Module 15
Arrays and Strings

Introduction

The C++ language is an extension of C and thus carries the legacy of the C language. One of the legacies is the arrays. The array is a simple structure containing a collection of homogeneous items together as a single unit. A great feature of the arrays is the ability of the programmer to manipulate the complete collection using the index value and some looping structure. Another important legacy of C is the string. The string in C is an array of characters terminated by a typical sentinel value called null ('\n'). The C string library contains many functions for dealing with C type strings, for example, copy one string into another, comparing two strings with each other, finding out if there is a typical substring which is part of a given string etc. The arrays are part of the C++ design and they are used like they are in C. However, the C type strings are not preferred in C++ as a better form of string is provided by the C++ designers. The C++ string is basically an object of the class (which is also known as a string class) available to C++ programmers. C++ contains a new library called standard template library or STL which contains many other classes apart from the string. The string is one of the most popular classes of the STL. The string class has many advantages as compared to the C type string. We will throw some light on how the string objects are better than C type strings in this module. We will also explore two important things in this module. We will see how arrays are extended in C++ to have objects as their elements. We will also learn about how the C++ string objects are used and how one can program using the C++ string objects and their member functions.

Before we embark on the discussion of string objects, let us be clear that the C language array structure and strings are assumed to be known. If you have no idea about C arrays and strings, it is strongly recommended that you study them before attempting to learn the content of this module.

Arrays of objects

The arrays in C++ can contain everything a C array could, additionally, it can also contain objects as its members. An array of objects is basically objects stored as elements of arrays. Let us look at an example to illustrate the point. Look at the program 15.1 which describes a class called players. There are 3 data members and two function members of this class. Every player has a typical jersey number, name and address associated with him. We have defined two function members, one which takes the details of the member (all three data member values) and the other displays the values of those data members. You can see that both functions are quite an integral part of most of the classes that we have defined. When we study operator overloading, we will see how can we simplify this process by overloading the operators << and >>. The definition of the class contains both private and public

sections exactly like other class definitions that we have seen in previous modules. We also access members defined in public sections using the dot operator like the previous modules in this case as well. The additional parts are the array definition and using the members of an array of objects as any other object. Here is the program for your perusal. We can see how the objects which are part of the array are accessed and used in the program.

```
//Program 15.1
// ObjectArray

#include <iostream>
#include <string>
using namespace std;

class Player
{
private:
   int JerseyNo;
   string pNm;
   string pAdd;
public :
   void Get(int Jersey, string t_pNm, string t_pAdd)
   {
      JerseyNo = Jersey;
      pNm = t_pNm;
      pAdd = t_pAdd;
   }

   void pDisplay()
   {
      cout << "Jersey Number is " << JerseyNo << "\n";
      cout << "Player name is " << pNm << "\n";
      cout << "Player Address is " << pAdd << "\n";
   }
};

int main() {

   int i;

   Player pArray[4];

   pArray[0].Get(1,"Virat Kohli","India");

   pArray[1].Get(2,"Chris Gayle","West Indies");

   pArray[2].Get(3,"A B Deviliers","South Africa");

   pArray[3].Get(4,"Steve Smith","Australia");

   for (i = 0; i < 4; i++)
   {
      pArray[i].pDisplay();
   }

   return 0;

}
```

Description

Let us try to see the program 15.1 from the point of view of our understanding of arrays. We have defined a class called a player with three data members and two member functions in

the beginning. An important statement follows the main definition.

```
Player pArray[4];
```

This statement is quite similar to statements which define an array in C, except for the case that the elements of the array are of type Player here. You can see that the definition has nothing new. Now we have pArray as an array. Every element is an object of type Player now. That means pArray [0] is a Player, pArray [1] is a Player and so on. We need to treat those elements as objects of type Player now. We can do so by calling member functions in the following fashion.

```
pArray[0].Get(1,"Virat Kohli","India");
pArray[i].pDisplay();
```

Kindly look at both the statements. In the first statement, an int constant is used as an index and in the second case an int variable is used but in both cases, we are able to access an array element with dot notation (like we did earlier) to call a member function using that syntax.

Thus, we can use arrays like we did with C, apart from all valid C data types, objects can also be an element of the C++ array. Array elements can be used like normal objects and we can use dot notation to access public data and function members.

Another point. You probably have noticed that we have used following statements to define two data members of the Player class.

```
    string pNm;
string pAdd;
```

Which type of string are these? They are not arrays like in C, neither they have the null character to indicate the end of the string. They are objects of type string. The string is a class from STL (Standard Template Library) from C++. We will study STL in modules 16,17 and 18 at a later stage but let us try to see how can we define and use string objects in C++.

Need for the string objects

C Type strings are the character arrays with the null character as an indicator to the end of the string. This mechanism is used by C to have string variables and also have many functions to manipulate these types of strings. Let me repeat, the C-Type string objects are still available to C++, we have this additional construct at our disposal while dealing with strings. This option, the string object, is also called the C++ strings, they are the objects of

STL[1] the class also called string. C++ strings is a better option than the C type strings and that is the reason why it is chosen to be used by most developers instead of C type strings.

C++ strings, as the ensuing discussion will prove, is a better option than the C type strings and that is the reason why it is chosen to be used by most developers if not all. Let us see shortcomings of the string representation as character arrays. We need to have a string library and the prototypes to use the C++ strings. Here is what we have done at the beginning of our program

```
#include <string>
```

Above statement inserts string prototypes in the std namespace so we can use them. The discussion about what is the std namespace and how these insertion works are beyond the scope of this course. You may refer to the reference 1 for more information on namespaces[2].

Limitations of string representation as character arrays

If we need to define a string in C, we can do it as follows.

```
char PNm[30];
char PAdd[60];
```

So we actually define an array of characters. We can use these names (PNm and PAdd) as strings and use them in normal operations. However, there are a few limitations of this approach. Let us try to see.

Non-availability of == operator

We cannot compare strings like normal variables. For example, following statements are not allowed.

```
if (PNm == PAdd) ….
```

We cannot compare strings using the == operator. The solution is to use a function called strcmp in a fashion which is not as straightforward as the == operator. One typical problem with the strcmp function is that when the comparison is right it returns a value zero and we need to negate the output to check if the strings are the matching. For example, following is the normal statement in C.

```
if !(strcmp(PNm,PAdd) ….
```

[1] STL is an acronym of Standard Template Library, a unique type of library which allows templatized library functions possible to be used by different types of collections of C++ objects.

[2] Namespace is an enclosure where functions and variables are defined in C++. The standard namespace or std is the most common namespace.

The ! operator is needed here. The problem with this construct is that it is not a logical way to state that the strings are matching. It is hard for somebody who is new to C to understand what is happening in that statement. A novice has more probability to mistake that the check is made for string being dissimilar (rather than same). A better form is really desired.

Let us repeat, there are many functions like strcmp and strcpy which manipulates the C type strings. They are known as C Type string functions. They are different than C++ string functions. They are the member functions of the string class which can be used by the C++ string objects. These C++ string functions allow manipulation of the string object in much more user-friendly and simpler fashion.

Non-availability of assignment

When we have two strings and we want to allocate the value of the first string to another, we cannot do it using normal assignment, unlike other variables. For example, look at the following (invalid) statement.

```
Player1.PNm = Player2.PNm
```

It is not a valid statement if both PNm is of C Type string. We need to use following construct for the same operation.

```
strcpy(Player1.PNm,Player2.PNm)
```

There is a possibility of incorrect assignment in the case of strcpy function. We may assign string X to string Y instead of assigning string Y to string X if we exchange the arguments by mistake. It is also not very readable.

Initialization is not straightforward

Another problem with C type strings is that it is not easy to initialize strings by other strings. For example, following statements are not acceptable in C type string case. Both are examples of initialization possible to be done for other types of variables. Initialization defines a new string with a value of an already defined string. Following statements are possible with string objects.

```
string Player1.PNm (Player2.PNm)
string Player1.PNm = Player2.PNm
```

One may incorrectly compare above-mentioned statements being equivalent to following using the C type string but it is not.

```
char *Player1.PNm = Player2.PNm
```

Above statement initializes Player1.PNm pointer to Player2.PNm but do not define a new string. In other words, we still have the same string, pointed to by two different pointers. What we really need is to have a new string with the value of an old string. That is not possible in C Type string in the fashion that described above (initialization). We need to define both strings and copy one string into another separately. One can understand the difference in one more way.

C strings are not possible to be initialized but it is not the case when objects are initialized with other objects. Though both objects contain the same value after initialization, changing one object does not change the other object.

String functions are not member functions

The strcpy and strcmp are not truly string functions. They are library functions which are used with arguments as strings. These functions are char array functions assuming a null character at the end. When that null character is not present in a char array, the function will not be able to work properly. For example, a function called strlen counts characters till the null character. It is possible that the array length is 100 but first 70 characters contain the string characters. The 71st character is the null character and thus the strlen correctly identify the length of the string as 70. The strcpy function is also written in the form that only characters till the null characters are copied in the resultant string. If strlen or strcpy functions are called with character arrays without having the null character, they cannot work as expected.

Readability is compromised

When the string is represented as character arrays, it works but not the same way as other built-in types. Using strings demands mastering different syntax and one must learn to deal with errors related to placement of null character and actual size of the string being one more than what is specified in the array definition (to accommodate the null character). If we have a solution which provides string operations like other data types, the users find it more readable and also user-friendly.

C++ string objects are designed to address all these limitations and thus provide better and more readable ways to working with strings. Subsequent sections of this module deal with how C++ string objects are possible to be defined and used. We will also see how C++ objects overcome the limitations of C type strings.

String Objects

The string class, as we have mentioned already, is part of STL and thus provides a class string. The objects of this class, obviously, is known as string objects and are the constructs used to represent strings instead of C type strings (character arrays).

The string class is defined in a manner that string objects work like natural strings. It contains many functions that a normal user expects from a string object. If you have looked at how we have defined and used the string objects in program 15.1, you can vouch for that yourself.

If you look back the code that we have defined and used in previous modules, you can see that string objects are indeed user-friendly. We have defined and used string objects without specifying anything about them. We have treated strings as if they are normal data types. One does not really need a special introduction to strings to use them in the C++ program. Those examples showcase how user-friendly and natural these string objects are.

Having mentioned that, it is important for us to understand that string class does not only provide user-friendly operations for creating and using strings like we have been doing so far but over and above, it also provide quite a few member functions to make it much more powerful than one can think of at the first glance. Let us delve deeper to see that.

Defining strings

One can define strings in a similar fashion as other types of user-defined and built-in variables in C++. There are three different ways possible in which one can define strings in C++, here are they.

Normal way

Examples that we have seen earlier are the normal way to define strings. Here is another example

```
string PNm;
string PAdd;
```

One just defines both variables as strings, that is all. We have seen quite a few examples of this type of definition so far.

