## Module-1:

## LECTURE-1

**Introduction:**

Programmers write instructions in various programming languages to perform their computation tasks such as:

(i) Machine level Language
(ii) Assembly level Language
(iii) High level Language

**Machine level Language :**

Machine code or machine language is a set of instructions executed directly by a computer's central processing unit (CPU). Each instruction performs a very specific task, such as a load, a jump, or an ALU operation on a unit of data in a CPU register or memory. Every program directly executed by a CPU is made up of a series of such instructions.

**Assembly level Language :**

An assembly language (or assembler language) is a low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to-one) correspondence between the language and the architecture's machine code instructions. Assembly language is converted into executable machine code by a utility program referred to as an assembler; the conversion process is referred to as assembly, or assembling the code.

**High level Language :**

High-level language is any programming language that enables development of a program in much simpler programming context and is generally independent of the computer's hardware architecture. High-level language has a higher level of abstraction from the computer, and focuses more on the programming logic rather than the underlying hardware components such as memory addressing and register utilization.

The first high-level programming languages were designed in the 1950s. Now there are dozens of different languages, including Ada , Algol, BASIC, COBOL, C, C++, JAVA, FORTRAN, LISP, Pascal, and Prolog. Such languages are considered high-level because they are closer to human languages and farther from machine languages. In contrast, assembly languages are considered low-level because they are very close to machine languages.

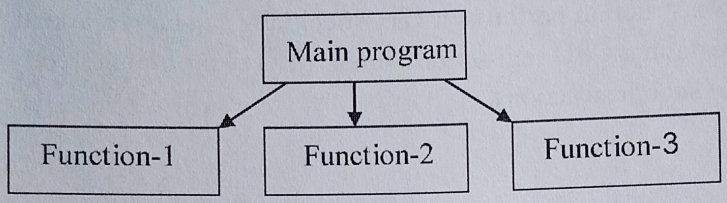
The high-level programming languages are broadly categorized in to two categories:

(iv) Procedure oriented programming(POP) language.
(v) Object oriented programming(OOP) language.

## Procedure Oriented Programming Language

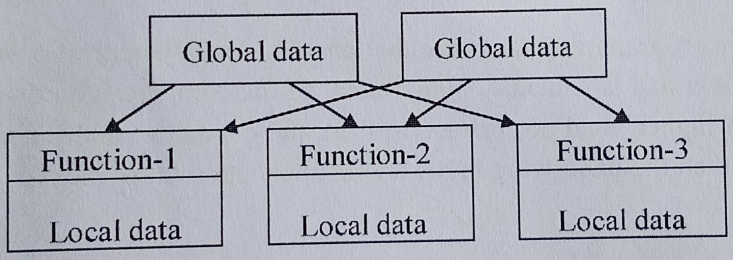
In the procedure oriented approach, the problem is viewed as sequence of things to be done such as reading , calculation and printing.

Procedure oriented programming basically consist of writing a list of instruction or actions for the computer to follow and organizing these instruction into groups known as functions.




The disadvantage of the procedure oriented programming languages is:

