OOP(C++)

(CHAPTER 1)

" CONCEPTS OF OOPS"

OBJECT ORIENTED PROGRAMMING

- It is an approach to program organization & development that attempts to eliminate some of the pitfalls of proc. Oriented programming methods by incorporating the best of the structured programming features with several powerful new concepts.
- It is a new way of organizing & developing programs and has nothing to do with any particular language.
- It improves s/w productivity & decreases cost.

What is OOP?

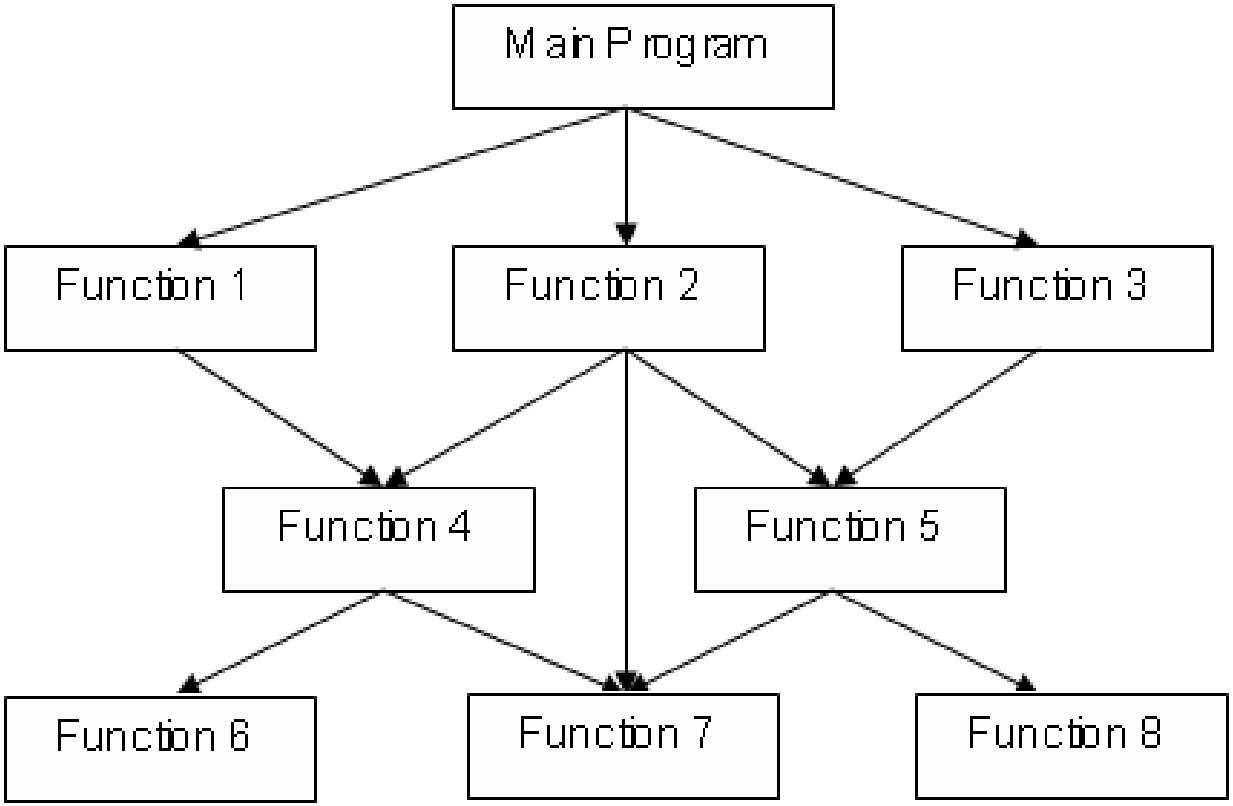
- What is Object Oriented Programming?
  - Object-oriented programming (OOP) is a programming language model in which programs are organized around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

