OOPS Unit-I

OOPS:-

⇒ OOPS is the Principle of design & development of Prog. using Modular approach.

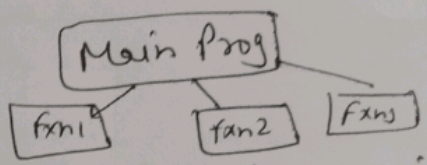
⇒ This approach provides advantages in creation & development of s/w for real life applns.

⇒ The basic element of OOP is the data.

⇒ The Program are built by combining data and functions that operate on the data

⇒ Some of the OOP's languages are C++, Java, C#, Smalltalk, Perl & Python.

Procedural Programming:-

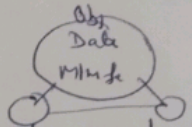


⇒ The Procedural Programming focus on processing of instrn in order to perform a desired computations.

⇒ It follows Top down approach to decompose main fxn into lower level components for modular coding purpose.

⇒ Therefore more emphasizes on doing things like algorithms. This technique is used in conventional Prog. languages such as C & Pascal.

Object Oriented Programming:-



⇒ OOP is a concept that combines both the data and the fxn that operates on that data into a single unit called the object. Comm. of objects done through fxns.

⇒ An object is a collection of set of data known as member data & the function that operate on these data known as member fxn.

⇒ OOP follows bottom up design technique.

⇒ Class is a major concept that plays important role in this approach.

⇒ class is a major concept that plays important role in this approach.

⇒ class is a template that represents a group of object which share common properties & relationship.

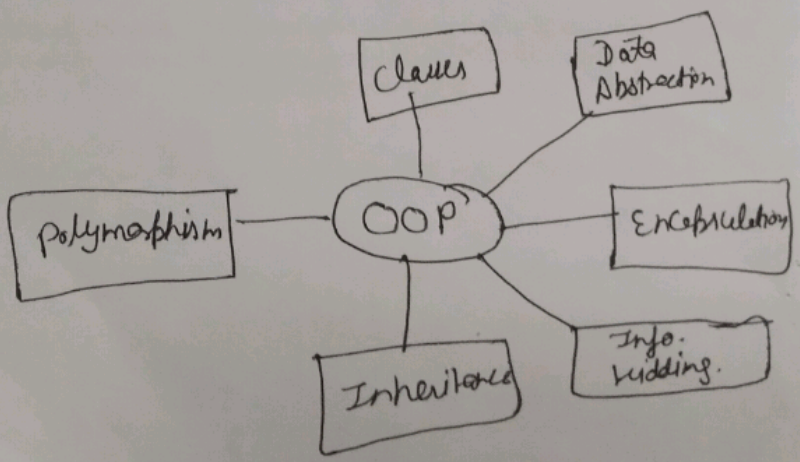
Difference b/w Procedural Programming & object oriented prog.

POP	OOPS
1.) Large Programs are divided into smaller Programs known as function.	1.) Programs are divided into objects.
2) Data is not hidden & can be accessed by external function.	2.) Data is hidden & cannot be accessed by external functions.
3.) Follows top-down approach in the Program design.	3.) Follows bottom up approach in the Program design.
4.) Data may communicate with each other through fcns.	4.) objects may communicate with each other through functions.
5) Emphasize is on procedure rather than data.	5) Emphasize is on data rather than Procedure.

Basic Concepts of OOP's :-

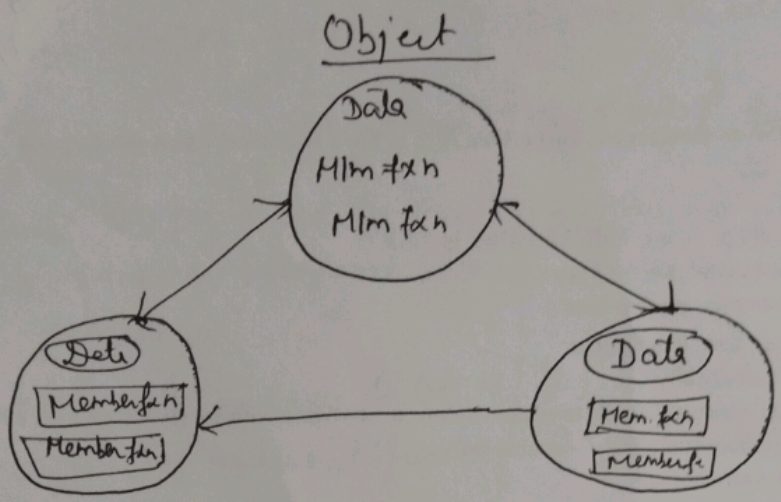
The following are the major characteristics of OOP's.

1. Objects
2. class
3. Data Abstraction
4. Data encapsulation
5. Inheritance
6. overloading
7. Polymorphism
8. Dynamic Binding.
9. Message Passing.



⇒ class is a major concept that allows inheritance.

Object oriented Programming :-



Characteristics of Object-oriented Programming :-

⇒ Emphasis on data rather than procedure.
⇒ Programs are divided into entities known as objects.
⇒ Data structures are designed such that they characterize object.
⇒ Fxns that operates on data of an object are tied into data structure
⇒ Data is hidden & cannot be accessed by external fxns.
⇒ Objects communicates with each other through functions
⇒ New data & fxns can be easily added whenever necessary.
⇒ Follows bottom up design in program design.