1. Global data access
2. It does not model real word problem very well
3. No data hiding



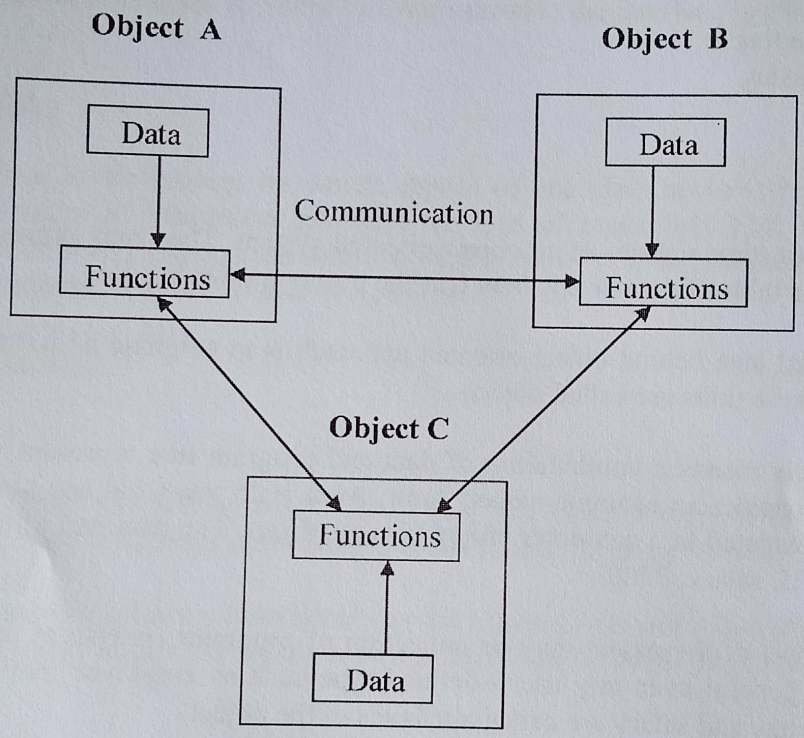

## Characteristics of procedure oriented programming:

1. Emphasis is on doing things(algorithm)
2. Large programs are divided into smaller programs known as functions.
3. Most of the functions share global data
4. Data move openly around the system from function to function
5. Function transforms data from one form to another.
6. Employs top-down approach in program design

## LECTURE-2

### Object Oriented Programing

"Object oriented programming as an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand".




Features of the Object Oriented programming

1. Emphasis is on doing rather than procedure.
2. programs are divided into what are known as objects.
3. Data structures are designed such that they characterize the objects.
4. Functions that operate on the data of an object are tied together in the data structure.
5. Data is hidden and can't be accessed by external functions.
6. Objects may communicate with each other through functions.
7. New data and functions can be easily added.
8. Follows bottom-up approach in program design.

## LECTURE-3

# BASIC CONCEPTS OF OBJECTS ORIENTED PROGRAMMING

1. Objects
2. Classes
3. Data abstraction and encapsulation
4. Inheritance
5. Polymorphism
6. Dynamic binding
7. Message passing

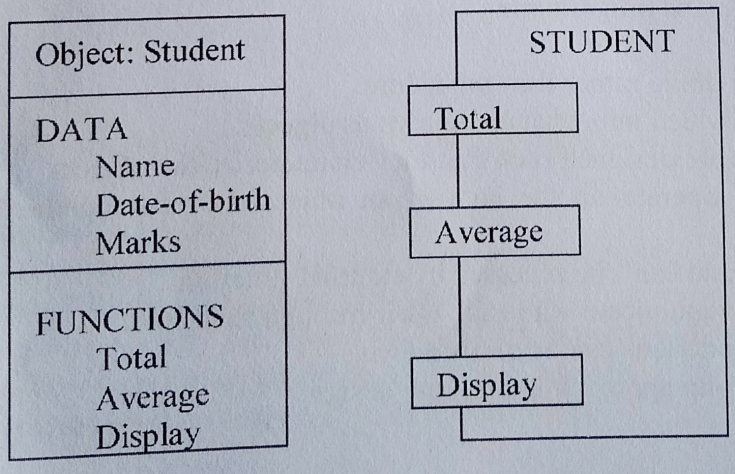
## OBJECTS

Objects are the basic run-time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program must handle.

The fundamental idea behind object oriented approach is to combine both data and function into a single unit and these units are called objects.

The term objects means a combination of data and program that represent some real word entity. For example: consider an example named Amit; Amit is 25 years old and his salary is 2500. The Amit may be represented in a computer program as an object. The data part of the object would be (name: Amit, age: 25, salary: 2500)

The program part of the object may be collection of programs (retrive of data, change age, change of salary). In general even any user –defined type-such as employee may be used. In the Amit object the name, age and salary are called attributes of the object.




