Introduction:

Program : A program is a set of instructions that a computer follows in order to perform aparticular task.
Programmers write instructions in various programming languages to perform their computationtasks.
Various types of Languages are :
(i) Machine level Language
(ii) Assembly level Language
(iii) High level Language

Machine level Language :
Machine Language is a collection of binary digits (01001101) or bits . Machine Language is the only language a computer understands.
Most machine languages consist of binary codes for both data and instructions. E.g., to add two numbers we would need a series of binary codes such as:
0010 0000 0000 0100
0100 0000 0000 0101
0011 0000 0000 0110

Assembly level Language :
An assembly language is a low-level programming language designed for a specific type of processor.
It uses mnemonic codes(MOV,ADD,JMP) that is English-like abbreviations to represent the machine-language instructions.
Assembly language has to be converted into executable machine code , a translator program called an assembler is used to convert each instruction from the assembly language to the machine language.
for E.g.: To add two numbers A and B in assembly lang we use LOAD instruction, ADD instructionto add
LOAD R1, ALOAD R2 , B

ADD R1, R2

High level Language :
High-level language is more English-like.
Each high- level language has to be translated to low-level language .
We Use compilers to translate high-level language into machine language. Compilers translate the whole program first, then execute the object program.

E.g., To add two numbers A and B simply we can use '+' operator read two values in to A and B
Sum = A + B

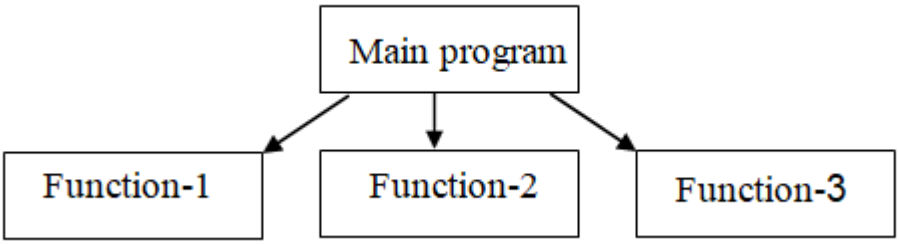
A programming language such as C, enables to write programs which is understandable to programmer(human) and can perform any sort of task. such languages are considered as High-level because they are close to human languages. In contrast, assembly languages are considered low- level because they are very close to machine languages.
The first high-level programming languages were designed in the 1950s.
Examples : Ada , Algol, BASIC, COBOL, C, C++, JAVA, Python, FORTRAN, LISP, Pascal, Prolog.

The high-level programming languages are broadly categorized in to two categories:
(i) Procedure oriented programming (POP) language.
(ii) Object oriented programming (OOP) language.

Procedure Oriented Programming Language

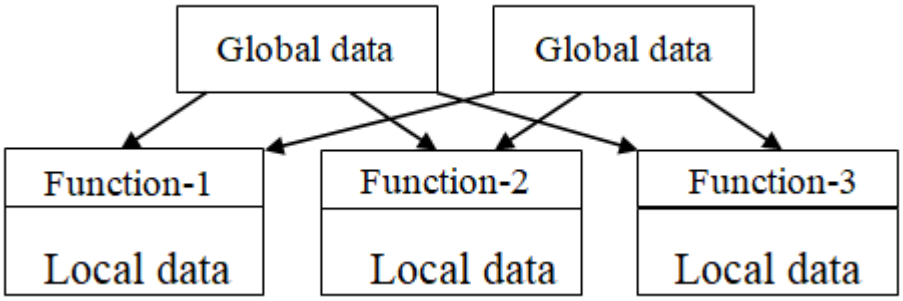
In the procedure oriented approach, the problem is viewed as sequence of things to be done such as reading , calculating and printing. A number of functions are written to accomplish this task.

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.



Some Characteristics exhibited by procedure-oriented programming are:

- Emphasis is on doing things (algorithms).
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data.
- Data move openly around the system from function to function.
- Functions transform data from one form to another.
- Employs top-down approach in program design.

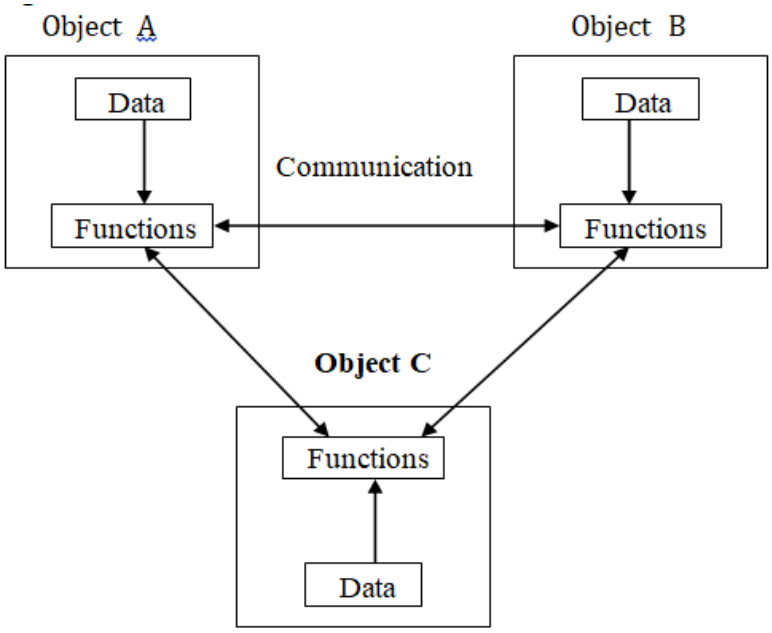


The disadvantages of the procedure oriented programming languages is:

