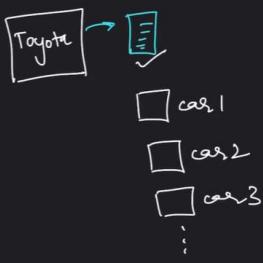


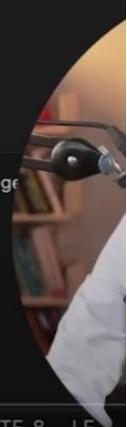
## Classes & Objects

- objects are **entities** in the real world
- class is like a **blueprint** of these entities



Ye wrong hai access modifier bhi dena padta hai class me

```
3 using namespace std;
4
5 class Teacher {
6     //properties/ attributes
7     string name;
8     string dept;
9     string subject;
10    double salary;
11
12    //methods/ member functions
13    void changeDept(string newDept) {
14        dept = newDept;
15    }
16};
17
18 int main() {
19     Teacher t1;    █
20
21     return 0;
22 }
```



```
o oops.cpp X
ApnaCollege > oops.cpp > main()
10     double salary;
11
12     //methods/ member functions
13     void changeDept(string newDept) {
14         dept = newDept;
15     }
16 };
17
18 int main() {
19     Teacher t1;
20     t1.name = "Shradha";
21     t1.subject = "C++";
22     t1.dept = "Computer Science";
23     t1.salary = 25000;
24
25     cout << t1.name << endl;
PORTS PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
oops.cpp:25:16: error: 'name' is a private member of 'Teacher'
    cout << t1.name << endl;                                     ^
oops.cpp:7:12: note: implicitly declared private here
    string name;                                                 ^
5 errors generated.
apnacollege@Amans-MacBook-Pro ApnaCollege %
```

Ln 26, Col 14 (118 selected) Spaces: 4 UTF-8 LF

## Access Modifiers

private      *data & methods accessible inside class*

public      *data & methods accessible to everyone*

protected    *data & methods accessible inside class & to its derived class*

By default sab private hai

```
class Teacher {
private:
    double salary;
public:
    string name;
    string dept;
    string subject;
    void changeDept(string newDept) {
        dept = newDept;
    }
};

int main() {
    Teacher t1;
    t1.name = "Shradha";
    t1.subject = "C++";
    t1.dept = "Computer Science";
}
```

```
};

int main() {
    Teacher t1;
    t1.name = "Shradha";
    t1.subject = "C++";
    t1.dept = "Computer Science";
    t1.salary = 25000;
}

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
```

```
college@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
.oops.cpp:24:8: error: 'salary' is a private member of 'Teacher'
    t1.salary = 25000;
          ^
.oops.cpp:7:12: note: declared private here
double salary;
          ^
error generated.
college@Amans-MacBook-Pro ApnaCollege %
```

3 / 20422 Access Specifier >

Ln 24 Col 23 (18 selected)

How to still get salary in main despite making it private

```
acollege % oops.cpp Teacher setSalary(double)
class Teacher {
private:
    double salary;

public:
    string name;
    string dept;
    string subject;

    void changeDept(string newDept) {
        dept = newDept;
    }

    //setter
    void setSalary(double s) {
        salary = s;
    }
    //getter
    double getSalary() {
        return salary;
    }
};

int main() {
}

PORTS PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
acollege@Amans-MacBook-Pro ApnaCollege %
```

Getting salary indirectly through these public functions

```
24     return salary;
25 }
26 };
27
28 int main() {
29     Teacher t1;
30     t1.name = "Shradha";
31     t1.subject = "C++";
32     t1.dept = "Computer Science";
33     t1.setSalary(25000);
34 }

PORTS PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
Shradha
25000
apnacollege@Amans-MacBook-Pro ApnaCollege %
```



## Encapsulation

### Encapsulation

Encapsulation is **wrapping up** of data & member functions in a single unit called class.

data)  
properties + member  
frx

C



### Encapsulation

Encapsulation is **wrapping up** of data & member functions in a single unit called class.

data hiding → private



```
24     return salary;
25 }
26 };
27
28 int main() {
29     Teacher t1;
30     t1.name = "Shradha";
31     t1.subject = "C++";
32     t1.dept = "Computer Science";
33     t1.setSalary(25000);
34 }
```

PLOTS PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out  
Shradha  
25000  
apnacollege@Amans-MacBook-Pro ApnaCollege %



```
24     return salary;
25 }
26 };
27
28 int main() {
29     Teacher t1;
30     t1.name = "Shradha";
31     t1.subject = "C++";
32     t1.dept = "Computer Science";
33     t1.setSalary(25000);
34 }
```

PLOTS PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out  
Shradha  
25000  
apnacollege@Amans-MacBook-Pro ApnaCollege %



Data hiding also done using encapsulation

```
class Account {
private:
    double balance;
    string password; //data hiding

public:
    string accountId;
    string username;

}

int main() {
```

PLOTS PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

ecollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out  
dha

## Constructor

### Constructor

Special method invoked automatically at time of object creation. Used for Initialisation.

- Same name as class
- Constructor doesn't have a return type
- Only called once (automatically), at object creation
- Memory allocation happens when constructor is called



```
24     |     return salary;
25   }
26 };
27
28 int main() {
29   Teacher t1; //constructor call
30   t1.name = "Shradha";
31   t1.subject = "C++";
32   t1.dept = "Computer Science";
33   t1.setSalary(25000);
34
35   cout << t1.name << endl;
36   cout << t1.getSalary() << endl;
37   return 0;
38 }
```

PORTS PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

If we don't make it C++ automatically does this for us

Ya programmer constructor banata hai ya compiler

Constructor ko Hamesha public hi banana hai

Highlighted is constructor

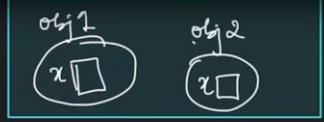
```
naCollege > oops.cpp > Teacher > Teacher()
3     using namespace std;
4
5     class Teacher {
6     private:
7         double salary;
8
9     public:
10        Teacher() {
11            dept = "Computer Science";    I
12        }
13
14        string name;
15        string dept;
16        string subject;
17
18        void changeDept(string newDept) {
19            dept = newDept;
20        }
21
22        //setter
23        void setSalary(double s) {
24            salary = s;
25        }
26    }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL zsh - Apni

OOPs Tutorial in One Shot | Object Oriented Programming | in C++ Language | for Placement Interviews

## Constructor

A obj1;  
A obj2;



Special method invoked automatically at time of **object creation**. Used for Initialisation.

- Same name as class
- Constructor doesn't have a return type
- Only called once (automatically), at object creation
- Memory allocation happens when constructor is called

class A{  
 int x;  
}



36:19 / 2:04:22 - Constructor >

Memory allocation is for objects not classes and it happens when constructor is called

```
public:
    //non-parameterized
    Teacher() {
        dept = "Computer Science";
    }

    //parameterized
    Teacher(string n, string d, string s, double sal) {
        name = n;
        dept = d;
        subject = s;
        salary = sal;
    }

    string name;
    string dept;
    string subject;
```

TERMINAL

```
maccollege@Amans-MacBook-Pro AnnaCollege % g++ -std=c++11 oops.cpp && ./a.out
```

```
public:
    //non-parameterized
    Teacher() {
        dept = "Computer Science";
    }

    //parameterized
    Teacher(string n, string d, string s, double sal) {
        name = n;
        dept = d;
        subject = s;
        salary = sal;
    }

    string name;
    string dept;
    string subject;
```

TERMINAL

```
maccollege@Amans-MacBook-Pro AnnaCollege % g++ -std=c++11 oops.cpp && ./a.out
```

A class can have multiple constructors also given different parameters hai this is called constructor overloading

Constructor overloading is example of polymorphism

```
public:  
    //non-parameterized  
    Teacher() {  
        dept = "Computer Science";  
    }  
  
    //parameterized  
    Teacher(string n, string d, string s, double sal) {  
        name = n;  
        dept = d;  
        subject = s;  
        salary = sal;  
    }  
  