## CLASS:

A group of objects that share common properties for data part and some program part are collectively called as class.

In C ++ a class is a new data type that contains member variables and member functions that operate on the variables.

## DATA ABSTRACTION :

Abstraction refers to the act of representing essential features without including the back ground details or explanations. Classes use the concept of abstraction and are defined as size, width and cost and functions to operate on the attributes.

## DATA ENCAPSALATION :

The wrapping up of data and function into a single unit (called class) is known as encapsulation. The data is not accessible to the outside world and only those functions which are wrapped in the class can access it. These functions provide the interface between the objects data and the program.

## INHERITENCE :

Inheritance is the process by which objects of one class acquire the properties of another class. In the concept of inheritance provides the idea of reusablity. This mean that we can add additional features to an existing class with out modifying it. This is possible by desining a new class will have the combined features of both the classes.

## POLYMORPHISIM:

Polymorphism means the ability to take more than one form. An operation may exhibit different instance. The behaviour depends upon the type of data used in the operation.

A language feature that allows a function or operator to be given more than one definition. The types of the arguments with which the function or operator is called determines which definition will be used.

Overloading may be operator overloading or function overloading.

It is able to express the operation of addition by a single operater say '+'. When this is possible you use the expression x + y to denote the sum of x and y, for many different types of x and y; integers , float and complex no. You can even define the + operation for two strings to mean the concatenation of the strings.

## DYNAMIC BINDING :

Binding refers to the linking of a procedure call to the code to the executed in response to the call. Dynamic binding means the code associated with a given procedure call is not known untill the time of the call at run-time. It is associated with a polymorphic reference depends upon the dynamic type of that reference.



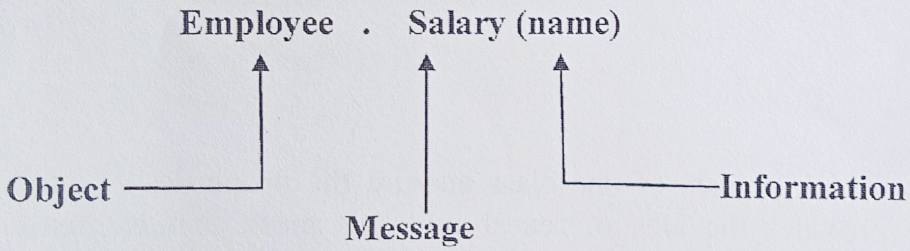


P.T.O

## MESSAGE PASSING :

An object oriented program consists of a set of objects that communicate with each other.

A message for an object is a request for execution of a procedure and therefore will invoke a function (procedure) in the receiving object that generates the desired result. Message passing involves specifying the name of the object, the name of the function (message) and information to be sent.



## LECTURE- 4

## BENEFITS OF OOP:

Oop offers several benefits to both the program designer and the user. Object-oriented contributes to the solution of many problems associated with the development and quality of software products. The principal advantages are :

1. Through inheritance we can eliminate redundant code and extend the use of existing classes.
2. We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
3. This principle of data hiding helps the programmer to build secure programs that can't be invaded by code in other parts of the program.
4. It is possible to have multiple instances of an object to co-exist with out any interference.
5. It is easy to partition the work in a project based on objects.
6. Object-oriented systems can be easily upgraded from small to large systems.
7. Message passing techniques for communication between objects makes the interface description with external systems much simpler.
8. Software complexity can be easily managed.

## APPLICATION OF OOP:

The most popular application of oops up to now, has been in the area of user interface design such as windows. There are hundreds of windowing systems developed using oop techniques.

Real business systems are often much more complex and contain many more objects with complicated attributes and methods. Oop is useful in this type of applications because it can simplify a complex problem. The promising areas for application of oop includes.

