① OOP's concepts

- The major purpose of C++ programming is to introduce the concept of object orientation to the C programming language.

- Object oriented Programming is a paradigm that supports many concepts like inheritance, data binding, Polymorphism, etc.

- The Programming paradigm where everything represented as an object is known as truly object-oriented Programming language.

- Smalltalk is the first truly object-oriented Programming language.

② Object, Class, Inheritance, Polymorphism, Abstraction, Encapsulation

- Object :- Any entity that has state & behaviour.

- Class :- Collection of objects. It is a logical entity. Class instances must be created in order to access and use the user-defined data types, data members & member functions.

__/__/____

- Inheritance :- when one object acquire all of the properties & behaviours of parent object i.e known as inheritance.
  It is used to achieve run-time Polymorphism.

  ① Sub-class - subclass or derived class refers to class that receive properties from another class.

  ② Superclass - (Base class) or super class refers to class from which a subclass inherits its properties.

- Polymorphism :- when one task is performed by different ways i.e known as Polymorphism.

Abstraction :- Hiding internal details and showing functionality is known as abstraction.

Data abstraction is the process of exposing to the outside world only of the information that is absolutely necessary while concealing implementation.

In C++, we use abstract class & ~~term~~ interface to achieve abstraction.

- Encapsulation :-

Binding (or wrapping) code & data together into single unit known as 'encapsulation'.

Encapsulation is the process of tying together data and the function that work with it in object oriented Programming.

③ Advantages of OOPs over Procedure - oriented Progg.

- OOPs makes development & maintences easier, where Procedure - oriented Programming language it is not easy to manage if code grows as project grows.

- OOPs Provide data hiding whereas in Procedure oriented Programming language a global data can be accessed from anywhere.

- OOP Provide ability to simulate real world event much more effectively as compared to Procedure Programming language.

④ Why do we need OOPs in C++?

Issues :- ① Previous method could couldn't address real world issue.
② No code reusability.
③ No data hiding.

__/__/____

Solutions :- ① With the use of class & object, code maintance is simple.
② Inheritance allows code reusability.
③ Data hiding is provided by encapsulation & abstraction.

⑤ Why is C++ a partial OOP?

- The main function must always be outside the class, this mean that we may do without classes & objects.

- Global variables that can be accessed by any other object within the program, encapsulation is broken here.

⑦ C++ Object

- In C++, Object is a real world entity.
- In other words object is an entity that has state & behaviour.
data ↳ ↳ functionality

- It is a runtime entity

- All the member of class can be accessed through object.

OOPS.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
class Student{
public:
  int R_No;
  string Name;

  void display(){
    cout<<"Name of the Student is "<<this->Name<<endl;
    cout<<"Roll of the "<<this->Name<<" is "<<this->R_No<<endl;
  }
};
int main(){
  Student Bhushan;
  Bhushan.R_No = 77 , Bhushan.Name = "Bhushan";
  Bhushan.display();
  return 0;
}
```

```
Name of the Student is Bhushan
Roll of the Bhushan is 77
```

Line 19, Column 1

__/__/____

(8) C++ Constructor

- In C++, constructor is a special method which is invoked automatically at the time of object creation.
- It is used to initialize the data members of new object.
- Constructor lack a return type since they don't have a return value.

- There can be 2 types of constructor in cpp
  - Default
  - Parameterized

8.1 Default Constructor

- A constructor which has no argument is known as default constructor.
- It is invoked at the time of object creation.

OOPS.cpp

```
#include<bits/stdc++.h>
using namespace std;
class Student{
public:
  int R_No;
  string Name;

  Student(){
    cout<<"Default Constructor Invoked Automatically"<<endl;
  }

  void display(){
    cout<<"Name of the Student is "<<this->Name<<endl;
    cout<<"Roll of the "<<this->Name<<" is "<<this->R_No<<endl;
  }
};
int main(){
  Student Bhushan;
  return 0;
}

```

```
Default Constructor Invoked Automatically
```

Line 19, Column 12

8.2 Parameterized Constructor

- A constructor which has Parameter is called Parameterized constructor

OOPS.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
class Student{
public:
  int R_No;
  string Name;

  Student(int r_no,string name){
    cout<<"Paremetrized Constructor Invoked "<<endl;
    this->R_No=r_no,this->Name=name;
  }

  void display(){
    cout<<"Name of the Student is "<<this->Name<<endl;
    cout<<"Roll of the "<<this->Name<<" is "<<this->R_No<<endl;
  }
};
int main(){
  Student Bhushan=Student(77,"Bhushan");
  Bhushan.display();
  return 0;
}

```

```
Paremetrized Constructor Invoked
Name of the Student is Bhushan
Roll of the Bhushan is 77
```

Line 19, Column 36

__/__/____

Characteristic of a Constructor ?

