FORM Name= "frmCapital">
    <CENTER> State
        <SELECT Size= 1 Name= "cboState">
            <OPTION>Kerala</OPTION>
            <OPTION>Karnataka</OPTION>
            <OPTION>Tamilnadu</OPTION>
            <OPTION>Maharashtra</OPTION>
        </SELECT>
        <BR><BR>
        Capital
        <INPUT Type= "text" Name= "txtCapital">
        <BR><BR>
        <INPUT Type= "button" Value= "Show" onClick= "capital()">
    </CENTER>
    </FORM>
</BODY>
</HTML>
```

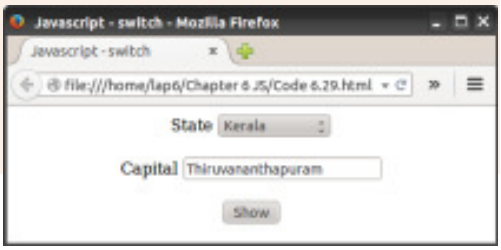


Fig. 6.24: Web page to find the capital of a state

Let us consider the following statement from the above program.

```
document.frmCapital.cboState.selectedIndex;
```

Here, 'cboState' is the name of the drop-down list. 'selectedIndex' gives the index of the selected item in the drop-down list. If the first item is selected, the index is 0, if the second item is selected, the index is 1, and so on. So, the above line assigns the index of the selected item in the variable n.

The following is a web page that allows the user to enter the name and age of a student. The name must contain at least 5 characters. The age should be a number in the range 15 to 20.

Example 6.16: To create a web page that validates name and age

```
<HTML>
<HEAD> <TITLE>Javascript - Validation</TITLE>
<SCRIPT Language= "JavaScript">
function checkData()
{
    var T_name, T_age, N_age;
    T_name = document.frmValid.txtName.value;
    if (T_name == "")
    {
        alert("Please enter name!");
        return;
    }
    if (T_name.length < 5)
    {
        alert("Name must contain at least 5 characters!");
        return;
    }
    T_age = document.frmValid.txtAge.value;
    if (T_age == "")
    {
        alert("Please enter age!");
        return;
    }
    if (isNaN(T_age))
    {
        alert("Please enter a number as the age!");
        return;
    }
    N_age = Number(T_age);
    if (N_age < 15 || N_age > 20)
    {
        alert("The age must be between 15 and 20!");
        return;
    }
}
</SCRIPT>
</HEAD>
```

```
<BODY>
    <FORM Name= "frmValid">
        <CENTER>Name
            <INPUT Type= "text" Name= "txtName">
            <BR><BR>
            Age
            <INPUT Type= "text" Name= "txtAge">
            <BR><BR>
            <INPUT Type= "button" Value= "Save"
                                  onClick= "checkData()">
        </CENTER>
    </FORM>
</BODY>
</HTML>
```

Fig. 6.25: Web page to validate name and age

The output of the above code is given in Figure 6.25. It checks all necessary validations for the data. At first, it tests whether there is an entry in the Name field or not. Then it tests whether the length of the name entry has minimum 5 characters or not. Then, it tests whether there is an entry in the age box. Then it checks whether the age entry is a number or not. Finally, it tests whether the age entry is in the range 15 to 20 or not.

6.9 Ways to add scripts to a web page

Scripts can be placed inside HTML code in different ways. In the previous examples we have placed the JavaScript code in the head section of the web page. Apart from head section, scripts can be placed inside the <BODY> tag or as an external file. Here we discuss the different ways of embedding scripts in web pages.

6.9.1 Inside <BODY>

Placing scripts inside the <BODY> tag has been discussed in the beginning of this chapter. Here, the scripts will be executed while the contents of the web page is being loaded. The web page starts displaying from the beginning of the document. When the browser sees a script code in between, it renders the script and then the rest of the web page is displayed in the browser window.

Let us discuss this method with an example. The following is a web page to get the result of a candidate. The user can enter a register number in the text box. On clicking the **Get Result** button, a JavaScript function should check whether there is any entry in the register number box. If there is, it must be a number and it should have seven digits. The output of the web page is given in Figure 6.26.

Example 6.17: To create a web page that accepts a register number after validation

```
<HTML>
<HEAD> <TITLE>Javascript - Validation</TITLE> </HEAD>
<BODY>
    <FORM Name= "frmValid">
        <SCRIPT Language= "JavaScript">
            function checkData()
            {
                var rno;
                rno = document.frmValid.txtRegno.value;
                if (rno == "")
                {
                    alert("Please enter Register No.");
                    return;
                }
                if (isNaN(rno))
                {
                    alert("Invalid Register No.");
                    return;
                }
                if (rno.length < 7)
                {
                    alert("The Register No. must have 7 digits");
                    return;
                }
            }
        </SCRIPT>
        <CENTER>
            <BR>Enter Register Number
            <INPUT Type= "text" Name= "txtRegno">
            <BR><BR>
            <INPUT Type= "button" Value= "Get Result"
                                    onClick= "checkData()">
        </CENTER>
    </FORM>
</BODY>
</HTML>
```

A script can also be at the bottom of <BODY> tag. If the scripts are loaded inside the <BODY> tag or in the <HEAD> tag, they will be loaded along with the HTML

code. This causes a visual delay in loading the web page. If the scripts are placed just before </BODY> tag, the contents of a web page like text, images, etc. will be displayed on the screen faster. However, the disadvantage of this method is that the scripts that are to be executed while loading the web page may not work.