    string name;  
    string dept;  
    string subject;
```

TERMS PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

zsh - ApnaCollege@Amane-MacBook-Pro\_ApnaCollege % g++ -std=c++11 cons.cpp && ./a.out

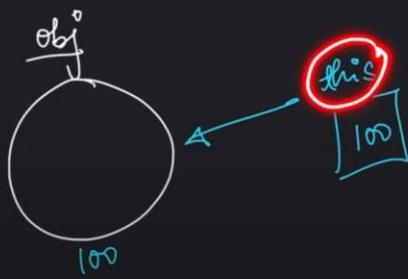
This

Current object ko point karta hai

## Constructor

**this** is a special pointer in C++ that points to the current object.

*this->prop is same as \*(this).prop*



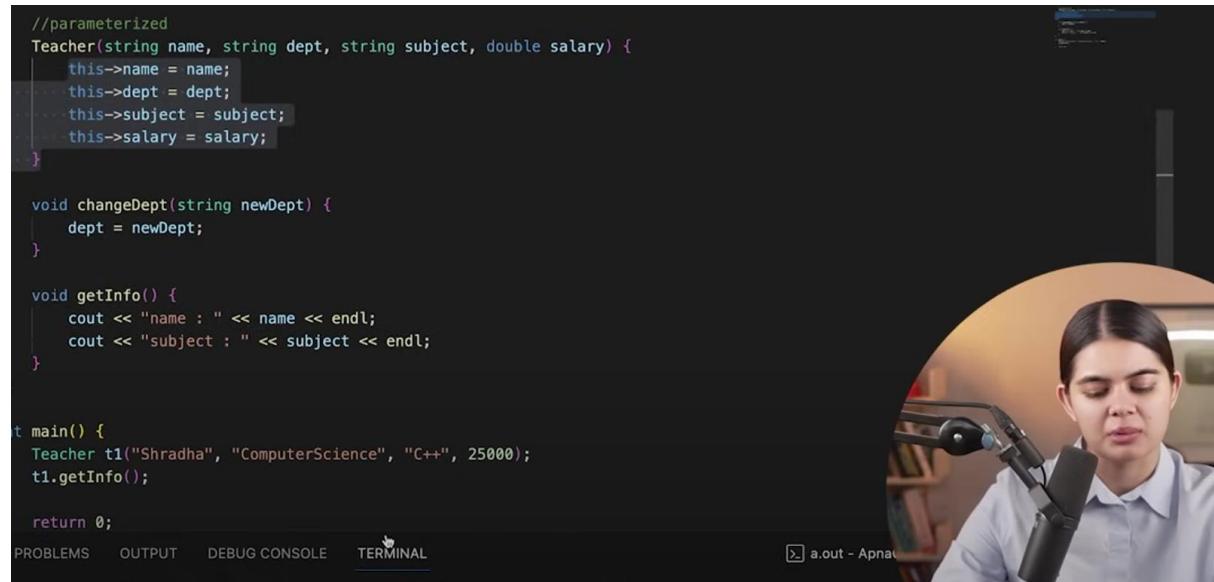
If we want to keep parameter names similar to names of properties in class then compiler can get confused

Name=name

Dept=dept

It is confusing

Left name is property of obj and right is function parameter



The screenshot shows a C++ code editor with the following code:

```
//parameterized
Teacher(string name, string dept, string subject, double salary) {
    this->name = name;
    this->dept = dept;
    this->subject = subject;
    this->salary = salary;
}

void changeDept(string newDept) {
    dept = newDept;
}

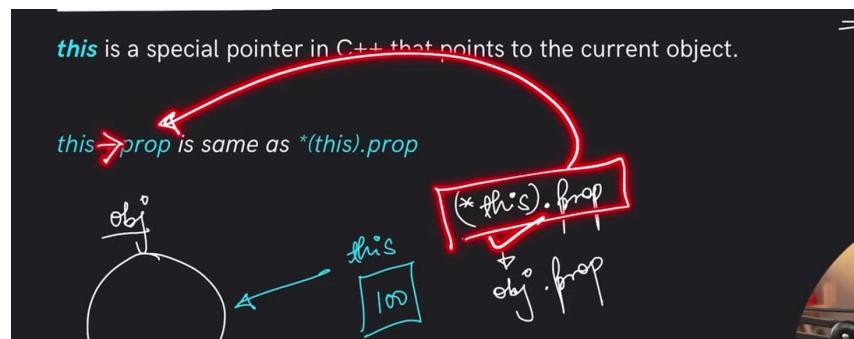
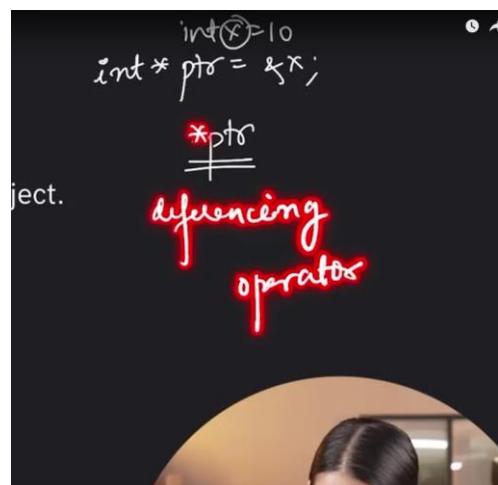
void getInfo() {
    cout << "name : " << name << endl;
    cout << "subject : " << subject << endl;
}

int main() {
    Teacher t1("Shradha", "ComputerScience", "C++", 25000);
    t1.getInfo();

    return 0;
}
```

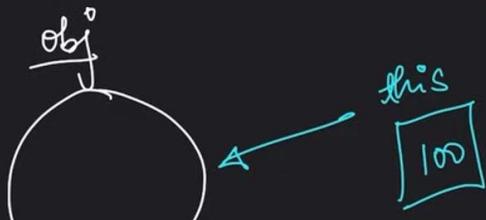
Below the code, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab is selected. To the right of the code editor is a circular video feed of a person speaking into a microphone.

This always point to current object



**this** is a special pointer in C++ that points to the current object.

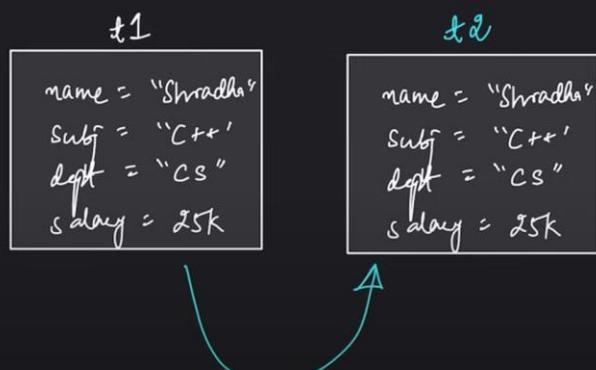
**this** → **prop** is same as **\*this.prop**



Copy constructor

## Copy Constructor

Special Constructor (default) used to copy properties of one object into another.



```
//copy constructor
Teacher(Teacher &obj) { //pass by reference| }
```

```
}
```

```
void changeDept(string newDept) {
    dept = newDept;
}
```

## Custom copy constructor

```
}

//copy constructor
Teacher(Teacher &orgObj) {
    cout << "i am custom copy constructor...\n"
    this->name = orgObj.name;
    this->dept = orgObj.dept;
    this->subject = orgObj.subject;
    this->salary = orgObj.salary;
}

}

//copy constructor
Teacher(Teacher &orgObj) {
    cout << "i am custom copy constructor...\n"
    this->name = orgObj.name;
    this->dept = orgObj.dept;
    this->subject = orgObj.subject;
    this->salary = orgObj.salary;
}
```

2 types of copy shallow and deep default is shallow

OPs Tutorial in One Shot | Object Oriented Programming | in C++ Language | for Placement Interviews

*default cst*

## Shallow & Deep Copy

A shallow copy of an object copies all of the member values from one object to another.

A deep copy, on the other hand, not only copies the member values but also makes copies of any dynamically allocated memory that the members point to.

```
anaCollege > oops.cpp > Student > Student(Student &)
2     using namespace std;
3
4     class Student {
5     public:
6         string name;
7         double* cgpaPtr;
8
9     Student(string name, double cgpa) {
10         this->name = name;
11         cgpaPtr = new float;
12         *cgpaPtr = cgpa;
13     }
14
15     Student(Student &obj) {
16         this->name = obj.name;
17         this->cgpaPtr = obj.cgpaPtr;    █
18     }
19
20     void getInfo() {
21         cout << "name : " << name << endl;
22         cout << "cgpa : " << *cgpaPtr << endl;
23     }
24 };
25
26
27 int main() {
28     Student s1("rahul kumar", 8.9);
29     Student s2(s1); //neha kumar
30
31     s1.getInfo();
32     *(s2.cgpaPtr) = 9.2;           █
33     s1.getInfo();
34     return 0;
35 }
```

```
ORTS PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL zsh - ApnaC
apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
ame : rahul kumar
gpa : 8.9
ame : rahul kumar
gpa : 9.2
apnacollege@Amans-MacBook-Pro ApnaCollege % █
```

## Shallow & Deep Copy

8.9  
555  
loop

A **shallow** copy of an object copies all of the member values from one object to another.

A **deep** copy, on the other hand, not only copies the member values but also makes copies of any dynamically allocated memory that the members point to.



Deep copy

```
oops.cpp X
ApnaCollege > oops.cpp > Student > Student(Student &)

6 public:
7     string name;
8     double* cgpaPtr;
9
10    Student(string name, double cgpa) {
11        this->name = name;
12        cgpaPtr = new double;
13        *cgpaPtr = cgpa;
14    }
15
16    Student(Student &obj) {
17        this->name = obj.name;
18        cgpaPtr = new double;
19        *cgpaPtr = *obj.cgpaPtr;
20    }
21
22    void getInfo() {
23        cout << "name : " << name << endl;
24        cout << "cgpa : " << *cgpaPtr << endl;
25    }
26}
27
28 int main() {
```

```

26 }
27
28 int main() {
29     Student s1("rahul kumar", 8.9);
30     Student s2(s1); // "neha kumar"
31
32     s1.getInfo();
33     *s2.cgpaPtr = 9.2;
34     s1.getInfo();
35
36     s2.name = "neha";
37     s2.getInfo();
38     return 0;
39 }

```

PORTS PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

● apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
name : rahul kumar
cgpa : 8.9
name : rahul kumar
cgpa : 8.9
● apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
name : rahul kumar
cgpa : 8.9
name : rahul kumar
cgpa : 8.9
name : neha
cgpa : 9.2
apnacollege@Amans-MacBook-Pro ApnaCollege % clear