PROCEDURE ORIENTED PROGRAMMING

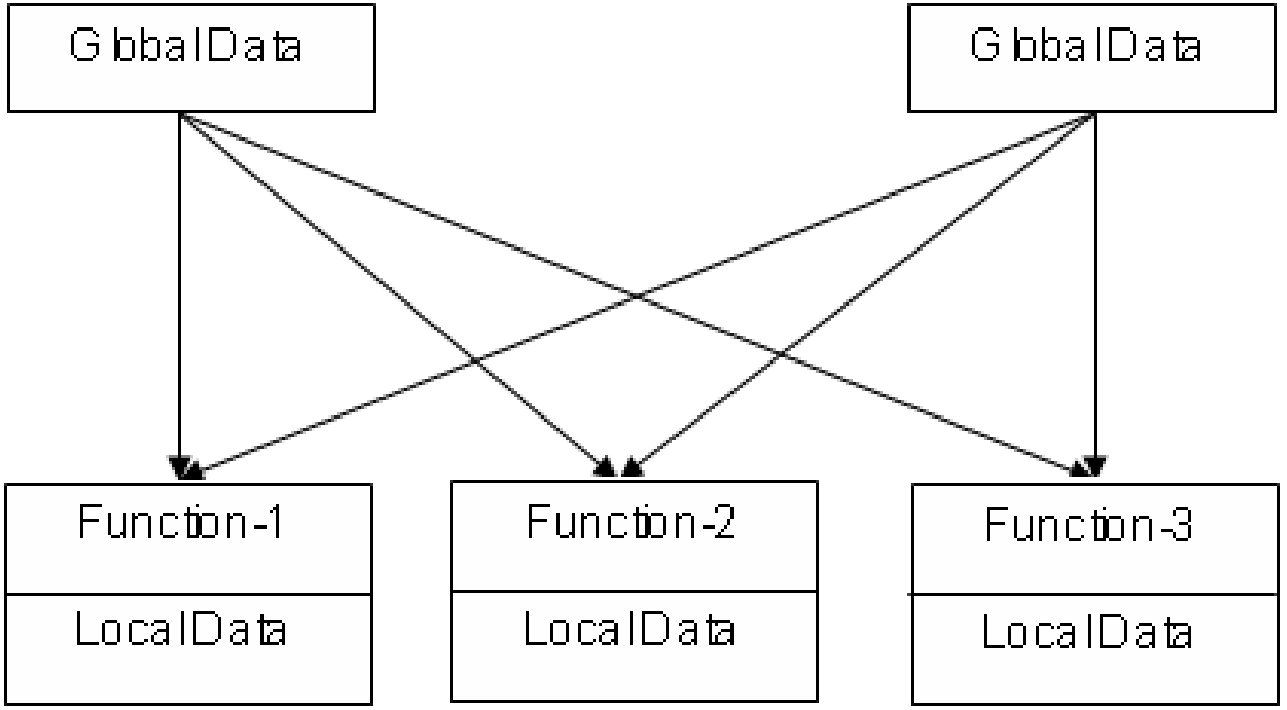
Characteristics of POP:

1. Also called as HLL such as Pascal, FORTRAN & C etc.
2. Large program are divided into small programs called as functions.
3. Emphasis is on doing things (Algorithms).
4. Most of functions share global data.
5. Data moves openly around the system from function to function.
6. Program design in Top-Down approach.
7. Does not model the real world problems very well.