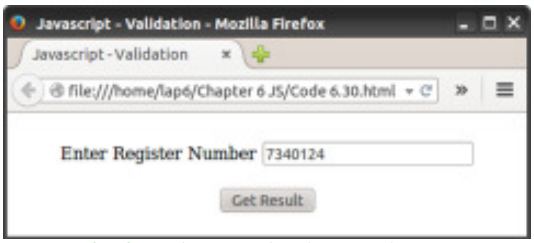


Fig. 6.26: Web page displaying the Form to accept register number

6.9.2 Inside <HEAD>

It is a usual practice to include scripts in the head section of the web page. We have been adding scripts in the head section in all our examples. The main reason for this is that the body section of most of the HTML pages contains a large volume of text that specifies the contents to be displayed on the web page. Mixing a function definition also with this will create a lot of confusion for the web designer for making any kind of modification in the page.More over, the head section of a web page is loaded before the body section. Therefore, if any function call is made in the body section, the function will be executed as the function definition is already loaded in the memory. In Example 6.17, the code within <SCRIPT> and </SCRIPT> has to be moved to the <HEAD> section of the HTML code.

6.9.3 External JavaScript file

We can place the scripts into an external file and then link to that file from within the HTML document. This file is saved with the extension '**.js**'. Placing JavaScripts in external files has some advantages. This is useful if the same script is used across multiple HTML pages or a whole website. It separates HTML and code which makes HTML and JavaScript easier to read and maintain. Storing JavaScript as separate files can speed up page loading. Note that in the above example (Example 6.17), the JavaScript code is stored as a separate file with the name "check.js". The contents of this file is as shown below:

```
function checkData()
{
    var rno;
    rno = document.frmValid.txtRegno.value;
    if (rno == "")
    {
        alert("Please enter Register No.");
        return;
    }
```

```
        if (isNaN(rno))
        {
            alert("Invalid Register No.");
            return;
        }
        if (rno.length < 7)
        {
            alert("The Register No. must have 7 digits");
            return;
        }
    }
```

Note that this file contains only JavaScript code and does not contain <SCRIPT> tag. <SCRIPT> tag is used inside the HTML file only. The file can be linked to HTML file using the <SCRIPT> tag. The Type attribute specifies that the linked file is a JavaScript file and the Src attribute specifies the location and file name of the external JavaScript file. The modified HTML code is given below.

```
<HTML>
<HEAD><TITLE>Javascript - Validation</TITLE>
    <SCRIPT Type= "text/JavaScript" Src= "checkdata.js">
    </SCRIPT>
</HEAD>
<BODY>
    <FORM Name= "frmValid">
        <CENTER>
            <BR>Enter Register Number
            <INPUT Type= "text" Name= "txtRegno">
            <BR><BR>
            <INPUT Type= "button" Value= "Get Result"
                                    onClick= "checkData()">
        </CENTER>
    </FORM>
</BODY>
</HTML>
```

If Src attribute is present, then contents of <SCRIPT> tag is ignored. That is, you cannot attach an external file and execute code in single <SCRIPT> tag. Two separate <SCRIPT> tags are needed for this. One with Src for external file and another with the code.

Know your progress

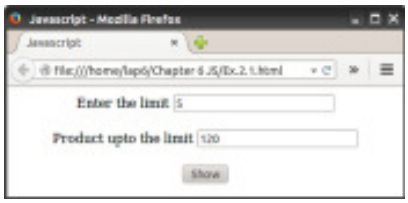
1. When will `onMouseEnter` event of JavaScript work?
2. The property of the drop-down list (`<SELECT>` element) used to get the index of the selected item is _____.
3. Compare `onKeyDown` and `onKeyUp` events of JavaScript.
4. What is the use of `Number()` function?
5. What is the advantage of placing JavaScript code just above the `</BODY>` tag?

Let us conclude

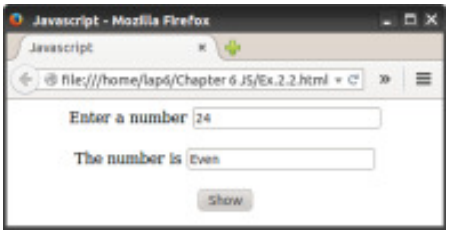
This chapter introduces JavaScript as a client side scripting language that is used mainly for validations. The tag used in HTML to include JavaScript code and the popular functions used are explained here. The datatypes in JavaScript and the use of variables are also discussed in detail. The use of operators and control structures is similar to that of C++. The different built-in functions and the events in JavaScript are explained through examples. The different ways of including JavaScript code in HTML page are presented in detail.

Let us practice

1. Develop a web page to display the following screen. User can enter a number in the first text box. On clicking the show button, product of all numbers from 1 to the entered limit should be displayed in the second text box.



2. Develop a web page to display the following screen. User can enter a number in the first text box. On clicking the show button, Even or Odd should be displayed in the second text box depending on whether the number is even or odd.

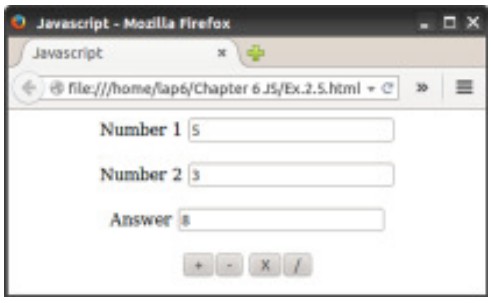


3. Develop a web page to display the following screen. The user can enter an age in the text box. If the user enters an alpbabet, instead of a number in the text

box, on clicking the show button, it should display a message "Invalid Age" to the user. Other wise it should display a message "Correct Data".