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

Object Oriented Programming / Object-oriented technology (OOT)

- The major motivating factor in the invention of object oriented approach is to over come the drawbacksof procedural oriented approach.
- The principal of Object Oriented Programming Approach is to combine both data and functions into asingle unit. such a single unit is called object.
- object oriented programming treats data as critical element in the program development and does notallow it freely to flow around the system or application or program.
- object oriented programming combines both data and functions in to a single unit this process is called Encapsulation.
- Ex : languages like C++, JAVA, PYTHON follow this approach.
- The organisation of data and functions in Object oriented programming is shown in the following figure.



Features of the Object Oriented programming

1. In OOP importance is given to data rather than procedure.
2. Programs are divided into what are known as objects.
3. New data and functions can be easily added whenever necessary
4. Data of an object can be accessed only by functions associated with that object.
5. Data is hidden and can't be accessed by external functions.
6. Objects may communicate with each other through functions.
7. Follows bottom-up approach in program design.

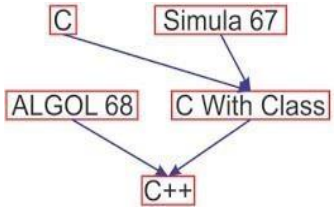
Difference Between Procedure Oriented Programming (POP) & Object Oriented Programming (OOP)

	Procedure Oriented Programming	**Object Oriented Programming**
1	program is divided into small parts called **functions**.	program is divided into parts called **objects**.
2	Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	Importance is given to the data rather than procedures or functions because it works as a **real world**.
3	follows **Top Down approach**.	OOP follows **Bottom Up approach**.
4	It does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
5	Data can move freely from function to function in the system.	objects can move and communicate with each other through member functions.
6	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data And function.
7	Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function, it can be kept public or private so we can control the access of data.
8	It does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
9	Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.

10	Example of Procedure Oriented Programming are : C, VB, FORTRAN, Pascal.	Example of Object Oriented Programming are : C++, JAVA, VB.NET, C#.NET.

Overview of C++ language:

- C++ is an object oriented programming language and is considered as an extension of C.
- C++ was developed by Bjarne Stroustrup in 1979, at AT & T Bell labs.
- Initially stroustrup combined the features of C programming language as well as Simula67(a first object oriented language) for developing a most powerful Object oriented programming language it was called "C with classes" as it had all the properties of the C language.



- In 1982, Stroustrup renamed it as "C++" (++ being the increment operator in C)
- C++ introduced the concept of Class and Objects.
- It encapsulates high and low-level language features. So, it is seen as an intermediate level language.
- C++ is used for developing applications such as editors, databases, personal file systems, networking utilities, and communication programs. Because C++ shares C's efficiency, much high-performance systems software is constructed using C++.

Benefits of object oriented programming (OOPs)

Reusability: In OOP" s programs functions and modules that are written by auser can be reused by other users without any modification.

Inheritance: Through this we can eliminate redundant code and extend the use of existing classes.

Data Hiding: The programmer can hide the data and functions in a class from other classes. It helps theprogrammer to
build the secure programs.

Reduced complexity of a problem: The given problem can be viewed as a collection of different objects. Each object is responsible for a specific task. The problem is solved by interfacing the objects.
This technique reduces the complexity of the program design.

Easy to Maintain and Upgrade: OOP makes it easy to maintain and modify existing code as newobjects
can be created with small differences to existing ones. Software complexity can be easily managed.

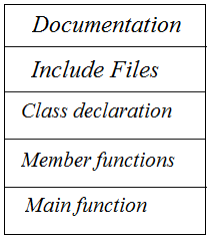
Message Passing: The technique of message communication betweenobjects makes the interface with external systems easier.

Modifiability: it is easy to make minor changes in the data representation or the procedures in an OOprogram. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods.

Uses / Application of OOPs:
Now a days OOP is much popular because we can use their concept to solve in any type of problemssome of them are much like:
1. Real time system

2. Simulation and modeling
3. Object oriented database
4. Hypertext and hypermedia
5. AI and expert system
6. Decision support and office automation
7. CAM /CAD System

Basic Structure Of C++ Language :



The structure of C++ program can have four sections.

1. **Documentation:** In this section we write comment about program that ignore by compiler. In comments we can write program name, uses, developer name, date, any help etc. Comments are written in /*--------*/(multiline comment) and //------
(single line comment).

// Documentation

/* Program to demonstrate POP organization and Top to Down approach*/

2. **Include file section:** In this section all needful files are included in the program that containsdeclaration of class and functions used in the program.