Initialization

Initialization is another way to define a string. In this case, the strings are defined and given a value from some other C++ string object, a C type string object or a string constant. Here are examples.

```
string PNm (SomeOtherPNm); // using other string object
string PNm = SomeOtherPNm; // same as above

char *Address = “Some Address”
string PAdd(Address); // using C type string

string PNm (“Virat Kohli”); // using string constant
string PNm = “Virat Kohli”; // same as above
```

Using constructor

All examples in above section invoke the constructor for string object. The designers have defined a few string constructors and they are called with appropriate arguments as and when the specific constructor is invoked. When a constructor is also defined for one argument, it is also possible to use as a conversion function.

Substring related member functions

Once we have seen how the strings are possible to be defined, there are some ways one can manipulate strings to get the substrings and use them in the program. There are two different ways one can manipulate STL objects including strings. The first method is using a member function, which is quite common across many other libraries. Second is to use generic algorithms. We will look at some of the member functions in this module. We will have a detailed look at the generic algorithms at a much later stage of this course when we study STL. However, we will give examples of using strings with member functions here to showcase how a programmer can put them to use. Before we discuss these functions, let us clearly understand one thing. Most of the functions that we show in the subsequent sections have other overloaded versions for different types and numbers of arguments. We are only showing the versions which are most commonly used. One should study the help file of the compiler that they use for learning about other versions.

Finding a character or a substring

The generic algorithm (a non-member function), find() is quite useful in many ways with many STL objects, so as strings. However, string also has a member function which is also called find(). The member function find() is used to find the location of a character or a substring in a given string. Find takes the string under consideration as the calling object and the substring or char as the first argument. It returns the position at which that character or substring resides in the given string. The first position of the string is numbered as zero like arrays. That means the humans who read the string and find the location of a typical character at position 10, the find function finds it at position 9.

When the substring or the character provided as the argument is really a part of the invoking string object, the position where the argument is present in the invoking string is returned as mentioned above. However, if the character or the substring is not part of the string, a strange response is provided. The output will be the largest value a string position can obtain is returned (normally a very big number indicating the maximum size of the string object). Let us check following statements.

```
string String1 = "Vishwanathan is a Chess Player";
unsigned long position = String1.find("Chess");
cout << "Position is " << position << "\n";
unsigned long Otherposition = String1.find('P');
cout << "Position is " << Otherposition << "\n";
```

output: -
Position is 18

Position is 24

We have a string "Vishwanathan is a Chess Player" at our disposal and we tried to find out where the word Chess and character P exists in the string. The find function returns the position of the substring in the first case and the character in the second case as expected.

When find () is supplied with a string or a character not present in the string

Look at the following code as well as the output. Both, the substring and the character are not really present in the string.

```
string String1 = "Chirag throw a ball";
string String2("Cat jumps over");
cout << String1.find("Nowhere") << endl; // substring not present
cout << String1.find('B')<< endl;// char not present
output: -
18446744073709551615
18446744073709551615
```

The large value **18446744073709551615** is the maximum size of the string on the machine where this code snippet was running.

Let us take a few more examples to illustrate few other important characteristics of the string object and the functions that are used with it.

Using function at()

A function at() takes an unsigned long integer value indicating the position and returns the character at that position. This functionality is exactly opposite to find. The find takes char and returns the position and the function at returns the character at a given position. In following code snippet, the value i is used to indicating the position in the string. The for loop starts with i value as 0 and went on till the length of the string and display the characters at i^{th} position. As we are looping through each position of the string, we get the complete string as an output of the loop.

```
for (unsigned int i=0; i<String1.length(); i++)
      cout << String1.at(i);
   cout << endl;
```

Output: -
Chirag throw a ball

Using function insert ()

A function insert() takes two arguments, a position, and a substring. It adds the substring at a given position. Here is an example.

```
String1.insert(14, " red ");
cout << String1<< endl;
```

output: -

Chirag throw a red ball

Using function append ()

The function append () takes a substring as an argument and appends the substring to the string. Here is an example.

```
String2.append(" the table!");
cout << String2 << endl;
```

output: -
Cat jumps over the table!

Using function replace ()

The replace () function takes three arguments, position from where to start, number of characters from that position and the substring which replaces the part which is being replaced. Following example illustrates how one can replace a table with a chair in the string2

```
String2.replace(15,10, "the chair"); // table with chair
cout << String2 << endl;
```

output: -
Cat jumps over the chair

Using function erase()

The erase() function works like the replace function but replacing the content with nothing. It has two arguments, position to start with and the number of characters from that position. The function removes those characters from the string object. Here is an example where 10 characters from positon 15 are removed.

```
String2.erase(15,10); // now removing chair
cout << String2 << endl;
```

output: -
Cat jumps over

Dealing with multiple strings

There are many cases where multiple strings are manipulated in the program. For example, when one string is compared with the other string or checking if one string is assigned to another string etc. The C++ string object contains many member functions which can be used for operations involving multiple strings. Let us try to see how those member functions can be used in a program.

Using operators

+ operator for concatenation

The first operator function provided for string object is operator +() which allows a programmer to use + to concatenate two strings. Here is an example.

```
string String1 = "Chirag throw a ball";
string String2("Cat jumps over");
string String3 = String1 + String2;
cout << String3;
```

output: -
Chirag throw a ballCat jumps over

= operator for assignment

The = operator is possible to be used to assign one string object into another. Here is an example.

```
string String1 = "Chirag throw a ball";
string String3;
String3 = String1;
cout << String3;
```

output: -
Chirag throw a ball

== operator for comparison

```
string String1 = "Chirag throw a ball";
string String3;
String3 = String1;
if (string3 == string1) cout << "both are equal";
```

output: -
both are equal

> operator for comparison

We can use > to see if the LHS of the > is a string which is lexicographically greater than the string which is at the RHS of the operator. Here is an example.

```
cout << "\n as per lexicographically order";
if (String1 > String2)
   cout << "string1 is greater than String2\n";
else
   cout << "\nString1 is lesser than String2\n";
```

output: -
as per lexicographically order string1 is greater than String2

!= operator for checking inequality

Like == operator, the != operator is also possible to check if two items on either side of the operators are not equal. Here is an example.

```
if (String1 != String2)
   cout << "Both strings are not equal\n";
```

output: -
Both strings are not equal

[] operator for array-like behavior

```
for (unsigned int i=0; i< String1.length(); i++)
   cout << String1[i];
cout << endl;
```

output: -
Chirag throw a ball

Using functions

Now we will see a few functions which can help us deal with multiple strings.

Function compare ()

The C++ does not have the strcmp function for the string objects but a similar compare() function which provides similar functionality. Exactly like strcmp, it checks for all three cases, both strings being equal, first string being lexicographically (in the alphabetic order like the library) larger or the second string being larger. Compare function is called with another string as an argument. When it returns zero, both strings are equal, when it returns lesser value than 0, the first string is lexicographically smaller and when it returns greater value than 0, the first string is lexicographically larger than the second. Following code describes these three cases.

```
int val = String1.compare(String2);

if (val == 0)
 cout << "Both the strings are equal\n";
else if (val > 0)
 cout << "Second String is lexicographically greater \n";
else if (val < 0)
 cout << "Second String is lexicographically lesser \n";
```

output: -
Second String is lexicographically greater

Function swap ()

The function swap is also a useful function. It takes one string argument and swaps the calling string object value with the string object value of the argument.

```
cout << "Strings before swapping \n";

cout <<"First string is " << String1 << endl;
cout <<"Second string is " << String2 << endl;

String1.swap(String2); // String2.swap(String1) will have same effect

cout << "Strings after swapping \n";

cout <<"First string is " << String1 << endl;
cout <<"Second string is " << String2 << endl;
```

output: -
Strings before swapping
First string is Vishwanathan is a Chess Player
Second string is England Won the worldcup
Strings after swapping
First string is England Won the worldcup
Second string is Vishwanathan is a Chess Player

Yielding Characteristics of the string

In this section, we will be dealing with a few functions which help us deal with the characteristics of the string object. We will look at four functions which are used the most.

Function empty ()

If we want to see if the string is empty, this function is handy. This function does not take any argument and returns a bool value which is true if the string is empty. Otherwise, it returns false.

Function size() and max_size ()

The function size returns the size of the string. Another function length() can also be used for the same. The function max_size() returns maximum size of the string for a given compiler under a given machine. Here is an example.

```
        cout << "\nCurrently the size of the string is " << tString.size();
        cout << "\nCurrently the string is empty: "<< tString.empty();
        cout << "\nThe maximum size of string is " << tString.max_size();
```

output: -

Currently, the size of the string is 30
Currently, the string is empty: false
The maximum size of string is 18446744073709551599

Function resize ()

The function resize contains a single argument, it resizes the string with the supplied argument size.

```
        void Resize(string tString, int NewSize)
{
   cout << "The original string is :"<< tString << endl;
   cout << "Original size of the string is : " << tString.size()<< endl;

   tString.resize(NewSize);

   cout << "Now the string size is : " << tString.size()<< endl;
   cout << "The resized string is :"<< tString << endl;
}
Resize (String1, 100);
Resize (String2,10);
```

The original string is: England Won the worldcup
Original size of the string is: 24
Now the string size is: 100
The resized string is: England Won the worldcup
The original string is: Vishwanathan is a Chess Player
Original size of the string is: 30
Now the string size is: 10
The resized string is: Vishwanath

Summary

We have seen how arrays can be defined and used with objects as members at the beginning of this module. We have learned to define and use string objects after that. We

have seen through numerous examples how string objects work in accordance with the idea of providing user-defined types to work as close to built-in types as possible. We have seen operators as well as a few member functions of the string class.

ePG Pathshala

Subject: Information Technology

Paper: Numerical Methods

Module 1 : Introduction to Numerical Methods and Errors

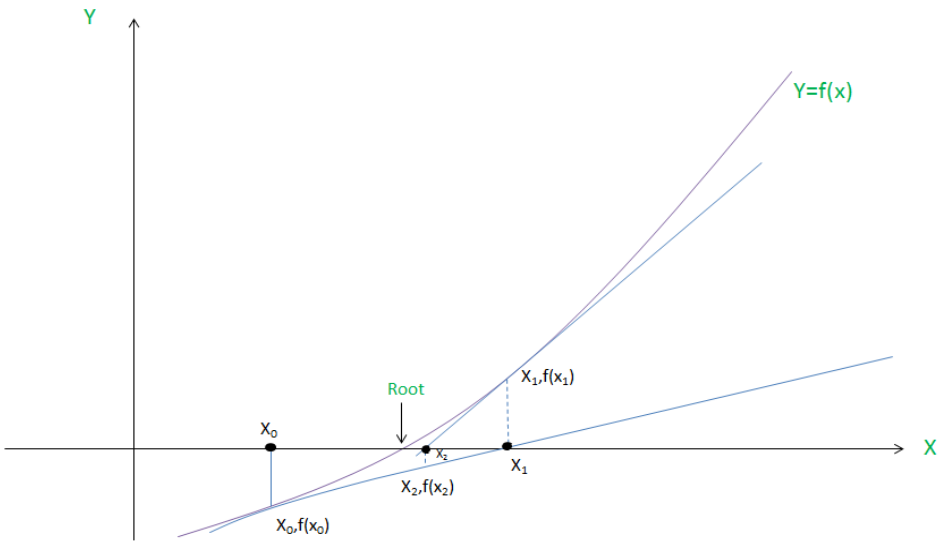
1. Introduction:

The first step in solving many problems in in Science, engineering, business, economics, biomedicines etc. is to build a mathematical model. This mathematical model is built using several principles and laws of science and engineering. Implementation of this mathematical model leads to formulation of mathematical problems to be solved. The mathematical problems arrived at could be one of the following types, but not limited to only these. Let us have a glimpse at these problems one by one. Our modules in this paper are based on studying numerical methods to solve these problems.