① Constructor has the same name as the class it belong to.
② Constructor can be declared anywhere in the Private as well as Public sections.
③ Because don't return values, they lack a return type.
④ Declaring a constructor virtual is not Permitted.
⑤ One can't Inherit Constructor.

8.3 Copy Constructor

– A copy constructor is an overloaded constructor used to declare & Initialize an object from another object.

OOPS.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
class Student{
public:
  int R_No;
  string Name;

  Student(int r_no,string name){
    cout<<"Paremetrized Constructor Invoked "<<endl;
    this->R_No=r_no,this->Name=name;
  }

  Student(Student &s){
    cout<<"Copy Constructor Invoked "<<endl;
    this->R_No=s.R_No,this->Name=s.Name;
  }
  void display(){
    cout<<"Name of the Student is "<<this->Name<<endl;
    cout<<"Roll of the "<<this->Name<<" is "<<this->R_No<<endl;
  }
};
int main(){
  Student Bhushan=Student(77,"Bhushan");
  Student newCopy=Student(Bhushan);
  newCopy.display();
  return 0;
}
```

```
Paremetrized Constructor Invoked
Copy Constructor Invoked
Name of the Student is Bhushan
Roll of the Bhushan is 77
```

Line 25, Column 21

- Two types of copies are produced by the Constructor:

Shallow Copy, Deep Copy

Shallow Copy

- the default Copy Constructor can only produce the shallow copy.
- A shallow copy is defined as the process of creating the copy of an object by copying data of all the member variables as it is.

OOPS.cpp

```
*/
class demo{
public:
  int x , y , *point;
  demo(){
    point=new int;
  }
  void setter(int newx , int newy ,int pval){
    this->x=newx , this->y=newy , *point=pval;
  }
  void display(){
    cout<<this->x<<" "<<this->y<<" "<<this->point<<" "<<(*point)<<endl;
  }
};
int main(){
  demo d;
  d.setter(1,2,3);
  cout<<"Base values :"<<endl;
  d.display();
  demo d1=d;
  cout<<"Derived values :"<<endl;
  d1.display();
  return 0;
}
/*
  Base values :
  1 2 0x6000020a0020 3  {shared the same memory}
  Derived values :
  1 2 0x6000020a0020 3  {shared the same memory} called as shallow copy this can
                        replace by the writing [user - defined copy constructor]
*/
```

```
Base values :
1 2 0x600003cac020 3
Derived values :
1 2 0x600003cac020 3
```

Line 34, Column 81; Build finished

__/__/____

Deep Copy

- Deep Copy dynamically allocates the memory for the copy & then copies actual values.
- both the source & copy have distinct memory location.

OOPS.cpp

```
    demo(){
      point=new int;
    }
    demo(demo &d){
      this->x=d.x , this->y=d.y ;
      point=new int;
      *point=*(d.point);
    }
    void setter(int newx , int newy ,int pval){
      this->x=newx , this->y=newy , *point=pval;
    }
    void display(){
      cout<<this->x<<" "<<this->y<<" "<<this->point<<" "<<(*point)<<endl;
    }
};
int main(){
    demo d;
    d.setter(1,2,3);
    cout<<"Base values :"<<endl;
    d.display();
    demo d1=d;
    cout<<"Derived values :"<<endl;
    d1.display();
    return 0;
}
/*
    Base values :
    1 2 0x600003c14020 3  {shared the Different memory}
    Derived values :
    1 2 0x600003c14030 3  {shared the Different memory} called as Deep copy
*/
```