1. Real – Time systems.
2. Simulation and modeling
3. Object oriented databases.
4. Hypertext,hypermedia and expertext.
5. Al and expert systems.
6. Neural networks and parallel programming.
7. Dicision support and office automation systems.
8. CIM / CAM / CAD system.

# LECTURE-5

## Basics of C++

C ++ is an object oriented programming language, C ++ was developed by Jarney Stroustrup at AT & T Bell lab, USA in early eighties. C ++ was developed from c and simula 67 language. C ++ was early called 'C with classes'.

## C++ Comments:

C++ introduces a new comment symbol //(double slash). Comments start with a double slash symbol and terminate at the end of line. A comment may start any where in the line and what ever follows till the end of line is ignored. Note that there is no closing symbol.

The double slash comment is basically a single line comment. Multi line comments can be written as follows:

```
// this is an example of
// c++ program
// thank you
```

The c comment symbols /* ....*/ are still valid and more suitable for multi line comments.

```
/* this is an example of c++ program */
```

## Output Operator:

The statement cout <<"Hello, world" displayed the string with in quotes on the screen. The identifier cout can be used to display individual characters, strings and even numbers. It is a predefined object that corresponds to the standard output stream. Stream just refers to a flow of data and the standard Output stream normally flows to the screen display. The cout object, whose properties are defined in iostream.h represents that stream. The insertion operator << also called the 'put to' operator directs the information on its right to the object on its left.

## Return Statement:

In C++ main ( ) returns an integer type value to the operating system. Therefore every main ( ) in C++ should end with a return (0) statement, otherwise a warning or an error might occur.

## Input Operator:

The statement

```
cin>> number 1;
```

is an input statement and causes. The program to wait for the user to type in a number. The number keyed in is placed in the variable number1. The identifier **cin** is a predefined object in C++ that corresponds to the standard input stream. Here this stream represents the key board.

The operator >> is known as get from operator. It extracts value from the keyboard and assigns it to the variable on its right.

## Cascading Of I/O Operator:

```
cout<<"sum="<<sum<<"\n";
cout<<"sum="<<sum<<"\n"<<"average="<<average<<"\n";
cin>>number1>>number2;
```

## Structure Of A Program :

Probably the best way to start learning a programming language is by writing a program. Therefore, here is our first program:

```
// my first program in C++

#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World!";
  return 0;
}
```

Output:-Hello World!

The first panel shows the source code for our first program. The second one shows the result of the program once compiled and executed. The way to edit and compile a program depends on the compiler you are using. Depending on whether it has a Development Interface or not and on its version. Consult the compilers section and the manual or help included with your compiler if you have doubts on how to compile a C++ console program.

The previous program is the typical program that programmer apprentices write for the first time, and its result is the printing on screen of the "Hello World!" sentence. It is one of the simplest programs that can be written in C++, but it already contains the fundamental components that every C++ program has. We are going to look line by line at the code we have just written:

**// my first program in C++**

This is a comment line. All lines beginning with two slash signs (//) are considered comments and do not have any effect on the behavior of the program. The programmer can use them to include short explanations or observations within the source code itself. In this case, the line is a brief description of what our program is.

**#include <iostream>**

Lines beginning with a hash sign (#) are directives for the preprocessor. They are not regular code lines with expressions but indications for the compiler's preprocessor. In this case the directive #include<iostream> tells the preprocessor to include the iostream standard file. This specific file (iostream) includes the declarations of the basic standard input-output library in C++, and it is included because its functionality is going to be used later in the program.

**using namespace std;**

All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name *std*. So in order to access its functionality we declare with this expression that we will be using these entities. This line is very frequent in C++ programs that use the standard library, and in fact it will be included in most of the source codes included in these tutorials.