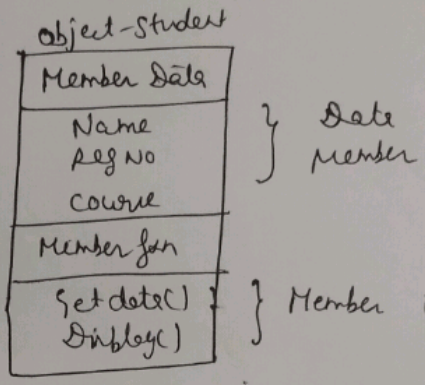
Bottom up approach :-

⇒ Reverse top-down approach.
⇒ Lower level tasks are first carried out & are then integrated to provide the solution of a single Prog.
⇒ Lower level structures of the program are evaluated first then higher level structures are created.
⇒ It promotes code reuse.
⇒ It may allow unit testing.
⇒

Objects :-

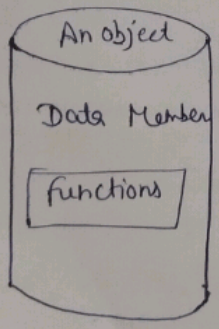
⇒ objects are basic building blocks for designing Program.
⇒ An object is a collection of data members & associated member functions
⇒ An object may represent a Person, Place or a table of data.
⇒ Each object is identified by a unique name.
⇒ Each object must be a member of a Particular Class.

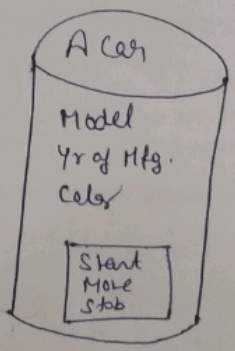
Example :- Apple, orange, Mango are the objects of the class fruit.



Objects are runtime entities of an object oriented system.

Ex Representation of objects :-





⇒ When a Program is executed the objects interact by sending message to one another.
⇒ Ex. Customer Account → objects in a Prog.
Send amount Reg for bank Data Account

Object → Basic unit of OOP 4

That is both data & fun that operate on data are bundled as a unit called as object

Declare Class :-

Class _ Name object _ Name;

Test T;

Ex :- Human being → is a class

Man → object

women → object

Color – Class

Red] – Object
blue

Ex :- Airport – Class

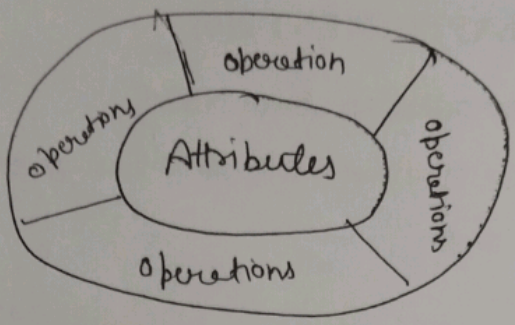
App – obj.
Mn

```
# include <iostream>
using namespace std;
class Test {
    Private :
      int n=10;
  Public :
      void show () {
        Cout << " The value of n: "<<n<<endl;
        }
    };
  main () {
class — Test T; — obj  // Declaring obj
obj — T.Show();
  }
```

Class declared & Method access

. – keyword
to
for Method
access
through
class

Objects



Class:- Classes are user defined data types and it behaves like build in types of Prog. Lang.

⇒ Object contains code & data which can be made user define type using class.

⇒ Objects are variables of class.

⇒ Once a class has been defined we can create any no. of objects for the class.

⇒ A class is collection of objects of similar type.

How we can create object of an class:-

We can create object of an class using following syntax.

Syntax:- class_name object_name;
↓ ↓
already created User defined name

Example:- If Student is class

Student ram, sham;
↓ ↓ ↓
class Name of object

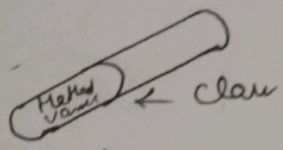
Ex: Planet, Sun, Moon.
↓ ↓
class M|M

NOTE We can create any no. of objects for class.

Encapsulation:- Encapsulation is the first pillar or principle of object oriented Programming.

Definition:- Encapsulation is a process of binding data members (variables, Properties) and member function (method) in a single unit.

Best example → class



Data encapsulation enables data hiding & information hiding.

Data hiding is a method used in object oriented Programs to hide info within Computer Code.

Example:- Medical Store

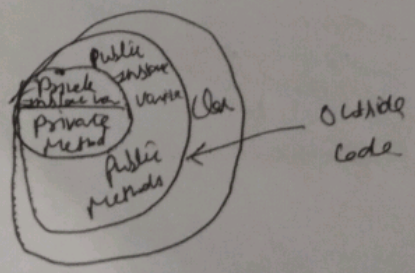
Let say you have to buy some medicines. You go to the medical store and ask the chemist for the medicines. Only the chemist has access to the medicines in the store based on your prescription.

The chemist knows that medicines to give to you.

This reduce the risk of you taking any medicine box. that is not intended for you.

⇒ In this example:-

Medicines → Member variables
Chemist → Member Methods
You → External Application or piece of code, etc.



... is the first pillar or principle

⇒ Through Encapsulation, Data is not accessible to the outside world and only those function which are wrapped in the class can access it.

⇒ Encapsulation solve the problem at the implementation level.

⇒ A class can specify how accessible each of its members (variable, Properties and methods) is to code outside of the class.

⇒ Encapsulation means hiding the important features of a class which is not been needed to be exposed outside of a class and exposing only necessary things of a class.

⇒ Hidden Part of a class acts like encapsulation & exposed part of a class like abstractions.

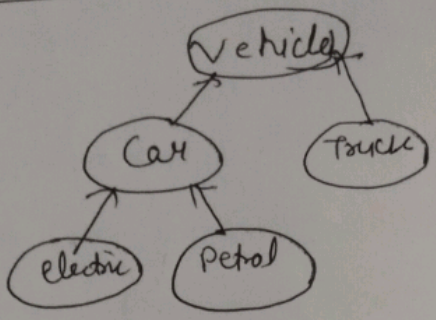
Abstraction :- Abstraction refer representation of necessary features without including more details or explanations.