```
Base values :
1 2 0x600003c14020 3
Derived values :
1 2 0x600003c14030 3
```

Line 38, Column 75

⑨ C++ Copy Constructor vs Assignment Operator

Copy Constructor	Assignment Operator
It is an overloaded constructor.	It is a bitwise operator.
It initializes the new object with the existing object.	It assigns the value of one object to another object.

Syntax of copy constructor:
Class_name(const class_name &object_name)
{
// body of the constructor.
}
	Syntax of Assignment operator:
	Class_name a,b;
	b = a;

◦ The **copy constructor** is invoked when the new object is initialized with the existing object. ◦ The object is passed as an argument to the function. ◦ It returns the object.	The **assignment operator** is invoked when we assign the existing object to a new object.
Both the existing object and new object shares the different memory locations.	Both the existing object and new object shares the same memory location.
If a programmer does not define the copy constructor, the compiler will automatically generate the implicit default copy constructor.	If we do not overload the "=" operator, the bitwise copy will occur.

(10) C++ Destructor

- A Destructor is used to destructs the object of classes.
- It can be defined only once in the class.
- Prefixed with (~) tilde sign.
- C++ destructor cannot have Parameters.

OOPS.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
class Student{
public:
  int R_No;
  string Name;

  Student(){
    cout<<"Default Constructor Invoked "<<endl;
  }

  ~Student(){
    cout<<"Destructor Invoked "<<endl;
  }

};
int main(){
  Student Bhushan;
  Student Sanket;
  return 0;
}

```

```
Default Constructor Invoked
Default Constructor Invoked
Destructor Invoked
Destructor Invoked
```

Line 13, Column 12

(11) this-> keyword

- In C++ Programming this is a keyword that refers to the current instances of the class.

- It can be used to pass current object as a parameter to another method.
- It can be used to refer current class instance variable.
- It can be used to declare indexers.

OOPS.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
class Student{
public:
  int R_No;
  string Name;

  Student(int r_no,string name){
    cout<<"Parametrized Constructor Invoked "<<endl;
    cout<<"Using this-> keyword"<<endl;
    this->R_No=r_no,this->Name=name;
  }

  void display(){
    cout<<"Name of the Student is "<<this->Name<<endl;
    cout<<"Roll of the "<<this->Name<<" is "<<this->R_No<<endl;
  }
};
int main(){
  Student Bhushan=Student(77,"Bhushan") ;
  Bhushan.display();
  return 0;
}
```

```
Parametrized Constructor Invoked
Using this-> keyword
Name of the Student is Bhushan
Roll of the Bhushan is 77
```

Line 10, Column 40

(12) C++ Static

- In C++, Static is a keyword or modifier that belongs to the type not instance.

- Instances is not required to access a static members

- In C++, Static can be field, method, constructor, class, properties, operator and event.

Advantage

- Memory-efficient: Now we don't need to create instance for accessing the static members.

- It belongs to the type, so it will not get memory each time when instance is created.

OOPS.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
class Student{
public:
  int R_No;
  string Name;
  static string College;
  Student(int r_no,string name){
    this->R_No=r_no,this->Name=name;
  }
  void display(){
    cout<<"Name of the Student is "<<this->Name<<endl;
    cout<<"Roll of the "<<this->Name<<" is "<<this->R_No<<endl;
    cout<<"College of the "<<this->Name<<" is "<<College<<endl<<endl;
  }
};
string Student::College="PICT";
int main(){
  Student Bhushan=Student(77,"Bhushan") ;
  Student Om=Student(33,"Om");
  Student SJ=Student(43,"SJ");
  Bhushan.display();
  Om.display();
  SJ.display();
  return 0;
}
```

```
Name of the Student is Bhushan
Roll of the Bhushan is 77
College of the Bhushan is PICT

Name of the Student is Om
Roll of the Om is 33
College of the Om is PICT

Name of the Student is SJ
Roll of the SJ is 43
College of the SJ is PICT
```

Line 7, Column 25

OOPS.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
class Student{
public:
  int R_No;
  string Name;
  static string College;
  static int studentCounts;

  Student(int r_no,string name){
    this->R_No=r_no,this->Name=name;
    studentCounts+=1;
  }

  void display(){
    cout<<"Name of the Student is "<<this->Name<<endl;
    cout<<"Roll of the "<<this->Name<<" is "<<this->R_No<<endl;
    cout<<"College of the "<<this->Name<<" is "<<College<<endl<<endl;
  }
};
string Student::College="PICT";
int Student::studentCounts=0;
int main(){
  Student Bhushan=Student(77,"Bhushan") ;
  Student Om=Student(33,"Om");
  Student SJ=Student(43,"SJ");
  cout<<"Total Objects are "<<Student::studentCounts<<endl;
  return 0;
}
```