STRCTURE OF PROC. ORIENTED PROGRAM



Relation between Data & Functions in Proc. Programming

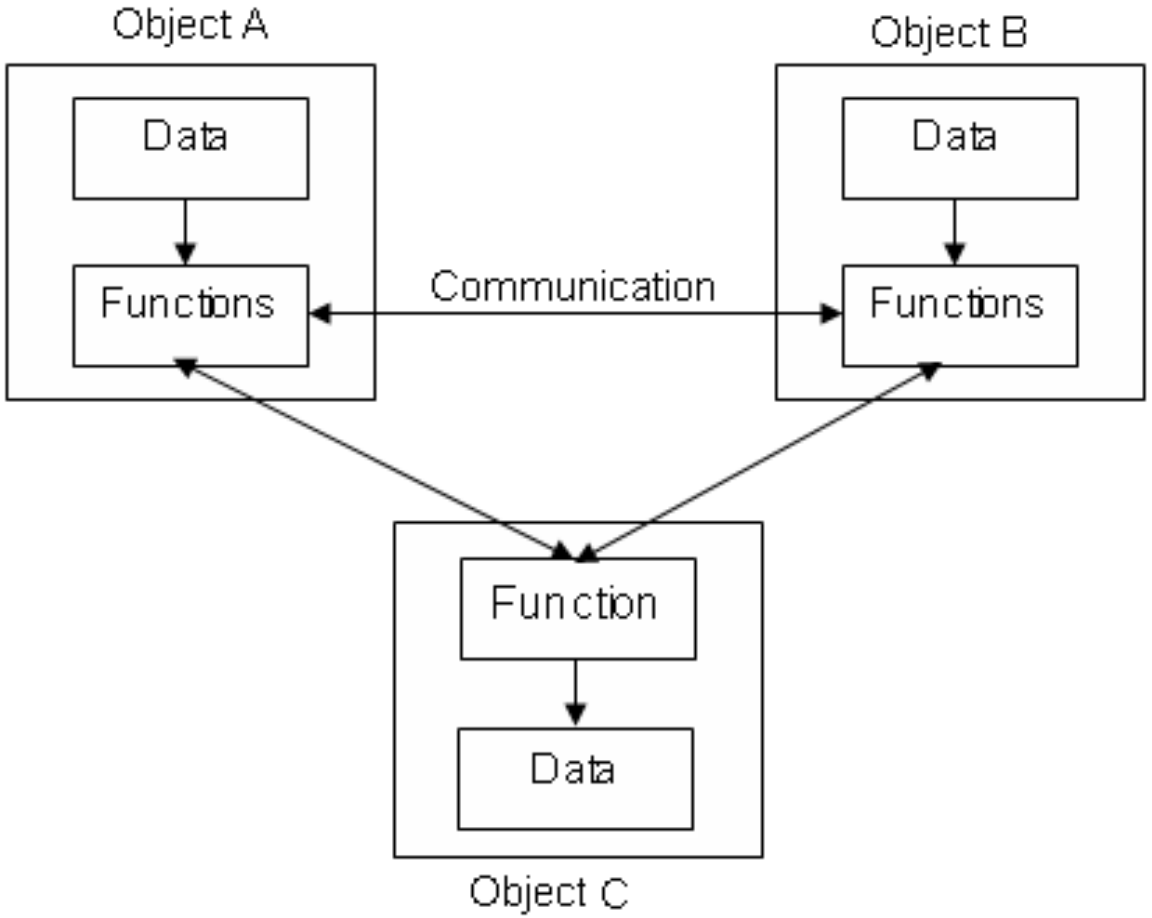


OBJECT ORIENTED PROGRAMMING

Characteristics of OOP:

1. Emphasis is on data rather than procedure.
2. Program are divided into what known as objects.
3. Functions that operate on data of an object are tied together in the data structure.
4. Data is hidden & cannot be accessed by external functions.
5. Object may communicate with each other through functions.
6. New data & functions can be easily added whenever necessary.
7. Program design in Bottom-Up approach.

ORGANIZATION OF DATA & FN IN OOP



DIFFERENCE

PROCEDURE ORIENTED PROG.	OBJECT ORIENTED PROG.
Emphasis is on doing things (Algorithms).	Emphasis is on data rather than procedure
Large program are divided into small programs called as functions	Program are divided into what known as objects.
Top-Down approach	Bottom-Up approach
New data & functions can be easily added whenever necessary with modifications in functions	New data & functions can be easily added whenever necessary
Most of functions shares global data. Data is not hidden	Data is hidden & cannot be accessed by external functions
It does not model real world problems very well	Used to represent real life entities in system design

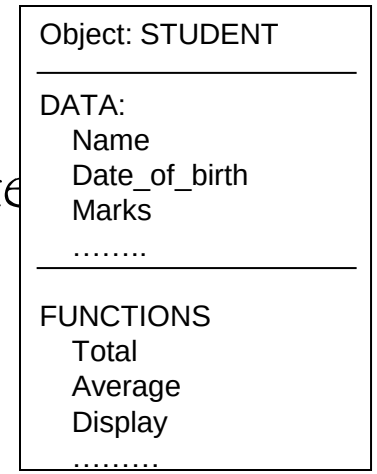
BASIC CONCEPTS OF OOP

- It includes,

1. Objects
2. Classes
3. Data Encapsulation
4. Data Abstraction
5. Inheritance
6. Polymorphism
7. Dynamic Binding

OBJECTS

- Basic runtime entities in object oriented system.
- May represent person, place, bank account or any item that program has to handle.
- Programming problem is analyzed in terms of objects & nature of communication between them.
- It take up space in memory & have an associated address like structure in C.
- Each object contain data & fucntion to manipulate data.
- Objects can interact without having to know the details of each other's data or code.



CLASSES

- It is user defined datatype which contains the entire set of data & code of an object.
- Object is variable/blueprint/instance of the type class.
- Once class is defined we can create any no. of objects of that class.
- It is collection of objects of similar type.
- For ex. Mango, apple & orange are members of class fruit.
- Syntax : Classname Object1, Object2 ….