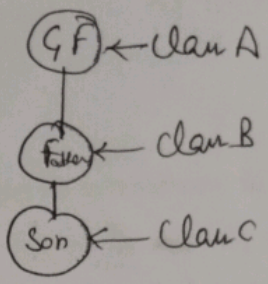
⇒ Data abstraction is a Programming (and design) technique that relies on the separation of interface & implementation.

⇒ When you press a key on your keyboard the character appear on the screen you need to know only this, but exactly it works based on the electronically is not needed.
This is called abstraction.

⇒ Another example is when you use the remote control of your tv, you do not bother about how pressing a key in the remote changes the channel on the T.V. You just know that pressing the + volume button will increase the

⇒ # Inheritance:-

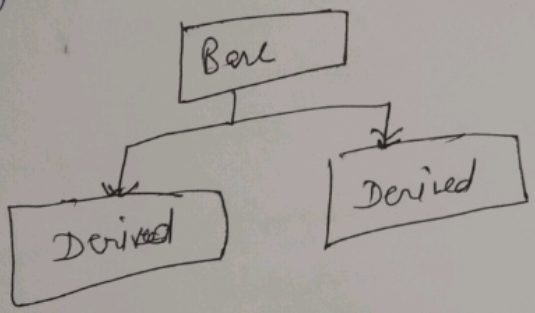




⇒ The mechanism of deriving a new class from an old class is called inheritance or derivation. The old class is known as base class or super class while new class is known as derived class or sub class.

⇒ The inheritance is the most powerful features of OOP.

⇒ The derived class shares some of the properties of the base class. Therefore a code from a base class can be reused by a derived class.



⇒ For example:-

Consider an example of family of three members having a mother, father & son named Jack.

Jack father → tall & dark
Jack Mother → short & fair
Jack is tall & fair so he is said to have inherited the features of his father and mother both.

Note: Through effective use of inheritance, you can save lot of time in your programming & also reduce errors which in turn will increase the quality of work & productivity.

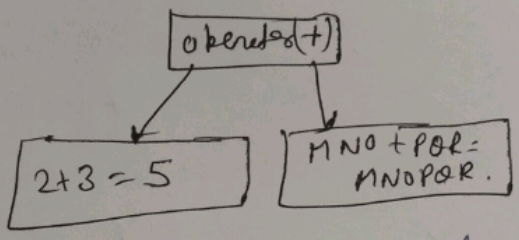
⇒ The different types of Inheritance are — 10/

1.) Single inheritance
2.) Hierarchical Inheritance
3.) Multible Inheritance
4.) Multilevel Inheritance.

Polymorphism :- Polymorphism is a greek term which means ability to take more than one form.

~~Poly~~ ability of an operator and fun to take multible forms is known as.

for example :- + is used to make sum of two number as well as it is used to combine two strings.

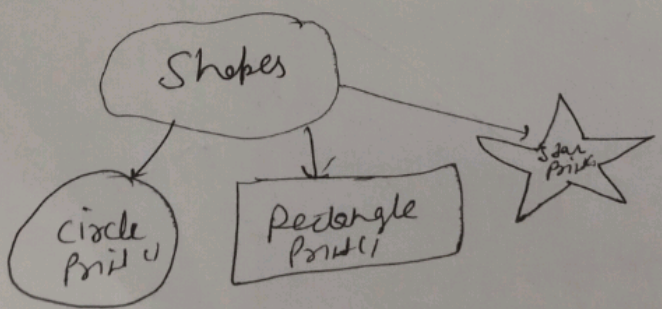


⇒ This is known as operator overloading because same operator may behave differently on different instances.

Same way functions can be overloaded.

⇒ for example, sum() function may takes two arguments or three arguments etc. i.e sum(5,7) or sum(4,6,8)

single function Print() draws different objects.

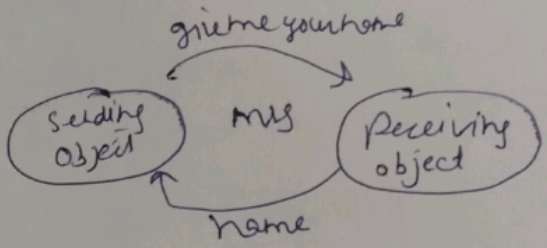


Dynamic Binding :- Binding means link b/w procedure call & code to be execute.

It the process of linking of a fun call to the actual code of the fun at run time.

⇒ That is, in dynamic binding the actual code to be executed is not known to the compiler until run time.
It is also called Late binding.
Dynamic binding is the process of connecting one prog to another.

For example:-
Compiler comes to know at runtime that which function of sum will be call either with two arguments or with three arguments.



Message Passing:-
⇒ Objects can communicate with each other by passing Message same as people passing message with each other.
⇒ Objects can send or receive message or information
⇒ Message passing involves the following basic steps.
(1.) Create classes that define object & their behaviour.
(2.) Creating object from class definitions.
(3.) Establishing communication among objects.
(4.) Concept of message passing makes it easier to talk about building system that directly model or simulate the real world counterparts.

⇒ for example – Consider two classes Product and order. The object of the Product class can communicate with the object of the order class by sending a request for placing order.

#Benefits of OOP:-

⇒ User can create new data type or user define data type by making class.

⇒ Code can be reuse by using inheritance.

⇒ Data can be hiding from outside world by using encapsulation.

⇒ Operators or fxns can be overloaded by using poly mathism, so same fxn or operators can be used for multitasking

⇒ OO System can be easily upgrade from small system to large system.

⇒ It is easy to Partition work for same project.

⇒ Message Passing techniques make communication easier

Software Complexity can be easily managed.

⇒ Maintenance cost is less.

⇒ Simple to implement.