```
Total Objects are 3
```

Line 27, Column 53

(13) C++ Structs

- In C++, classes and Struct are blueprint that are used to create the instances of class (Struct are used for lightweight objects)

- Structs in c++ are value type than reference type,

OOPS.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
struct Student{
public:
  int age;
  string name;
  Student(string nname,int aage){
    this->age=aage , this->name=nname;
  }
};
int main(){
  struct Student Agraj("Agraj",23);
  struct Student Anuj("Anuj",20);
  return 0;
}
```

```
[Finished in 1.7s]
```

Line 15, Column 2

Structure	Class
If access specifier is not declared explicitly, then by default access specifier will be public.	If access specifier is not declared explicitly, then by default access specifier will be private.

Syntax of Structure:
 	
struct structure_name
{
// body of the structure.
}
	Syntax of Class:
	
	class class_name
	{
	// body of the class.
	}

The instance of the structure is known as "Structure variable".	The instance of the class is known as "Object of the class".

(14) C++ Enumeration

- Enum in C++ is a data type that contains fixed set of constants

- enum improves type safety,
- enum can be easily used in switch,
- enum can have fields, constructors and methods
- enum may implement many interfaces but can't extend any class because internally extends enum class.

OOPS.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
class Student{
public:
  enum year {First,Second,Third,Final};
private:
  int R_No;
  string Name;
  year yearOfStudy;
public:
  Student(int r_R_No,string n_Name,year y_yearOfStudy){
    this->R_No=r_R_No ,this->Name=n_Name,this->yearOfStudy=y_yearOfStudy;
  }

  void display(){
    cout<<this->R_No<<" "<<this->Name<<" "<<getStudyYearAsString()<<endl;
  }
private:
  string getStudyYearAsString(){
    switch (yearOfStudy){
      case First: return "First Year";
      case Second: return "Second Year";
      case Third: return "Third Year";
      case Final: return "Final Year";
      default: return "Unknown";
    }
  }
};
int main(){
  Student Bhushan(77,"Bhushan",Student::Final);
  Student ABC(23,"ABC",Student::First);
  Bhushan.display();
  ABC.display();
  return 0;
}
```

```
77 Bhushan Final Year
23 ABC First Year
```

Line 27, Column 4

(15) C++ friend function

- If a function is defined as friend function in c++, then the protected & private data of class can be accessed using the function.

Characteristics

1. The function is not in 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 and has to use an object name and dot membership operator with the member name.

OOPS.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
class Student{
  int R_No;
  string Name;
public:
  Student(int r_R_No,string n_Name){
    this->R_No=r_R_No,this->Name=n_Name;
  }

  friend void display(Student currobject);

  void tryToCallFriendFunction() {
    display(*this);
  }
};
/* The function is not in the scope of the class to which it has been declared as a friend.
   So ,i.e it is not possible => Bhushan.display();
*/
void display(Student obj){
  cout<<obj.R_No<<" "<<obj.Name<<endl;
}
int main(){
  Student Bhushan(77,"Bhushan");
  Student ABC(23,"ABC");
  Bhushan.tryToCallFriendFunction();
  display(Bhushan);
  display(ABC);
  return 0;
}
```

```
77 Bhushan
77 Bhushan
23 ABC
```

Line 26, Column 37

OOPS.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
class B;// forward declaration
class A{
  int value1;
public:
  A(int val){
    this->value1=val;
  }
  friend void min(A,B);
};
class B{
  int value2;
public:
  B(int val){
    this->value2=val;
  }
  friend void min(A,B);
};
void min(A a,B b){
  cout<<"Minimum value is : ";
  if(a.value1 <= b.value2) cout<<a.value1<<endl;
  else cout<<b.value2<<endl;
}
int main(){
  A a(87);
  B b(78);
  min(a,b);
  return 0;
}
/*
In the above example, min() function is friendly to two classes,
i.e., the min() function can access the private members of both the classes A and B.
*/

```