Ex: #include<iostream.h>

Contain declaration of cout and cin stream object.

3. **Class declaration section:** In this section we create classes. Every class contains data and functiondeclaration. In class we can also define member functions.

Ex: class A

```
{
Int a,b;
public:
void getdata(int);
void showdata();
int sum();
};
```

4. **Member function definition:** In this section, all explicit member functions of the class are definedoutside the class using scope resolution operator (::).

Ex:

```
void A::getdata(int p1,int p2)
{
a=p1;
b=p2;
}
void A::showdata()
 {
  cout<<"\n"<<a<<"\t"<<b;
}
int A::sum()
{
return a+b;
}
```

5. **Main function section**: In this section we can declare object of class and call public member functions of the class.

Ex:

```
void main()
{
{
A ob;
ob.getdata(10);
 ob.showdata();
int c=ob.sum();
cout<<"\nSum="<<c;
}
```

In this way a OOPs C++ program is designed in four sections.

OOPs(Object Oriented Programming) C++ Program:

On the basis of above structure, an OOP's C++ will design in following order.

```
// Documentation
/* Program to demonstrate OOPs organization and Bottom-Up approach
*/
//include files
#include<iostream.h>
//class declaration
class A
{
int a,b;
public:
void getdata(int);
void showdata();
int sum();
};

//Member function definition
void A::getdata(int p1,int p2)
{
a=p1;
b=p2;
}
void A::showdata()
{
cout<<"\n"<<a<<"\t"<<b;
}
int A::sum()
{
return a+b;
}

//Main function
void main()
{
A ob;
ob.getdata(10);
ob.showdata();
 int c=ob.sum();
cout<<"\nSum="<<c;
}
```

Output:

a=10 b=20

sum=30

Scope Resolution operator (::)

If a function that uses same name of global variable and local variable then by local variable will be accessed. But if we want to use global variable then use scope resolution operator (::) with variable name.

```
#include<iostream.h>
int n=10;              //global variablevoid main()
{
int n=20;                      // Local Variable
cout<<"\n Access local variable:"<< n;
cout<<"\n Access global variable:"<< ::n;
}
```

Output:

Access local variable: 10 Access global variable: 20

In this program, name of local and global variable having the same name (n) therefore inside any functionn will be consider as local variable and ::n will be consider as global variable.

Input (cin) & Output(cout) statement in C++:

In c++, iostream.h header file is used to support all input output streams classes in our program.

Output statement: using "*cout<<"* stream object we can display data on monitor. (just like printf in c). 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.

Input statement: using "*cin >>*" stream object we can assign keyboard value to the variable of the program.(just like scanf in c). The operator >> is known as get from operator. It extracts value from the keyboard and assigns it to the variable on its right.

Example:

```
#include<iostream.h>
void main()
{
int a;
cout<<"\n Input Integer:";
cin>>a;
cout<<"\nInteger value is "<<a;
}
```

Output:

Input Integer: 11 Integer value is 11

If number of data are more than each data must be separated by << or >>.cout << "a =" << a << "\n"; will display on monitor

as-

a=11

cin>>a>>b;

This statement reads two values from keyboard and store to first in variable a then tovariable b.

Key Concepts of Object Oriented Programming

Key concepts of object oriented programming are :

1. class

2. object
3. Data Encapsulation
4. Data Abstraction
5. Inheritance
6. Polymorphism

1. class : It is a user defined data type, which consists of both data and functions into a single unit.

In OOP data must be represented in the form of a class . A class contains variables for storing data and functions to specify various operations that can be performed and hence it is only logical representation of data and not physical representation.

so class is collection of Data members and Member functions. class is a blue print of an object.
Once a class has been defined we can create any number of objects.

Each object is associated with the data of type class which they were created. class provides the concept of Encapsulation.
class provides the concept of Data hiding with private declarations.

2. Object:

A class is only logical representation of data. Hence to work with the data represented by the class you must create a variable for the class which is called as an object.

So object is a variable of type class.

Object is the basic unit of object-oriented programming.

In a class to access data members first memory have to be allocated. No memory is allocated when a class is created.
Memory is allocated only when an object is created, i.e., when an instance of a class is created.
Object is physical copy and class is logical copy

3. Data Encapsulation : A class can contain variables for storing data and functions to specify various operations that can be performed on data. This process of wrapping up of data and functions that operate on data as a single unit is called as Encapsulation. When using Data Encapsulation, data is not accessed directly, it is only accessible through the functions present inside the class. Data Encapsulation enables the important concept of data hiding possible.

4. Data Abstraction : Abstraction refers to the act of representing essential features without including the back ground details or explanations. It represents a functionality of a program without knowing implementation details. It is an approach that speaks about hiding of the complexity and consume only the functionality.

5. Inheritance: Creating a new class from an existing class or base class is called Inheritance. The base class is also known as *parent class* or *super class*, The new class that is formed is called *derived class*. Derived class is also known as a *child class* or *sub class*. Inheritance helps in reducing the overall code size of the program, which is an important concept in object-oriented programming.
The concept of inheritance provides the idea of reusablity. Instead of rewritting the code you can create the class from the existing class and extending the functionality of existing class. This mean that we can add additional features to an existing class with out modifying it. This is possible by designing a new class that will have the combined features of both the classes.