Areas for applications of OOP:-

1. Real time systems.
2. Simulation & modeling.
3. Object oriented database
4. Hypertext, hypermedia & expert text.
5. AI & expert system.
6. Neural N/w & Parallel Programming.
7. Decision support System.
8. office automation System.
9. CIM / CAM / CAD systems.

What is C++?

⇒ C++ is most Popular Cross-platform Prog. Lang. which is used to create high performance apps & SW like Games, OS E-Commerce SW etc

⇒ High level & object oriented Programming Language.

⇒ It was developed by Bjarne Stroustrup.

⇒ It is extension of C Language.

⇒ C++ gives a high level of control over system resource & mem.

Why learn C++?

⇒ C++ is one of the most used & Popular Programming Lang.

⇒ C++ is used in making O.S, embedded system & SVS.

⇒ C++ is OOP Lang. that implements the OOPs Concepts

⇒ C++ is Portable & can be used to create apps that can be adapted to multiple platforms.

⇒ C++ is easy to learn so that you can choose it as your first Prog. Lang.

⇒ C++ makes Programming easy for Programmers to switch to C++ because its syntax is similar to C, Java & C#.

Difference between C++ & C Language:

C++	C
① C++ was developed as an extension of C.	① C is a developed independent language
② C++ supports classes & Objects	② C does not support OOP Concepts
③ namespace is used by C++, to avoid name collisions.	③ The namespace feature is not supported by C.

Standard C++ Programming is divided into three important Part

1.) The core library includes the data types, variables & literals etc.

2.) The standard library includes the set of fxn manipulating strings, files etc.

3.) The standard Template Library (STL) includes the set of methods manipulating a data structure.

⇒ By the help of C++ programming language we can develop different types of secured & robust applications.

1. Window Appln.
2. Client server Appln
3. Device driver
4. Embeded firmware etc

⇒ using namespace std :- Line can be omitted and replaced with a std keyword followed by the :: operator for some objs.

```
#include<iostream>
using namespace std; / std::{cout<<" ";
int main() {
Cout << "Hello C++ Programming ";
return 0;
}
```

```
#include<iostream.h>
#include<conio.h>
void main() {
clrscr();
cout<<"Welcome to C++ Programming";
getch();
}
```

⇒ #include<iostream.h> → Include standard input/output library. It provides cin & cout methods for reading input & writing to output respectively.

⇒ #include<conio.h> → include the console input/output library. The getch() fxn is defined in conio.h file.

⇒ using namespace std :- It means that we can use names of obj/variable

#) What is c++?

from the standard library.

⇒ void main() → The main() function is the entry point of every prog.

⇒ The void keyword:- Specify that it returns no value.

⇒ Cout << "welcome to c++ Programming" → is used to print the data welcome to c++ Programming". on the Console. It is an object used together with the insertion operator (<<) to o/p / Print fun.

⇒ getch() → The getch() fun asks for a single charecter. until you press any key, it blocks the screen.

⇒ How to Compile & Run Programe:?

There are 2 ways to compile & run the c++ Prog. by menu and by shortcut.

① By menu:- click on Compile menu then Compile sub menu to Compile the c++ Prog.

then click on the run menu then run sub menu to run the c++ program.

② By shortcut:- Ctrl + F9 key Compile & run the Prog. directly

view the user screen anytime → alt + F5

⇒ Esc → to return to the turbo c++ Console.

⇒ /n → new line

⇒ int main() → This is called a function. Any code inside its curly brackets { } will be executed.

Ex:-
```
int main()
  {
  cout << "hello world!";   → body
   return 0;  // end the main fun.
   } // end of the main fun
```

Note → Compiler ignore white Space.

Call a function :-

Declared fans are not executed immediately. They are "saved for later use", and will be executed later, when they are called.

To call a function, write the fan name followed by two Parentheses () and a semicolon;

Create a function

```
Void myfunction ()
  {
Cout << "I just got executed!";
  }

int main ()
{
myfunction (); // Call the function
return 0;
}
```

O/P → I just got executed

Declaration :- The return type, the name of the function and Parameters.

Definition :- The body of the function (Code to be executed)

```
Void myfunction (){ → fan declaration
    {
// the body of the fan → definition)
    }
```

Note If a user defined fan, such as myfunction() is declared after the main() fan, an error will occur

Call a function:-
It is possible to separate the declaration & the defn of the fun for code optimization.

Return by reference in c++:-

Pointer & References in c++ hold close relation with one another. The major difference is that the Pointer can be operated on like adding values whereas references are just an alias for another variable.

Function in c++ :- can return as it's returns a Pointer.

When function returns a reference it means it returns a implicit Pointer.

Return by reference is very different from call by reference.

Note:- We should never return a local variable as a reference, reason being as soon as the fun returns local variable will be erased, however we still will be left with reference which might be a security bug in the code.

Inline function in c++:-

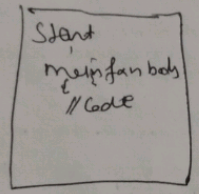
C++ Provides inline functions to reduce the function call overhead. An inline function is a function that is expanded in line when it is called.
When the inline function gets invoked or substituted at the point of the inline function call.

This substitution is performed by the C++ compiler at compile time.

An inline fun may increase efficiency if it is small.

Syntax :

```
inline return-type function-name (Parameters)
{
//function code
}
```



Inline is only a request to a compiler, not a command.

Why Inline function are Used —

Function Overloading :-

Suppose a c++ class has multible member functions with the same name, but different parameters (with change in type, sequence, or number) and the Programmer can use them to perform the same kind of Operations. It refers to function overloading.

In this case two functions may have the same identifier (name) and if the number of argument Passed to the function or the types of arguments differ, overloading is possible in c++ Programs.

It is the ability to create multible functions with the same name & slightly different implem-entations.