```
Minimum value is : 78
```

Line 25, Column 12

(17) Inheritance

- In C++, Inheritance is a process in which one object acquires all the properties and behaviours of its parent object automatically.

- In such way, you can reuse, extend, modify, the attributes and behaviours which are defined in other class.

- Main Advantage → Code Reusability

Types of Inheritance
1. Single
2. Multiple
3. Hierarchical
4. Multilevel
5. Hybrid

Base class visibility	Derived class visibility		
	Public	Private	Protected
Private	X	X	X
Protected	Protected	Private	Protected
Public	Public	Private	Protected

Private member is not inheritable if we modify the visibility mode by making it public but this take away the advantage of data Hiding. ie C++ introduce protected

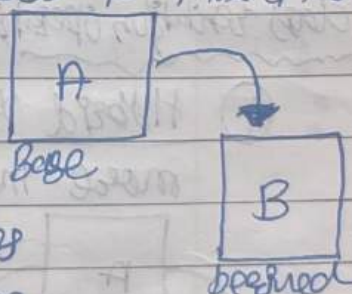
Structure → class derivedName :: visibility mode baseName
{ ... }

- when the base class is privately inherited by the derived class, public members of the base class become private members of the derived class.
i.e public members of the base class are not accessible by the object of the derived class only by the member functions of derived class.

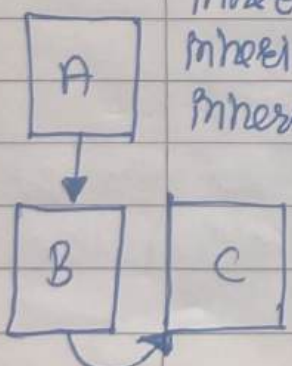
- when the base class is publicly inherited by the derived class, public members of the base class also become the public member of the derived class.
ie public members of the base class are accessible by both objects as well as member functions of base class.

- In C++, the default mode visibility is private.
- The private member of base class never inherited.

① Single Inheritance :- Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.

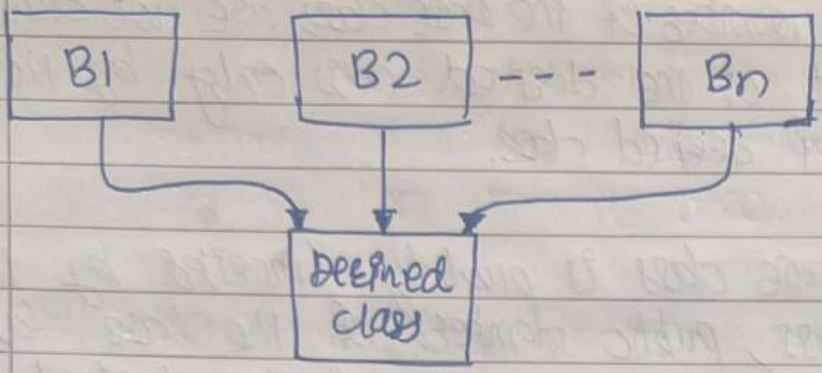


② Multilevel Inheritance :- when one class inherits another class which is further inherited by another class it known as multilevel inheritance in C++.