6. Polymorphism : Polymorphism comes from the Greek words “poly” and “morphism”. “poly” means many and “morphism” means form i.e.. many forms.

Polymorphism means the ability to take more than one form. Advantage of this is you can make an object behave differently in different situations, so that no need to create different objects for different situations.

Class and ObjectClass:

- In OOP data must be represented in the form of a class . It is a collection of data and member functions that manipulate data. The data components of class are called data members and functions that manipulate the data are called member functions.
- Class is a group of objects that share common properties and relationships .In C++, a class is a new data type that contains member variables and member functions that operates on the variables.
- A class contains variables(data members) for storing data and functions(member functions) to specify various operations that can be performed and hence it is only logical representation of data and not physical representation.
- A class is a user defined data type, which consists of both data members and member functions into a single unit.
- class is a blue print of an object.
- Once a class has been defined we can create any number of objects.
- Each object is associated with the data of type class which they were created.

Syntax:-

```
class class-name
{
private:
variable declarations;
function declaration ;
public:
variable declarations;
function declaration;
};
```

The keyword 'class' is used to declare a class. Inside the class access specifiers are used to declare variables and methods. The declaration of a class is enclosed with curly braces and terminated with a semicolon.

The class body contains declarations of variables and functions and these are called members of class. The keywords private and public are known as labels that are followed by colon(:).

The class members that have been declared as private can be accessed only with in the class. The classmembers that have been declared as public can be accessed outside the class also

example:

```
class student
{
private:
int sid;
char sname[20];
float avg;
void readvalues();
public :
void printvalues();
};
```

Object

- An object is an identifiable entity with specific characteristics and behavior. An object is said to be aninstance of a class.
- A class is only logical representation of data. Hence to work with the data represented by the class you mustcreate a variable for the class which is called as an object.
- So object is a variable of type class.
- Object is the basic unit of object-oriented programming.
- In a class to access data members first memory have to be allocated. No memory is allocated when a class is created.
- Memory is allocated only when an object is created, i.e., when an instance of a class is created. Object is
- physical copy and class is logical copy.

Syntax for creating an object :

className objectName;

Ex: student s;

OR

cls_name obj_name1, obj_name2;

Ex: student s1,s2;

OR

class class_name

{

}x ,y ,z;

would create the objects x ,y ,z of class.

Ex: calss student
{
// class body
}s1,s2,s3;

class name followed by variable name. Objects are identified by its unique name.
An object represents a particular instance of a class. There can be more than one instance of an object. Characteristics of an object are represented in a class as **Properties**.
The actions that can be performed by objects becomes functions of the class and is referred to as **Methods**.
For example consider we have a Class of *Cars* under which *Santro Xing*, *Alto* and *WaganR* represents individual Objects. In this context each *Car* Object will have its own, Model, Year of Manufacture, Color, Top Speed, Engine Power etc., which form **Properties** of the *Car* class and the associated actions i.e., object functions like Start, Move, Stop form the **Methods** of *Car* Class.

Accessing Class Members

The main() cannot contain statements that access class members directly. Class members can be accessed only by an object of that class. To access class members, use the dot (.) operator. The dot operator links the name of an object with the name of a member.

syntax:

[object name] [operator] [Member name]

Example:- x.getdata(100,75.5);

It assigns value 100 to number, and 75.5 to cost of the object x by implementing the getdata() function .

Example:

```
Class xyz
{
int x;
int y;
public :
int z;
};
void main()
{
xyz p;
p. x =0;          //error, x is private
p. z=10;          //ok , z is public
}
```

Defining Member Function:

Member can be defined in two places

- Outside the class definition
- Inside the class function

Declaration of member function inside the class :

The member functions defined inside the class are treated as inline function. If the member function is small then it should be defined inside the class .

Otherwise it should be defined outside the class.

Member function inside the class can be declared in public or private section.

The following program illustrates the use of a member function inside the class in public section.

Write a program to access private members of a class using member function.

```
#include<iostream>
using namespace std;
class employee
{
private: //private section starts
int eno;
float esal;
public: //public section starts
void display()
{
eno=1023; // Access to private members
esal=15000;
cout<<"\n Employee Number : "<<eno;
cout<<"\n Employee salary : "<<esal;
}
};
int main()
{
Employee e1;
e1.display();
return0;
}
```

Explanation : In the above program display() is the member function defined inside the class in public section. We know that object has permission to access the public members of the class. Under main() function object e1 is declared. The public member function can access the private members of the same class. The function display() initializes the private member variables and display the contents on the console.

Member function outside the class

If the function is defined outside the class following care must be taken.

a) The prototype declaration(function declaration) must be done inside the class.

b) The function name must be preceded by class name and its return type separated by scope access operator.