Ex: Consider we have two functions that add numbers of different type.

```
# include <iostream>
using namespace std;
int plusfuncInt (int x, int y)
    {
    return x+y;
    }
double plusfuncDouble (double x, double y)
  {
  return x+y;
  }
int main()
  {
int myNum1 = PlusfuncInt (8,5);
  double myNum2 = PlusfuncDouble (4.3, 6.26);
Cout << "Int: " << myNum1 << "\n";
  Cout << "Double: " << myNum2;
  Return 0; }
```

O/P Int: 13
double: 5C

Note: Multiple functions can have the same name as long as the number and/or type of Parameters are different.

Recursion :-

Recursion is the technique of making a functioncall itself. This technique provides a way to break Complicated Problems down into simple Problems which are easier to solve.

Recursion Example :-

Adding two numbers together is easy to do, but adding a range of number is more complicated. Here, recursion is used to add a range of numbers together by breaking it down into the simple task of adding two numbers.

Ex:-

```
#include <iostream>
using namespace std;
int sum (int k) {
 if (k>0) {
return k+ sum (k-1); }
 else {
 return 0;
 }
}
int main() {
int result = sum (10);
Cout << result;
  return 0;
  }
```

O/P → 55

When the sum() function is called, it adds Parameter k to the sum of all numbers smaller than k & return the result.
When k becomes 0, the function just return 0.

10 + sum(9)
10 + (9 + sum(8)
10 + (9 + (8 + sum(7)))

10+9+8+7+6+5+4+3+2+1+sum(0)
10+9+8+7+6+5+4+3+2+1+0

Here the function does not calls itself when k is 0, the Program stop there & result return.

NOTE:- If User defined function is declared after the main fun, an error will occur.

Friend function :-

If a fun is defined as a friend function in C++, then the Protected and private data of a class can be accessed using the function.

By using the keyword friend compiler knows the given function is a friend function.

For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword fun.

1

Declaration of friend function :-

```
class class_name
{
friend data_type function_name (argument/s);
};
```

In this the friend function is preceded by the keyword friend.

Characteristics of a friend function :-

1) The function is not the scope of the class to which it has been declared as a friend.
2) It cannot be called using the object as it is not in the scope of that class.
3) It can be invoked like a normal function without using the object
4) It cannot access the member names directly & has to use an object name & dot membership operator with the member name.
5) It can be declared either in the Private or the Public part.

NOTE:- A friend function of class can access both Private & Protected member of the class in which it has been declared as a friend.

Ex:-

```
#include<iostream>
using namespace std;
class A
{
 int x=5;
 friend class B;   //friend function.(B is declared as a frnd fun inside the class A)
 };
class B
{
Public:
 void display(A &a)
 {
 Cout<<" Value of x is :"<<a.x;
  }
  };
 int main()
 {
  Aa;                    Here
  B b;
  b.display(a);
  return 0;
  }
```

O/P x=5

Here B class B can access the private members of class A.

Unit-IInd

#Specifying a class:-

⇒A class is a user defined data type which hold both the data and function.
The data inside the class are called member data and the function are called member function.

Example:-

Create a class called "MyClass";

```
class MyClass{ The class
  public:  // Access specifier
  int myNum; // Attribute (int variable)
  string myString; // Attribute (string variable)
  };
```

Class Keyword:- It is used to create a class called MyClass.

Public:- Access specifier (which specifies that members (attributes & Methods) of the class are accessible from outside the class.

Inside the class, there is an integer variable myNum and a string variable myString.
When variables are declared within a class, they are called attributes.

Ex
```
#include <iostream>
#include <string>
using namespace std;
class MyClass {  The class
  public        // Access specifier
```

```
int myNum; // Attribute (int variable)
string myString // Attribute (String Variable)
};
int main() {
MyClass myObj; // Create an object of MyClass
myObj.myNum = 15;        Access attributes & set values
myObj.myString = "some text";
Cout << myObj.myNum << "\n";   ] Print values
Cout << myObj.myString;
return 0;
}
```

O/P → 15
some text

Create a Car class with some attributes:-

Ex:-

```
#include <iostream>
#include <string>
using namespace std;
class Car {
public:
string brand;
string model;
int year;
};
int main() {
Car carObj1;
carObj1.brand = "BMW";
carObj1.model = "X5";
carObj1.year = 1999;

Car carObj2;
carObj2.brand = "Ford";
carObj2.model = "Mustang";
```

```
carObj2.year = 1969;
coutObj1.ha
cout << carObj1.brand << " " << carObj1.model << " " <<
carObj1.year << "\n";
cout << carObj2.brand << " " << carObj2.model << " " <<
carObj2.year << "\n";
return 0;
}
```

Output BMW X5 1999
Ford Mustang 1969

Member function :- A member function of a class is a function that has its definition or its Prototype written within the class definition like any other variable.
It operates on any object of the class of which it is a member, and has access to all the members of a class for that object.

```
class Box {
  public:
  double length; // Length of a box
  double breadth; // Breadth of a box
  double height; // Height of a box
  double getVolume(void); // Returns box volume
    };
```

Member functions can be defined within the class definition or separately using scope resolution operator,

Encapsulation:-

In C++, Encapsulation involves combining similar data and function into a single unit called a class.

By Encapsulating these functions & data, we protect that data from change.

The concept is also known as data or information Hiding.

There are three basic techniques to encapsulate data in object programming.

Data members, methods and classes can be encapsulated

Encapsulation is one of the fundamentals of OOPS. It refers to the bundling of data with the method that operate on that data.

Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties' direct access to them.

Two important property of Encapsulation

① Data Protection:- Encapsulation protects the internal state of an object by keeping its data members private. Access to and modification of these data members is restricted to the class's public methods, ensuring control and secure data manipulation.