  Example: Fruit Mango;

DATA ENCAPSULATION

- Wrapping of data & functions in single unit (class).
- Data is not accessible to outside world & only those function which are wrapped in class can access it.
- This insulation of the data from direct access by the program called as Data hiding or Information hiding.

Data Encapsulation

```
class Box {
   public:
      double getVolume(void) {
         return length * breadth * height;
      }

   private:
      double length;      // Length of a box
      double breadth;     // Breadth of a box
      double height;      // Height of a box
};
```

The variables length, breadth, and height are **private**. This means that they can be accessed only by other members of the Box class, and not by any other part of your program. This is one way encapsulation is achieved.

DATA ABSTRACTION

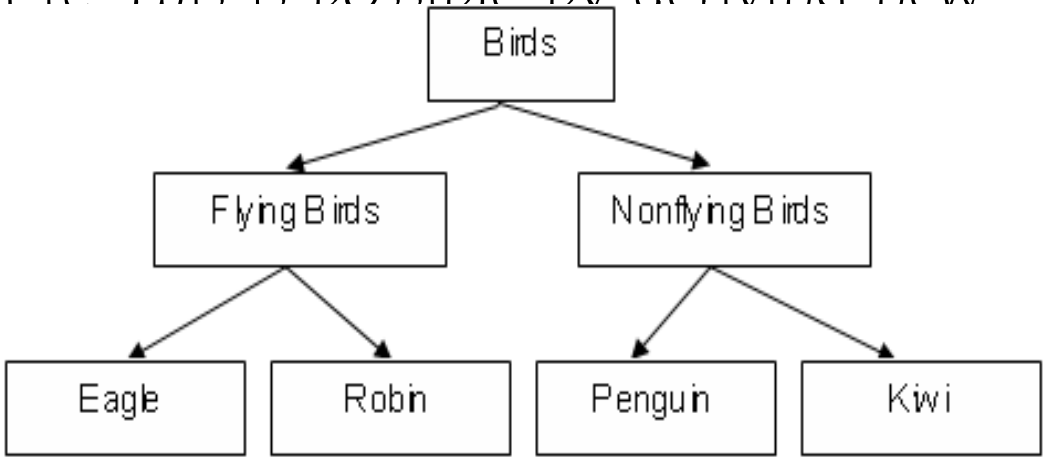
- Representing the essential feature without including the background details or explanations.
- Classes uses the concepts of abstraction and are defined as list of abstract attribute (data members) and functions (Methods or Member functions) to operate on this attribute.
- For example, your program can make a call to the sort() function without knowing what algorithm the function actually uses to sort the given values.
- Similarly cout, cin

Benefits of Data Abstraction

- Data abstraction provides two important advantages –
- Class internals are protected from inadvertent user-level errors, which might corrupt the state of the object.
- The class implementation may evolve over time in response to changing requirements or bug reports without requiring change in user-level code.

INHERITANCE

- Process by which object of one class acquire the properties of objects of another class.
- Support the concept of hierarchical classification.
- Provide the idea of reusability.
- It means that we can add additional features to an existing class without modifying it. This is possible by deriving new class from existing one.
- For Example,



INHERITANCE

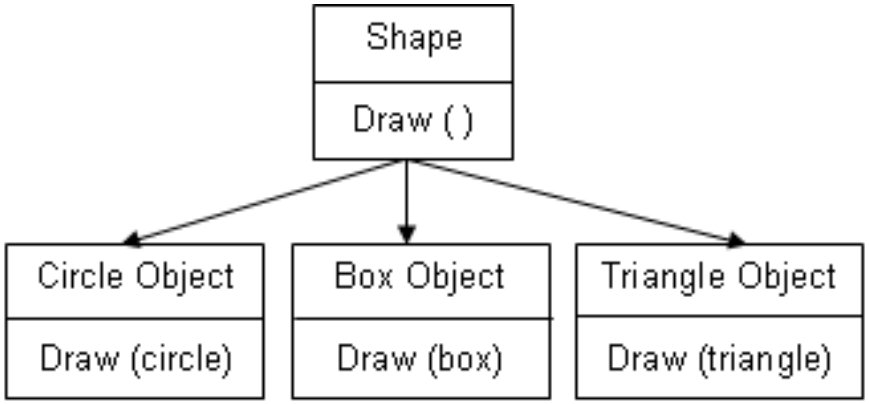
```
class subclass_name : access_mode base_class_name
{
  //body of subclass
};
```

```
class Parent
{
    public:
      int id_p;
};

// Sub class inheriting from Base Class(Parent)
class Child : public Parent
{
    public:
      int id_c;
};

//main function
int main()
   {

        Child obj1;

        // An object of class child has all data members
        // and member functions of class parent
        obj1.id_c = 7;
        obj1.id_p = 91;
        cout << "Child id is " <<  obj1.id_c << endl;
        cout << "Parent id is " <<  obj1.id_p << endl;

        return 0;
```