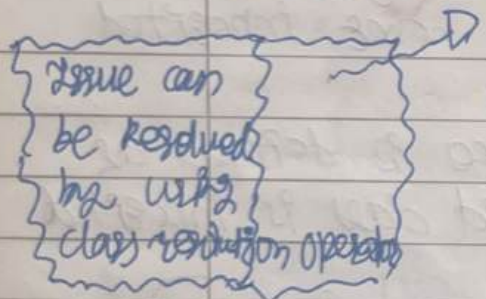
__/__/____

③ Multiple Inheritance :- Multiple Inheritance is the process of deriving a new class that inherited by 2 or more classes.



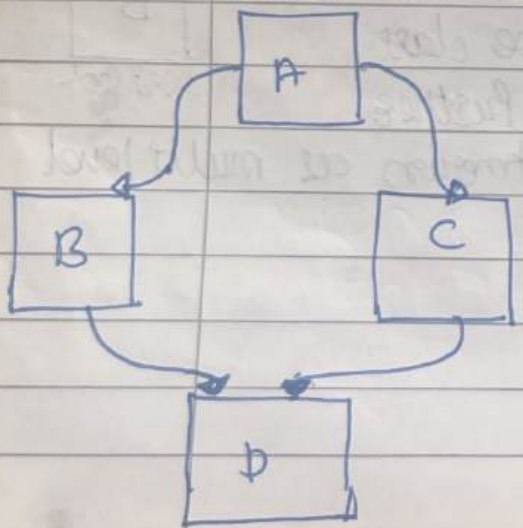
Ambiguity Resolution in Inheritance :-
Ambiguity can be occurred in using the multiple inheritance,

when a function with the same name occures in more than one base class.

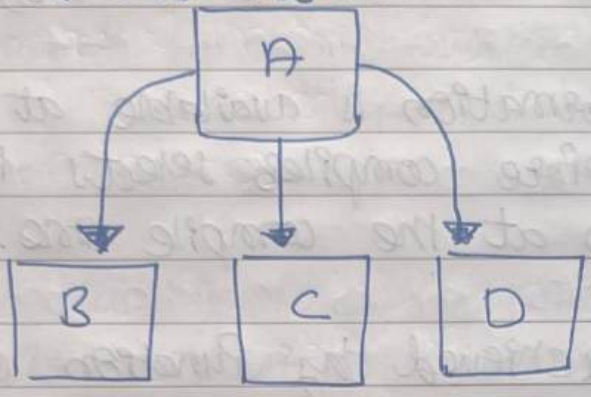


Class Resolution operator (: :)

④ Hybrid Inheritance :- Hybrid Inheritance is a comb^{n} of more than one type of inheritance.



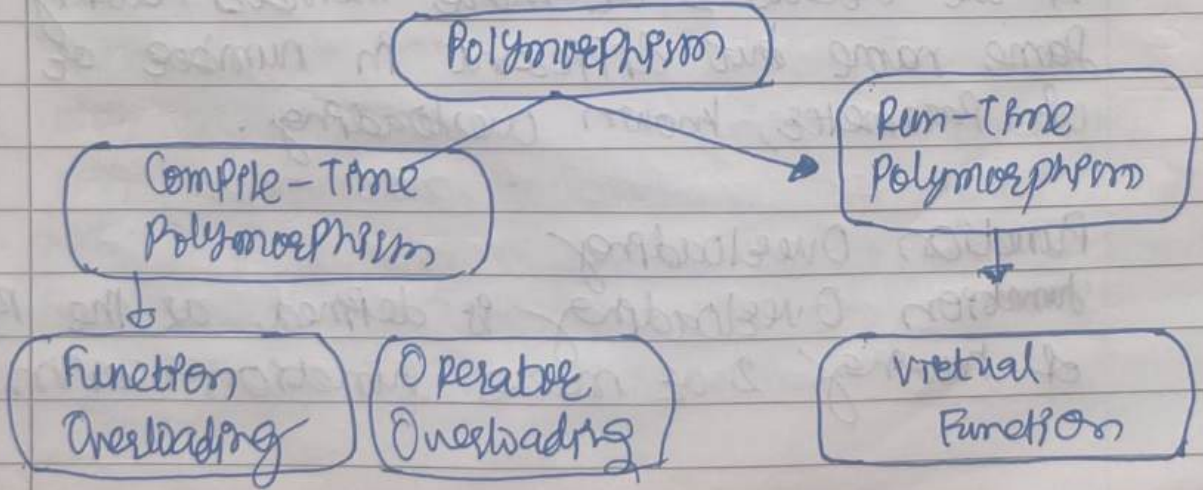
(5) Hierarchical Inheritance :- Hierarchical Inheritance is defined as the process of deriving more than one class from a base class.



C++ Aggregation (HAS-A Relationship)
In c++, aggregation is a process in which one class defines another class as any entity reference. It is another way to reuse class.

(18) Polymorphism

The term Polymorphism is the combination of "Poly" + "morphs" which means many forms. It is a greek word. In object-oriented programming, Polymorphism is main concepts.



code.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
class car{
protected:
    string model;
    int year;
    int maxspeed;
    string color;
    double fuelLevel;
public:
    void getFuelLevel(double d){
        this->fuelLevel=d;
    }
};
class lamborgini{
protected:
    string brandName="Lamborgini";
};
class urus:public car,public lamborgini{
public:
    void display(){
        cout<<"BrandName is Urus is "<<brandName<<endl;
        cout<<"Fuel Level of the Urus is "<<fuelLevel<<endl;
    }
};
int main(){
    urus u1;
    u1.getFuelLevel(100);
    u1.display();
    return 0;
}
```

```
BrandName is Urus is Lamborgini
Fuel Level of the Urus is 100
```