② Information Hiding:- Encapsulation hides the internal implementation details of a class from external code. Only the public interface of the class is accessible, providing abstraction & simplicity

Features of Encapsulation

C++ Program to demonstrate Encapsulation.

```
#include <iostream>
using namespace std;
class Encapsulation {
private:          // Data hidden from outside world.
  int x;
public:           // fun function to set value of var x.
  void set (int a ) { x=a; } // Funct to return value
int get () {return x;}       of variable x.
};                // Drier Code
int main()
{
```

the usage of the class while allowing the internal implementation to be modified without impacting external code.

Ex: If we give input, and output should be half of input.

```
#include<iostream>
using namespace std;
class temp {
 int a;
 int b;
 public :
int solve(int input)
 {
  a= input;
  b = a/2;
  return b;
 }
 };
 int main() {
 int n;
 cin>>n;
 temp half ;
int ans = half.solve(n);
cout<< ans<< endl;
 }
```

```
int main(){
cout<< abbles;
NO works
```

```
or int main() {
int abbles=5;
Cout <<abbles;
}
```

Scope :- Part of the Prog. from where a variable is accessible

When we can use it — Scope of variable

can't use → out of the scope of variable.

Program for Data Hiding :-

```
#include <iostream>
using namespace std;
class Base {
int num;          // by default Private
public:
void getData();
void showData();
};
void Base::getData()
{
cout<<"Enter any Integer value"<<endl;
cin>>num;
}
void Base::showData()
{
cout<<"The value is "<<num<<endl;
}
int main() {
Base obj;
obj.getData();
obj.showData();
return 0;
}
```

O/P

Enter any Integer value

2

The value is 2

Scope Resolution
↳ Tells scope of which class or fun/var we are used.

get - Class Name

put - Class Name

show -

```
Encapsulation obj;
obj.set(5);
cout << obj.get();
return 0;
}
```

O/P - 5

get() & set() → which represent inside the class. They are bound together which is nothing but encapsulation.

Data hiding :-

Data hiding is a technique of hidding internal objects details i.e data members.

Data hiding ensures, or we can say guarantees to restrict the data access to class members. It maintains data integrity.

Data hidding means hidding the internal data within the class to prevent its direct access from outside the class.

Data hidding also helps to reduce the system complexity to increase the robustness by limiting the interdependencies b/w software components.

Data hidding is achieved by using the private access specifier.

Eg. In this example there is a class with a variable and 2 fun. Here the variable num is private so it can be accessed only by the member of the same class & it can't be accessed anywhere else. Hence it is unable to access the variable outside the class which is data hidding

Scope of variable in c++:-

```
int main() {
int apples = 5;
Print c);
}
```

Not working
b/c it is limited
to only main fun.

```
void Print() {
int q = 50;
Cout << apples;
}
```
only limited to print fun.

```
int main() {
int P = 5;
float P = 5.7;
Cout << P;
```

→ (In same scope, 2 variables are present.)
→ At the same time if scope is diff then it will be allowed.

gives an error
same scope so
it not allowed.

Static Member function in c++:-

The static keyword is used with a variable to make the memory of the variable static once a static variable is declared its memory can't be changed.

Static members of a class are not associated with the object of the class. Just like a static variable once declared is allocated with which they can't be changed every object points to the same memory

```
class Person {
static int index_number; // It will be treated as same
                         for all the object associate
                         with the class
};
```

C++ Prog. to demonstrate Static member in a Class.

```
#include <iostream>
using namespace std;
class Student {
public:        //Static member
Static int total;
Student () { total += 1; }  //Constructor called
  };
int Student::total=0;
int main()
{ Student S1;      //Student 1 declared
  Cout<<" No. of students:"<< S1.total<<endl;
  Student S2; //Student 2 declared
 Cout <<" No. of students: "<<S2.total <2 endl;
  Student S3;      //Student S3 declard
  Cout << " No of students: " << S3. total<<endl;
  Return 0;
 }
```

O/P No of students: 1
: 2
: 3

Properties of Static Member function:-

The reason we need static member functions
1.) Static members are frequently used to store info that is shared by all objects in a class.
2) for instance, you may keep track of the quantity of newly generated objects of a specific class type using a static data member as a counter.
The static member can be increased each time an object is generated to keep track of the overall no. of objects

Program to show the working of Static member function

```
# include <iostream>
using namespace std;
Class Box
{
  Private:
    Static int length;
    Static int breadth;
```

```
Static int height;
public;
Static void print()
 {
Cout<<" The value of the length is : "<<length<<endl;
 Cout<<" the value of the breadth is : "<<breadth<<endl;
 Cout<<" The value of the height is : "<<height<<endl;
   }
  };

int Box :: length=10;
int Box :: breadth =20;
int Box :: height =30;
```

} Initialize the Static member function.

```
int main()
 {
 Box b;
Cout<<" Static member function is called through object
      name : \n"<<endl;
 b.print();
Cout<<"\n Static member function is called thro Class
      name : \n"<<endl;
 Box::print();
  return 0;
 }
```

O/P Static — through object name:

```
The value of length is : 10
                  b    : 20
                  h    : 30
```

Static mem — through Class name:

```
The value of the length : 10
                        : 20
                        : 30
```

Need to define object:
When a class is defined only the specification for the object is defined; no memory or storage is allocated
To use the data and access functions defined in the class we need to create objects.

The main use of object is to create an instance of a class.
Objects can represent a class in an independent form.
The basic blue print that contains the info. of the type of data that can be stored in an object is given by the class

Can we use class without Object?-
In the class methods you are not accessing any class properties and the methods are not virtual. So you can call those methods without creating objects.
But in case if you are accessing class properties inside the method & if the methods are virtual then first we need to create the object before calling the methods.