2. Overview of different problems discussed in this paper of numerical methods:

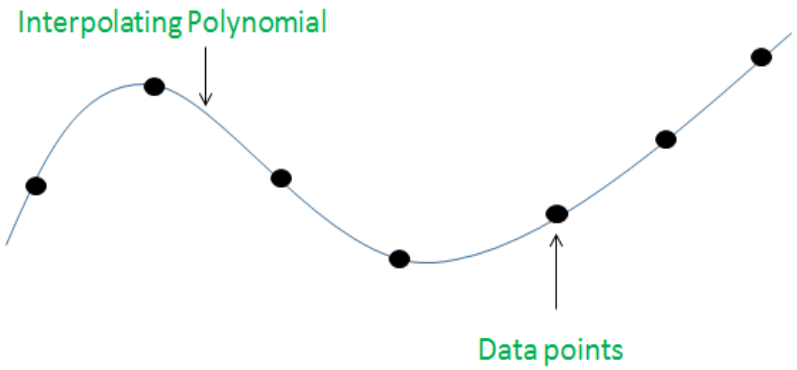
2.1. Roots of an equation f(x) = 0

One comes across the problem of finding roots of an equation f(x) = 0. In modules 3 to 9, we would be learning different methods for the same, namely, Bisection method, method of False Position, Secant Method, Successive Approximation method, Newton Raphson method and special techniques to find roots of a polynomial as they deserve special attention



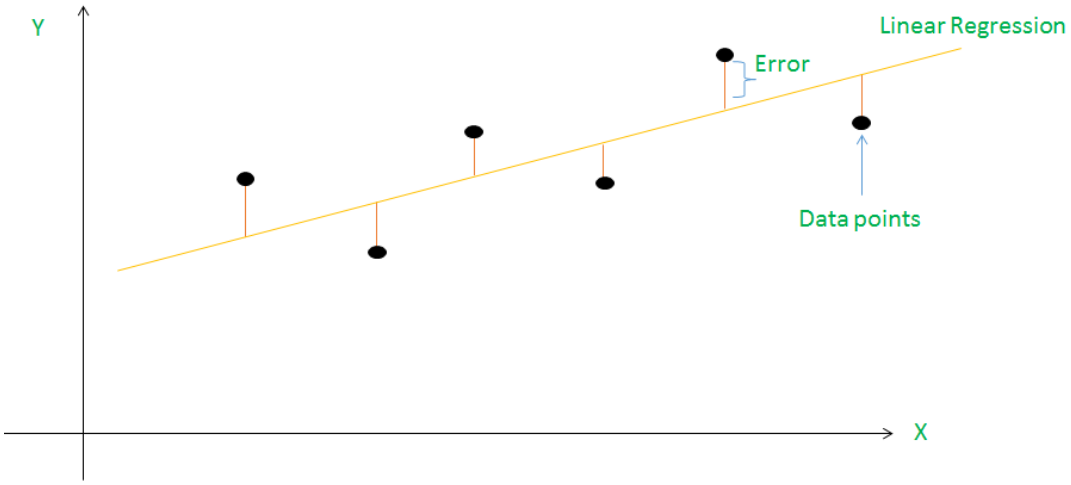
2.2. Interpolation

It turns out that due to many reasons (to be discussed in detail in module 10), function values are available at discrete points and function value is required at an argument, for which the function value is unknown. One of the ways to estimate function value is through interpolation. Interpolation is discussed in modules 10-16 and inverse interpolation in module 17.



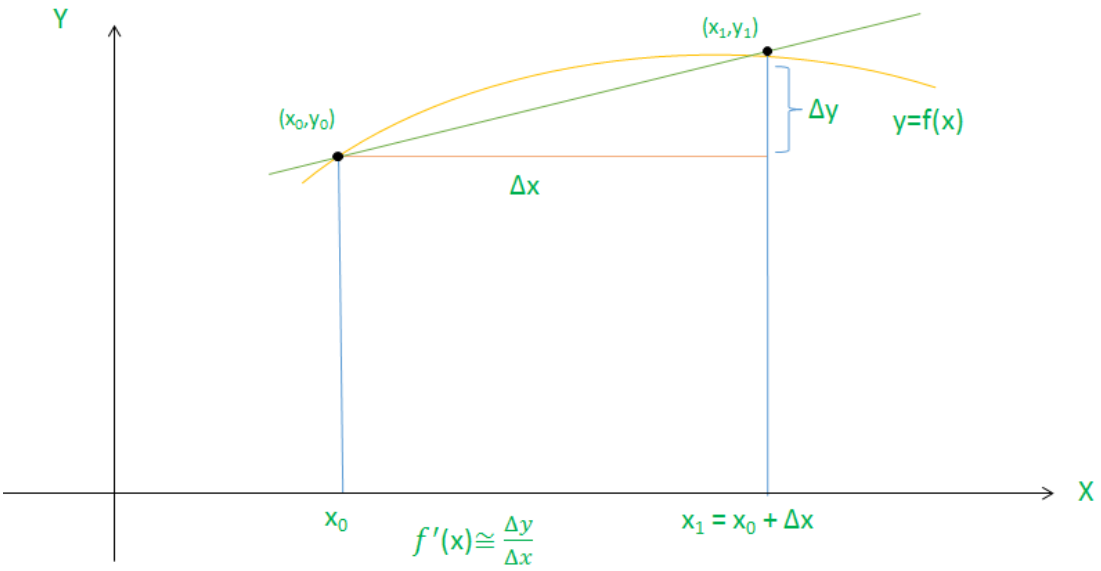
2.3. Least Square Curve fitting

Many times function (values available at discrete points) is to be modeled by a curve whose algebraic equation is known and parameters of the curve are determined by method, known as least square curve fitting. Least square curve approximation and other approximation methods are discussed in modules 18-20. The simplest least square curve fitting is when the curve being fitted is a straight line.



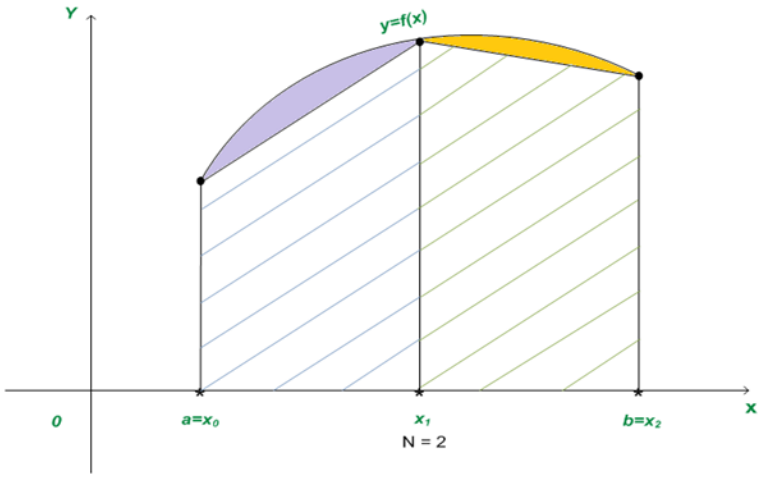
2.4. Numerical Differentiation

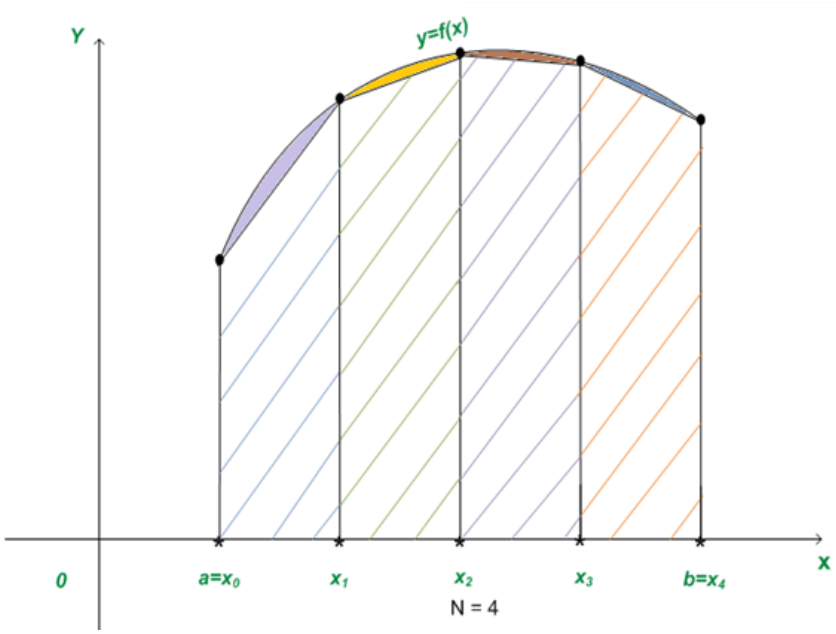
In several cases, one looks for the value of derivative of function at a point. The process of estimating derivative of a function at a point, be it tabular or non-tabular is called numerical differentiation. Numerical differentiation is covered in module 21.