Line 17, Column 35

code.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
class car{
protected:
    string model;
    int year;
    int maxspeed;
    string color;
    double fuelLevel;
public:
    void display(){
        cout<<"Hey, Compiler I am in car Class"<<endl;
    }
    void getFuelLevel(double d){
        this->fuelLevel=d;
    }
};
class lamborgini{
protected:
    string brandName="Lamborgini";
public:
    void display(){
        cout<<"Hey, Compiler I am in lamborgini Class"<<endl;
    }
};
class urus:public car,public lamborgini{
public:
    void view(){
        car :: display();
        lamborgini :: display();

    }
};
int main(){
    urus u1;
    u1.view();
```

```
Hey, Compiler I am in car Class
Hey, Compiler I am in lamborgini Class
```

Line 26, Column 26

code.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
class car{
protected:
    string car_color;
public:
    void get_car(){
        cout<<"Enter Color of Car"<<endl;
        cin>>car_color;
    }
};
class lamborgini:public car{
protected:
    string brand_Name;
public:
    void get_lamborgini(){
        cout<<"Enter BrandName"<<endl;
        cin>>brand_Name;
    }
};
class urus{
protected:
    string which_Models;
public:
    void get_urus(){
        cout<<"Which Model Pearl OR Graphite"<<endl;
        cin>>which_Models;
    }
};
class urusModels:public lamborgini,public urus{
public:
    void display_urusModels(){
        get_car();
        get_lamborgini();
        get_urus();
        cout<<"Car Color is "<<car_color<<" BrandName is "<<
        brand_Name<<" and Model is "<<which_Models<<endl;
    }
};
int main(){
    urusModels u1;
    u1.display_urusModels();
    return 0;
}
```

```
Enter Color of Car
Black
Enter BrandName
Lamborgini
Which Model Pearl OR Graphite
Pearl
Car Color is Black BrandName is Lamborgini and Model is Pearl
```

Line 35, Column 20

code.cpp

```cpp
#include <iostream>
using namespace std;
class Shape{
public:
    int a;
    int b;
    void get_data(int n,int m){
        a=n;
        b=m;
    }
};
class Rectangle:public Shape{
public:
    int rect_area(){
        int result = a*b;
        return result;
    }
};
class Triangle:public Shape{
public:
    int triangle_area(){
        float result=0.5*a*b;
        return result;
    }
};
int main()
{
    Rectangle r;
    Triangle t;
    int length,breadth,base,height;
    cout<<"Enter the length and breadth of a rectangle: "<<endl;
    cin>>length>>breadth;
    r.get_data(length,breadth);
    int m = r.rect_area();
    cout<<"Area of the rectangle is : "<<m<<endl;
    cout<<"Enter the base and height of the triangle: "<<endl;
    cin>>base>>height;
    t.get_data(base,height);
    float n = t.triangle_area();
    cout <<"Area of the triangle is : "<<n<<endl;
    return 0;
}
```

```
Enter the length and breadth of a rectangle:
12 34
Area of the rectangle is : 408
Enter the base and height of the triangle:
2 3
Area of the triangle is : 3
```

OOPS.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
class car{
public:
  string model;
  int year;
  int maxspeed;
  string color;
  double fuelLevel;
};
class lamborgini:public car{
public:
  void brandName(){
    cout<<"BrandName is Lamborgini"<<endl;
  }
};
class urus:public lamborgini{
public:
  urus(string _model,int _year,int _maxspeed,string _color,double _fuelLevel){
    this->model=_model,this->year=_year,this->maxspeed=_maxspeed,
    this->color=_color,this->fuelLevel=_fuelLevel;
  }
  void display(){
    brandName();
    cout<<"Model of the Car is : "<<this->model<<endl;
    cout<<"Year of the Manufacture is : "<<this->year<<endl;
    cout<<"Maxspeed of the car is : "<<this->maxspeed<<endl;
    cout<<"Color of the car is : "<<this->color<<endl;
    cout<<"FuelLevel of the car is : "<<this->fuelLevel<<endl;
  }
};
int main(){
  urus urus1("Urus123",2008,235,"Black",9.3);
  urus1.display();
  return 0;
}
```

```
BrandName is Lamborgini
Model of the Car is : Urus123
Year of the Manufacture is : 2008
Maxspeed of the car is : 235
Color of the car is : Black
FuelLevel of the car is : 9.3
```

code.cpp

```cpp
#include <iostream>
using namespace std;
class Address {
    public:
   string addressLine, city, state;
     Address(string addressLine, string city, string state)
    {
        this->addressLine = addressLine;
        this->city = city;
        this->state = state;
    }
};
class Employee
    {
        private:
        Address* address;  //Employee HAS-A Address
        public:
        int id;
        string name;
        Employee(int id, string name, Address* address)
       {
            this->id = id;
            this->name = name;
            this->address = address;
       }
     void display()
       {
            cout<<id <<" "<<name<< " "<<
              address->addressLine<< " "<< address->city<< " "<<address->state<<endl;
       }
   };
int main(void) {
    Address a1= Address("C-146, Sec-15","Noida","UP");
    Employee e1 = Employee(101,"Nakul",&a1);
            e1.display();
   return 0;
}
```

```
101 Nakul C-146, Sec-15 Noida UP
```

Line 34, Column 31

- Compile Time Polymorphism :-
The overloaded functions are invoked by matching the type and number of arguments.

This information is available at the compile time & therefore compiler selects the appropriate function at the compile time.

It is achieved by function overloading & operator overloading which is also known as static binding or early binding.

- Run Time Polymorphism :-
Run Time Polymorphism is achieved when the objects methods is invoked at the run time instead of compile time. It is achieved by method overriding which is also known as dynamic binding or late binding.

- Overloading
If we create 2 or more members having the same name but different in number or type of parameter, known overloading.

Function Overloading
Function Overloading is defined as the process of having 2 or more function with the

Same name, but different in parameter known as function overloading.

The advantage of function overloading is that it increases the readability of the program.

Operator Overloading

Operator overloading is a compile time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined data type.

Operator that can't be overloaded,

- Scope Operator (::)
- sizeof
- member selector (.)
- member pointer selector (*)
- ternary operator (?:)

Rules for Operator Overloading

1. Existing operator can be overloaded.
2. We cannot use friend function to overload certain operators.
3. When unary operator are overloaded through member function take no explicit arguments, but if may are overloaded by friend function takes one arguments.
4. binary, takes 2 arguments.

__/__/____

Function Overriding

If derived class defines same function as defined in its base class, it is known as function overriding in C++.

It is use to achieve run-time Polymorphism.

It enables you to provide specific implementation of the function which is already provided by its base class.

C++ virtual function

- A C++ virtual function is a member function in the base class that you redefine in a derived class.
- It is used to tell compiler to perform dynamic linkage or late binding on the function.
- when the function is made virtual, C++ determine which function is to be invoked at the runtime.

Rules of virtual function

virtual function,

1. must be members of some class.
2. cannot be static members.
3. They are accessed through object Pointer.
4. They can be friend of another class.

⑤ virtual function must be defined in the base class, even though it is not used.

⑥ we can't have virtual constructor but we have virtual destructor.

✤ Pure virtual function
when the function has no definition, such function is known as "do-nothing" function, is known as Pure virtual function.

⑲ Data Abstraction
- Data Abstraction is a process of providing only the essential details to the outside world and hiding the internal details i.e representing only the essential details in the Program.

- c++ provides a great level of abstraction, e.g pow(), min(), max(), --- etc.

✤ Data abstraction can be achieve using 2 ways

- Abstraction using classes

- Abstraction in header files.

__/__/____

Abstraction using classes :-

An abstraction can be achieve using classes.
A class is used to group all the data members & member function into a single unit by using the access specifiers.
A class has the responsibility to which data is to visible outside.

Abstraction in header file :-

e.g pow(), header files hides all the implementation details from user.

Advantages of Abstraction

- A Programmer does not need to write low-level codes.
- Internal implementation can be changed without affecting the user level code.
- The main aim of data abstraction is to Reuse the code.
- Avoids Duplication.