The following is syntax:

```
return type class_name :: function(args...)
{
        ---
        ---


}
```

Example: Write a program to define member function of a class outside the class.

```
#include<iostream>
using namespace std;
class employee
{
private:
int eno;
char ename[20];
```

```
float esal;
public:
void display();            // prototype declaration
};
void employee :: display()         // definition outside the class
{
cout<<"Enter Emp No : ";
cin>>eno;
cout<<"Enter Emp Name : ";
cin>>ename;
cout<<"Enter Emp sal : ";
cin>>esal;
cout<<"\n Employee Number : "<<eno;
cout<<"\n Employee Name : "<<ename;
cout<<"\n Employee salary : "<<esal;
}

int main() {
employee e1;  // object declaration
e1.display();   // call to public member function return 0;
}
```

Access specifiers :

Access specifiers are used to identify access rights for the data and member functions of the class. There arethree main types of access specifiers in C++ programming language:

- public
- private
- protected

1. Public : Public members are accessible from outside the class. The keyword public is used to allow objectsto access the member variables of a class directly.
The member variables and functions declared as public can be accessed directly by the object.

Example: Write a program to declare all members of a class as public. Access the elements using object.

```
#include<iostream>
using namespace std;
class item
{
public:              // public section begins
int codeno;
float price;
int qty;
};                 // end of class

int main() {
item one;              // object declaration
```

```
 one.codeno=123;  // member initialization

one.price=123.45;
one.qty=150;
cout<<"\n Codeno = "<<one.codeno;cout<<"\n Price = "<<one.price;
cout<<"\n Quantity = "<<one.qty;
return 0;
}
```

2. Private :

A *private* member within a class denotes that only members of the same class have accessibility. The *private*
member is inaccessible from outside the class.
To prevent member variables and functions from direct access the private keyword is used.

Example:

```
#include<iostream>
 using namespace std;
class item
{
private:                    // private section begins
int codeno;
float price;
int qty;
};                  // end of class

int main()

 {
item one;               // object declaration
one.codeno=123;          // it will show error because we can not access private members
one.price=123.45;        //it will show error because we can not access private members
one.qty=150;             // it will show error because we can not access private members
cout<<"\n Codeno = "<<one.codeno;
cout<<"\n Price = "<<one.price;
 cout<<"\n Quantity ="<<one.qty;
return 0;
}
```

when the above program is compiled, the compiler will display the error message codeno, price, qty are private inthis scope.
so, the private members are not accessible by the object directly. Now how to access the private members. To access private data members a member function must be declared in the class in public section.
** Member functions are used to initialize the data members.

Example:

```
#include<iostream>

using namespace std;

class item

{
private: // private section begins

int codeno;

float price;

int qty;

public :              // public section starts

void display()
{
// member function

codeno=123; // member initialization


price=123.45;

qty=150;
cout<<"\n Codeno = "<<codeno;

cout<<"\n Price = "<<price;

cout<<"\n Quantity = "<<qty;
}

};              // end of class


int main()

{

item one;             // object declaration

one.display();
return 0;
}
```

3. Protected: A protected access specifier is a stage between *private* and *public* access. If member functions defined in a class are *protected*, they cannot be accessed from outside the class but can be accessed from the derived class. We can not access protected section members from outside the class by any object. Protected keyword is used in inheritance concepts.

Abstraction

Abstraction is one of the most important concepts of object-oriented programming. It refers to showing only relevant information to the outside world. In simple words, we can say it means hiding any background information from the outside world.

Data abstraction refers to showing only relevant information about data to the outside world or hiding the implementation.

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

Thus, we can say a television clearly separates its internal implementation from its external interface and you can play with its interfaces like the power button, channel changer, and volume control without having any knowledgeof its internals.

Types of Abstraction in C++

There are 2 types of abstraction in C++:

1. **Data Abstraction:** It hides the information about the data.
2. **Control Abstraction:** It hides the information about the implementation.

Ways of Achieving Data Abstraction in C++

Data Abstraction can be achieved in two ways:

1. **Abstraction using classes** : We can implement Abstraction in C++ using classes. Class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to outside world and which is not.
2. **Abstraction in header files** : An another type of abstraction is header file.
   For example:- pow() function available is used to calculate the power of a number without actually knowing which algorithm function uses to calculate the power. Thus, we can say that header files hides all the implementation details from the user.

Abstraction using Access modifiers

Access modifiers are the main pillar of implementing abstraction in C++. We can use access specifiers to enforce restrictions on class members.

- Members declared as **public** in a class, can be accessed from anywhere in the program.
- Members declared as **private** in a class, can be accessed only from within the class. They are not allowed tobe accessed from any part of code outside the class.

C++ Program to demonstrate Abstraction

Example of data abstraction using classes.

```
#include <iostream>
using namespace std;
class abstraction {
int a, b;
public:
// method to set value of private members
void value(int x, int y)
 {
a = x;
b = y;
}
void display()
{
```

```
cout << "The value of a = " << a << endl;
cout << "The value of b = " << b << endl;
}
};

int main()
{
abstraction obj;
obj.value(2, 4);
obj.display();

return 0;
}
```

Output

The value of a = 2
The value of b = 4

In the above example, we are not allowed to access the variables a and b directly, however one can call the function value() to set the values in a and b and the function display() to display the values of a and b.