POLYMORPHISM

- Ability to take more than one form.
- Operation may exhibit different behavior in different instance.
- For Example, + operator.
- **Operator Overloading:** It is process of making the Operator to exhibit different behaviors in different instance.
- **Function Overloading:**

  Single function name to perform the different types of tasks.



Function overloading

- Write a function with same name and change the arguments type/count and implement function overloading

```
using namespace std;
class Geeks
{
    public:

    // function with 1 int parameter
    void func(int x)
    {
        cout << "value of x is " << x << endl;
    }

    // function with same name but 1 double parameter
    void func(double x)
    {
        cout << "value of x is " << x << endl;
    }

    // function with same name and 2 int parameters
    void func(int x, int y)
    {
        cout << "value of x and y is " << x << ", " << y << endl
    }
};

int main() {

    Geeks obj1;
```

Function overloading

```
    // Which function is called will depend on the parameters pa
    // The first 'func' is called
    obj1.func(7);

    // The second 'func' is called
    obj1.func(9.132);

    // The third 'func' is called
    obj1.func(85,64);
    return 0;
}
```

Dynamic Binding

- Also called as Late Binding.
- Binding: It refers to the linking of procedure call to the code to be executed in response to the call.
- It means that the code associated with a given procedure call is not known until the time of the call a run time.
- It is associated with polymorphism & Inheritance.

BENEFITS OF OOP

- Eliminate redundant code by using Inheritance.
- Build the programs from std working modules that communicate with one another. It leads to saving development time & higher productivity.
- Data hiding helps to build secure programs.
- It is possible to have multiple instances of an object to co-exist without any interference.
- Easy to partition the works based in project on objects.
- Can be easily upgraded from small to large systems. - Scalable
- Software complexity can be easily managed.

OBJECT ORIENTED LANGUAGE

- Language that is specially designed to support OOP concepts make it easier to implement them.
- The language are divided into two types depending on the feature they support
  1. Object Based Programming Language
  2. Object Oriented Programming Language

OBJECT ORIENTED LANGUAGE

- **Object Based Programming Language :**
- Features that are required for OBPL,
  Encapsulation, Data Identity, Data Hiding, Operator Overloading
- Language that support prog. With Object is said to be OBPL.
- Do not Support Inheritance & Dynamic Binding.
- Example : Ada

- **Object Oriented Programming Language:**
  Object Based Feature + Inheritance + Dynamic Binding
- Example: C++, Java.

APPLICATION OF OOP

- Real Time Systems
- Object Oriented Database
- AI & Expert System
- Neural Network & Parallel Programming
- CAD/CAM Systems
- Design Support & Office Automation Systems

What is C++?

- Object Oriented Prog. Language.
- Develop by Bjarne Stroustrup at AT & T Bell Lab. In NJ, USA in Year 1980.
- Extension of C with addition of the class construct feature.
- Initially called as 'C with Classes'.
- C++ adds on to C are classes, inheritance, polymorphism, function overloading & operator overloading.
- C++ program are easily maintainable & expandable.
- It is expected to C++ will replace C as a general programming language in the near future.

SIMPLE C++ PROGRAM

- Program to print the string on screen.

```
#include<iostream.h>   //Include the Header File
#include<conio.h>

void main()
{
        clrscr();
        cout<< "C++ is better than C. \n"; // C++ Statement
        getch();
}
```