4. Develop a login page as shown in the figure. The page must contain one text box for entering the username and one password box for entering the password. The username must contain at least 4 characters and the password must contain at least 6 characters. The first two characters in the password must be numbers. On clicking the show button, if the valid data are given in boxes, a message "Correct Data" should be displayed. Otherwise, "Wrong Data" message should be displayed.

5. Develop a web page to implement a simple calculator. The page should have two text boxes to enter two numbers. It should also have 4 buttons to add, subtract, multiply and divide the two numbers. The answer should be displayed in a third text box on clicking the button. The web page should be as shown in the given figure.



Let us assess

1. Write the value of the variable z in each of the following:

a.
```
var x, y, z;
x = 5;
y = 3;
z = ++x - y--;
```

b.
```
var x, y, z;
x = "12";
y = 13;
z = x + y;
```

c.
```
var x, y, z;
x = 20;
y = 8;
x %= y;
z = x++;
```

d.
```
var x, y, z;
x = 1;
y = 4;
z = !(x < y);
```

e.
```
var x, y, z;
x = 5;
y = 6 ;
z = (x > y) || (y % 2 == 0);
```

2. Predict the output of the following

a.
```
<HTML>
<BODY>
<SCRIPT Language= "JavaScript">
var i;
for (i = 10; i >= 1; i--)
      document.write(i + "<BR>");
</SCRIPT>
</BODY>
</HTML>
```

b.
```
<HTML>
<BODY>
<SCRIPT Language= "JavaScript">
var i, s = 0;
for (i = 1; i <= 100; i += 2)
      s += i;
document.write("Sum = " + s);
</SCRIPT>
</BODY>
</HTML>
```

c.
```
<HTML>
<BODY>
<SCRIPT Language= "JavaScript">
var n, s = 0;
n = 0;
while (n <= 50)
{
      s = s + n;
      n = n + 5;
}
```

```
document.write("Sum = " + s);
</SCRIPT>
</BODY>
</HTML>
```

d.
```
<HTML>
<BODY>
<SCRIPT Language= "JavaScript">
var n, f = 1;
n = 5;
while ( n > 0)
{
      f = f * n;
      n--;
}
document.write("Product  = " + f);
</SCRIPT>
  </BODY>
  </HTML>
```

3. Following is an html code segment in a web page

```
<FORM Name= "frmStud">
<INPUT Type= "text" Name= "studentName">
</FORM>
```

Fill in the blanks to store the value of the text box to the variable n.

```
var n;
n = ..........................;
```

4. Suppose you have written a JavaScript function named checkData(). You want to execute the function when the mouse pointer is just moved over the button. How will you complete the following to do the same?

```
<INPUT Type= "button" .......... = "checkData()">
```

5. Explain <SCRIPT> tag and its attributes.
6. Write the syntax of a built-in function in JavaScript.
7. Classify the following values in JavaScript into suitable data types.
   "Welcome", "123", "true", 67.4, .98, false, "hello"
8. What is meant by undefined data type in JavaScript mean?
9. Explain operators in JavaScript.

10. Write JavaScript functions to perform the following
    a. To check whether a variable N contains a number.
    b. To convert the string "scert" to all capitals.
    c. To convert the string "HTML" to all small letters.
    d. To display a message "Welcome to functions".
    e. To display the third character in the string "Computer".
11. Write JavaScript code to display the length of the string "Computer".
12. A web page contains a button. Write HTML code for the button which executes a function Message( ) on the occurance of the following events.
    a. When user clicks the mouse on the button.
    b. When user moves the mouse over the button.
13. What are the advantages of writing JavaScript code in the head section of an HTML page?
14. Design an HTML page that contains a text box to enter the marks in a given subject.
    a. Write HTML code for this web page.
    b. Provide validations for this text box in a separate JavaScript file and link it with the HTML file. The validations are (i) it should not be empty (ii) it should be a number (iii) it should be between 0 and 60.
    c. List the advantages of writing the script in a separate file.

7 Web Hosting

Significant Learning Outcomes

After the completion of this chapter, the learner:

- describes the use of a web server and the concept of web hosting.
- classifies different types of hosting.
- explains the way to buy hosting space.
- registers a domain and hosts a website using FTP client software.
- explains the features of free hosting.
- identifies the use of Content Management Systems.
- describes the need for responsive web design.

We familiarised ourselves with creating web pages in the earlier chapters. A website consisting of several web pages are designed to give information about an organisation, product, service, etc. Suppose we have developed a website for our school using HTML. How can we make this website available on the Internet? These web pages are to be stored in the web servers connected to the Internet, to be made available to others. This chapter presents an overview of web hosting, its different types and general features. For accessing a website we need a domain name. How domain names are chosen and registered are also presented here. The various FTP client softwares available to transfer the files of the website (web pages, images, etc.) from our computer to the server are also discussed. After learning this chapter, one will be able to register a domain name and host a website.

7.1 Web hosting

To develop a website for our school we need to design web pages. In the previous chapters, we discussed how to design web pages. Any text editor or a web designing tool can be used to develop a home page, a page for the courses

in the school, facilities available, contact address, etc. and link them using a menu to form an attractive website.

After developing the school website in our computer, it has to be made available in the Internet. The designed website has to be uploaded to a web server to make it available to Internet users all over the world. For this, either storage space is rented on a web server to store the web pages that we have created or our own web server is set up. Setting up a web server is very expensive compared to hosting a website in a rented storage space.