```

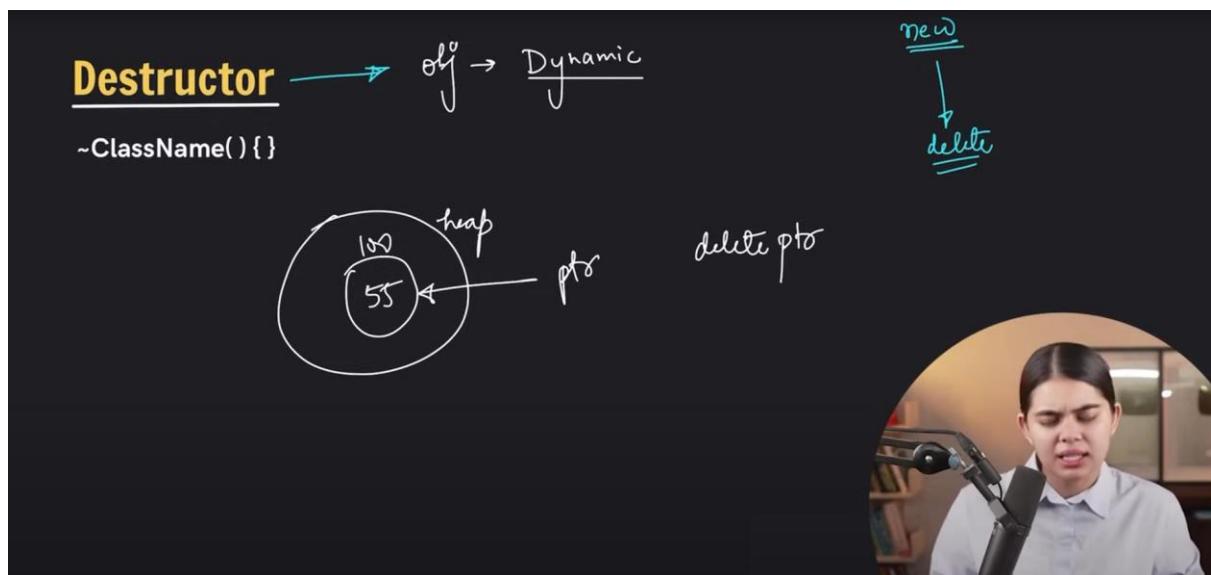
## Destructor

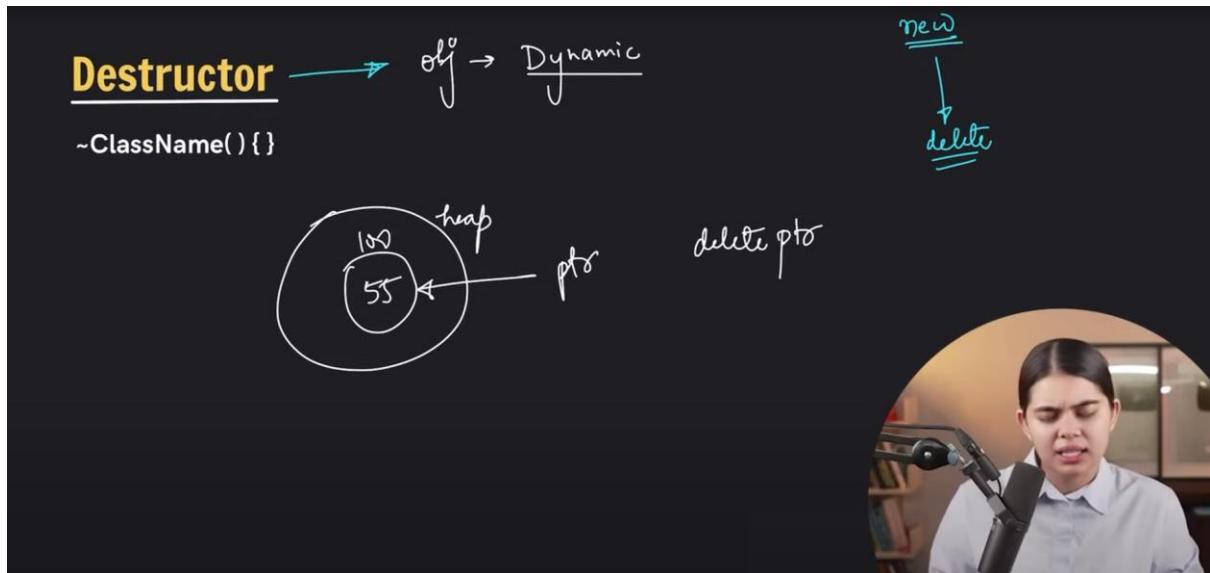
Compiler does this automatically

But agar dynamic memory allocation hai toh dynamic deallocation karna padega

It does not delete pointer but delete memory to which the pointer points

Cout<<ptr still works





Called automatically

`~` starting with it

```

12     cgpaPtr = new double;
13     *cgpaPtr = cgpa;
14 }
15
16 //destructor
17 ~Student() {
18     cout << "Hi, I delete everything\n";
19 }
20
21 void getInfo() {
22     cout << "name : " << name << endl;
23     cout << "cgpa : " << *cgpaPtr << endl;
24 }
25 };
26
27 int main() {
28     Student s1("rahul kumar", 8.9);
29     s1.getInfo();
30     return 0;
31 }

PORTS PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL
apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
name : rahul kumar
cgpa : 8.9
Hi, I delete everything
apnacollege@Amans-MacBook-Pro ApnaCollege %

```

```

//destructor
~Student() {
    cout << "Hi, I delete everything\n";
    delete cgpaPtr;
}

void getInfo() {
    cout << "name : " << name << endl;
    cout << "cgpa : " << *cgpaPtr << endl;
};

int main() {
    Student s1("rahul kumar", 8.9);
    s1.getInfo();
    return 0;
}

```

॥

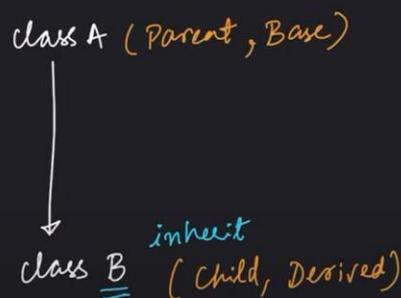
If not done memory leak can happen

Inheritance

Use for code reusability

## Inheritance

When properties & member functions of **base** class are passed on to the **derived** class.



In this we are doing in public mode of inheritance

Pehle parent class ka constructor call hota fir child class ka

Opposite in destructor

```
class Person {
public:
    string name;
    int age;

    // Person(string name, int age) {
    //     this->name = name;
    //     this->age = age;
    // }
    Person() {}

};

class Student : public Person {
    //name, age, rollno
}
```

No properties made like name in student class but still we get

```
class Student : public Person {
public:
    int rollno;

    void getInfo() {
        cout << "name : " << name << endl;
        cout << "age : " << age << endl;
        cout << "rollno : " << rollno << endl;
    }
};

int main() {
    Student s1;
    s1.name = "rahul kumar";
    s1.age = 21;
    s1.rollno = 1234;

    s1.getInfo();
    return 0;
}
```

```
MacCollege > cd C:\Users\HP\CLionProjects\oop3> Person> Person()
3     using namespace std;
4
5     class Person {
6     public:
7         string name;
8         int age;
9
10        // Person(string name, int age) {
11        //     this->name = name;
12        //     this->age = age;    std::__1::basic_ostream<char, std::__1::char_traits<char>>
13        // }
14        Person() []
15        |     cout << "parent constr"
16    []
17 };
18
19 class Student : public Person {
20 public:
21     int rollno;
22
23     Student() {
24         cout << "child constructor..\n";
25     }
26 }
```

```
MacCollege > cd C:\Users\HP\CLionProjects\oop3> Person> Person()
3     using namespace std;
4
5     class Person {
6     public:
7         string name;
8         int age;
9
10        // Person(string name, int age) {
11        //     this->name = name;
12        //     this->age = age;    std::__1::basic_ostream<char, std::__1::char_traits<char>>
13        // }
14        Person() []
15        |     cout << "parent constr"
16    []
17 };
18
19 class Student : public Person {
20 public:
21     int rollno;
22
23     Student() {
24         cout << "child constructor..\n";
25     }
26 }
```

Agar parameterised constructor hai parent me toh use call karenge aise

Parent constructor me type bhejne ki need nahi hai

```
13     }
14 };
15
16 class Student : public Person {
17 public:
18     int rollno;
19
20     Student(string name, int age, int rollno) : Person(name, age){
21         this->rollno = rollno;
22     }
23     void getInfo() {
24         cout << "name : " << name << endl;
25         cout << "age : " << age << endl;
26         cout << "rollno : " << rollno << endl;
27     }
28 };
29
30 int main() {
31     Student s1("rahul kumar", 21, 1234);
32 }
```