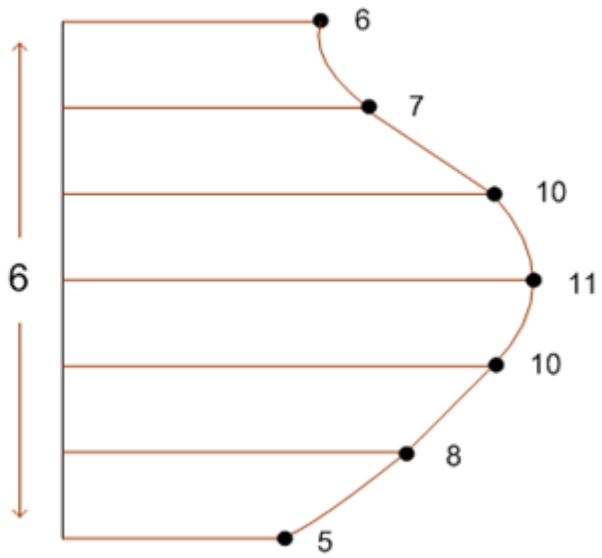


2.5. Numerical Integration

As derivative is to be estimated, similarly, integral of function is required to be estimated over an interval, with the help of function values at the discrete set of points. (Integrand expression may be available or not available) Numerical Integration methods come to our rescue. Numerical Integration methods are elaborated in modules 22-26

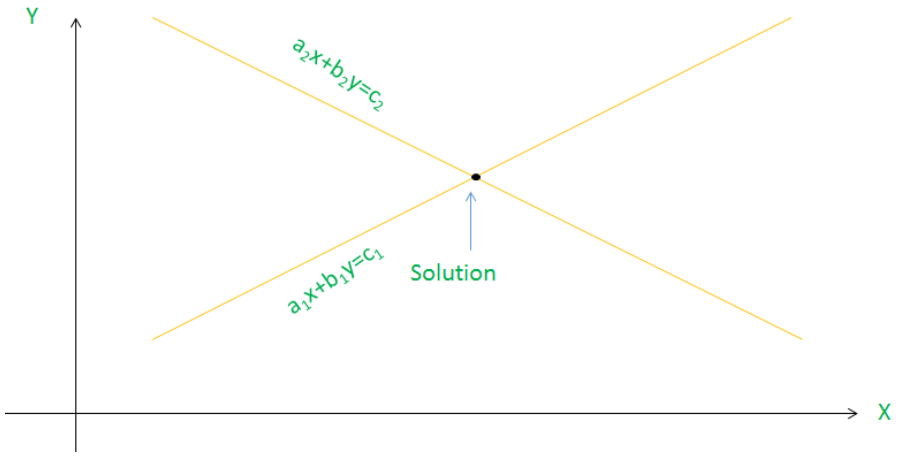






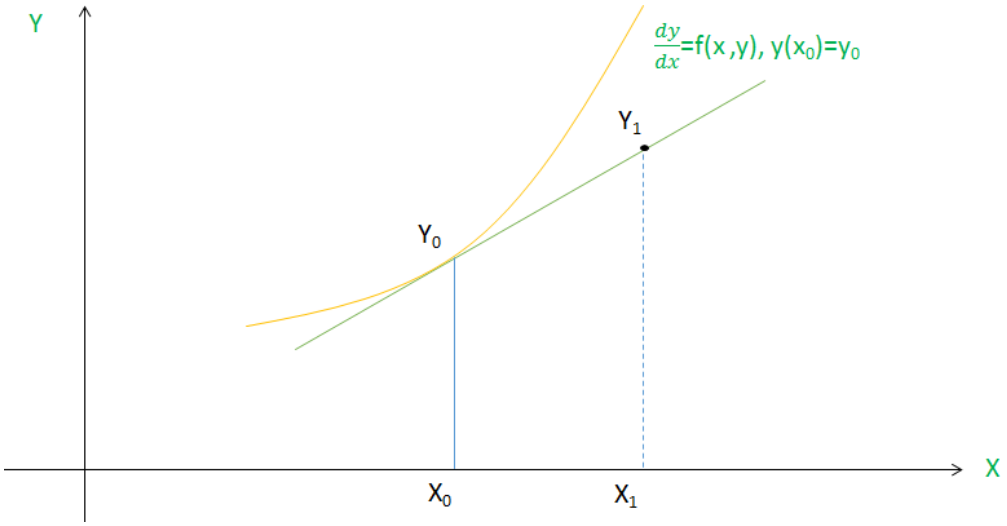
2.6. Solution of Simultaneous Equations

You are quite familiar with simultaneous linear equations and have solved them in school days using elimination techniques. In real life situations, one faces simultaneous linear as well nonlinear equations in large number of variables to be solved, which is tedious using analytical methods. Solution of simultaneous equations, determination of eigen values and eigen vectors of matrices (matrices are associated with coefficients occurring in equations) has been presented in modules 27-32



2.7. Solution of Differential Equations

Many scientific and other problems on mathematical modeling give rise to differential equations, whose solution is to be determined. Methods for finding numerical solutions are discussed in modules 33-35.



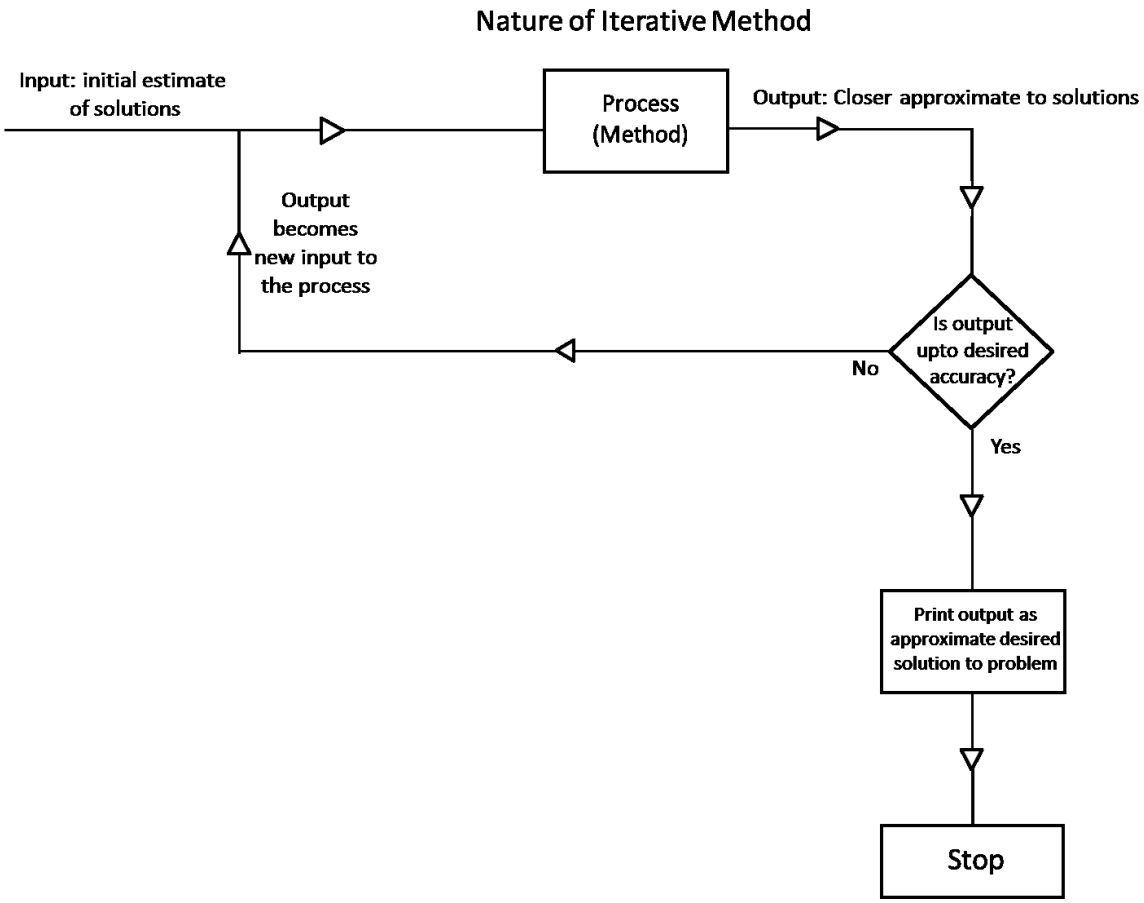
3. Why numerical methods?

Many times analytical methods to solve a given problem may not exist at all (not known so far), or are too laborious and complex to apply. In many situations, information (Data) available does not admit applicability of direct analytic methods. Like, if function is not known and only values are available at discrete arguments, analytical methods are of no use. Finally, analytic methods exist but are quite time consuming due to huge data/complex functions involved. As a result, numerical methods are of great rescue and results are obtainable to the desired accuracy in many situations.

4. How are numerical methods different from analytical methods?

Let us understand some characteristics of Numerical Methods. Numerical methods are applicable to numerical problems, that is, the problem to be solved by any numerical method has no scope of containing arbitrary parameters. Solution cannot be determined in terms of parameters. For example, no numerical method would be applicable for finding root of a quadratic equation $ax^2 + bx + c = 0$ as the equation contains parameters a, b, c . Nevertheless, any equation with known coefficients, for example, $2x^2 - 3x + 5 = 0$ would be solvable by an appropriate numerical method. In short, numerical answer to a numerical problem is obtained under numerical method. If the same type of problem with different data set is to be solved, the entire method is to be reapplied. Numerical methods act like algorithms and given problem acts like data set. It is likely, that there would be many methods to solve a given type of problem, all justified by mathematical analytical theory and to select the most suitable method for the occasion requires considerable skill.

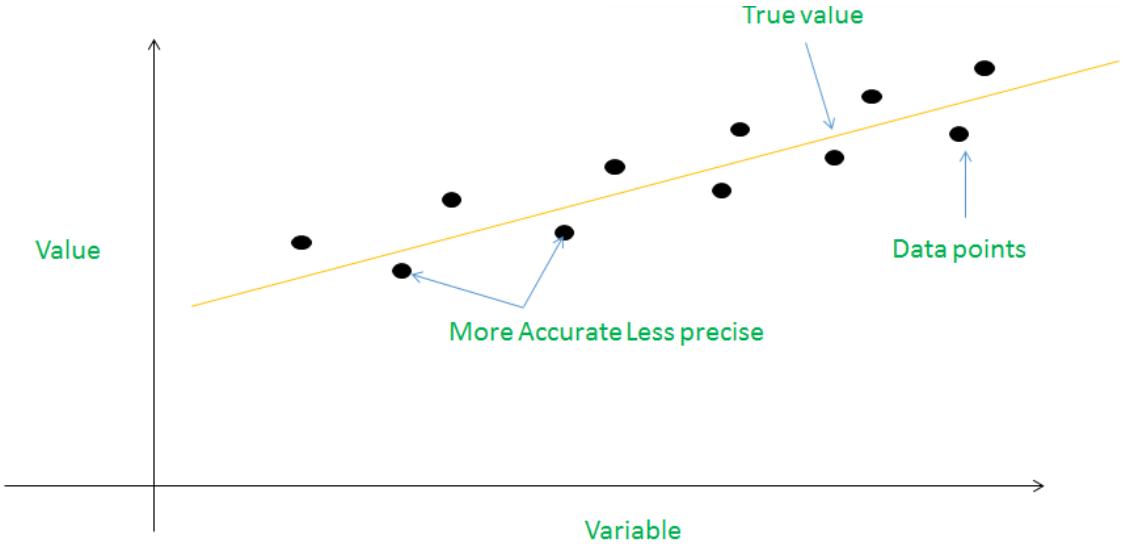
Secondly, analytical methods boast of giving exact answer whereas numerical methods can only ensure approximate answer. But, in many of numerical methods, good approximate answer to desired accuracy can be obtained. Also, the computations are fast, many of the methods being iterative in nature are the best candidate to be implemented on computer. One can expect the answer to complex problem within time limit. Some methods are direct methods also. The following diagram gives basic nature of iterative processes applied in numerical methods.