C++ Program to calculate the power of a number.

Example of data abstraction in header files.

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
 {
int n = 2;
int power = 3;
// pow(n, power) is the power function
 int result = pow(n, power);
cout << "Cube of n = " << result;
return 0;
}
```

Output

Cube of n = 8

In the above example, pow() function is used to calculate 2 raised to the power 3. The pow() function is present inthe math.h header file in which all the implementation details of the pow() function is hidden.

Why Abstraction?

- Data abstraction means **hiding of data.**
- Abstraction is implemented automatically while writing the code in the form of class and object.
- It shows only important things to the user and hides the internal details.

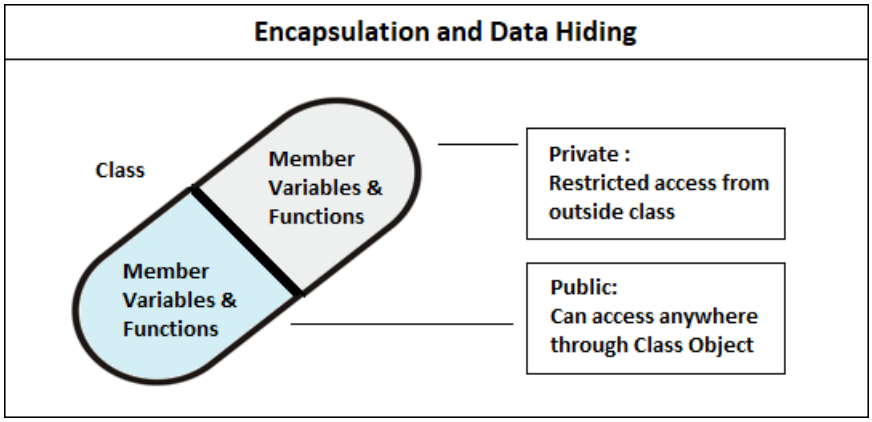
Advantages of Data Abstraction

Data abstraction has several advantages including:

- **Simplicity:** Helps the user to avoid writing the low-level code.
- **Reusability**: Avoids code duplication and increases reusability.
- **Modularity:** Allows for changes to the internal implementation of the class independently without affecting the user.
- **Security:** Enhances the security of an application or program by only providing essential details to the user and keeping sensitive information hidden.
- **Readability:** Reduces the complexity as well as the redundancy of the code, therefore increasing the readability.

Encapsulation

Encapsulation is a process of combining member functions and data members in a single unit called a class. The purpose is to prevent access to the data directly. Access to them is provided through the functions of the class. It is one of the popular features of Object-Oriented Programming (OOPs), which helps in **data hiding**.



Real-life example

Suppose you go to an automatic teller machine (ATM) and request money. The machine processes your request and gives you money.

Here, ATM is a class. It takes data from the user (money amount and PIN) and displays data as icons and options. It processes the request (functions). So, it contains both **data** and **functions** wrapped/integrated under a single ATM. This is called **Encapsulation.**

C++ Encapsulation

```
// Program to calculate the area of a rectangle
#include <iostream>
using namespace std;

class Rectangle
{
public:
// Variables required for area calculation
 int length;
int breadth;

// Constructor to initialize variables
```

```
Rectangle(int len, int brth)
{
Length=len;
Breadth=brth;
}
// Function to calculate area
int getArea()
{
return length * breadth;
}
};

int main() {
// Create object of Rectangle class
Rectangle rect(8, 6);

// Call getArea() function
cout << "Area = " << rect.getArea();

return 0;
}
```

Output

```
Area = 48
```

In the above example, we are calculating the area of a rectangle.
To calculate an area, we need two variables: length and breadth and a function: getArea(). Hence, we bundledthese variables and function inside a single class named Rectangle.
Here, the variables and functions can be accessed from other classes as well. Hence, this is not **data hiding**.This is only **encapsulation**. We are just keeping similar codes together.

Benefits of encapsulation:

Encapsulation provides several benefits, including:

- Improved code maintainability: Encapsulation helps in improving the code maintainability by providing aclear separation between the implementation details of a class and its clients.
- Data hiding: Encapsulation enables data hiding, which protects the data members of a class from beingaccessed and modified by the clients of the class.
- Code reuse: Encapsulation helps in code reuse by providing a modular design that can be easily extendedand modified.
- Security: Encapsulation provides security by preventing unauthorized access to the data members of a class.

Constructor :

If we want to automatically initialize data members i.e., with out calling member function we want to initializedata members. It is possible only through Constructors .
A constructor is a special member function.
Constructors are used to initialize the objects of its class and this is called automatic initialization of object. Constructors constructs the values of data members of class i.e., when constructor is used data members are automatically initialized, when the object of the class is created.

Characteristics of constructors :

1. Constructor name and class name should be same.
2. Constructors should be declared in public section.
3. Constructors never have any return value (or) data type including void.
4. The main function of Constructor is to initialize objects and allocate appropriate memory toobjects.
5. Constructors can have default values and can be overloaded.
6. Constructor may or may not have arguments (or) parameters.
7. Constructor is executed only once when the object is created.
8. Constructors are automatically invoked they are not invoked with dot(.) operator.
9. Constructor is invoked automatically when object of class is created.

Syntax Of Constructor

the syntax for defining constructor inside the class body is as follows:

```
clasc classname
{
……public :
classname([parameter_list]) // constructor definition
{
. . . . . .
}
. . . . . .
};
```

Constructors can be defined either inside the class definition or outside class definition using class name and scoperesolution :: operator.