Ye abtak public mode of inheritance thi

Modes of inheritance

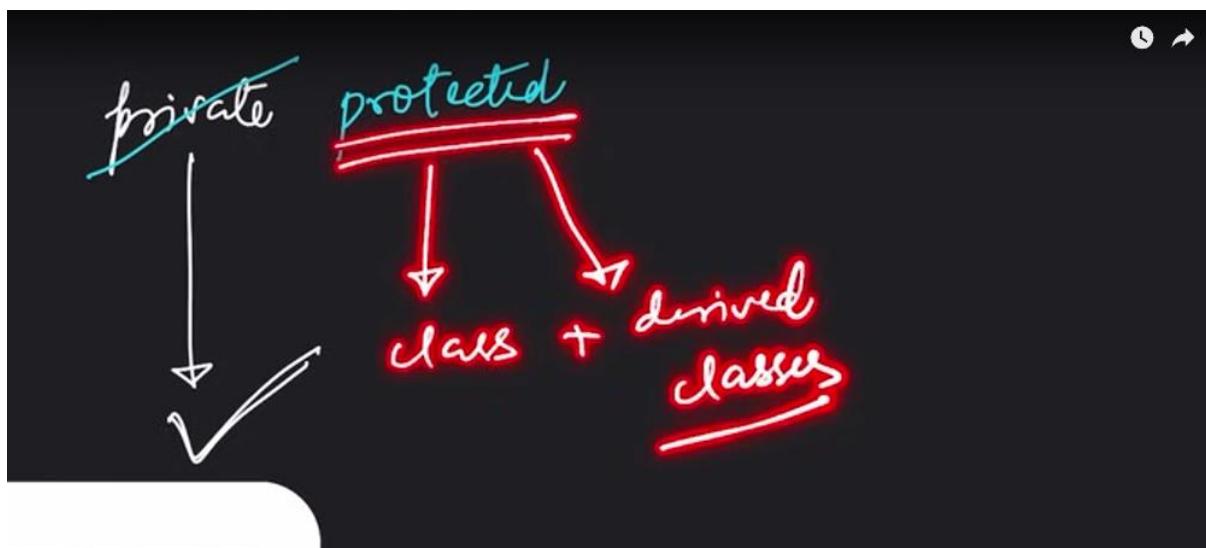
## Inheritance

### Mode of Inheritance

Base Class	Derived Class	Derived Class	Derived Class
Private	Private Mode	Protected Mode	Public Mode
Protected	Not Inherited	Not Inherited	Not Inherited
Public	Private	Protected	Public

If we have some private property that we just want to use for inheritance and not give them access anywhere else we make them protected

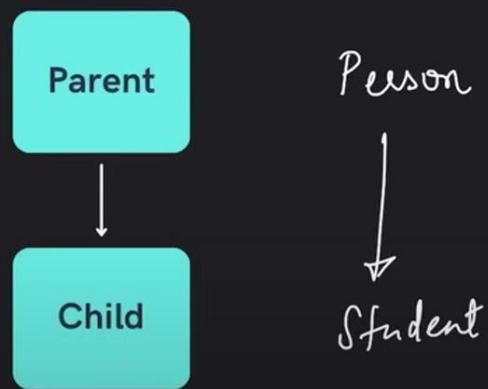
Sirf inheritance ke liye pass karna inhe



Types of inheritance

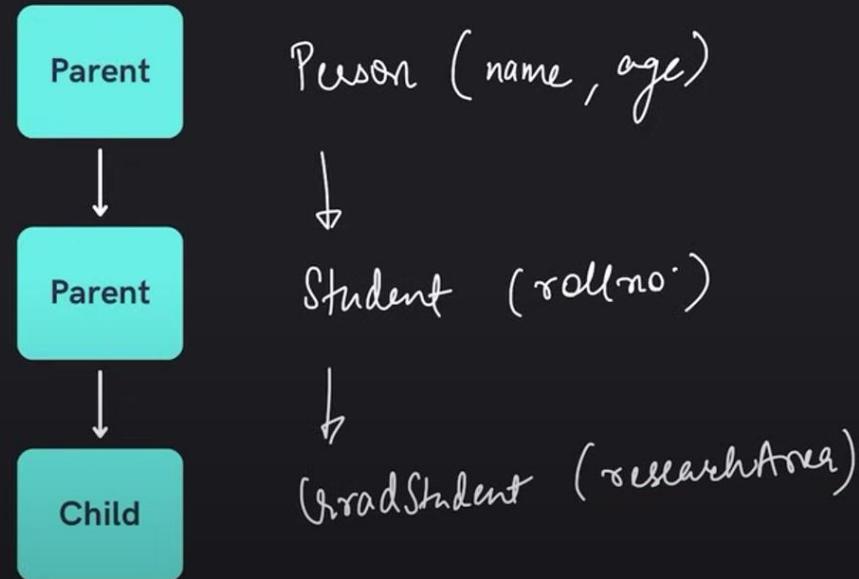
## Types of Inheritance

Single Inheritance



## Types of Inheritance

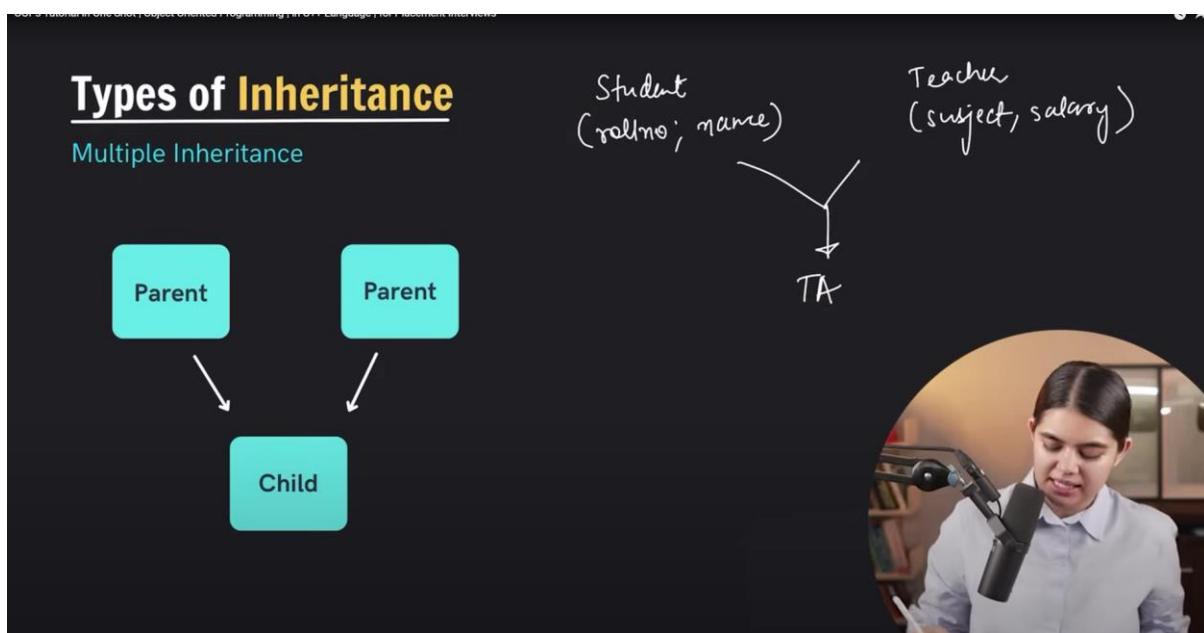
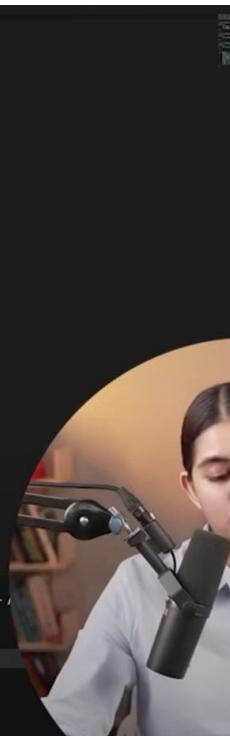
Multi-level Inheritance



```
2 #include <string>
3 using namespace std;
4
5 class Person {
6 public:
7     string name;
8     int age;
9 };
0
1 class Student : public Person {
2 public:
3     int rollno;
4 };
5
6 class GradStudent : public Student {
7 public:
8     string researchArea;
9 };
0
10 int main() {
11     GradStudent s1;
12     s1.name = "tony stark";
13     s1.researchArea = "quantum physics";
14 }
```

TERMINAL

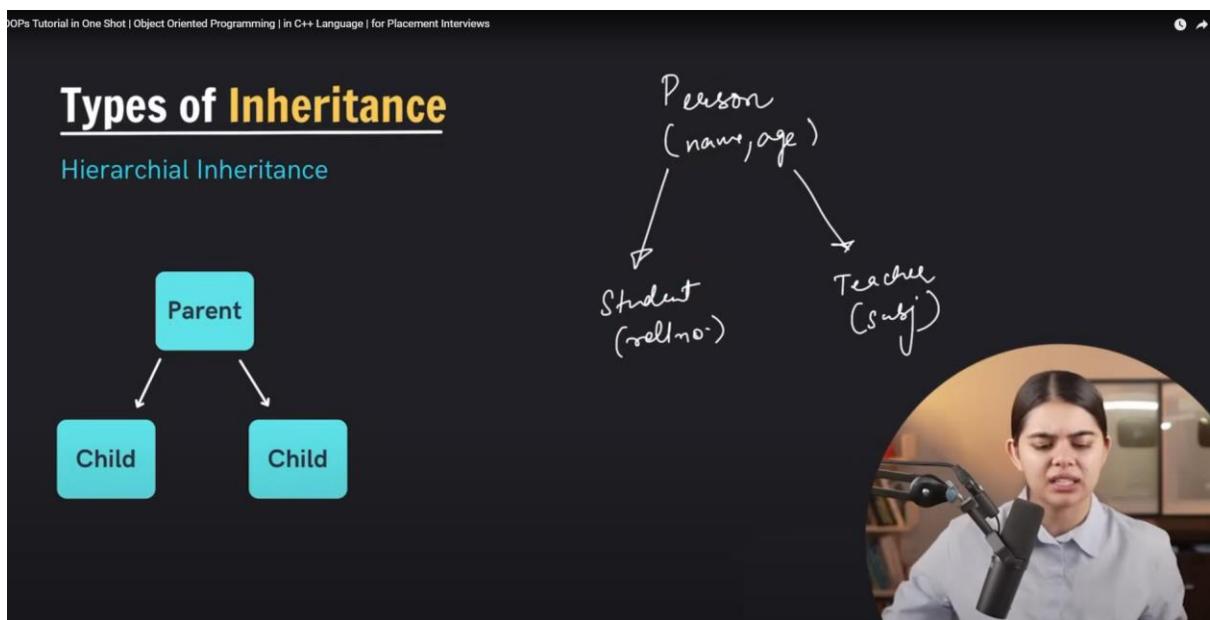
```
anacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
tony stark
quantum physics
anacollege@Amans-MacBook-Pro ApnaCollege %
```



```
ApnaCollege > C: oops.cpp > Student
3  using namespace std;
4
5  class Student {
6  public:
7      string name;
8      int rollno;
9  };
10
11 class Teacher {
12 public:
13     string subject;
14     double salary;
15 };
16
17 class TA : public Student, public Teacher {
18 };
19
20
21 int main() {
22     TA t1;
```