Web hosting is the service of providing storage space in a web server to serve files for a website to be made available on the Internet. The companies that provide web hosting services are called web hosts. Web hosts own and manage web servers. These web servers offer uninterrupted Internet connectivity, software packages for offering additional services such as databases and support for programming languages such as PHP, Java, ASP.NET, etc.

7.1.1 Types of web hosting

Suppose the entire content of our school website including html files, images, etc., require 4 MB of hard disk space on the web server. Web hosts provide only standard packages of 10 MB, 20 MB, etc. of space on the web server. We may need to choose the web host considering the packages offered by them that suit our purpose. The number of visitors expected to visit our website is also a factor. If our website requires a database, contains scripts, etc. then the support of such features are also to be considered while choosing a web host.

The type of web hosting has to be decided based on requirements like the amount of space needed for hosting, the number of visitors expected to visit the website, the use of resources like databases, programming support, etc. Web hosts provide different types of hosting packages. They can be shared hosting, virtual hosting and dedicated hosting.

a. Shared hosting: Shared web hosting is the most common type of web hosting. It is referred to as shared because many different websites are stored on one single web server and they share resources like RAM and CPU. The features available on shared web servers are generally basic and are not flexible to suit a website that require specific features like high bandwidth, large storage space, etc. Shared hosting is most suitable for small websites that have less traffic. Shared servers are cheaper and easy to use, since they are configured with the most popular options. The updates and the security issues of the software installed in the web server are taken care of by the hosting company itself. A drawback is that since the bandwidth is shared by several websites, if any of these has a large volume of traffic, it will slow down all other websites hosted in the shared server.

b. Dedicated hosting: Dedicated web hosting is the hosting where the client leases the entire web server and all its resources. The web server is not shared with any other website. Websites of large organisations, government departments, etc. where there are large numbers of visitors, opt for dedicated web hosting. Here, the client has the freedom to choose the hardware and software for the server and has full control over the web server. Dedicated servers provide guaranteed performance, but they are very expensive. The advantage of dedicated servers is that such servers are usually hosted in data centers where the service provider facilitates Internet connectivity, round-the-clock power supply, etc. and the technical expertise for managing web servers. The cost of setting up and managing these facilities is thus reduced for the client. Since the bandwidth is not shared with other websites, it speeds up the access of the website. If the client is allowed to place their own purchased web server in the service providers facility, then it is called co-location.

c. Virtual Private Server: A Virtual Private Server (VPS) is a physical server that is virtually partitioned into several servers using the virtualization technology. Each VPS works similar to a dedicated server and has its own separate server operating system, web server software and packages like e-mail, databases, etc. installed in it. Unlike shared hosting, VPS provides dedicated amount of RAM for each virtual web server. Each of these VPS works as a fully independent web server, as if each were running on a separate physical server. The users of VPS are provided with the rights to install and configure any software on their VPS. They are also given the right to restart their VPS without affecting other virtual servers running on the same physical server.

VPS hosting provides dedicated bandwidth to each website on the server. This provides the advantages of a dedicated hosting, even though the actual server is shared. This type of hosting is suitable for websites that require more features than that provided by shared hosting, but does not require all the features of dedicated hosting. VPS provides almost the same services at a lesser cost than that of dedicated hosting. Some popular server virtualization softwares are VMware, Virtualbox, FreeVPS, User-mode Linux, Microsoft Hyper-V, etc.

Figure 7.1 gives a symbolic representation of the different types of web hosting packages.

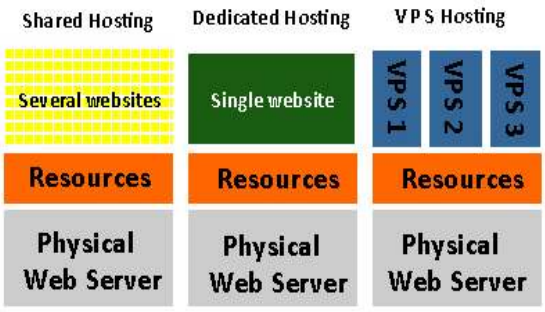


Fig. 7.1 : Types of web hosting

7.1.2 Buying hosting space

We have designed and stored the web pages of our school website in a folder in the computer. These files are to be copied to a web server to make it available on the Internet as shown in Figure 7.2. For this, the appropriate type of web hosting has to be chosen. It is preferred that the website is hosted using the services of a shared hosting service provider as it is cheaper and appropriate. Once the type of hosting server is decided, hosting space has to be purchased from a web hosting service provider.

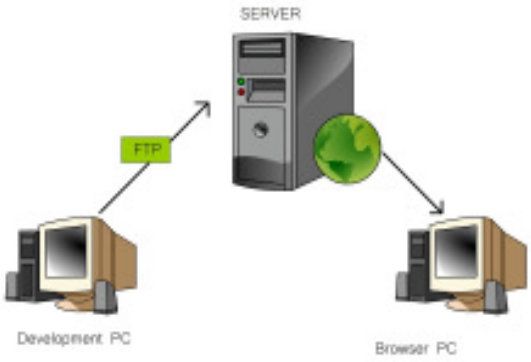


Fig. 7.2 : Making a website available on the Internet

While purchasing hosting space, several factors have to be taken into consideration. First, we have to decide on the amount of space required. We need to select a hosting space that is sufficient for storing the files of our website. If the web pages contain programming content, we need a supporting technology in the web server. The program used in the web page may require Windows hosting or Linux hosting. Here we need to choose between a Windows server or a Linux server as shown in Figure 7.3. If our website contains only HTML code, we can choose any server. Other features like database support, e-mail facility, etc. can also be considered while choosing the web host.