```
class classname
{
. . . . . .
public:
classname ([parameter_list]);//constructor declaration
. . . . . .
};
classname: :classname([parameter_list])//constructor definition
{
. . . . . .
}
```

Example: Constructor definition inside the class

```
class A
{
public:
 int x;
A() // constructor
{
// object initialization
}
};
```

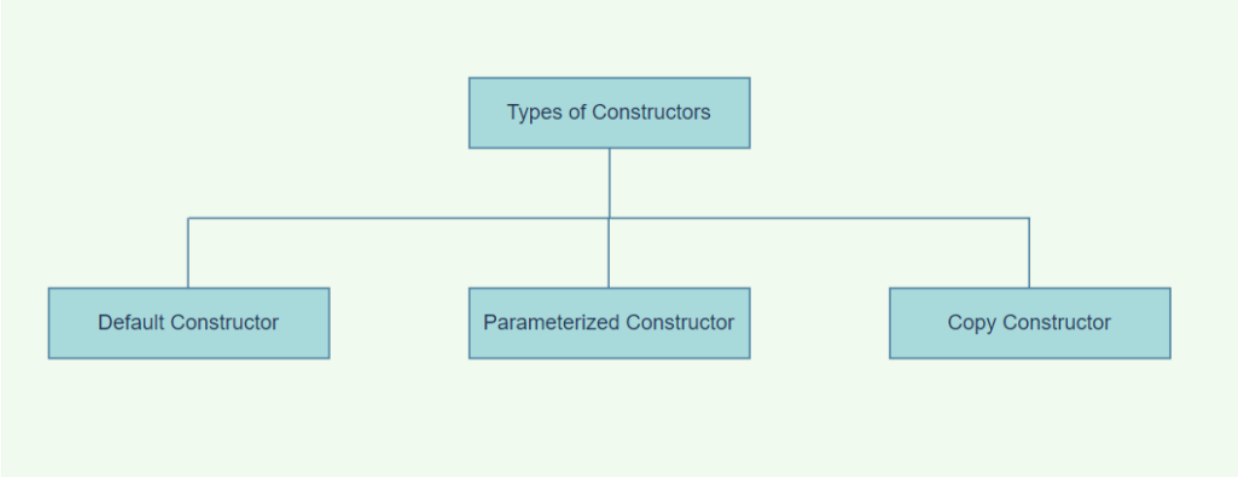
Example: Constructor definition outside the class

```
class A
 {
public:
int i;
A(); // constructor declared
};
A::A() // constructor definition
{
i = 1;
}.
```

Types of Constructors

There are 3 types of constructors in C++, They are :

- Default Constructor
- Parameterized Constructor
- Copy Constructor



Default Constructor

Constructor with no arguments (or parameters) in the definition is a default constructor. It is the type of constructor in C++ usually used to initialize data members (variables) with real values. The default constructor can be identified by the name of the class followed by empty parentheses.

Note: The compiler automatically creates a default constructor without data member (variables) or initialization if no constructor is explicitly declared.

Syntax of Default Constructor –

```
class CLASS_NAME
{
  ……
public :
CLASS_NAME()  //Default constructor
{
                    . . . . . .
}

//other member functions
};
```

Example

```
// C++ program to demonstrate the use of default constructor

#include <iostream>
using namespace std;

// declare a class
class  Wall {
  private:
    double length;

  public:
    // default constructor to initialize variable
    Wall() {
      length = 5.5;
      cout << "Creating a wall." << endl;
      cout << "Length = " << length << endl;
    }
};

int main() {
  Wall wall1;
  return 0;
}
```

Output

```
Creating a Wall
Length = 5.5
```

Explanation

Here, when the wall1 object is created, the Wall() constructor is called. This sets the length variable of the object to 5.5.

Parameterized Constructor

In C++, a constructor with parameters is known as a parameterized constructor. This is the preferred method to initialize member data. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.

The syntax for declaring parameterized constructor inside the set:

```
class class_name
{

public:
  class_name(variables) //Parameterized constructor declared.
  {

  }
};
```

The syntax for declaring parameterized construct outside the class:

```
class class_name
{

};
class_name :: class_name() //Parameterized constructor declared.
```

```
{

}
```

Example

```
// C++ program to calculate the area of a wall

#include <iostream>
using namespace std;

// declare a class
class Wall {
  private:
    double length;
    double height;

  public:
    // parameterized constructor to initialize variables
    Wall(double len, double hgt) {
      length = len;
      height = hgt;
    }

    double calculateArea() {
      return length * height;
    }
};

int main() {
  // create object and initialize data members
  Wall wall1(10.5, 8.6);
  Wall wall2(8.5, 6.3);