PORTS PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL

apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out



For consistency classes ka name capital me hi Rakha jata hai

```

class Person {
public:
    string name;
    int age;
};

class Student : public Person {
public:
    int rollno;
};

class Teacher : public Person{
public:
    string subject;
};

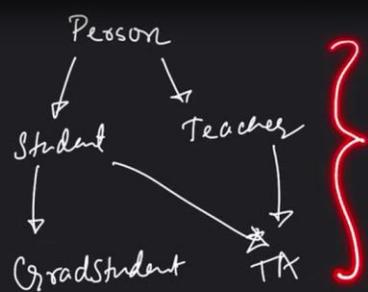
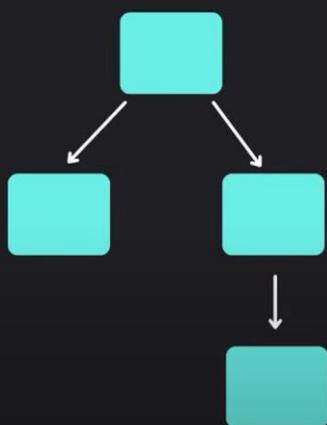
class TA : public Student, public Teacher {
}

```

OOPS Tutorial in One Shot | Object Oriented Programming | in C++ Language | for Placement Interviews

## Types of Inheritance

### Hybrid Inheritance



## Polymorphism

OOPS Tutorial in One Shot | Object Oriented Programming | in C++ Language | for Placement Interviews

*multiple forms*

# Polymorphism

Polymorphism is the ability of objects to take on **different forms** or behave in different ways **depending on the context** in which they are used.

- Compile Time Polymorphism
- Run Time Polymorphism



Constructor overloading is example of polymorphism (compile time )

OOPS Tutorial in One Shot | Object Oriented Programming | in C++ Language | for Placement Interviews

# Polymorphism

*Constructor Overloading*

Polymorphism is the ability of objects to take on **different forms** or behave in different ways **depending on the context** in which they are used.

- Compile Time Polymorphism → constructor
- Run Time Polymorphism



**Function overloading** is implementation of compile time polymorphism and it is when we are defining 2 or more functions with same name in same class but they only differ in terms of their parameters (no. or type of parameters)

## Compile Time Polymorphism

- Function Overloading

The diagram shows a class definition with two methods, fun(). The first method has one parameter, indicated by a red arrow labeled "param". The second method has no parameters, indicated by a red arrow pointing to an empty set of parentheses.



```
3 using namespace std;
4
5 class Print {
6 public:
7     void show(int x) {
8         cout << "int : " << x << endl;
9     }
10
11    void show(char ch) {
12        cout << "char : " << ch << endl;
13    }
14 };
15
16 int main() {
17     Print p1;
18     p1.show('&');
19     return 0;
20 }
```

PORTS PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL

```
apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
int : 101
apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
char : &
```

## Compile Time Polymorphism

- Function Overloading

```
class {
    fun() { param }
    fun() { }
}
```

Operator Overloading

int  $x = y$  ]  $x = 10$   
 string  $a = b$  ]  $a = "abc"$



Compile time polymorphism ex

Constructor overloading

Function overloading

Operator overloading

Overloading me within class 2 function

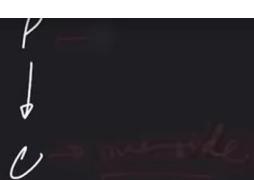
## Run Time Polymorphism

- Function Overriding

Parent & Child both contain the same function with different implementation.

The parent class function is said to be overridden.

Overloading      Overriding  
 class      ↓  
 f      Inheritance



If object is of child class child function is called if parent class obj then parent class function called

```
2 #include <iostream>
3 using namespace std;
4
5 class Parent {
6 public:
7     void getInfo() {
8         cout << "parent class\n";
9     }
10 };
11
12 class Child : public Parent {
13 public:
14     void getInfo() {
15         cout << "child class\n";
16     }
17 };
18
19 int main() {
20     Child c1;
21     c1.getInfo();
}
PORTS PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL
apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
child class
apnacollege@Amans-MacBook-Pro ApnaCollege %
```



Child class function override parent class

```
using namespace std;
class Parent {
public:
    void getInfo() {
        cout << "parent class\n";
    }
};

class Child : public Parent {
public:
    void getInfo() {
        cout << "child class\n";
    }
};

int main() {
    Parent p1;
```

## Run Time Polymorphism

- Virtual Functions

A virtual function is a member function that you expect to be redefined in derived classes.

vishab

Almost like overriding

```
ApnaCollege > C++ oops.cpp > Child > hello()
9     }
0
1     virtual void hello() {
2         cout << "hello from par\n";
3     }
4 };
5
6 class Child : public Parent {
7 public:
8     void getInfo() {
9         cout << "child class\n";
0     }
1
2     void hello() {
3         cout << "hello from par\n";
4     }
5 };
6
7 int main() {
8     Parent p1;

```

WORKS PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL

ApnaCollege@Amans-MacBook-Pro ApnaCollege %

zsh - A

## Run Time Polymorphism

- Virtual Functions

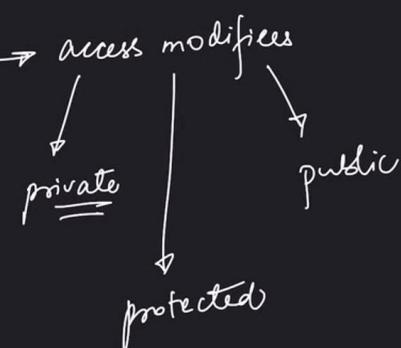
- Virtual functions are Dynamic in nature.
- Defined by the keyword "virtual" inside a base class and are always declared with a base class and overridden in a child class.
- A virtual function is called during Runtime



## Abstraction

Hiding all unnecessary details & showing only the important parts

sensitive



Data hiding only hides but abstraction also shows useful info

OOPs Tutorial in One Shot | Object Oriented Programming | in C++ Language | for Placement Interviews

## Abstraction

using **Abstract** Classes

abstract class ~~X~~

- Abstract classes are used to provide a base class from which other classes can be derived.
- They cannot be instantiated and are meant to be inherited.
- Abstract classes are typically used to define an interface for derived classes.



Ye class ka object nahi banta ye dusri class ka blueprint banti hai

Made only for inheritance

NOT POSSIBLE TO MAKE OBJECTS

Abstract class ka kaam hai bas ek idea dena

Ye sirf ek blue print deti hai konsi chije base class me honi chaiye

OOPs Tutorial in One Shot | Object Oriented Programming | in C++ Language | for Placement Interviews

## Abstraction

using **Abstract** Classes

instance → ~~obj~~

- Abstract classes are used to provide a base class from which other classes can be derived.
- They cannot be instantiated and are meant to be inherited.
- Abstract classes are typically used to define an interface for derived classes.