ClassName ObjectName [no of objects];

Array of objects in c++:-
Array is a collection of similar data items stored in a contiguous memory locations and elements can be accessed randomly using indices of an array.

→ When a class is defined only the specification for the object is defined, no
ClassName ObjectName [number of objects];
The array of object stores objects.
An array of a class type is also known as an array of objects

Program.

```
#include <iostream>
using namespace std;
 class Employee
 {
 int id;
 char name [30];
  Public:
 void getdata ();
 };
 void Employee :: getdata.
   {cout << "Enter Id : ";
   cin >> id;
    cout << "Enter Name ; ";
    cin >>name
     }
    void Employee
```

Constructor in C++ :-

Constructor is a special method that is invoked automatically at the time of object creation.

It is used to initialize the datamembers of new objects generally.

Constructor has the same name as the class or structure

Constructor is a member fun of a class whose name is same as the class name.

Constructor do not return value hence they do not have a return type.

```
<class-name> ( List of Parameters );
```

Constructor can be defined inside the class declaration or outside the class declaration.

a) Syntax for defining the constructor within the class.

```
< class-name > (List of Parameters)
    {
        // Constructor definition
    }
```

b) Syntax for defining the constructor outside the class ?

```
< class-name > : : <class-name > list of Parameter)
    {
        // Constructor definition
    }
```

Characteristics of Constructor:-

The name of the Constructor is same as its class name.
Constructors are mostly declared in the public section of the class though it can be declared in the Private section of the class.
Constructors do not return values, hence they do not have a return type.
A Constructor gets called automatically when we create the object of the class.
Constructor can be overloaded.
Constructor can not be declared virtual.

Types of Constructor:-

Default Constructor
Parameterized Constructor
Copy Constructor

Constructor do not have a return value hence they do not have a return type.

Inside

<class_name> (list_of_Parameters) { // Constructor definition }

Outside

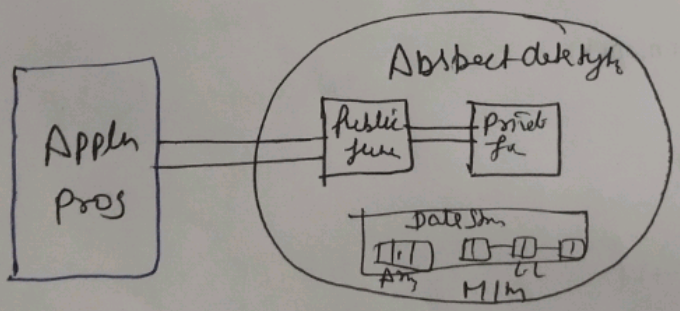
<class_name> :: <class_name> (list of Parameters) { // Code }

Data Abstraction:-

Ex- AC

Abstract Data type is a type (or class) for objects whose behaviour is defined by a set of values & a set of operations.

ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized ~~ADT is~~ in memory & what algorithms will be used for implementing the operations. It is called abstract because it gives an implementation-independent view



The user of data type does not need to know how that data type is implemented for example, we have been using primitive values like int float char data types. Only with the knowledge that these data types can operate and be performed on without any idea of how they are implemented.

Destructors:- It is used to destroy object that has been created by a constructor. Like a constructor, the destructor is a member function whose name is same as the class name but is preceded by a tilde.

```
~integer () { }
```

A destructor never takes any argument nor does it return any value.

It will invoked implicitly by the compiler upon exit from the Prog. to clean up storage that is no longer accessible.

It release mem space for future use.

```
matrix ::matrix ()
{
  for (int i=0; i<d1; i++)
  delete P[i];
  delete P;
}
```

Implementation of Destructors.

```
#include <iostream>
using namespace std;
class test int count=0;
class test
int *a
{
public:
  test()  // Constructor create
  {
    count ++i;
    Cout << "\n \n Constructor Msg : Object number " << Count
      " Created ... ";
  }
```

The class declaration describes the type and scope of its members.

The class function definitions describe how the class functions are implemented.

Class specifies :- that what follows is an abstract data of type class_name.

The class body contains the declaration of variables & functions. These fxns & variables collectively called class members.

They are grouped under 2 section ⇒ Private, Public

The keywords Private & Public are known as visibility labels.

Private → class member can be accessed only from within the class.

Public :- Can access outside of the class.

The use of keyword Private is optional. By default the member of a class are private.

```
class item
{ int number;          // Variable declaration
  float cost;             Private by default
  Public :
  void getdata (int a, float b);   // function declaration
    Void putdata (void);           using Prototype
};                                 end with
```

Object:- An object is a single unit having both data & the Processes that operates on that data.

In C++, the data and functions are bundled together as a self contained unit called an object.

An object is an entity which has some properties and behaviour associated with it.

Objects are the basic run time entities in an object oriented system.

The main Purpose of using Objects —

1. They correspond to the real life entities.
2) They Provide interactions with the real World.
3) They Provide practical approach for the implementation of the solution.

All the objects have a state, behaviour and identity

1) **State of an object:**

The state or attributes are the built in characteristics or properties of an object.

For ex:- A T.V has the size, colour, Model etc

2) Behaviour of the object:-

The behaviour or operations of an object are its predefined function.

For ex:- A TV can show Picture, Change channels turn on for a channel etc.

In OOPS terminology the behaviour is implemented through methods.

Objects An ...

3) Object Identity :- Each object is uniquely identifiable for ex the fridge can not become the T.V.

Object

Data Members
They establish the state or attributes for the object.