  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;
  cout << "Area of Wall 2: " << wall2.calculateArea();

  return 0;
}
```

Output

```
Area of Wall 1: 90.3
Area of Wall 2: 53.55
```

Explanation

Here, we have created a parameterized constructor Wall() that has 2 parameters: double len and double hgt. The values contained in these parameters are used to initialize the member variables length and height.

When we create an object of the Wall class, we pass the values for the member variables as arguments. The code for this is:

```
Wall wall1(10.5, 8.6);
Wall wall2(8.5, 6.3);
```

With the member variables thus initialized, we can now calculate the area of the wall with the calculateArea() function.

Copy Constructor

If we have an object of a class and we want to create its copy in a new declared object of the same class, then a copy constructor is used. The copy constructor in c++ is a constructor that creates an object by initialising it with a previously created object of the same class. The compiler provides each class a default copy constructor and users can define it also. It takes a single argument which is an object of the same class.

Syntax:

```
class ClassName
 {
Access_specifier:
ClassName( ClassName &source)
{
// Copy data members from the source object
}
};
```

Here,

- **class** is the keyword used for the class definition.
- **ClassName** is the name of the class in which the constructor is defined and also the name of the constructor.
- **Access_specifier**: It is a data visibility controller which could be public, private, protected
- **ClassName &source:** It is a parameter of the copy constructor that accepts objects of the same class.

Example

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

// declare a class
class Wall {
  private:
    double length;
    double height;

  public:

    // initialize variables with parameterized constructor
    Wall(double len, double hgt) {
      length = len;
      height = hgt;
    }

    // copy constructor with a Wall object as parameter
    // copies data of the obj parameter
    Wall(Wall &obj) {
      length = obj.length;
      height = obj.height;
    }

    double calculateArea() {
      return length * height;
```

```
    }
};

int main() {
  // create an object of Wall class
  Wall wall1(10.5, 8.6);

  // copy contents of wall1 to wall2
  Wall wall2 = wall1;

  // print areas of wall1 and wall2
  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;
  cout << "Area of Wall 2: " << wall2.calculateArea();

  return 0;
}
```

Output

```
Area of Wall 1: 90.3
Area of Wall 2: 90.3
```

Explanation

In this program, we have used a copy constructor to copy the contents of one object of the Wall class to another. The code of the copy constructor is:

```
Wall(Wall &obj)
 {
  length = obj.length;
  height = obj.height;
}
```

Notice that the parameter of this constructor has the address of an object of the Wall class.
We then assign the values of the variables of the obj object to the corresponding variables of the object calling the copy constructor. This is how the contents of the object are copied.
In main(), we then create two objects wall1 and wall2 and then copy the contents of wall1 to wall2:

```
// copy contents of wall1 to wall2
Wall wall2 = wall1;
```

Here, the wall2 object calls its copy constructor by passing the address of the wall1 object as its argument i.e. &obj = &wall1.

Constructor Overloading

The use of multiple constructors in the same class is known as **Constructor Overloading**.
The constructor must follow one or both of the two rules below.

- All the constructors in the class should have a different number of parameters.
- It is also allowed in a class to have constructors with the same number of parameters and different data types.

Declaration

Here is the code syntax for the declaration of constructor overloading

```
class ClassName
  {
    public:
       ClassName()
     {
```

```
        body; // Constructor with no parameter.
      }
      ClassName(int x, int y)
     {
        body; // Constructor with two parameters.
      }
      ClassName(int x, int y, int z)
       {
        body; // Constructor with three parameters.
      }
      ClassName(ClassName & object)
     {
        body; // Constructor with the same class object as a parameter.
      }
      // Other member functions.
    };
```

Example :

Here is a program to overload three constructors, one to set the area with no parameters, the second to set the area with one parameter, and the third to set the area with two parameters.

```
// C++ program to demonstrate constructor overloading.

#include <iostream>
using namespace std;
class Area
{
   public:
      // Member Variable Declaration.
      int area;
      // Constructor with no arguments.
      Area()
 {
         area = 0;
      }
      // Constructor with one argument.
      Area(int side)
{
         area = side * side;
      }
      // Constructor with two arguments.
      Area(int length, int width)
{
         area = length * width;
      }
      // Member function declaration.
      int disp()
 {
         return area;
      }
};

int main()
 {
   Area obj1;
   Area obj2(6);
   Area obj3(8, 5);
```

```
    cout << "Area of obj1: " << obj1.disp() << endl;
    cout << "Area of obj2: " << obj2.disp() << endl;
    cout << "Area of obj3: " << obj3.disp() << endl;
    return 0;
}
```

Output

```
Area of obj1: 0
Area of obj2: 36
Area of obj3: 40
```

Explanation

In the above C++ program, we have created a class Area with one variable, area. We have also defined three constructors, Area(), Area(int side), and Area(int length, int width).

- The first constructor, Area(), is called when the object obj1 is created because we have not passed any argument.
- The second constructor, Area(int side), is called when the object obj2 is created because we have passed one argument.
- The third constructor, Area(int length, int width), is called when the object obj3 is created because we have passed two arguments with it.