Make any shape in this example but they should have draw function

OOPS Tutorial in One Shot | Object Oriented Programming | in C++ Language | for Placement Interviews

## Abstraction

using **Abstract** Classes

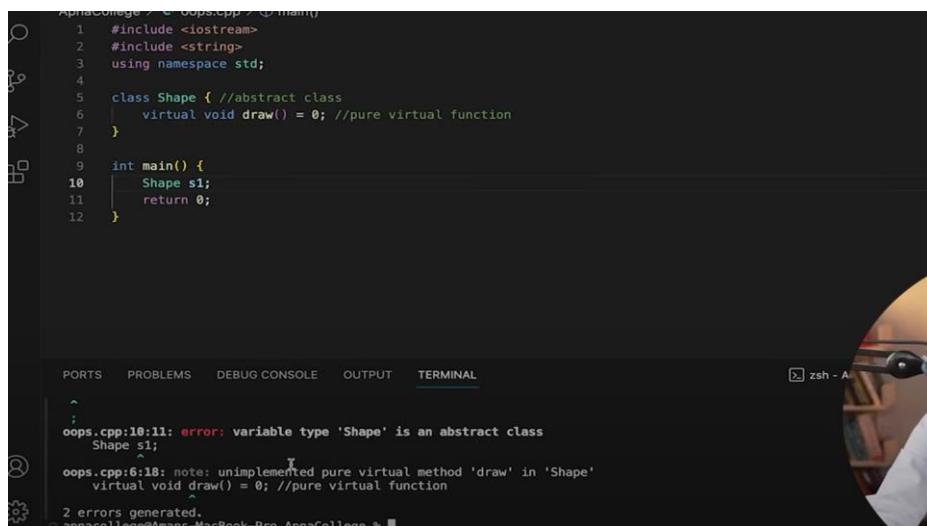
- Abstract classes are used to provide a base class from which other classes can be derived.
- They cannot be instantiated and are meant to be inherited.
- Abstract classes are typically used to define an interface for derived classes.

abstract Shape {  
 draw();  
}



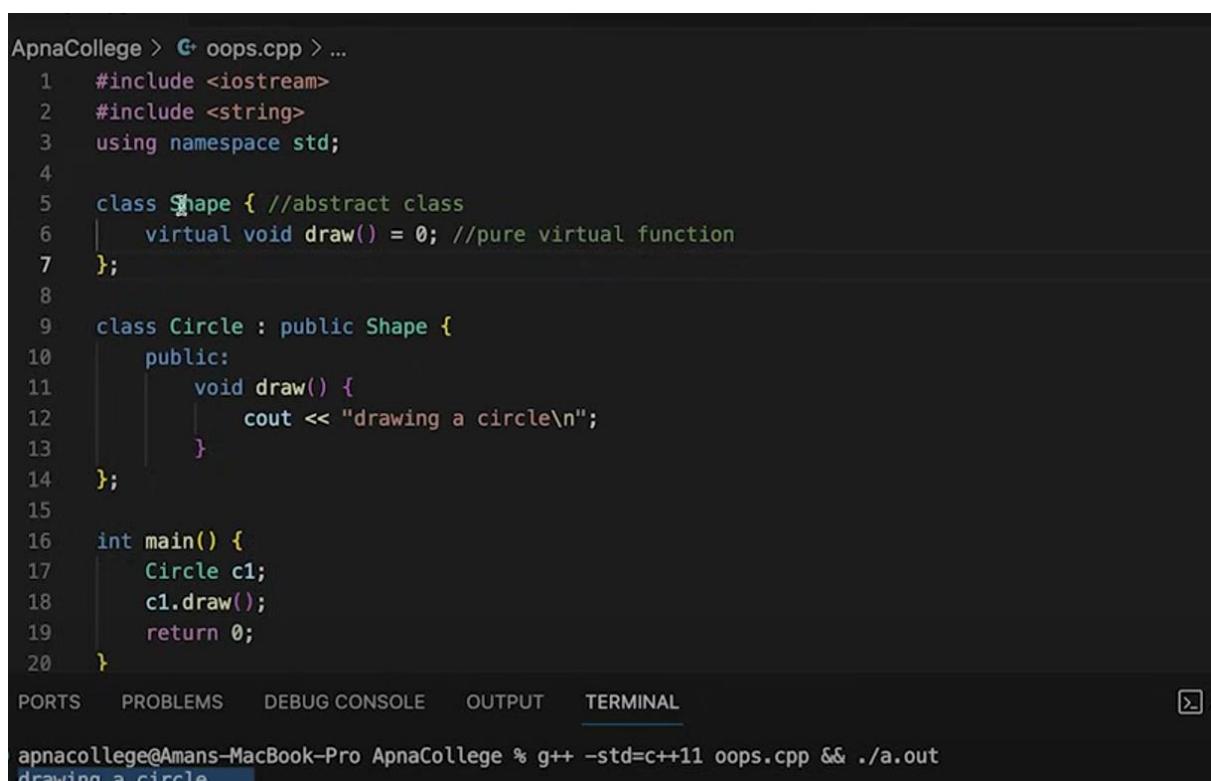
Jis class me pure virtual function hota hai who automatically abstract class ban jati hai

```
class Shape {  
public:  
    virtual void draw() = 0; //pure virtual function  
}  
  
int main()  
{  
    return 0;  
}
```



A screenshot of the Visual Studio Code interface. The left pane shows the code editor with the file 'oops.cpp'. The right pane shows the terminal output. The terminal output shows the following errors:

```
: oops.cpp:10:11: error: variable type 'Shape' is an abstract class
      Shape s1;
              ^
oops.cpp:6:18: note: unimplemented pure virtual method 'draw' in 'Shape'
    virtual void draw() = 0; //pure virtual function
              ^
2 errors generated.
apnacollege@Amans-MacBook-Pro ApnaCollege %
```



A screenshot of the Visual Studio Code interface. The left pane shows the code editor with the file 'oops.cpp'. The right pane shows the terminal output. The terminal output shows the following command and its execution:

```
ApnaCollege > oops.cpp > ...
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class Shape { //abstract class
6  |     virtual void draw() = 0; //pure virtual function
7  };
8
9  class Circle : public Shape {
10 |     public:
11 |         void draw() {
12 |             cout << "drawing a circle\n";
13 |         }
14 };
15
16 int main() {
17     Circle c1;
18     c1.draw();
19     return 0;
20 }

PORTS PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL
```

```
apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
drawing a circle
```

## Static

DOPs Tutorial in One Shot | Object Oriented Programming | in C++ Language | for Placement Interviews

# Static Keyword

- Static Variables

Variables declared as static in a function are created & initialised once for the lifetime of the program. [//in Function](#)

Static variables in a class are created & initialised once. They are shared by all the objects of the class. [//in Class](#)



- Static Objects

```
oops.cpp ✘
```

ApnaCollege > oops.cpp > main()

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 void fun() {
6     int x = 0;
7     cout << "x : " << x << endl;
8     x++;
9 }
10
11 int main() {
12     fun();
13     fun();
14     fun();
15     return 0;
16 }
```

PORTS PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL zsh - Ar

```
● apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
x : 0
x : 0
x : 0
○ apnacollege@Amans-MacBook-Pro ApnaCollege %
```

OOPS Tutorial in One Shot | Object Oriented Programming | in C++ Language | for Placement Interviews

## Static Keyword

- Static Variables

Variables declared as static in a function are created & initialised once for the lifetime of the program. //in Function

OOPS Tutorial in One Shot | Object Oriented Programming | in C++ Language | for Placement Interviews

File: oops.cpp

```
ApnaCollege > oops.cpp > fun()
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 void fun() {
6     static int x = 0; //init statement - 1 run
7     cout << "x : " << x << endl;
8     x++;
9 }
10
11 int main() {
12     fun();
13     fun();
14     fun();
15     return 0;
16 }
```

PORNS PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL

```
apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
X : 0
X : 1
X : 2
apnacollege@Amans-MacBook-Pro ApnaCollege %
```

```
class A {  
public:  
    int x;  
  