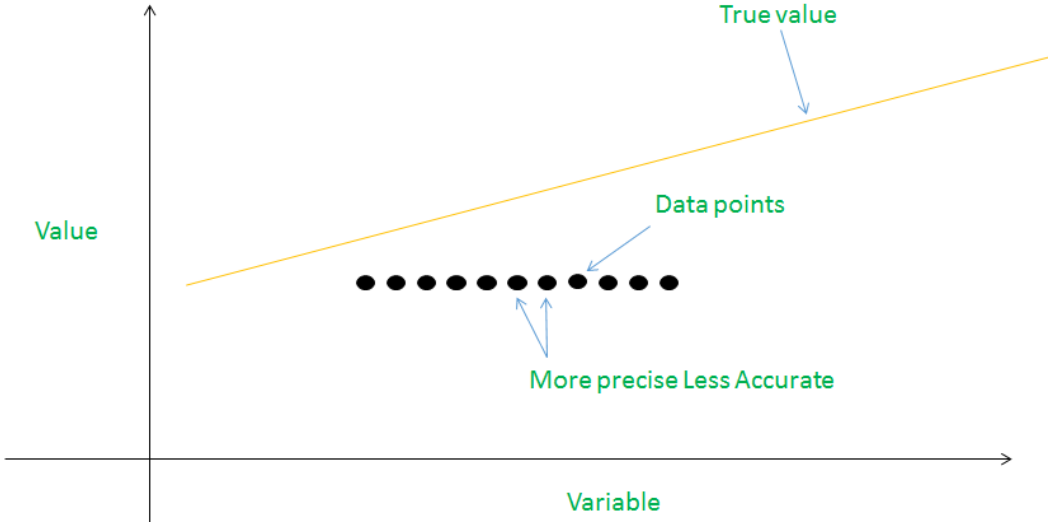


5. Quantification of Errors

Before the problem arrives to a numerical analyst, it undergoes different stages and errors may creep in. Also during computations, errors arise due to several reasons. Different sources of errors and types of error are discussed in our next module 2. Here we shall learn different measures of error. Before that, we also need to understand the difference between accuracy and precision.

Accuracy is a measure of how close the estimated answer of the problem is to true solution, the closer the answer to true value, more accurate it is. Precision is, how closely values agree to each other.





5.1. True Error

True error is defined as difference of true value (exact answer) and approximate answer. Thus, True Error: E_t = *True Value – Approximate Value*. For example, suppose, true value of a quantity = 7.893 and approximate value = 7.672 then

E_t= *True Value – Approximate Value*

= 7.893 – 7.672

= 0.321

Note that, True error can be positive or negative.

If true value = 7.893 as above, but approximate value = 7.975, then

E_t= 7.893 – 7.975 = – 0.082

5.2. Absoluter (true) Error

Usually one is interested in the magnitude of error. One is interested in, "How much is the error?" So, we have, Absolute (true) error given by

$$\textit{Absolute (true) error}: |E_t| = |\textit{True Value} - \textit{Approximate Value}|$$

5.3. Relative Error

Absolute error may not be desirable in each and every case, as it is measure of amount of error and does not take into account of order of value. It needs to be normalized. This leads to definition of Relative Error as

$$\textit{Relative Error} = \frac{E_t}{True\ Value}$$

$$= \frac{True\ value - Approximate\ value}{True\ Value}$$

5.4. Percentage Error

Multiplication of relative error by 100 gives % error denoted by e_t

$$e_t = \frac{True\ value - Approximate\ value}{True\ Value} * 100\%$$

And therefore, absolute relative error and absolute % error are given by $\left|\frac{True\ value - Approximate\ value}{True\ Value}\right|$ and $|e_t| = \left|\frac{True\ Error}{True\ Value}\right| * 100\%$; respectively.

Illustration 1:

Let true value be 20 meters, absolute true error = 1 cm (approximate value is 20.01 meters)

$$|\text{Relative error}| = \left|\frac{True\ value - Approximate\ value}{True\ Value}\right|$$

$$= \left|\frac{1}{2000}\right|$$

$$= 0.0005 = 0.05\%$$

Whereas, if true value = 1 cm and absolute true error = 1 cm (approximate value 2 cm) then

$$|\text{Relative error}| = \left|\frac{True\ value - Approximate\ value}{True\ Value}\right|$$

$$= \left|\frac{1}{1}\right|$$

$$= 1 = 100\%$$

5.5. Approximate errors

While solving numerical problems, (class room discussion for explaining is different) , usually true solution would be known in only limited cases, like while testing an application/software/ program's performance , program would be tested through test cases as input . If exact answer of a problem is known, why would one apply numerical method for the same? So, for all practical reasons, true answer is not known. Thus measures of errors need to be modified. Methods being iterative in nature, what can be better than current estimate as a substitute for true value and previous estimate as approximate value. As a result, Absolute approximate error is given by

|Approximate error| = |Current estimate – Previous estimate|; and

Approximate relative error = $\left|\frac{Approximate\ error}{Current\ Estimate}\right|$

Approximate % relative error = $\left|\frac{Approximate\ error}{Current\ Estimate}\right| * 100\%$, denoted as $|e_a|$

Illustration 2:

Following is the table of iterations being performed for finding roots of an equation f(x) = 0 by Bisection method (Module 3). The column containing c_k's are successive estimates for the root. Let us calculate approximate % error in some iterations.

Iteration No.	***a***	***f(a)***	***b***	***f(b)***	C_k	$f(c_k)$
0	0	-1	1	1	0.5	-0.375
1	0.5	-0.375	1	0.17188	0.75	0.1719
2	0.5	-0.375	0.75	0.17188	0.625	-0.1309
3	0.625	-0.1309	0.75	0.01245	0.6875	0.0125
4	0.625	-0.1309	0.6875	0.01245	0.65625	-0.0611
5	0.65625	-0.0611	0.6875	0.01245	0.67188	-0.0248
6	0.67188	-0.0248	0.6875	0.01245	0.67969	-0.0063
7	0.67969	-0.0063	0.6875	0.01245	0.6836	0.0031

Iteration No.	*a*	*f(a)*	*b*	*f(b)*	C_k	$f(c_k)$
8	0.67969	-0.0063	0.6836	0.00305	0.68165	-0.0016
9	0.68165	-0.0016	0.6836	0.00305	0.68263	0.0007
10	0.68165	-0.0016	0.68263	0.00072	0.68214	-0.0005
11	0.68214	-0.0005	0.68263	0.00072	0.68239	0.0001
12	0.68214	-0.0005	0.68239	0.00015	0.68227	-0.0001
13	0.68227	-0.0001	0.68239	0.00015	0.68233	0

Iteration - I

$$e_a = \frac{0.75 - 0.5}{0.75} * 100 = 33\%$$

Iteration - II

$$= \frac{0.625 - 0.75}{0.625} * 100 = 20\%$$

Iteration – last but one

$$= \frac{0.68227 - 0.68239}{0.68227} * 100 = 0.01758\%$$

Iteration – last

$$= \frac{0.68233 - 0.68227}{0.68233} * 100 = 0.00879\%$$

5.6. Stopping Criterion

There are many stopping criterions (achieved the desired accuracy, satisfied with the answer obtained). Two of them are:

(a) Absolute approximate error < Pre assigned tolerance say ϵ , where $\epsilon > 0$

(b) Absolute Relative Error < ϵ

If one is interested in obtaining answer correct to certain number of significant digits say m, then

$$|e_a| \leq \frac{1}{2} * 10^{2-m}$$

****

Role	Name	Affiliation
Principal Investigator	Dr. Savita Gandhi	Professor, Dept. of Computer Science, Gujarat University, Ahmedabad
Content Writer	Mr. Hardik Joshi	Asst. Professor, Dept. of Computer Science, Gujarat University, Ahmedabad
Content Reviewer	Dr. Hiren Joshi	Professor, Dept. of Computer Science, Gujarat University, Ahmedabad

Item	Description
Subject Name	Information Technology
Paper Name	Open Source Software
Module No	4
Module Name	GNU/Linux Structure & Installation
Pre-requisite	Knowledge of Computing
Objectives	To be able to install and operate the Linux OS using GUI
Keywords	Disk partition, boot, installation, centos, ubuntu, virtualization

GNU/Linux Structure & Installation

Finalizing the Linux Distribution

As we have seen in the previous module that Linux OS comes in various distributions. Before finalizing the installation of Linux, we must select the best distribution that fits our need. Our technical expertise and necessity will help in finalizing a particular Linux distribution. As such, Linux does not have different OS for server or for desktop systems, however, in server based distributions the GUI may not be included. There are various criteria to finalize an appropriate distribution for Linux. The following indicators can help us to select a proper distribution:

- What type of hardware is available ? (X86, ARM, PPC etc.)
- What will be the main function of the system (Server or Desktop)
- What type of packages will be installed?
- How much hard disk space is available?
- How long is the support cycle for each release?
- How often are packages updated?
- Is it for experimental use or long term stable system?
- Is there any need of customizable kernel from the vendor?

Linux can be classified on various criteria, the following diagram classifies based on the use (whether personal or network based). There are nearly 1000+ flavours of Linux as of now, most of them are suitable for Server OS or for Desktop OS. A curious user may surf internet and try to find out the distribution that is mostly used by others. We must select the distribution such that proper help is available through nearby communities.

