HANDWRITTEN NOTES

OOPS in C++

Tapesh Dua

⇒ Object-Oriented Programming

Main aim of OOP is to bind the data & functions that operate on them so that no other part of the code can access this data except this function.

Class — It is user defined data types, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.

```
class   Class_name {
        ---------
};
```
→ Syntax

Object — When a class is defined no memory is allocated but when it is instanciated (ie object is created) memory is allocated.

Syntax → `Class_name obj_name;`

Example → `Myclass obj1;`

Syntax 2 → `Class_name *pointer_name = new Class_name();`

Example → `MyClass *ptr1 = new MyClass();`

→ **Encapsulation** — In OOP, it is defined as binding together the data and the functions that manipulate them.

→ Leads to data abstraction or hiding.

```
class Encap {                    // hidden data
    private: int data;
    public:                      // set value
        void set(val){
            data = val;
        }
        void get(){              // get value
            cout<< data;
        }
};
```

Setters / Mutators

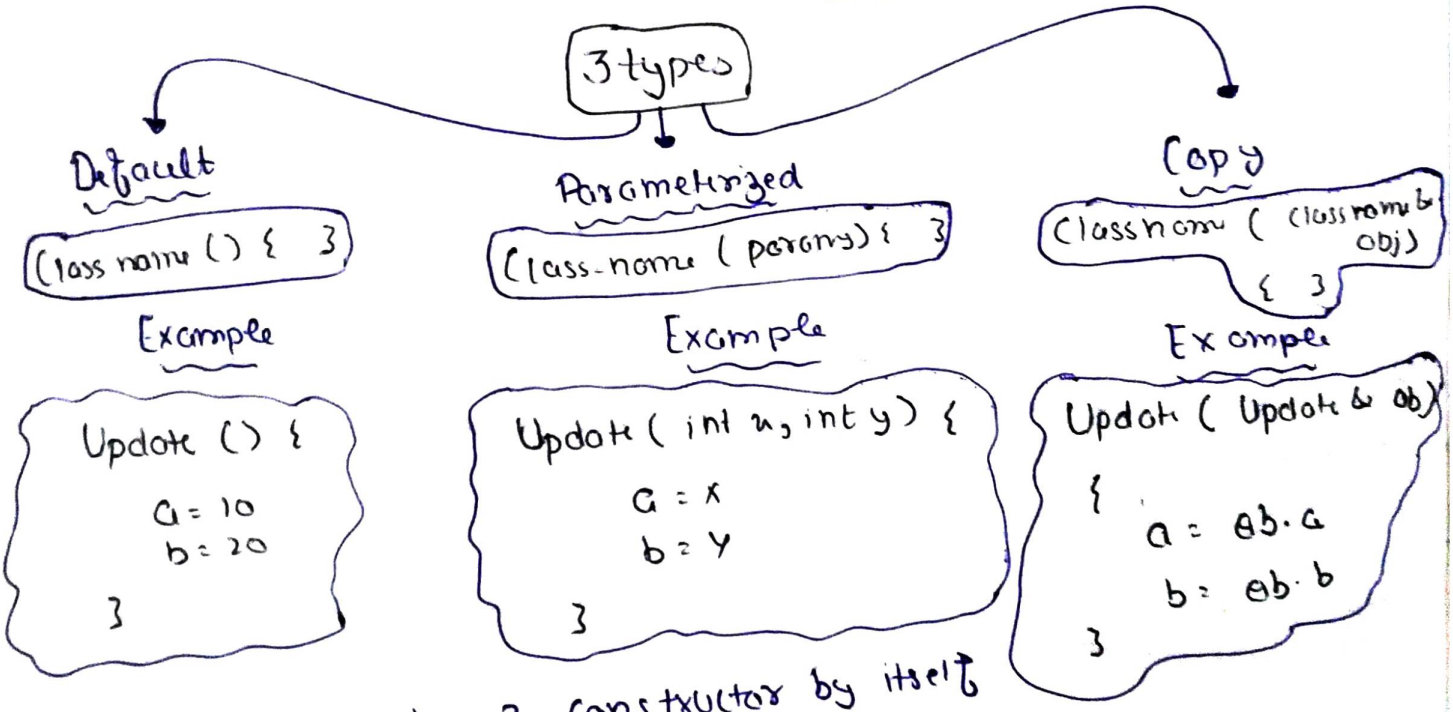
Getters / Accessors

```
Encap obj;
obj.set(2);
obj.get();
```

→ Abstraction : Abstraction means displaying only essential information and hiding the details (we don't see actual implementation)
- Abstraction using classes
- Abstraction using Header files (math h → Pow())

Advantages of data Abstraction → Avoid code duplication and increase reusability
→ Can change internal implementation of class independently.

→ Constructors : It is an special member functions of the class. It automatically invoked when an object is created.
↳ x Return type
↳ Same name as class
↳ If we do not specify, then C++ compiler generates a default constructor for us.



→ Compiler generate 2 constructor by itself
↳ Default ↳ Copy

But if any of constructor created by user, then default constructor will not be created by compiler.

→ Constructor overloading can be done just like function overloading.

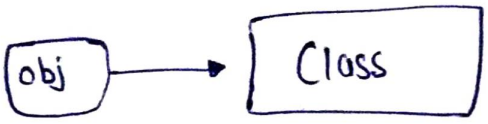
→ Copy constructor can be made private.

→ Copy Constructor must be passed by reference because if we pass value, then it would made to call copy Constructor which is non terminating.

→ Destructor It is a member function which destructs or deletes an objects.

↳ X takes args ↳ X return type ↳ Cannot Static
↳ Only one destructor is possible

→ Actually destructor doesn't destroy object, it is the last function that invoked before object destroy.



Destructor is used, so that before deletion of obj we can free space allocated for this class. Because if obj gets deleted the space allocated for obj will be free but class doesn't.

→ Access Modifier

If we do not specify any access modifier inside the class then by default member will be private.

→ Public: Can be accessed by any class
→ Private: Can be accessed by a function in a class (inaccessible outside class)
→ Protected: Inaccessible outside class but can be accessed by derived class.

→ Inheritance It is a process of inheriting properties and behaviour of existing class into a new class.

Syntax

```
Class Base { };
Class Derived : visibility-Mode Base { };
```

Example

```
Class Phone { };
Class SmartPhone : public Phone { };
```

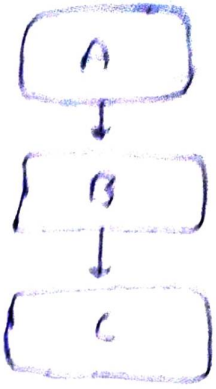
→ Types of Inheritance

a) Single Inheritance

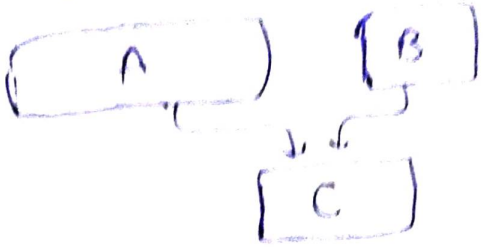

```
Class B : public A
{
};
```

b) Multilevel Inheritance

```
class B : public A
{
};
class C : public B
{
};
```



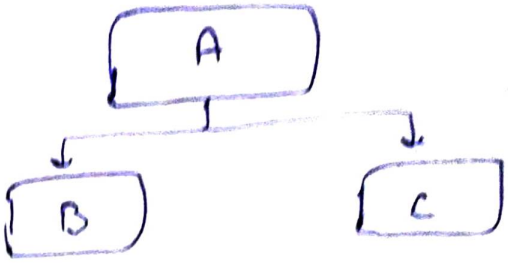
c) Multiple Inheritance




```
class A {        class B {
};               };

class C : public A, public B {
};
```

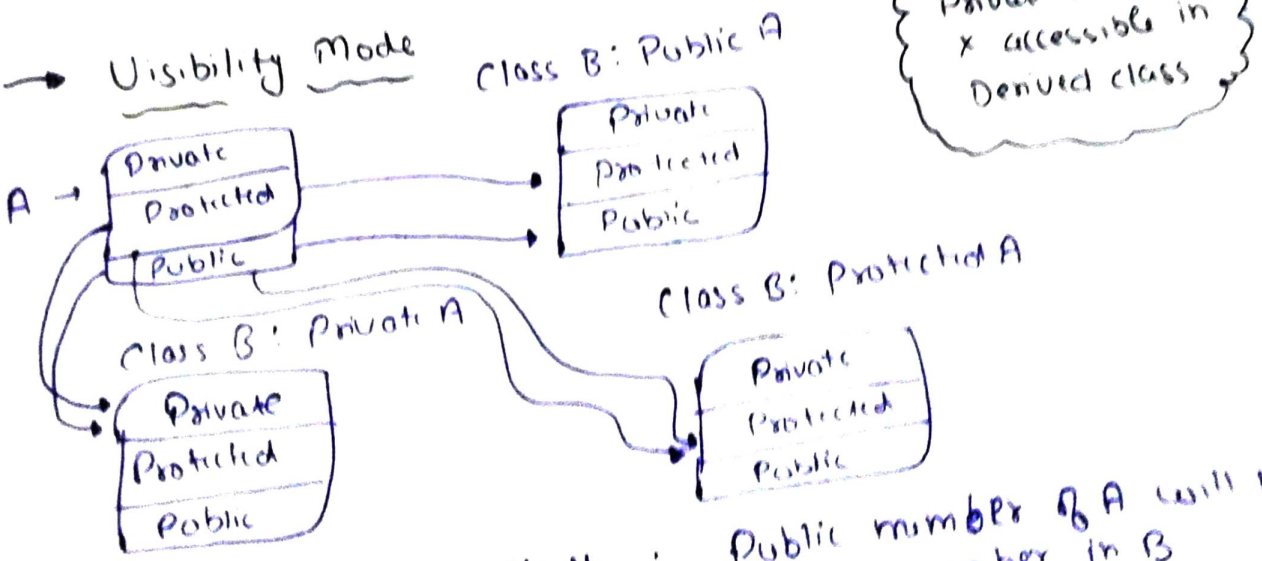
d) Heirarchial Inheritance




```
class B : public A {
};
class C : public A {
};
```

e) Hybrid (Virtual) Inheritance → Combination of more than one type of Inheritance

→ Visibility Mode



Private member x accessible in Derived class

↳ If B derived Publically : Public member of A will be Public member in B
Protected will be Protected

↳ If B derived Protectedly : Public → Protected
Protected → Protected

↳ If B derived Privately : Public → Private
Protected → Private.

Constructor and Destructor in Inheritance

Child class constructor will run during creation of object of child class, but as soon as obj is created child class constructor run and it will call constructor of it's parent class and after the execution of parent class constructor it will resume it constructor execution.

```
class child {

    child () : parent() {

    }
};
```

Parent constructor call

Constructor Evolution

Obj creation
↓
Child Constructor
↓
Parent Constructor
↓
Complete Parent
↓
Complete Child.

↳ In case of Destructor, first child destructor executes, then paren destructor executed.

→ **Polymorphism** : In simple words, we can define polymorphism as the ability of a message to be displayed more than one form.

↳ Compile time poly: → Operator overloading
→ Function overloading

↳ Run time poly: → Function overriding
↓
occurs when a derived class has a difination of one or more members of base class.

• **Function Overloading** → feature in C++ where two or more functions can have same name with diff params.

Achieved at compile time

```
void print (int i) {
    cout << i;
}

void print ( int i, int j ) {
    cout << i << j;
}

void print ( float i ) {
    cout << i;
}
```

```
print (10)
print (10, 11)
print (10.1)
```

Compile time Polymorphism.

- **Operator Overloading** — C++ have the ability to provide special meaning to the operator.

Example

```
class Complex {
    public:
        int real, imag

        Complex operator + (Complex & num) {
            Complex ans;
            ans.real = real + num.real
            ans.imag = imag + num.imag
            return ans;
        }
};

Complex num1(1,2), num2(3,4)
Complex num3 = num1 + num2
```

As + can't add complex no's directly, so we can define a function with name + with operator keyword before it

↳ So, we can use almost all operator like this.

- **Method Over Riding** (achieved at Run time)

↓

Redifinition of base class function in its derived class, with same, return type and same parameters.

Example

```
class Car {
    int gear;
    public:
        void change_gear (int g){
            gear++
    }
};

class sportsCar : public Car {
    int gear;
    public
        void change_gear (int g) {
            if (gear > 5)
                gear++
        }
};

sportCar SC;
SC.change_gear(4);
```

→ fn of sportsCar will be called

While calling change_gear() first it will check if an func with this name exist in calling class, otherwise it goes to base class.

→ **Virtual function** → Member function which is declared with a Virtual keyword in the base class and redeclared (override) in a derived class.

When you refer to a obj of derived class using pointer to base class, you can call a virtual function of that object and execute derived class's version of the function.

↳ Used to achieve Run time polymorphism

↳ virtual func can not be static and also cannot be friend func of another class.

Compile time (Early Binding)
VS
Runtime time (Late Binding)

```
class Base {
    public:
        virtual void print () {
            cout << " Base print"
        }
        void show () {
            cout << " Base show "
        }
};
class Derived {
    public:
        void print () {
            cout << " Derived Print"
        }
        void show () {
            cout << " Derived show"
        }
};
Derived d;
Base * bptr = &d;
```

bptr → print() ⟶ Runtime

bptr → show() ⟶ Compile time

output

Late Binding
↑ derived print
Base show ↓
Early Binding.

During Compile time bptr behaviour judged on the basis of which class it belong, so bptr represent base class.

If function is not virtual then it will allow binding at Compile time & print func of base class will get binded becoz bptr represent base class.

But at runtime, bptr points to obj of derived class so it will bind function

→ Pure virtual function and abstract class

- Sometimes implementation of all function cannot be provided in the base class. Such a class is called abstract class
- A pure virtual function is a virtual function for which we don't have any implementation, we only declare it.

```
class Test {
    public:
        virtual void fun() = 0;
};
```

→ Abstract Class

→ Pure virtual function.

1. A class is abstract if it has at least one pure virtual function.
2. We cannot declare object of abstract class
   Ex Test t; → show error.
3. We can have pointer or reference of abstract class.
4. We can access the other functions except virtual by object of its derived class.
5. If we don't override the pure virtual function in derived class then it becomes abstract.
6. An abstract class can have constructors.

→ Inline functions

Inline is a request not command.

It is function that is expanded in line when it is called when the inline function is called, whole code get inserted or substituted at the point of inline func call.

Syntax

```
inline return-type fun() {

}
```

→ Friend class and function

- A friend class can access private and protected members of other classes in which it is declared as a friend.

Syntax

Base class

```
class You {
    ...
    friend class Me;
};
class Me {
    ...
};
```

Me is a friend class of You. → Syntax

Friend class

- A friend function can be granted access to private & protected members of a class.
  - ↳ It can be global function or
  - ↳ A member function of another class

Syntax

Global func →

```
friend return_type func-name (args)
```

or

```
friend return-type class-name :: func_name (args
```

member func of another class.

Example

```
class Box {
    private:
        int width;
    Public:
        friend void printwidth (Box b);
};
```

```
void printwidth (Box b) {
    Cout << b.width;
}
```

↳ this pointer

↳ Every obj in C++ has access to its own address through an imp pointer called this pointer.

- Friend function doesn't have a 'this' pointer, because friends are not members of a class.
  Only member function have this pointer.

Example

```
Class Box {
    int l, b, h
    Public
        void set (int l, int b, int h) {
            this → l = l;
            this → b = b;
            this → h = h;
        }
};
```

Static Members

- Static variables in a func: When a variable is declared as static, space for it gets allocated for the life time of the program. (default initialized to 0)
  - ↳ Even if the function is called multiple times, the space for it is allocated once.
- Static variable in a class: Declared inside the class body
  - ↳ They must be defined outside the class.
  - ↳ They doesn't belong to any obj but to the whole class.
  - ↳ There will be only one copy of static member variable for the whole class.

Example

```
Class Student {
    Static int Roll-NO;   → Static Member variable
    string name;
    Public:
        Student (string s) {
            name = s
            Roll-NO ++
        }
};
int Student :: Roll-NO = 0;   → Initialized outside the class
int main () {
    Student S1 ("Tapesh")
    Cout << S1. Roll-NO;   → 1
    Student S2 ("Dua")
    Cout << S2. Roll-NO;   → 2
    Cout << Student :: Roll-NO;   → 2
```

Can access without an object

- Object can also be declared as static

Static Student S1;

→ Static function in a class → are allowed to access only the static data members or other static member functions.

→ Template in C++

template < typname T >

↓

Helps in data type, we can write generic function that can be used for different data type.

Example

```
template < typename T >
class Node {
public:
T data;
Node* next;
Node ( T data) {
    this → data = data
    next = NULL
}
};
```

Object declaration → Node<int> *head = new Node<int>(5);

→ Dynamic Constructor When allocation of memory is done dynamically using dynamic memory allocation 'new' in constructor.

```
class Topesh {
    char[] name;
    Topesh() {
        name = new char[6];
    }
};
```

Structure Vs Class

Most imp diff is security.

- Structure: ✗ secure, ✗ hide its member functions & variables
- Class: ✓ secure, ✓ its programming & designing details

Local classes in C++

A class declared inside a function becomes ~~total~~ local to that function and is called local class.

All the methods of local class must be defined inside the class only.

Function Overloading	VS	Function Overriding
• Occurs with in given class		• Occurs in two related class, super and sub class
• Diff methods have same name but diff params		• Same name & params & return type in super and sub class
• Signature of func are diff		• Signature of func should be same
• Can have diff access modifiers		• Must have same or more restrictive access modifiers.

Compile-time poly	VS	Runtime poly
• Occurs at compile time		• Occurs at run time
• Achieved through func & operator overloading		• Achieved through function overriding
• Faster execution time		• Slower execution time
• More memory efficient		• Less memory efficient