    void incX() {  
        x = x + 1;  
    }  
};
```

Normal without static in class

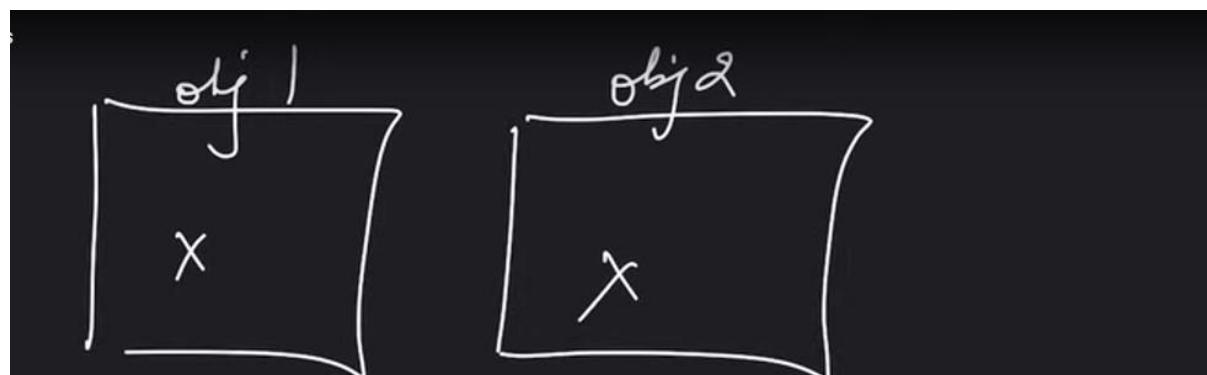
```
10     x = x + 1;  
11 }  
12 };  
13  
14 int main() {  
15     A obj1;  
16     A obj2;  
17  
18     obj1.x = 100;  
19     obj2.x = 200;  
20  
21     cout << obj1.x << endl;  
22  
23     return 0;  
24 }
```

PORTS PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL

a.out - A

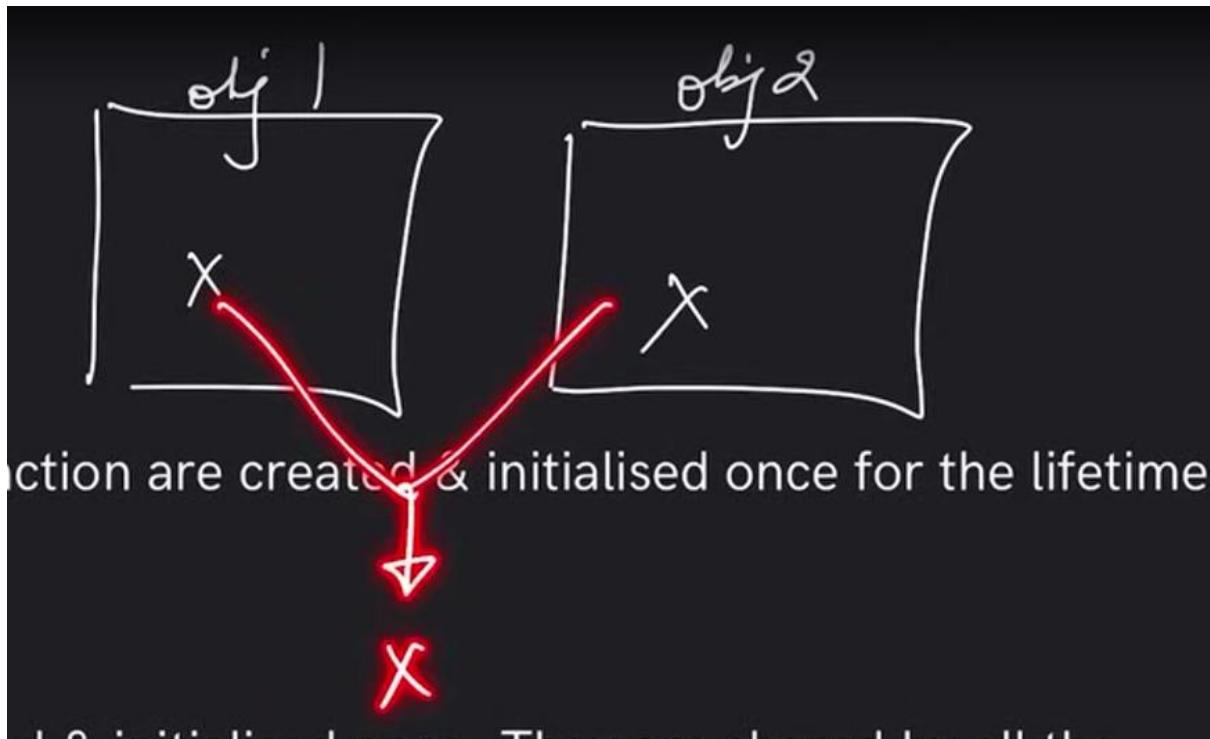
```
● apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out  
200  
● apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out  
100  
○ apnacollege@Amans-MacBook-Pro ApnaCollege %
```

Without static



Action are created & initialised once for the lifetime of the

With static



Static objects remain till the lifetime of program

OOPs Tutorial in One Shot | Object Oriented Programming | in C++ Language | for Placement Interviews

## Static Keyword

- Static Variables

Variables declared as static in a function are created & initialised once for the lifetime of the program. //in Function

Static variables in a class are created & initialised once. They are shared by all the objects of the class. //in Class

- Static Objects

*lifetim e of the program*

lifetim e of the program

## Without static

The screenshot shows the VS Code interface with the file 'oops.cpp' open. The code defines a class ABC with a constructor and a destructor. In the main() function, an object obj is created, which triggers the constructor and destructor output. The terminal shows the expected output: 'constructor', 'destructor', and 'end of main fnx'.

```
5   class ABC {
6   public:
7       ABC() {
8           cout << "constructor\n";
9       }
10      ~ABC() {
11          cout << "destructor\n";
12      }
13  };
14
15
16 int main() {
17     if(true) {
18         ABC obj;
19     }
20
21     cout << "end of main fnx\n";
22     return 0;
23 }
```

PORTS PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL

```
● apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
constructor
destructor
end of main fnx
● apnacollege@Amans-MacBook-Pro ApnaCollege %
```

## With static

The screenshot shows the VS Code interface with the file 'oops.cpp' open. The code is identical to the previous version, but it includes a static member in the main() function. When run, the static member is not executed, so only the constructor and destructor are printed. The terminal shows the output: 'constructor' and 'destructor'.

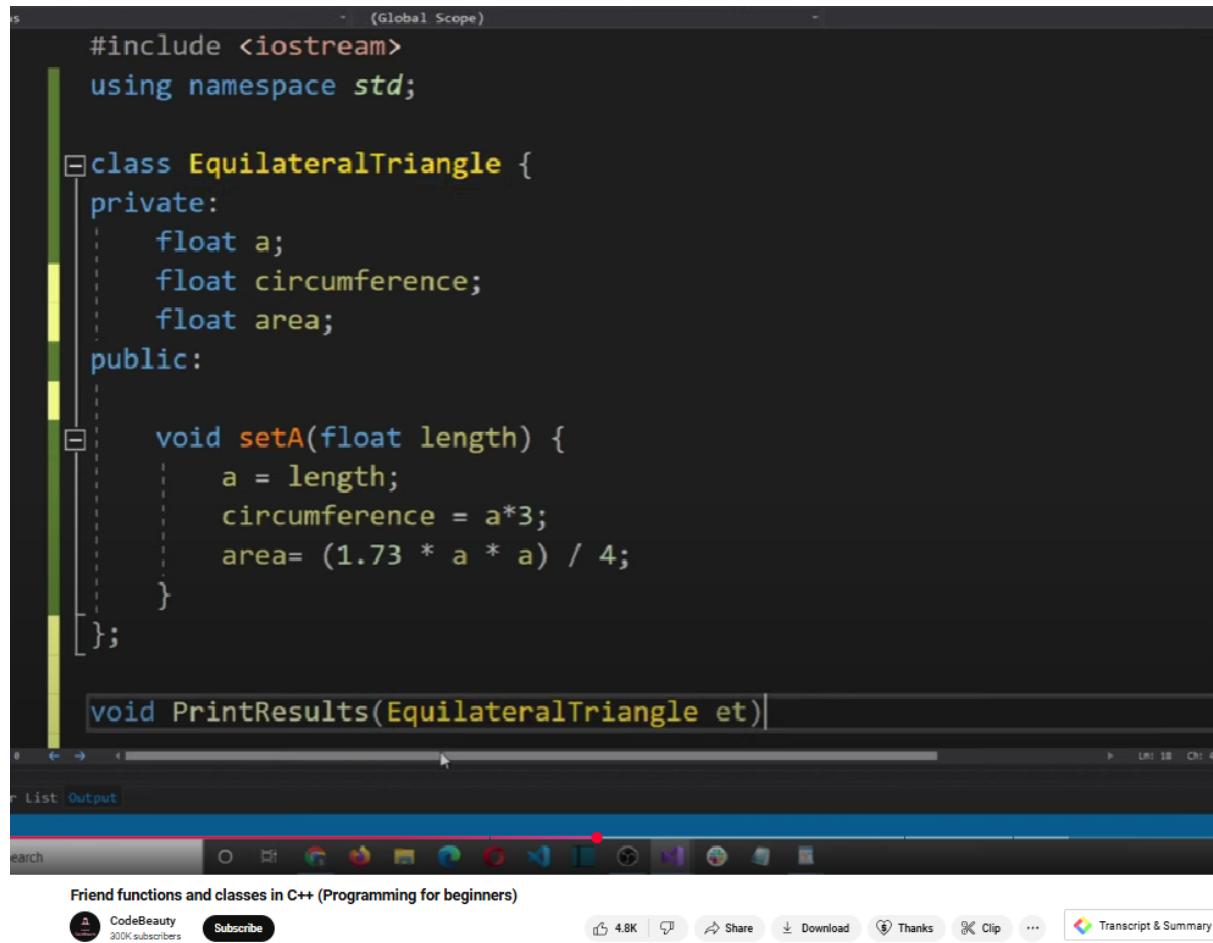
```
5   class ABC {
6   public:
7       ABC() {
8           cout << "constructor\n";
9       }
10      ~ABC() {
11          cout << "destructor\n";
12      }
13  };
14
15
16 int main() {
17     if(true) {
18         static ABC obj;
19     }
20
21     cout << "end of main fnx\n";
22     return 0;
23 }
```

PORTS PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL zsh - Ar

```
● apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
constructor
destructor
end of main fnx
● apnacollege@Amans-MacBook-Pro ApnaCollege % g++ -std=c++11 oops.cpp && ./a.out
constructor
destructor
end of main fnx
● apnacollege@Amans-MacBook-Pro ApnaCollege %
```

## Friend function and friend class

Similar to real world friend will have access to private properties of class



The screenshot shows a video player interface with a code editor window. The code editor displays the following C++ code:

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

class EquilateralTriangle {
private:
    float a;
    float circumference;
    float area;
public:

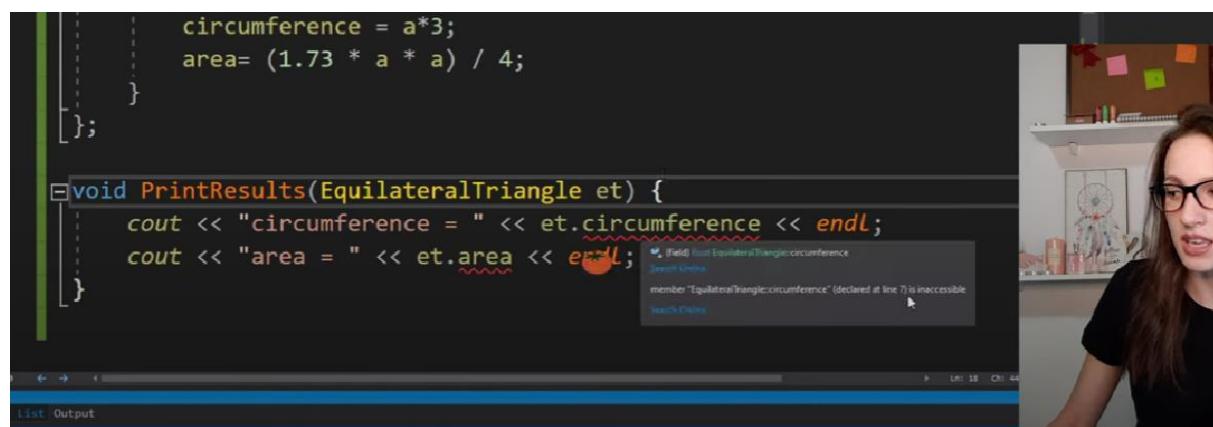
    void setA(float length) {
        a = length;
        circumference = a*3;
        area= (1.73 * a * a) / 4;
    }
};

void PrintResults(EquilateralTriangle et)
```

The video feed in the bottom right corner shows a person with long dark hair and glasses, wearing a black t-shirt, sitting at a desk with a corkboard in the background.