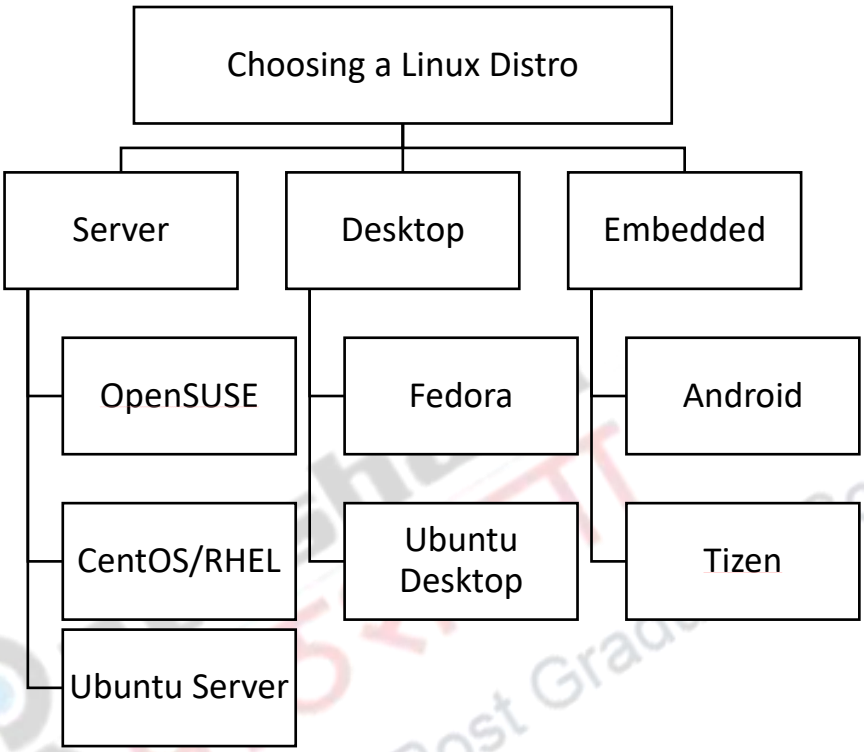


Figure 1: Linux Distributions

Installation of Linux for personal use can be non-commercial solution like Ubuntu, Fedora, OpenSuse, CentOS etc., however, if we want to install for organization, it is beneficial to get a commercial product. The following Linux distributions can be installed for academia:

- CentOS (https://www.centos.org/download/) – Good for Network Administrators
- BOSS (http://www.bosslinux.in/) – Indian Linux
- Trisquel (http://trisquel.info/en/download) - Complies to free software
- Ubuntu (http://www.ubuntu.com/download/desktop) - Adopted by many schools

Modes of Installation

Depending on the availability of the system, its hardware configuration and installed software, we may select either of the modes of installation that are listed below:

- Bare Metal Installation
- Dual boot with other OS
- Using Live Media like Live CD/DVD/USB
- Using a host machine hypervisor program (eg. VMWare, Virtual Box, Hyper-V etc.)

A computer system with no OS installed in it is generally called bare metal. Installing Linux on bare metal is completely risk free as there is no thread of losing any data. However, we have to check the hardware configuration and decide whether the hardware present in our system supports the distribution that we are planning to install. In case, if there already exists and OS on the computer, we can plan of installing Linux as dual boot provided there is an empty partition in the hard disk. We must ponder on the following points before installing Linux as dual boot system:

- Is there sufficient hard disk space so that two or more OS can reside on a Single hard disk
- Is there any OS that already exists (eg. Windows)
- Separate partition must exist or needs to be re-partitioned
- Risk of losing previously installed OS or data

There are chances of modifying the boot sector while installing Linux as dual boot system. The prior OS may not be visible if any mistake is made during the installation process.

Since few years, Linux distributions are also available such that there is not need to install Linux on hard disk. We can use Linux directly from the USB drive or CD/DVD. Such kind of Linux are known as live media, the CD/DVD or USB can be bootable. As the media is bootable, we must ensure that the system boots from media and not from the hard disk. Using Linux from live media is surely fun as it does not require any installation, there is also no risk of losing data residing on the hard disk, however while using live media, the following issues may arise:

- System performance may degrade (as data transfer rate is less)
- Persevering the files from live media to hard disk can become trickier

Since past few years, good hardware is available at cheaper costs. So, if we have a good hardware with more RAM and efficient processor, we can think of using virtualization software to install Linux.

Installing Linux using hypervisors

With the development of virtualization technologies, installation of Linux has become very easy. There is no risk of losing any data when Linux is installed as a virtual machine. However, if the hardware is not efficient, the system performance may degrade while using hypervisors. Let us understand how virtualization is done. If there exists some OS on the hardware, we can install virtualization software like VirtualBox, VMWare, Hyper-V within the existing OS. Let us take a scenario, suppose Windows is existing in our system. It is is a licensed version, we may not remove Windows. Under such circumstances, we download the virtualization software and install Linux within the software. After the installation gets over, we will be running two OS concurrently, windows as a host OS and Linux as a guest OS. Such kind of installation will require nearly 10 GB of hard disk space of Windows. The Linux installation will reside as a file in hard disk. Figure 2 illustrates the virtualization technique. The hypervisor or virtualization software fits in between the Host OS and the Guest OS.

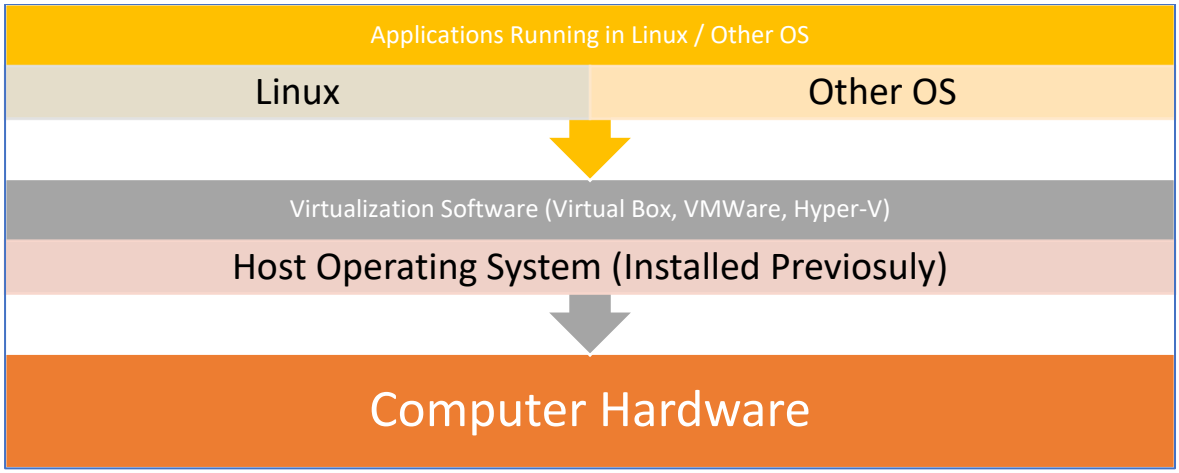


Figure 2: Virtualization

Modern computer hardware contains virtualization functionality. While using virtualization technique, we must remember to enable the virtualization flags of CPU from the BIOS. Latest version of Windows comes with Hyper-V as a feature, in such case there is no need to install any virtualization software but we can start using Hyper-V. Oracle VirtualBox is a free virtualization software that can be downloaded if Hyper-V is not present in the system. Let us see the steps used to install Linux with the help of virtualization.

Installation of Linux

Before we begin with the installation, we must have Linux in .iso format. Linux iso file can be kept in USB drive or on hard disk partition. There is no need for a separate hard disk partition, however, the hard disk must have atleast 10-12 GB of free space if we are planning entire Linux installation. Now, let us see step-by-step screens of installing Linux. The virtualization software demonstrated here is Oracle Virtual Box. The following things need to be taken care of:

- Hard disk must have enough space for installation
- CPU Virtualization must be enabled from BIOS
- We have downloaded proper ISO file
- The ISO file (64 bits) and our hardware (64 bits) must match
- Internet connection is desirable but not necessary

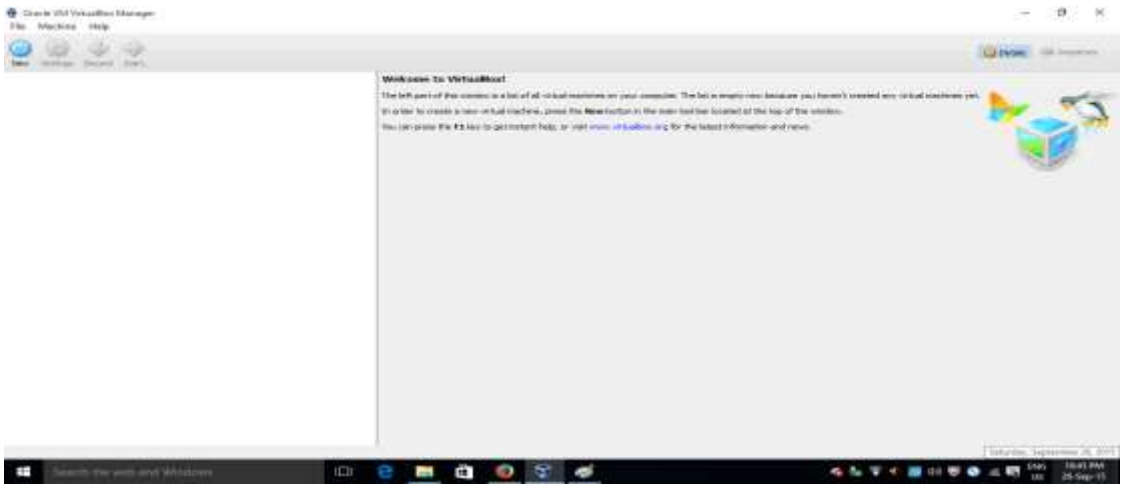


Figure 3: Step-1 (Start Virtual BOX Manager)

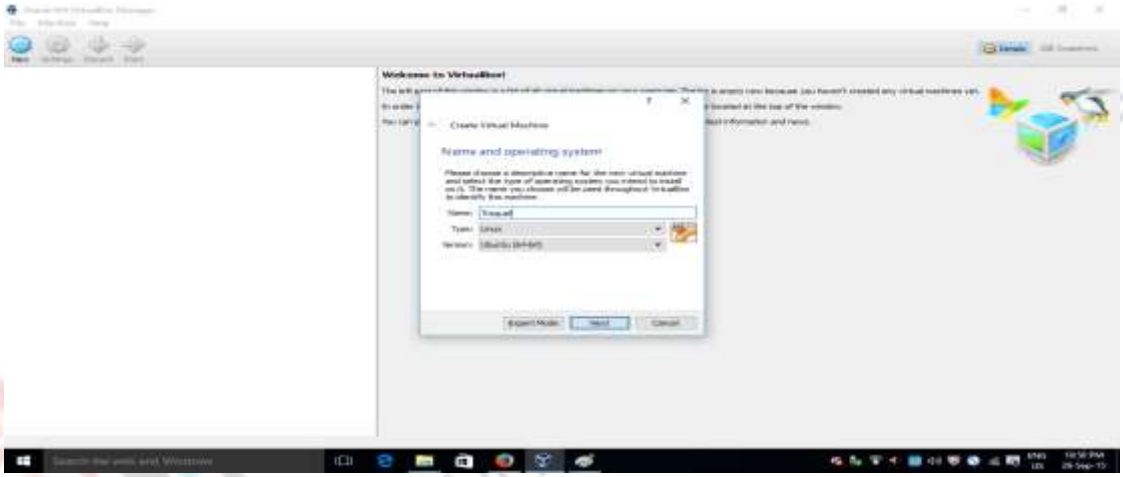


Figure 4: Step-2 (Create a Virtual Machine and provide Name to the OS)

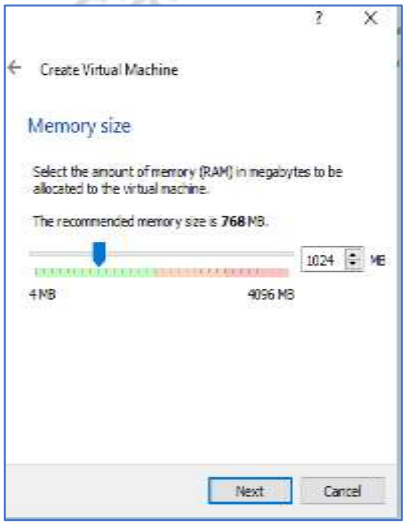


Figure 5: Step-3 (Assign RAM)

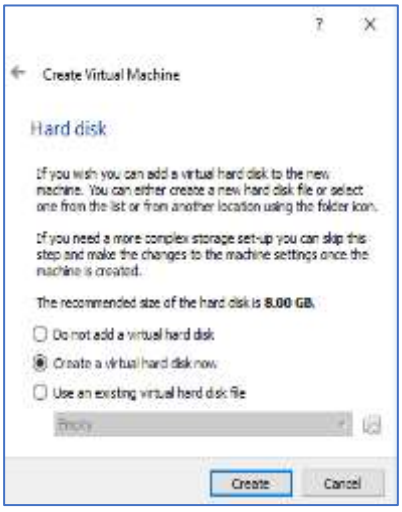


Figure 6: Step-4 (Assign Hard disk space)

Figure 7: Step-5 (Select Hard disk type)

Figure 8: Step-6 (Assign Storage type)

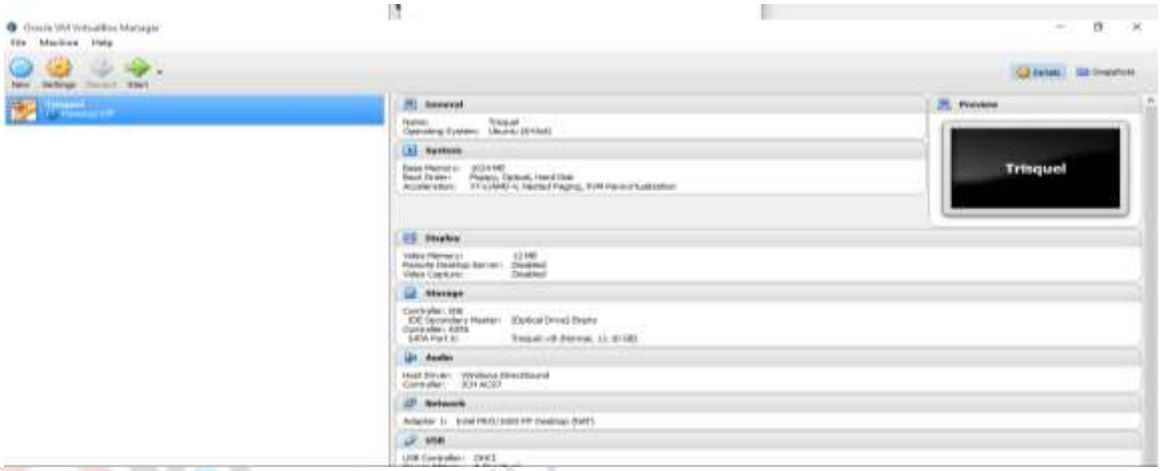


Figure 9: Step-7 (Start the machine)

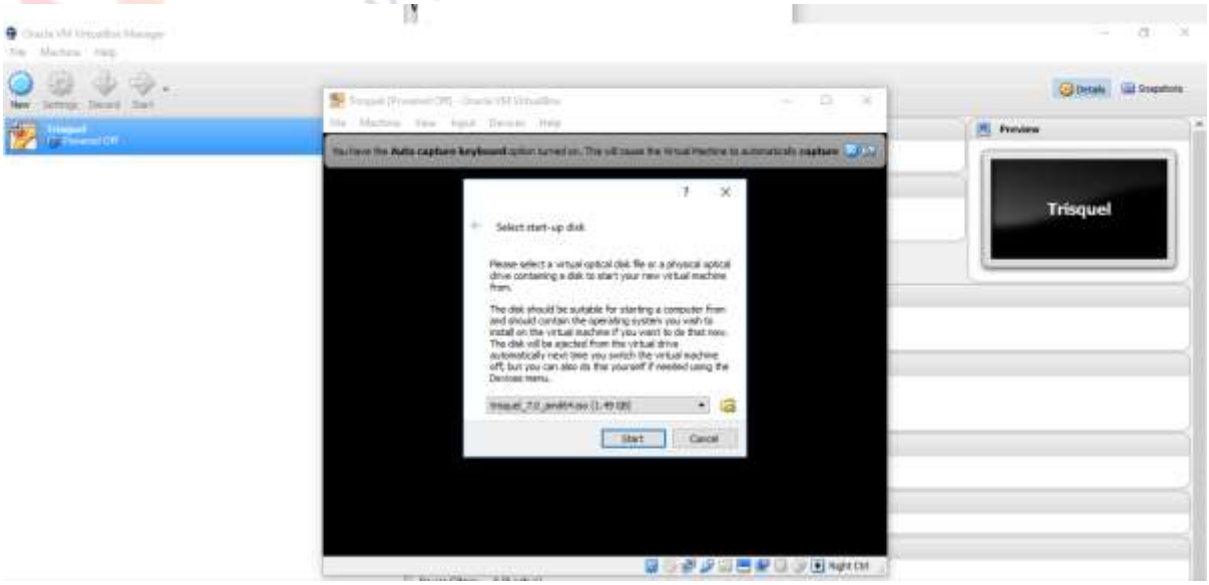


Figure 10: Step-8 (Select the location of ISO File)

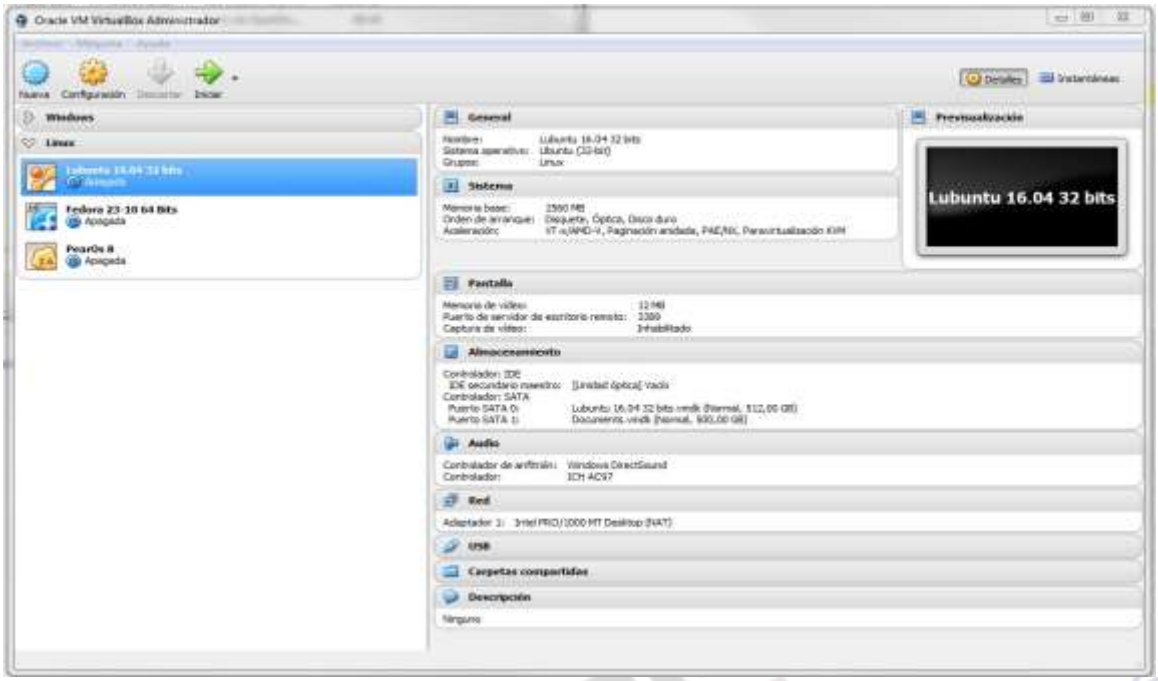


Figure 11: Step-9 (Installation will being)

During the installation process, we will be asked to supply root password, create users, configure the country/time zone, configure languages, select the software to be installed. It is recommended that the root password must be remembered since root user is the administrator account(super user account). It is also recommended to select software/packages before installing to avoid post installation hassles of installation of individual software through internet.

Linux Boot Process & Partitions

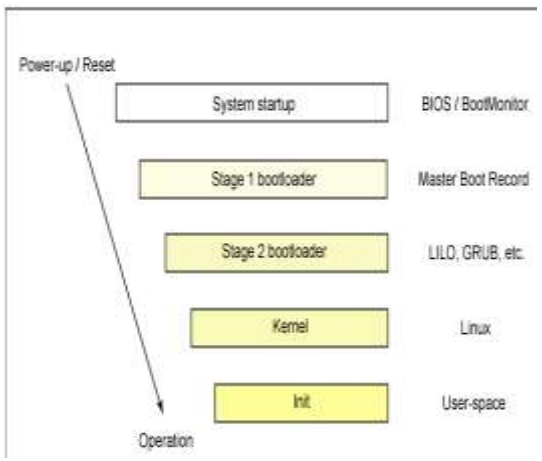
In this section, we will understand the booting process and certain terminology associated with the filesystem. During the installation, we can create partitions and can configure the directories as per the requirement of the organization. For instance, if the organization needs to create 100 users, we must assign more disk space to /HOME directory.

Particulars	Windows	Linux
Partition	Disk1	/dev/sda1
FileSystem type	FAT32/NTFS	EXT3/EXT4/XFS…
Base Folder where the OS	“C drive”	“/” root
Mounting Parameters	Drive Letter	MountPoint

The filesystem of Linux is entirely different from Windows. In Windows, we have C:, D: and so on as hard disk partitions whereas in Linux the partitions can be /boot, /bin, etc. So, in Linux we identify the mount point as ‘/’ (root).