MORE C++ STATEMENTS

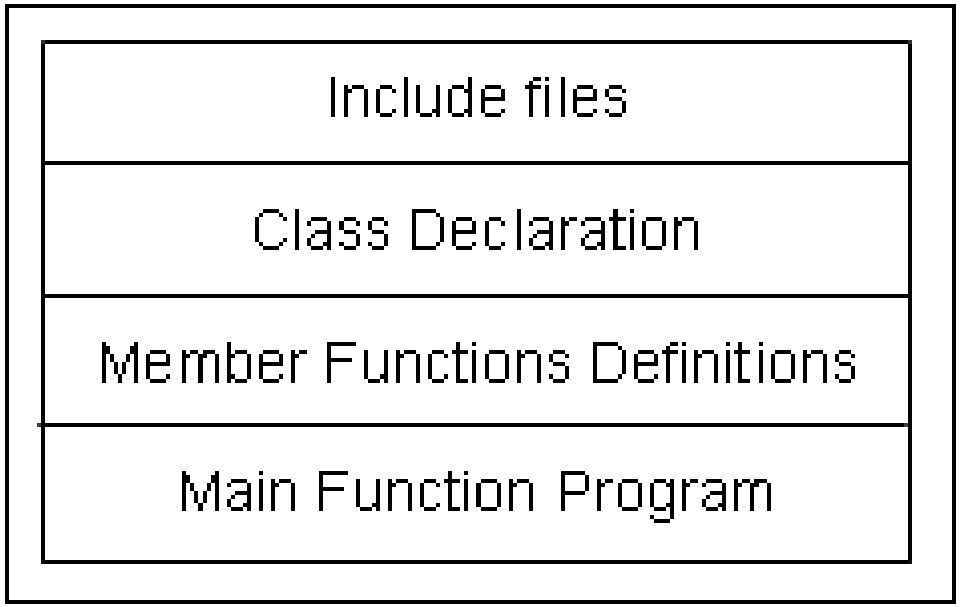
- Program to find the average of 2 no.

```
#include<iostream.h>  //Include the Header File
#include<conio.h>

void main()
{
        clrscr();
        float a,b,sum;
        cout<<"Enter two numbers: ";
        cin>>a
          >>b;
        sum=a+b;
        float avg;
        avg=sum/2;

        cout<<"Sum= "<<sum
           <<"\n";
        cout<<"Average= "<<avg<<"\n";
        getch();

}
```

PROGRAM FEATURES

- Start from main() function like C
- Comments (single Line & Multiline Comment)
- Output Operator (<< called as Insertion operator or Put to operator)(Like printf in C)
- Input Operator (>> called as extraction operator or Get from operator) (Like scanf in C)
- Variable declaration
- Header files
- Return type (Like Void)
- Cascading of I/O operators

Structure of C++ Program



EXAMPLE WITH CLASS

```
#include ‹iostream›
using namespace std;
class student
{
public:
  char name[20];
  int rollno;

    void accept()
    {
      cout ‹‹ "Enter the name:\t";
      cin ›› name;
      cout ‹‹ "Enter the roll no:\t";
      cin ›› rollno;

    }
```

```
void display()
  {
    cout ‹‹ "\n Student Name:\t" ‹‹
name;
    cout ‹‹ "\n Roll No:\t" ‹‹ rollno;
  }
};


int main()
{
  student a;
  a.accept();
  a.display();
}
```

Compiling and Linking

- Execution process of a C/C++ program
- Execution of a C/C++ program involves four stages using different compiling/execution tool, these tools are set of programs which help to complete the C/C++ program's execution process.
  1. Preprocessor
  2. Compiler
  3. Linker
  4. Loader
- These tools make the program running.

Compiling and Linking

- 1) Preprocessor
  - This is the first stage of any C/C++ program execution process; in this stage Preprocessor processes the program before compilation. Preprocessor include header files, expand the Macros.

- 2) Complier
  - This is the second stage of any C/C++ program execution process, in this stage generated output file after preprocessing (with source code) will be passed to the compiler for compilation. Complier will compile the program, checks the errors and generates the object file (this object file contains assembly code).

Compiling and Linking

- 3) Linker
  - This is the third stage of any C/C++ program execution process, in this stage Linker links the more than one object files or libraries and generates the executable file.

- 4) Loader
  - This is the fourth or final stage of any C/C++ program execution process, in this stage Loader loads the executable file into the main/primary memory. And program run.

Different files during the process of execution

- Source code - .ccp
- Preprocessor - .ii
- Compiler – Converts into the code to assembly code and generates .s and then finally generates object code .o
- Linker - .exe
- Loader – Executes the .exe

Iostream

- C++ program contains pre processor directive statements at the beginning.
- Such statements are preceded with # to indicate the presence of preprocessor statements to the compiler

- # include <iostream>

- To include header files. It contains the declarations of cin and cout statements

Namespace

- Standard class libraries are defined in **std** namespace.
- A **namespace** is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it.
- **Namespaces** are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries.