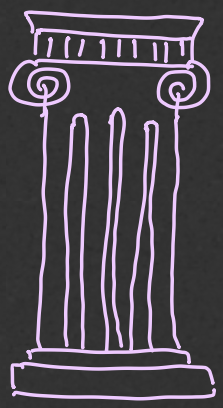
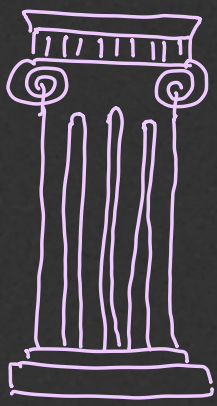
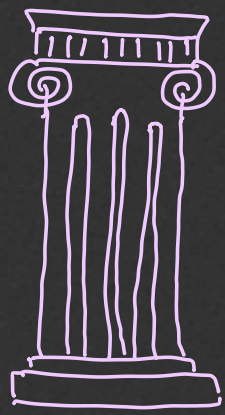
OOPs



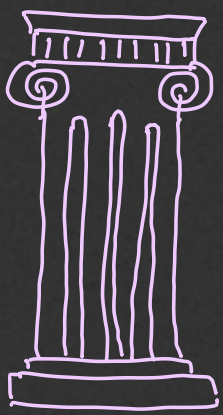
Encapsulation



Abstraction



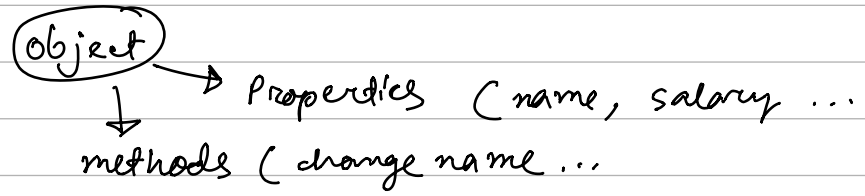
Inheritance



Polymorphism

Object Oriented Programming

Classes are blueprint for simulating real life entities called objects.

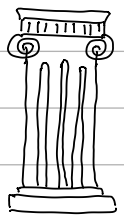


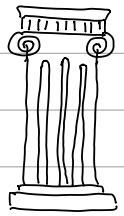
Access Modifiers

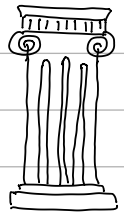
This is by default in C++

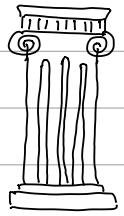
1. Private :- data & method accissible to class.
2. Public :- data & method accissible to everyone.
3. Protected :- data & method " " class & to its derived class.

4 Pillars of OOPS


Encapsulation


Abstraction


Inheritance


Polymorphism

Encapsulation :-

wrapping up of data & method with each other in a single unit.

private encapsulation == Data hiding.

* Teacher t1 ; → it calles a constructor even though specific constructor is not defined in class.

constructor :-

- ✓ analogus to a function but doesn't have a return type
- ✓ name is same as class name.
- ✓ only called once at object creation
- ✓ memory allocation happens when constructor is called.

#miscellaneous-STL

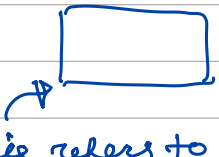
struct Teacher {
    int num;
    char c; };

Teacher t1 = { 10, 'a' };
This is valid.

multiple constructor definition is valid with difference in parameter called Constructor Overloading Type of Polymorphism.

E.g. class Teacher {
    Teacher ( ) {
        " constructor created; }
    Teacher (int salary, string name) {
        name = name
        salary = salary }

✓ this → name = nam
this → age = itsage



class Teacher object

This refers to here.

Copy constructor :-

```
Teacher T1("Bhuvia", 10);
Teacher T2(T1);
```

↳ T2 is created with values of T1

```
Teacher (Teacher & original){
    this->name = original.name
    ⋮
}
```

Deep Copy vs Shallow Copy :-

```
class Teacher {
    int a;
    int * b;
}
```

→ c++ compiler does shallow copy which is not making new memory for copied b uses original one's b's addr.

need to write deep copy constructor by yourself.

Destructor :-

automatically called when program is finished or object gets out of scope

e.g.
```
~Teacher(){
    delete ptrSalary;
}
```

```
class Teacher {
    int age;
    Teacher() ....
    ⋮
    ~Teacher(){
        cout << Deleted }};
```

data from heap is not deleted automatically hence any data in heap should be cleared through destructor using the pointer to heap memory.

Inheritance :-

```
class child : public parent{
};
```

this is mode of inheritence

```
outputf.in
Parent created
Child created
Child deleted
Parent deleted
```

```
1.cpp
#include<bits/stdc++.h>
using namespace std;

class person{
public:
    person(){
        cout<<"Parent created"<<endl;
    }
    string name;
    int age;

    ~person(){
        cout<<"Parent deleted"<<endl;
    }
};

class student:public person{
    public:
        student(){
            cout<<"Child created"<<endl;
        }
        int rollno;

        ~student(){
            cout<<"Child deleted"<<endl;
        }
};

int main() {
    student s;
    return 0;
}
```

Code

* a default constructor is needed in parent ; if parameterized cons is present in parent non parameterized cons should be defined. which is called everytime derived class is created.

✓ constructor for child can be simplified using parent's constructor

```
Parent created Non Parameterized constructor
Child created non-Parameterized constructor
Parent created parameterized constructor
Child created Parameterized constructor
Child deleted
Parent deleted
Child deleted
Parent deleted
```

```
#include<bits/stdc++.h>
using namespace std;

class person{
public:
    person(){
        cout<<"Parent created Non Parameterized constructor"<<endl;
    }
    person(int age, string name){
        name = name;
        age = age;
        cout<<"Parent created parameterized constructor"<<endl;
    }
    string name;
    int age;

    ~person(){ ... }
};

class student:public person{
    public:
        student(){
            cout<<"Child created non-Parameterized constructor"<<endl;
        }
        student(string name, int age, int rollno):person(age, name){
            rollno = rollno;
            cout<<"Child created Parameterized constructor"<<endl;
        }
        int rollno;

        ~student(){ ... }
};

int main() {
    student s;
    student s1("Rahul", 23, 1234);
    return 0;
}
```

```
class Parent{
    parent(int age, string name){
        :
    };
class child : public Parent{
    child(int rollno, string name, int age) : parent(age, name)
```

This age & name order must be followed

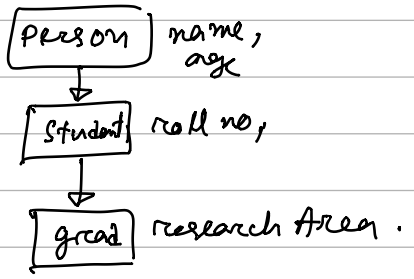
here order can be shuffled.

Mode of Inheritance :-

attributes in parent class →	private	protected	public
Inherited in Private mode:	Not Inherited	Private	Private
Protected mode:	Not Inherited	Protected	Protected
Public mode:	Not Inherited	Protected	Public

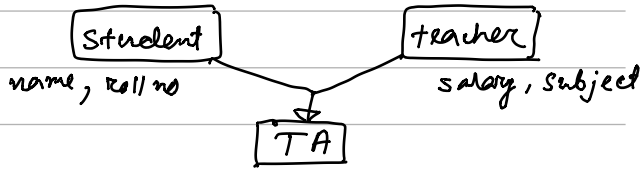
types of Inheritance :-

Multilevel Inheritance :-



if at any level parent is inherited in protected mode it will those attributes are continued as protected.

Multiple Inheritance :-



Eg. class TA : public Student, public Teacher {

};

Polymorphism :- (multiple form)

objects to take different forms, depending on the Context. or function's behaviours

e.g. constructor overloading.

Compile Time Polymorphism :

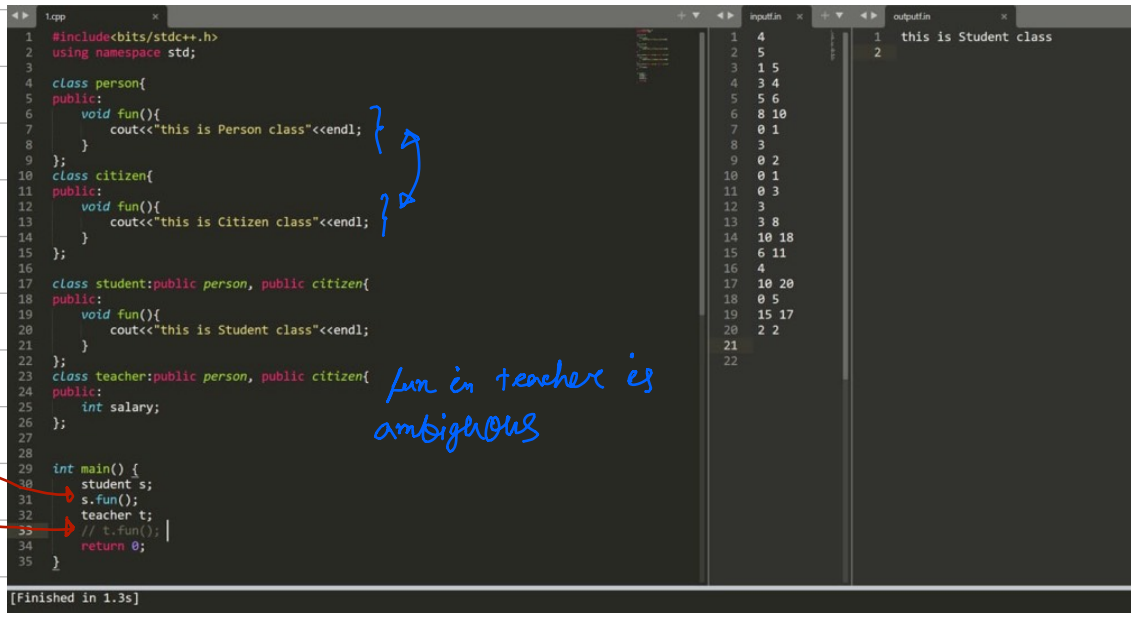
e.g. constructor overloading, operator overloading, function Overloading. → class {
fun1( ) → different parameter
fun1( )

```
class print{
    print(int x){ cout<<x ;}
    print(char x){ cout<<x ;}
```

Run Time Polymorphism :-

e.g function overriding

↳ child class function overrides parent class function.



This runs

This creates a problem

fun in teacher is ambiguous

e.g. virtual function which is called at runtime.

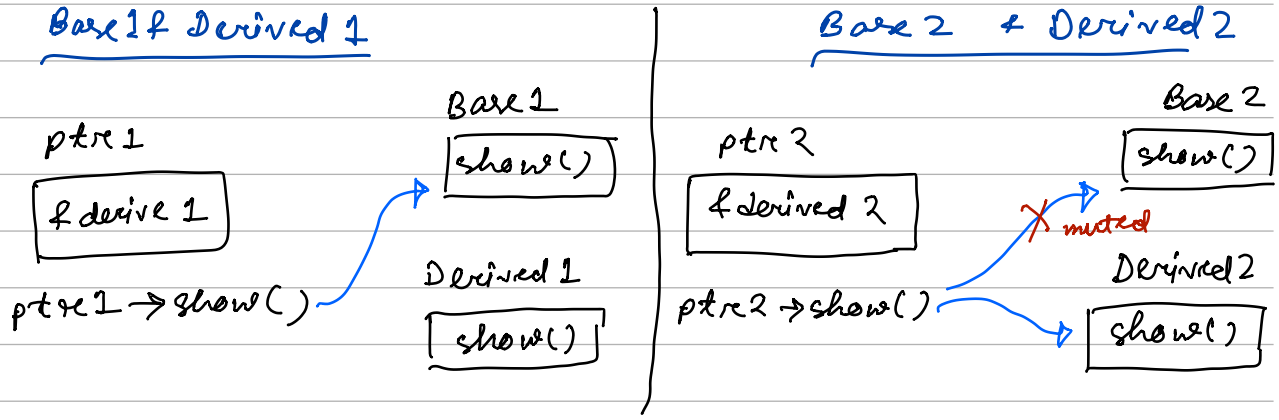
Need For virtual function :-

```cpp
#include<bits/stdc++.h>
using namespace std;

class base1{
    public:
    void show(){cout<<"this is base class"<<endl;}
};
class base2{
    public:
    virtual void show(){cout<<"this is base class"<<endl;}
};
class derived1:public base1{
    public:
    void show(){cout<<"this is Derived class 1"<<endl;}
};
class derived2:public base2{
    public:
    void show(){cout<<"this is Derived class 2"<<endl;}
};

int main() {
    base1* ptr = new derived1;
    ptr->show();

    base2* ptr2 = new derived2;
    ptr2->show();

    return 0;
}
```

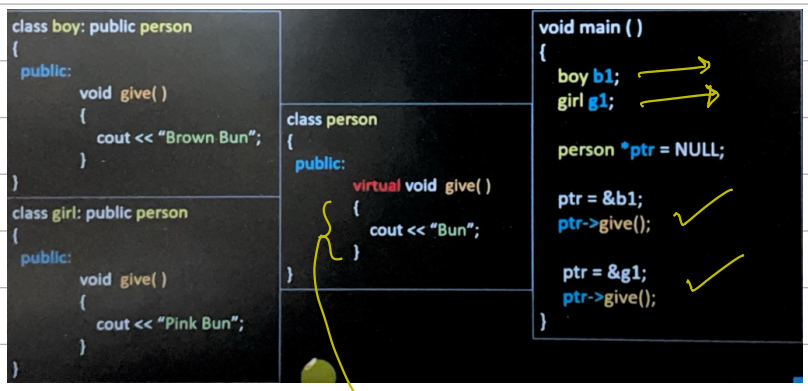
→ virtual function gets muted once object pointer is initialized to a derived class.

```
(base) PS C:\Users\Swarnar
this is base class
this is Derived class 2
(base) PS C:\Users\Swarnar
```

Reason for this output



✓ Another practical application.



#here we initialized 1 base pointer to invoke function of multiple child's function.

virtual void given() = 0 ; # this would have been purely Abstract class.

Virtual Destructor

```
virtual ~base(){ "deleting base" }

~derived(){ "deleting derived" }

base* ptr = derived
delete derived
```

→ without deleting base

with virtual deleting derived deliting base.

Abstraction :-

hiding unnecessary / sensitive details only showing important part.

access modifiers : implementing abstraction
(public, private, protected)

- Abstract classes
  - can not be instantiated are meant to be inherited.
  - typically used to define an interface for derived classes.

Operator Overloading Pre & Post Increment

- ✓ C++ allows more than 1 definition for most operator in the same scope, which is called Operator Overloading.
- ✓ You can redefine or Overload most of built in Operator
- ✓ It's a type of polymorphism.
- ✓ Operator that can not be overloaded are
  - → scope operator (::)
  - → sizeof
  - → member selector (.)
  - → Pointer selector (*)
  - → ternary operator (?:)

```cpp
#include<bits/stdc++.h>
using namespace std;

class Complex{
private:
    int real;
    int img;

public:
    Complex(int r = 0, int i = 0){
        this->real = r;
        this->img = i;
    }

    Complex operator + (Complex& C){
        return Complex(this->real + C.real, this->img + C.img);
    }
    void operator ++ (){
        real++;
        img++;
    }
    void operator ++ (int){
        real++;
        img++;
    }
    void show(){
        cout<<"real : "<<real<<" Img : "<<img<<endl;
    }
};

int main() {
    Complex c1, c2(2, 3), c3(2, 3);
    c1.show(); c2.show(); c3.show();

    Complex c4 = c1+c2+c3;
    c4.show();
    ++c4;
    c4.show();
    c4++;
    c4.show();
    return 0;
}
```

Pre increment

Post increment

This also works

```cpp
Complex Operator ++ ( ) {
    Complex temp;
    temp.real = ++real;
    temp.img = ++img;
    return temp;
}
```

This also works.

```
(base) PS C:\Users\Swarr
real : 0 Img : 0
real : 2 Img : 3
real : 2 Img : 3
real : 4 Img : 6
real : 5 Img : 7
real : 6 Img : 8
(base) PS C:\Users\Swarr
```

output to this

Function Templates in C++ :-

✓ generic programm / Type independent variable.
✓ Template is a blueprint for creating Generic class / function

→ ① Function Templates.
② class Templates.

Function Overloading

```
int add (int x, int y){
float add (float x, float y){
double add(double x, double y){
```

Function Template

```
template <typename T>
T add (T x, T y)
{ return (x+y); }
```

Eg. `add<int>(3, 7);`

For 2 different variables;

```
template <typename T, typename U>
U add(T x, U y){ return (x+y); }
```

↳ add<int, double>(2, 3.95) → 5.95
add<double, int>(3.95, 2) → 5

Class Template

```
template <typename T>
class Weight{
    private; T Kg;
    public:
        void setData(T x)
            { Kg = x; }
        T getData()
            { return Kg; }
```

Eg. `Weight <int> W1;`

`Weight <double> W2;`
`W1.setData(5)`
`W2.setData(5.5).`

→ Runtime polymorphism : virtual function :-

a virtual f^n is a member function that is expected to be redefined in derived classes.

- ✓ virtual f^n are Dynamic in nature.
- ✓ defined by Keyword virtual. always declared in Base class
- ✓ virtual functions are called during runtime.

```
# class shape {
        virtual void draw() = 0;
  };
```

} this is pure virtual f^n & shape is pure Abstract class.

```
Shape s1;
return 0;
```

} gives error coz shape is pure abstract class.

→ Static Keyword :-

```
void fun(){                    op
    int x=0                    0
    cout<< x;                  0
    x++ }                      0
main{ fun();
      fun();
      fun(); }
```

```
void fun(){                    op
    static int x=0;            0
    cout<< x;                  1
    x++ }                      2
main{ fun();
      fun();
      fun(); }
```

- variable declared as static in a function are created and initialized once for the lifetime of the programm. (in f^n)
- Static variables in classes are declared & initialized Once. they are shared by all the objects of the class. (in Class)

```cpp
#include<bits/stdc++.h>
using namespace std;

class A{
public:
    A(int x){
        this->x = x;
    }
    int x;
    void show(){
        cout<<"x : "<<x<<endl;
    }
};
class B{
public:
    static int y;
    void show(){
        cout<<"y : "<<y<<endl;
    }
};

int B::y = 5;
int main() {
    A a1(1), a2(2);
    B b1, b2;
    b2.y = 8;
    b1.y = 6;
    b1.y++;
    a1.show();
    a2.show();
    b1.show();
    b2.show();
    return 0;
}
```

this is the correct way to initialize static variable of a class

```cpp
#include<bits/stdc++.h>
using namespace std;

class B{
public:
    B(string s){
        this->s = s;
        cout<<"B is created : "<<s<<endl;
    }
    static int y;
    string s;
    void show(){
        cout<<"y : "<<y<<endl;
    }
    ~B(){
        cout<<"B is destroyed : "<<this->s<<endl;
    }
};

int B::y = 5;

int main() {
    if(true){
        static B b1("static");
        B b2("non static");
    }
    cout<<"end of main"<<endl;
    return 0;
}
```

```
B is created : static
B is created : non static
B is destroyed : non static
end of main
B is destroyed : static
```

code ;

static B b1;
is destroyed after main function finishes.

Friend Function :-

```
class teacher{
    private:
        int age
    public:
        void getInfo(){ cout<<age}
        friend void addX(teacher &T, int x);
};

void addX(Teacher &T, int x){
    T.age += x;          }
```

#Revision, #Tricky business

friend function.

#tricky

#sometime works sometime doesn't.

```
192
193  #include<bits/stdc++.h>
194  using namespace std;
195
196  class cl{
197      private:
198          int dist;
199      public:
200          cl(int x){
201              this->dist = x;
202          }
203          void show(){
204              cout<<"dist : "<<this->dist<<endl;
205          }
206      friend void printRes(cl &g);
207
208  };
209
210  void printRes(cl &G){
211      cout<<G.dist<<endl;
212      G.dist++;
213  }
214
215  int main() {
216      cl c(7);
217      c.show();
218      printRes(c);
219      c.show();
220
221      return 0;
222  }
```

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   COMMENTS

(base) PS C:\Users\Swarnarup\Desktop\DSA_Progress\DSA_progress\zOOP
dist : 7
7
dist : 8
```