Linux Boot Process

When Linux is successfully installed, we can restart the system. As soon as the system gets restarted, the hardware initializes and gradually the OS will start. The following steps are performed during the booting of Linux:



1. BIOS initializes
2. POST routine executes
3. Boot Loader from MBR is invoked
4. Boot Loader gives a choice of OS to start
5. Boot Loader loads the Linux Kernel
6. Boot Loader loads the initramfs (RAM based file system)

Once the Linux starts, it will display the login screen. We can login to the system and explore GUI provided with Linux. It is desirable to login as some other account and not as root. Since root login has higher privileges and there is a risk of making mistakes with the root login.

Let us summarize the key concepts covered in this module

- Various parameters are used to finalize on a specific Linux Distribution.
- Installation of Linux can be done using different methods, however, the available hardware becomes the key deciding factor.
- Linux can be installed on a partition of hard disk or can be executed from live media or from hypervisor program.
- Hypervisor programs like VirtualBox are the easiest way to learn Linux without taking any risk.

Subject : Information Technology
Paper : Object Oriented Concepts & Programming
Module : Introduction to C++ Programing

Programming Language History

Programming is a process of writing solution to a problem in a computer language so that it can be executed by computer.

With increasing level of complexities of applications the methodology of programming have changed. Initially after the invention of computers the first programming method was- machine level programming. With few hundreds lines of code this approach worked well but as the complexity of programs increased a new language – Assembly Language was invented which uses symbolic representation of machine instructions. It was comparatively easy to write and maintain the code using assembly language. As the program size and complexity kept on increasing, higher level languages were introduced such as FORTAN, C , BASIC , COBOL . A higher level language is one which is closer to human language and thus further from machine language.

C language is general purpose , fast and widely used language. It is well suited for system level application as the code written in C runs faster.

But it is not suitable for large software development as the complexity increases and software become hard to maintain.

There was another language - Simula which had the object oriented features helpful in large software development but was not very efficient in term of time. C++ was inspired by both these languages, thus it contain the object oriented features as well as the efficiency of C .

C++ was created by Bjarne Stroustrup . Its development began in 1979 at Bell laboratories in New Jersey . The language was initially known as “C with classes” . In 1983 the name was changed to C++ , as the language was more than classes . C++ was extension to C language and has the backward compatibility with C.

There are so many programming languages available but C and C++ are one of the most widely used language in software industry. They are used for creating everything from Operating System to embedded systems, games, desktop application and so on.

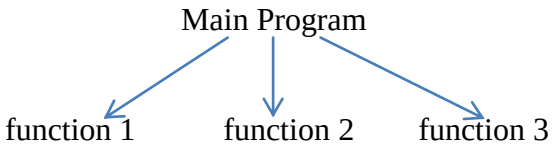
C++ provides support for both procedural programming as well as object oriented programming. Let’s understand the difference between Procedure Oriented and Object Oriented programming paradigms.

Procedure Oriented Approach vs Object Oriented Approach

There are different approaches to build a solution to a specific problem known as Programming Paradigms. Procedure Oriented and Object Oriented are two major paradigms.

The conventional approach (ex. FORTAN, COBOL and C) for solving a problem is Procedure Oriented Programming .

In Procedure Oriented Programming the focus is on Procedures also known as Function or Routines. It gives the step by step approach to solve the problem .It is also known as top down approach – a big problem is broken into smaller segments (functions)

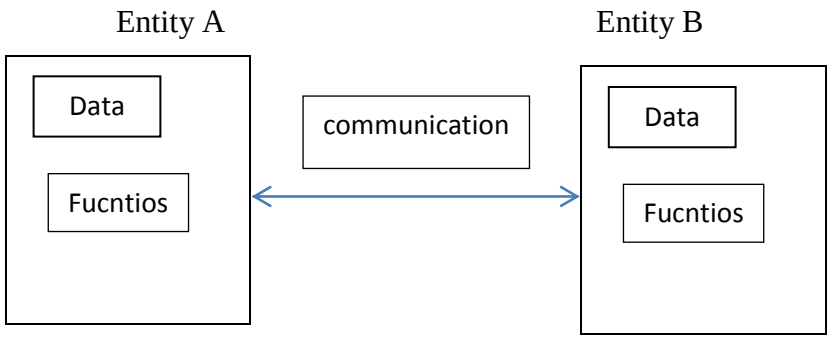


This approach is intuitive as we give a step by step solution to a problem.

Procedure Oriented relies on Procedures and Data which are two separate concepts. Data move freely around these procedures thus there is no data security constraint. There is no control over the way the data is processed or modified. In Object Oriented approach these two concepts are bundled in to a single concept of Entity (object).

In Object Oriented every problem is perceived in terms of collection of Entities / Objects. Every Entity encapsulates the data and behavior in a single unit called Class. Here the data can be processed only in the way that has been defined by the class, this makes the application more maintainable.

Object oriented approach gives the power that the data can be hidden and cannot be accessed by outside world directly. Entities communicate with each other with the help of functions.



Let's understand this with an example of an application for maintaining Students record. In Procedure Oriented programming the approach would be to break the system in smaller modules. We will think of the operation which needs to be performed in this application such as-
Read student's data, display student's data , calculate the grade of a given student, calculate the percentage of a given student , generate a mark sheet of a given student.

These are some of the task which is required in the application. Now the second part is to decide what data are required for the application? Here the data would be required to store student detail thus we need to define a structure –Student having the attributes like – name , roll number, course , marks etc.

Let's see the complete set of data and fucntions

```
struct Student
{
   char name[50];
   int roll;
   char course[50];
   int sem;
   float marks[6];
};

Struct Student readStudent();
void displayStudent (struct Student s);
char calculateGrade(struct Student s);
float calculatePercentage(struct Student s);
void generateMarksheet(struct Student s);
```

Here the data part i.e. structure Student and the functions which operate on this Student structure are written separately. Student data is moving freely among these function therefore no data security is there.

Let's see the same application design using Object Oriented programing approach. In Object Oriented approach we first find out the entities (objects) which take part into this application. Here the main entity is - Student
(In actual System there can be more entities)

After identifying the entities now we need to find the attribute and behavior of these entities.
The Student entity can have the following attributes:

Student id, name, course, semester, marks etc.

And the behavior –

readStudent()
displayStudent()
calculateGrade()
calculatePercentage()
generateMarksheet()

In Object Oriented these attributes and behavior are combined into a single unit – Class.

```
class Student
{
private:
   char name[50];
   int roll;
   char course[50];
   int sem;
   float marks[6];

public:

void readStudent();
void displayStudent ();
char calculateGrade();
float calculatePercentage();
void generateMarksheet();
} ;
```

Here the data part is kept private which means no outside function can directly access the attribute of this class.

Object Oriented Programming Concepts

Classes and Objects

A Class is a user defined data type and an Object is a variable of this Class.

For example suppose we want to store details of Book. There is no data type available for storing the details of book. We can define a C structure for this but a C structure defines only the attributes of an entity and not the behavior. So the functions which operate on this Book entity has to be defined separately .

Object Oriented approach allows us to create more powerful data type- Class, which gives more control over the data type. A class is an extension to C structure. The difference is that structure in C contains only attributes but a Class can contains attribute along with the behavior of the entity.

The class defines the blueprint of a real world entity. An Object is an instance of a class. If we need a real world entity then we need to instantiate object of a class.

Here is an example of a Time class:

Example:

```
class Time
{
private:
        int second;
        int minute;
        int hour;

public:

        void displayTime();
        void readTime();
        Time addTime(Time t);
        Time subtractTime(Time t );

};
```

Properties

Methods

Here the class contain the attributes of Time i.e. hours, minutes and seconds along with the methods to read time, display time, add two time entity and subtract time entity.

We will see in the coming lectures that C++ class data type is as powerful as the standard data types.

Encapsulation and Data Hiding:

Encapsulation refers to combining both data and function that manipulate the data together. In C++ encapsulation is achieved by Class. As we have seen that a Class in C++ contains both data and functions which operate on data. This makes the data highly secured from direct access. In the above Time example the attribute of Time entity and the methods are encapsulated in the single Time class.

Encapsulation led to another important Object Oriented programming concept **Data Hiding** where a programmer can decide what part of the class should be visible (public) and what must be hidden(private/protected) from outside world.

In the Time class example the properties of class are kept "private" , which means these properties are available only to the functions of this class but hidden from outside world.

The advantage of this concept is that the data is safe; programmer cannot access the data directly. A non-member function cannot access an object's private or protected data. (An exception to this is friend function which is covered in coming module)

Abstraction

The idea of abstraction is to have a higher level look at the task.

Data Abstraction allows us to create our own data type (using class), where we can define the variables and operations on this new data type. (Known as Abstract Data Type)

With data abstraction we can focus on what operation needs to be performed on the data without bothering about how these operations are implemented.

In the above Time Class example, if we want to find the difference of two Time objects then we just need to instantiate the Time objects and invoke the method subtractTime(Time t), but we don't need to think how this difference is found .

Abstraction is an excellent tool for managing complexity. Well designed abstraction can make complex and large size problem simpler and manageable.

Reusability:

Reusability refers to reusing the existing class. In C++ once a Class is defined, it can be reused in any application. for example once we define a Time Class as given in the example above, in any application if we need a Time object then we just need to instantiate this class.

C++ strongly supports reusability through Inheritance and STL (Standard Template Library). Inheritances allow a class to reuse data and methods of an existing class. STL contain built-in classes which can be used in any C++ application. STL contains the general purpose classes such as Stack, Vector, Queue etc.

Inheritance:

Real world objects do not exists in isolation. The basic idea of inheritance is defining a new object in terms of an existing object. The advantage of this feature is – code reusability and hierarchical relationship

For example if we define a Person class which contains the basic characteristics of a Person such as – name, dob , address etc. and methods such as readData() , DisplayData(), calculateAge() etc. Now if we want to define a Student class then either we can create a completely new class Student from scratch or we can use existing Person class and add the specific characteristics of Student.

Thus inheritance imposes a hierarchical relationship where a child class inherits data as well as behavior (methods) from its parent class.

C++ strongly supports reusability through Inheritance and STL (Standard Template Library). Inheritance allow a class to reuse data and methods of an existing class. STL contain built-in classes which can be used in any C++ application . STL contains the general purpose classes such as Stack, Vector , Queue etc.

Polymorphism :

The meaning of polymorphism in general terms is – one name multiple forms. One function call may results into different behavior in different instances. Polymorphism helps in making user defined data type more powerful. We will discuss Polymorphism in successive lectures.

Summary:

- C++ is an extension of C language and has the backward compatibility with C.
- C++ model is not pure object oriented model. It gives power of both Procedure Oriented and Object Oriented Programming
- C++ Classes allows us to create more powerful data type.
- Elements of object oriented paradigm such as Encapsulation, Data hiding, Abstraction, Inheritance, Polymorphism etc. makes the language suitable for large software development.