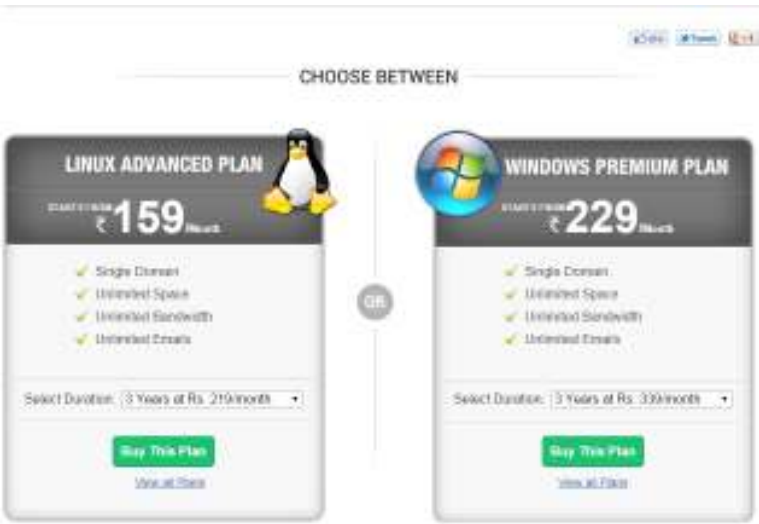


Fig. 7.3 : Hosting options

7.1.3 Domain name registration

We have now bought hosting space for our web pages and now we need an identification (URL) on the Internet. For this a suitable domain name has to be registered for our school. Domain names are used to identify a website in the Internet.

Most of the web hosting companies offer domain name registration services. After finalising a suitable domain name for our website, we have to check whether this domain name is available for registration or it is already in use by somebody else. The websites of the web hosting companies or web sites like www.whois.net provide an availability checker facility where we can check this. These websites check the database of ICANN that contains the list of all the registered domain names and gives a response as shown in Figure 7.4. If the domain name entered is available, we can proceed with the registration. The registration requires filling information for WHOIS database of domain names for ICANN. WHOIS information requires the name, address, telephone number and e-mail address of the registrant, as shown in Figure 7.5. This information can be made public or can be kept private according to the registrant's wish. After paying the annual registration fees online, the domain is purchased and is registered in our name. The shopping cart of the purchase is shown in Figure 7.6.

Fig. 7.4 : Domain name registration search result

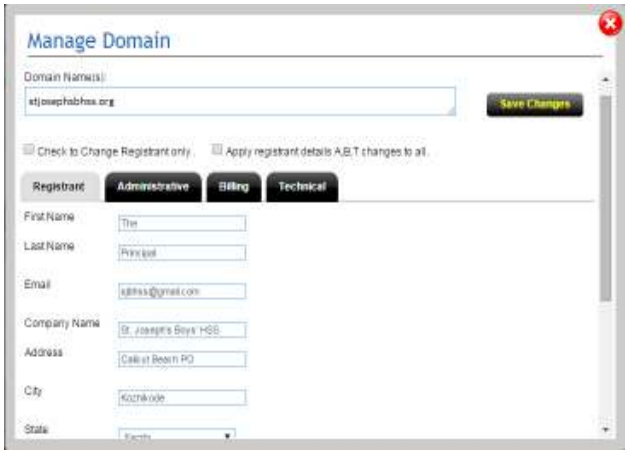


Fig. 7.5 : Providing WHOIS information

Thus, we have purchased a web server space for our school website and have registered a domain name for it. Now, when the user types our domain name, www.stjosephsbhss.org, in the browser window, it should display the webpage stored in the web server we have purchased. This will happen only if the DNS for server returns the IP address of our web server when the browser requests for it with www.stjosephsbhss.org.

Fig. 7.6 : Buying a domain name

Therefore, our domain name has to be connected to the IP address of the web server where the web pages are stored. This is done using 'A record' (Address record) of the domain. An 'A record' is used to store the IP address of a web server connected to a domain name. The 'A record' can be modified by logging into the control panel of the domain. Here, you can set the 'A record' of the domain name to point to the IP address of web server as shown in Figure 7.7. After this, the DNS servers will be able to resolve our domain name to connect to our web server.

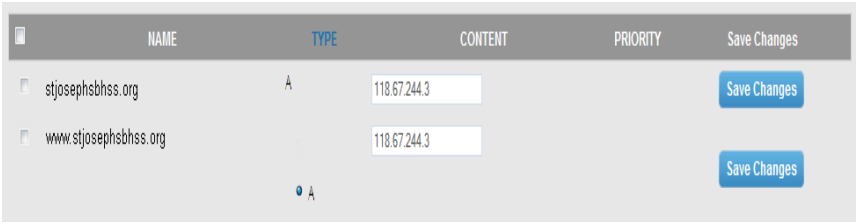


Fig. 7.7 : Changing 'A record' of the domain

A WHOIS search will provide information regarding a domain. It may include information like domain ownership, where and when registered, expiration date, etc. It is also used to determine whether a given domain name is available or not. A WHOIS lookup on www.kerala.gov.in returns the following.

```
Domain ID:D8944-AFIN
Domain Name:KERALA.GOV.IN
Created On:31-Dec-2003 05:00:00 UTC
Last Updated On:16-Jul-2014 11:37:59 UTC
Expiration Date:31-Dec-2016 05:00:00 UTC
Sponsoring Registrar:National Informatics Centre (R12-AFIN)
Status:OK
Registrant ID:R-R03120114034
Registrant Name:Government of Kerala
Registrant Organization:
Registrant Street1:Chief Minister's Office
Registrant Street2:
Registrant Street3:
Registrant City:Government Secretariate, Trivandrum, 69500
Registrant State/Province:Kerala
Registrant Postal Code:695001
Registrant Country:IN
```

Let us do

Prepare a comparative table of popular web hosts and their prices per year for their respective minimum hosting space. Provide pricing for both Windows and Linux web hosting.

Prepare a comparative table for the pricing for a year for .org and .com domain registrations in the popular service providers.

Search the WHOIS details of www.dhsekerala.gov.in and www.scert.kerala.gov.in in www.whois.net and prepare a chart of the details.

Know your progress

1. The companies that provide web hosting services are called _____.
2. List the factors that decide the type of web hosting.
3. VPS is
   a. Virtual Premium Service
   b. Virtual Private Service
   c. Virtual Premium Server
   e. Virtual Private Server
4. What is co-location?
5. What does WHOIS information contain?
6. Why is 'A record' important for a domain name?

7.1.4 FTP client software

After buying a space on the hosting server and a domain name, we need to transfer the files of the website from our computer to the web server. This requires an FTP client software.

We have already discussed FTP in Chapter 8, Computer Networks of Class XI. FTP is used to transfer files from one computer to another on the Internet. FTP client software establishes a connection with a remote server and is used to transfer files from our computer to the server computer. To connect to an FTP server, FTP client software requires a username and password and also the domain name. This is to be provided in the Site Manager dialog box as shown in Figure 7.8. FTP sends username and password to the server as plain text which is unsecure. Therefore nowadays, SSH FTP (SFTP) protocol which encrypts and sends usernames, passwords and data to the web server is used in the FTP software. SFTP uses Secure Shell (SSH) protocol which provides facilities for secure file transfer.

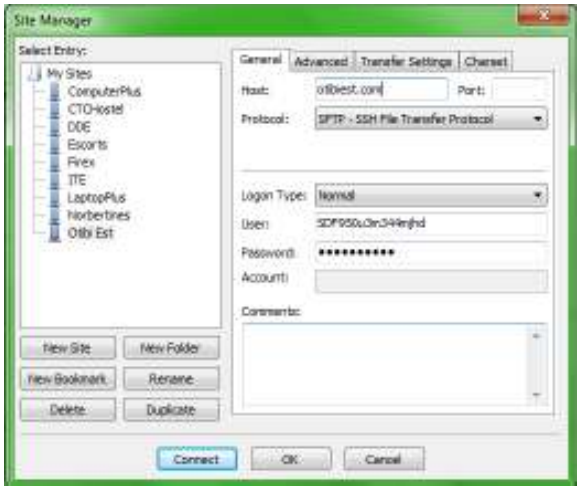


Fig. 7.8 : FTP Login

Once the FTP client is authenticated the IDE of FTP software appears as given in Figure 7.9. In this figure, the portion on the left side displays the folders and files in our computer and the right side displays the files in the web server computer. We

can use either the menu or 'drag and drop' files from the left side window to the right side window to upload the files to the web server. The files will then be transferred (copied) from our computer to the web server. The popular FTP client software are FileZilla, CuteFTP, SmartFTP, etc.

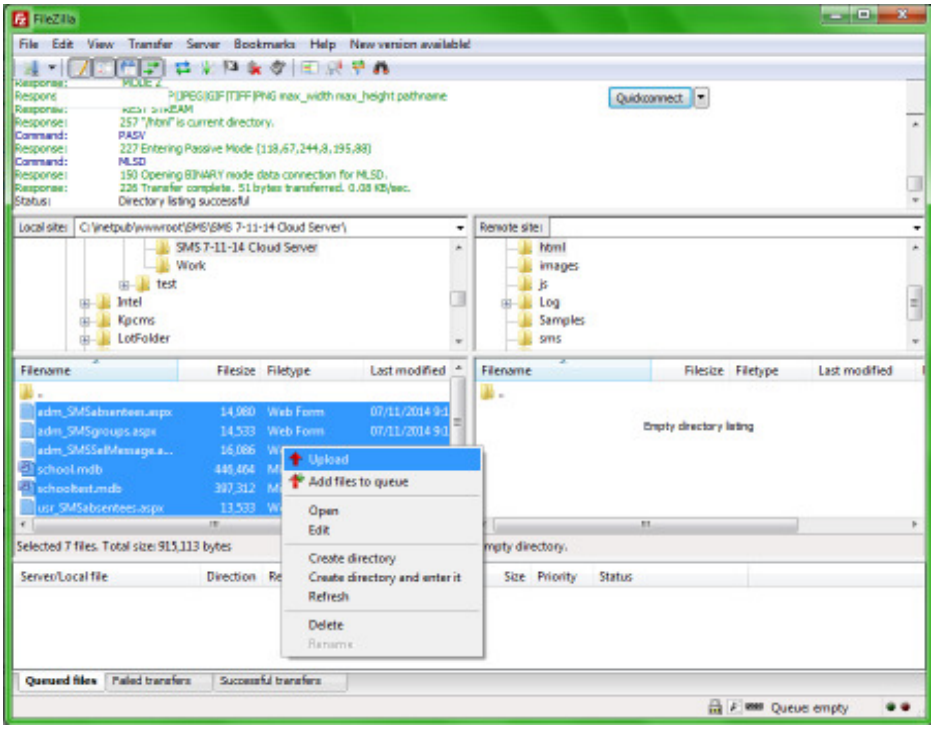


Fig. 7.9 : FTP client software IDE

Some web hosting companies provide their own control panel webpage through which users can upload the files. Such hosting companies do not allow third party FTP client software to upload files to their web servers. Figure 7.10 displays a control panel provided by a web hosting company to transfer files.

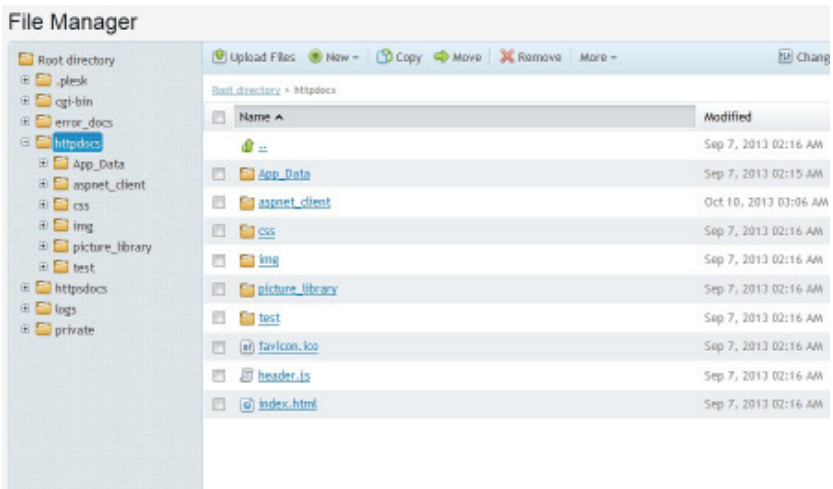


Fig. 7.10 : FTP control panel provided by a hosting company

7.2 Free hosting

Free hosting provides web hosting services free of charge. The service provider displays advertisements in the websites hosted to meet the expenses. Some free web hosting sites provide the facility to upload the files of our website in their server, but may place certain restrictions on the files. The size of the files that can be uploaded may be limited (supports upto 5 MB only), audio/video files (mp3, mp4, etc.) may not be permitted and so on. These websites usually provide control panels to upload files from our computer to the web server as shown in Figure 7.10. Some other websites only permit us to use the templates (pre-formatted designs) they provide for designing websites. They do not allow external files to be uploaded to their web server.

Free web hosting services usually provide either their own subdomain (oursite.example.com) or as a directory service (www.example.com/oursite) for accessing our websites. Some free web hosting companies provide domain name registration services also. Free web hosting is useful for sharing content on the web, among groups having similar interests like family unions, nonprofit organisations, etc. who are not able to spend money on web hosting. The availability of cheap web hosting services has reduced the need for free web hosting. Sites.google.com, yola.com, etc. are free web hosting services. A website of higher secondary school teachers that uses free hosting is given in Figure 7.11.

Fig. 7.11 : A free web site

7.3 Content Management System

Content Management System (CMS) refers to a web based software system which is capable of creating, administering and publishing websites. CMS provides an easy way to design and manage attractive websites.

Most CMS is available for free download at their websites. Copy the files of the CMS to the hosting space on our web server and configure the website as shown in Figure 7.12. Templates that provide a list of preset designs for websites are also

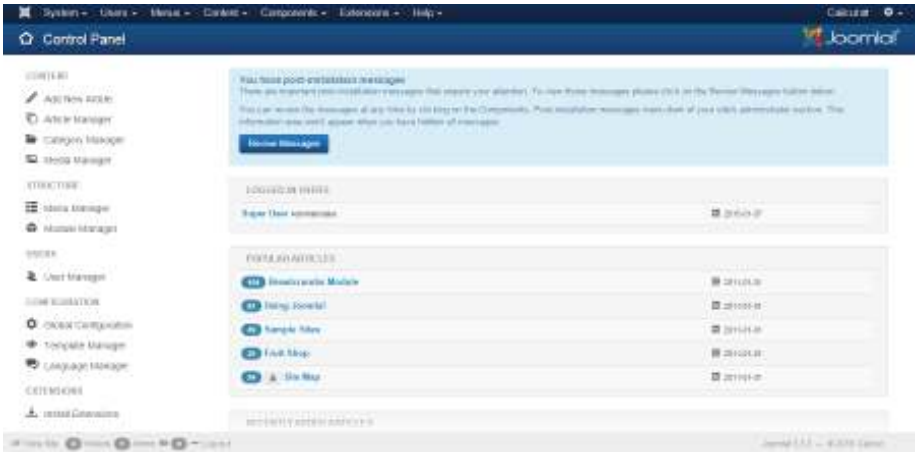


Fig. 7.12 : Configuring a website using Joomla!

available for download in some websites. Some sample templates available are displayed in Figure 7.13. The users need to choose a template from the given list, upload them to our website and add titles, images and other descriptions for the chosen design. This is as simple as designing a document in a word processor.

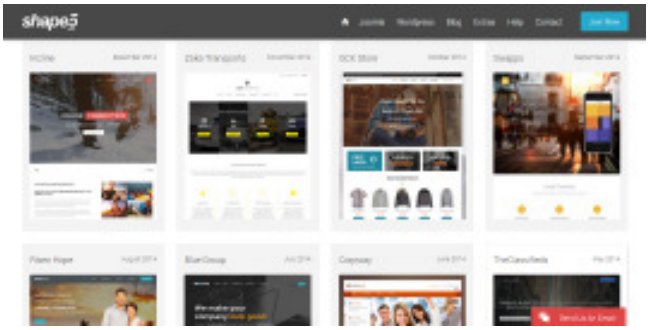


Fig. 7.13 : Templates available in Joomla!

Fig. 7.14 : A web site developed using Joomla CMS

CMS provides standard security features in its design, that help even people with less technical knowledge to design and develop secure websites. The CMS templates also reduce the need for repetitive coding and design for headings and menus that appear in all pages in a website, by internally providing features to add them in all pages. Templates are available for download for free or for a small price at

popular CMS websites. There are also third party vendors who customise the CMS for a fee. If you are hosting on a shared web server, verify whether the web server supports the CMS you have downloaded.

CMS is economical and now many organisations, bloggers, etc. use it for their websites. Some of the popular CMS software are WordPress, Drupal and Joomla! Figure 7.14 shows the website of Motor Vehicles department of Government of Kerala, developed using Joomla!

7.4 Responsive web design

Today we browse web pages using various devices like desktops, laptops, tablets and mobile phones. All these devices have different screen sizes. Traditional web pages are designed to be displayed on the screens of devices like desktops and laptops. These web pages are difficult to view when it is accessed using tablets and mobile phones. The user may have to use the scroll bar to move from one part of the web page to another. In earlier days, a separate website was created for the purpose of viewing in mobile devices. These websites contained web pages whose size matched the screen size of these devices. But maintaining two websites for a single organisation created issues. It would be better if the web page was able to adjust itself to the screen size of the device. This type of web page designing is called responsive web design. Figure 7.15 shows the appearance of a responsive website in different devices.

Fig. 7.15 : Responsive web design

Responsive web design is the custom of building a website suitable to work on every device and every screen size, no matter how large or small, mobile phone or desktop or television. The term 'responsive web designing' was coined by Ethan Marcotte, an independent designer and author, to describe a new way of designing for the ever-changing Web. Responsive web design can be implemented using flexible grid layout, flexible images and media queries. Flexible grid layouts set the size of the entire web page to fit the display size of the device. Flexible images and videos set the image/video dimensions to the percentage of display size of the device. Media queries provide the ability to specify different styles for individual devices. A horizontal menu in a web page for larger displays might have to be converted to a drop down menu for a mobile phone. These settings can be done using media queries inside the CSS file.

Screen sizes always vary - from a wearable device, mobile phones, tablets to laptops, desktops and televisions. Therefore, it is important that websites are designed to adapt to the screen size of the device.

Let us do

Design a website containing the name, age, total runs scored, no. of wickets taken, etc. of the members of the Indian cricket team and host the web site using any free hosting services.

Prepare a list of popular CMS providers and write their features.

In the web hosting space provided, upload the files of the school website you have prepared using any FTP software.

Let us conclude

After designing a website, it has to be hosted over the Internet using a suitable type of hosting. Websites of small organisations can be hosted using shared hosting, websites that have more traffic and need more security can opt for VPS hosting, whereas websites of large organisations or those with very high traffic require dedicated hosting. After buying a suitable web hosting space, FTP software can be used to connect to the web server and can securely transfer the files of the website to it. Domain name can be registered through a service provider and the 'A record' can be set to point the domain to the web server. There are free hosting websites that provide hosting free of cost. Content Management Systems (CMS) offer tools and standard security features in its design that helps even people with less technical knowledge to design and develop secure websites. Today, since we browse websites using devices like mobile phones, tablets, laptops, etc. that have different screen sizes, it is important to design websites that display themselves according to the size of the screen they are viewed from.

Let us assess

1. What do you mean by web hosting? Explain the different types of web hosting.
2. A supermarket in a city wishes to take its business online. It plans to accept orders for its products through a website and receive payments online.
   a. Which type of hosting is suitable for this website?
   b. Explain the reason for your choice.
3. Emil wishes to purchase the web hosting space required to host a website for his medical shop. List the features to be taken into consideration while buying hosting space on a web server.
4. How can we connect a website hosted in a webserver to a domain name?
5. What is the advantage of using SFTP protocol in FTP software?
6. Raju wishes to host a website for his family. What are the advantages that free web hosting companies provide?
7. What is CMS? What are the features of CMS? Give Examples.
8. Explain the need for applying responsive web design while developing websites today.
9. How is responsive web design implemented?

References

- Stroustrup, B. (2013). *The C++ Programming Language*. New Delhi : Addison-Wesley Professional
- Lafore, R. (2009). *Object-Oriented Programming in C++*. Chennai : Sams Publishing
- Balagurusamy, E. (2008). *Object Oriented Programming with C++*. New Delhi: Tata McGraw-Hill Education
- Sharma, A. K. (2011). *Data structure using C*. New Delhi : Pearson Education India
- Srivastava, S. K. & Srivastava, D. (2011). *Data Structures Through C in Depth*. New Delhi : BPB Publications
- Powel, T. A. (2010). *The Complete Reference : HTML & XHTML*. New Delhi: OSborne/Tata MC Graw-Hill
- Lloyd, I. (2008). *The Ultimate HTML Reference*. Melbourne : Sitepoint
- Holzner, S. (2000). *HTML Black Book*. New Delhi : DreamTech Press
- Frain, B. (2012). *Responsive Web Design with HTML5 and CSS3*. Mumbai : Packt Publishing
- Powel, T. A. & Schenider, F. (2008). *The Complete Reference JavaScript*. New Delhi : Tata McGraw-Hill
- Zakas, N. C. (2012). *Professional JavaScript for Web Developers*. Birmingham : Wrox