**int main ()**

This line corresponds to the beginning of the definition of the main function. The main function is the point by where all C++ programs start their execution, independently of its location within the source code. It does not matter whether there are other functions with other names defined before or after it – the instructions contained within this function's definition will always be the first ones to be

executed in any C++ program. For that same reason, it is essential that all C++ programs have a main function.

The word main is followed in the code by a pair of parentheses (()). That is because it is a function declaration: In C++, what differentiates a function declaration from other types of expressions are these parentheses that follow its name. Optionally, these parentheses may enclose a list of parameters within them.

Right after these parentheses we can find the body of the main function enclosed in braces ({}). What is contained within these braces is what the function does when it is executed.

**cout << "Hello World!";**

This line is a C++ statement. A statement is a simple or compound expression that can actually produce some effect. In fact, this statement performs the only action that generates a visible effect in our first program.

cout represents the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters (in this case the Hello World sequence of characters) into the standard output stream (which usually is the screen).

cout is declared in the iostream standard file within the std namespace, so that's why we needed to include that specific file and to declare that we were going to use this specific namespace earlier in our code.

Notice that the statement ends with a semicolon character (;). This character is used to mark the end of the statement and in fact it must be included at the end of all expression statements in all C++ programs (one of the most common syntax errors is indeed to forget to include some semicolon after a statement).

**return 0;**

The return statement causes the main function to finish. return may be followed by a return code (in our example is followed by the return code 0). A return code of 0 for the main function is generally interpreted as the program worked as expected without any errors during its execution. This is the most usual way to end a C++ console program.

You may have noticed that not all the lines of this program perform actions when the code is executed. There were lines containing only comments (those beginning by //). There were lines with directives for the compiler's preprocessor (those beginning by #). Then there were lines that began the declaration of a function (in this case, the main function) and, finally lines with statements (like the insertion into cout), which were all included within the block delimited by the braces ({}) of the main function.

The program has been structured in different lines in order to be more readable, but in C++, we do not have strict rules on how to separate instructions in different lines. For example, instead of

```
int main ()
{
 cout << " Hello World!";
 return 0;
}
```

We could have written:

```
int main ()
{
cout << "Hello World!";
return 0;
}
```

All in just one line and this would have had exactly the same meaning as the previous code.

In C++, the separation between statements is specified with an ending semicolon (;) at the end of each one, so the separation in different code lines does not matter at all for this purpose. We can write many statements per line or write a single statement that takes many code lines. The division of

code in different lines serves only to make it more legible and schematic for the humans that may read it.

Let us add an additional instruction to our first program:

```
// my second program in C++
#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World! ";
  cout << "I'm a C++ program";
  return 0;
}
```

Output:-Hello World! I'm a C++ program

In this case, we performed two insertions into cout in two different statements. Once again, the separation in different lines of code has been done just to give greater readability to the program, since main could have been perfectly valid defined this way:

```
int main ()
{
 cout << " Hello World! ";
cout << " I'm a C++ program ";
 return 0;
}
```

We were also free to divide the code into more lines if we considered it more convenient:

```
int main ()
{
  cout <<  "Hello World!";
  cout << "I'm a C++ program";
  return 0;
}
```

And the result would again have been exactly the same as in the previous examples.

Preprocessor directives (those that begin by #) are out of this general rule since they are not statements. They are lines read and processed by the preprocessor and do not produce any code by themselves. Preprocessor directives must be specified in their own line and do not have to end with a semicolon (;).

## STRUCTURE OF C++ PROGRAM

- Include files
- Class declaration
- Class functions, definition
- Main function program

Example :-

```
# include<iostream.h>

class person
```

```
{
char name[30];
int age;
public:
        void getdata(void);
        void display(void);
};


void  person :: getdata ( void )
{
        cout<<"enter name";
        cin>>name;
        cout<<"enter age";
        cin>>age;
}

void display()
{
        cout<<"\n name:"<<name;
        cout<<"\n age:"<<age;
}

int main( )
{

person p;
p.getdata();
p.display();
return(0);

}
```