Member function.
M/m fun. represents the code to manipulate the data. The behaviour of the obj is determined by the member fun.

```
class triangle
{
 private:
 float side1;
 float side2;
 float side3;
 float area;
 Public:
 void read_data();
 void area_triangle();
  void display();
```

Object Declared :- Triangle t; (class → object)

No of object can also be declared as :-

Triangle t1, t2, t3;

C++ garbage collection:-

The garbage collector automatically de-allocates the memory not being used in the Program.
De-allocation is done automatically through its automatic memory management method known as garbage collection.
Garbage collection is a memory management technique.
It is a separate automatic mem management method which is used in prog. lang.

Dynamic memory Allocation:-

Real life ex: when a list of employees is created as the new employees are hired in the organization & similarly reduce when a person leaves the organisation.
This is called managing the memory.

Reserving or providing space to a variable is called memory allocation. For storing the data memory allocation can be done in 2 ways.

1. Static allocation or compile time allocation.
2. Dynamic allocation or run time allocation.

1. Static allocation or compile time allocation:-

Static memory allocation means providing space for the variable.
The size & data type of the variable is known and it remains constants throughout the Prog.

2. Dynamic Allocation or run time allocation:-

The allocation in which memory is allocated dynamically

In this type of allocation, the exact size of the variable is not known in Advance.
Pointers play a major role in dynamic memory allocation.

Why dynamic m/m allocation :-

Dynamically we can allocate storage while the Prog is in running state but variables can not be created on the fly.

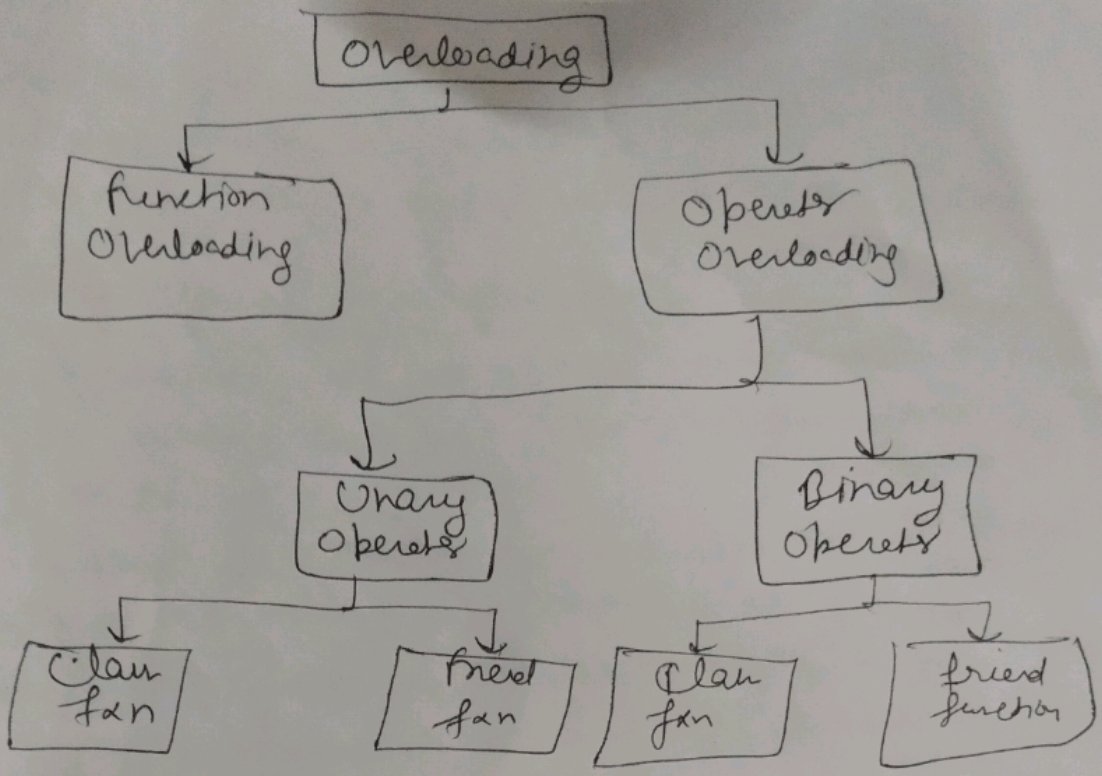
Runtime allocation or dynamic allocation of memory:
where the memory is allocated at runtime, and the allocation of memory space is done dynamically within the Prog. run.
The m/m segment is known as a heap or the free store.

Sta All variables declared inside the function will take up memory from the stack.

This is unused memory of the Prog. and can be used to allocate the memory dynamically when Prog. runs.

You can allocate memory at run time within the heap for the variable of a given type using a special Operator in C++, which returns the add. of the space allocated. This Operator is called new Operator.

If you are not in need of dynamically allocated memory anymore, you can use delete Operator which deallocates memory that was previously allocated by new operator



1. Unary operator overloading :-

A operator which contain only one operand is called unary operator overloading.

Synted

Class function :-

```
return type operator op()
{
    body;
}
```

keyword

op() ← arithmetic opn

outside the class :-

return type Operator op();

② friend function :-

friend return type operator op (arg list);

keyword arth opn

In friend fun we need to pass the arg. list.

② Binary operator overloading :-

A operator which contain two operands is called Binary operator overloading.

Syntax :-

Class function — friend function

return type operator op (Arg list) // only single Argument

{ body;
}

friend

friend return type operator op (Arg list);
// we can pass 2 Arguments

Prog: Unary operator :-

```
#include<iostream.h>
#include<conio.h>
class demo  // class
{ int a,b;  // Private data mem
  public:
  demo (int x, int y) // Constructor (arg.
  {
    a=x;
    b=y;   // x & y r value store in a & b
  }
  void show()
  {
  cout<<"A="<<a<<"  "<<"B="<<b<<endl;   // print value
  }
  void operator -()  // operator fun
  {                  operator
    a=-a;
    b=-b;   // Positive value itself gives Negative value
  }
```