## Other function

With

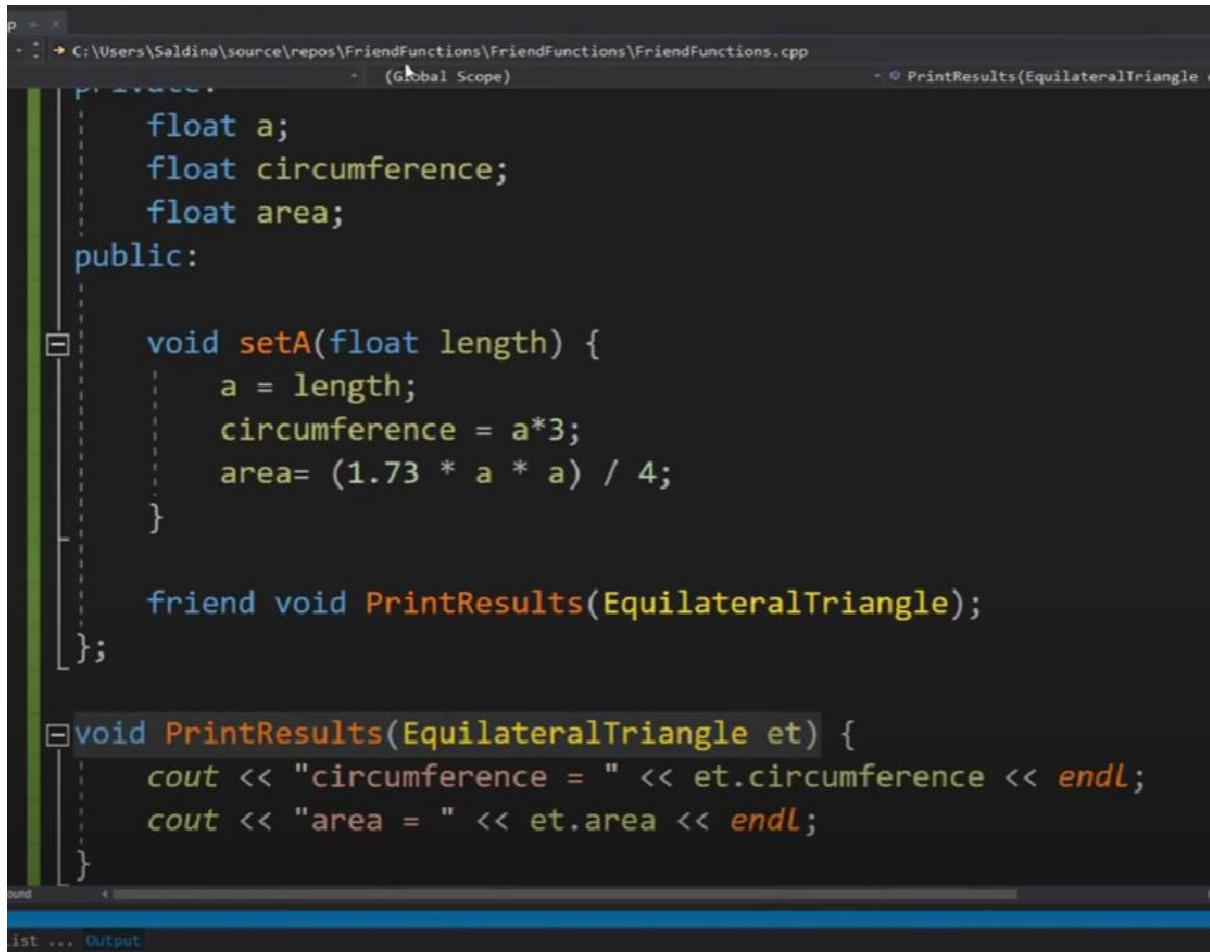


The screenshot shows a video player interface with a code editor window. The code editor displays the same C++ code as before, but with a tooltip appearing over the line `cout << "circumference = " << et.circumference << endl;`. The tooltip content is:

- Field from EquilateralTriangle::circumference
- Search Member
- member "EquilateralTriangle::circumference" (declared at line 7) is inaccessible
- View Decl

The video feed in the bottom right corner shows the same person as in the previous screenshot.

Write friend and just copy invocation of that



```
float a;
float circumference;
float area;
public:

void setA(float length) {
    a = length;
    circumference = a*3;
    area= (1.73 * a * a) / 4;
}

friend void PrintResults(EquilateralTriangle);

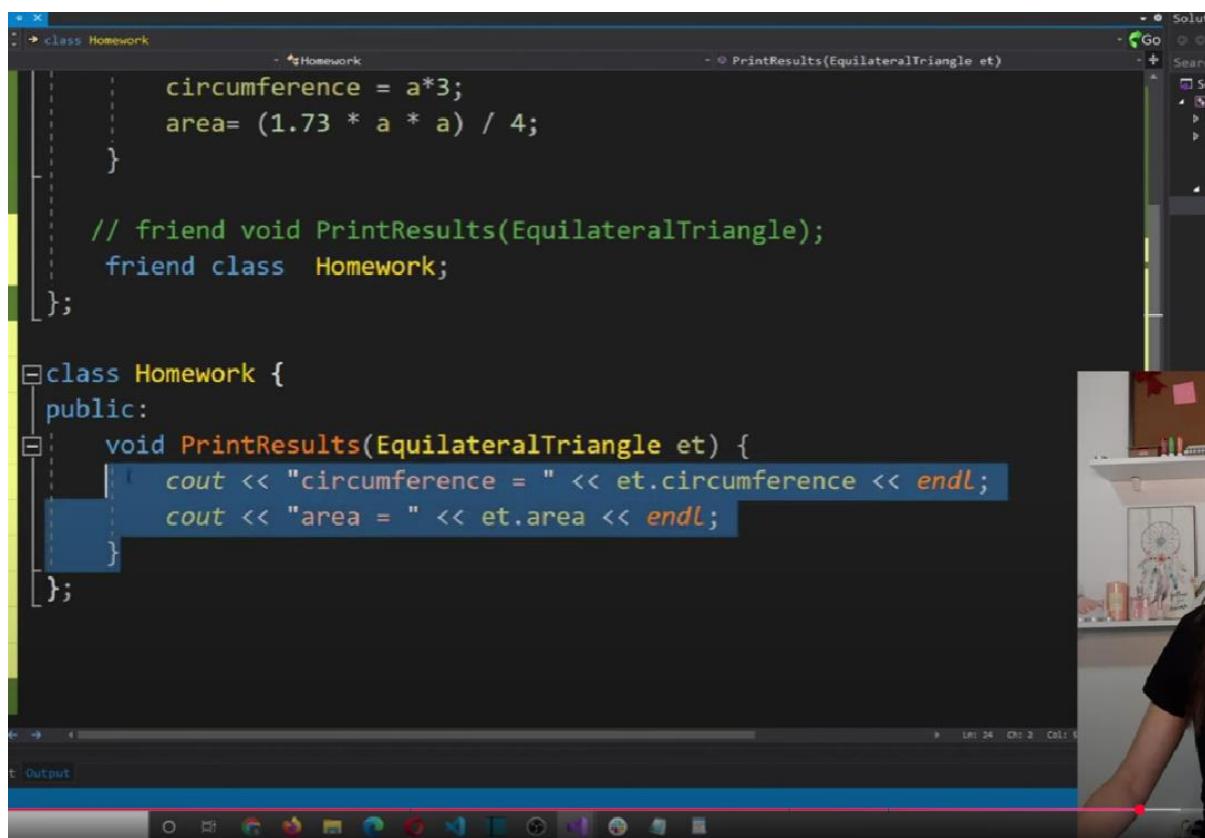
};

void PrintResults(EquilateralTriangle et) {
    cout << "circumference = " << et.circumference << endl;
    cout << "area = " << et.area << endl;
}
```

We have access to private properties called circumference and area in other func called printresult because we have made that function friend of our class

We don't have access to these private properties outside friend function

## Friend class



The screenshot shows a video player interface with a C++ code editor in the foreground. The code defines a class `Homework` with a friend function `PrintResults`. The friend function is highlighted in blue. A person's arm is visible on the right side of the screen.

```
class Homework {
public:
    void PrintResults(EquilateralTriangle et) {
        cout << "circumference = " << et.circumference << endl;
        cout << "area = " << et.area << endl;
    }
};

// friend void PrintResults(EquilateralTriangle);
friend class Homework;
```

Friend functions and classes in C++ (Programming for beginners)

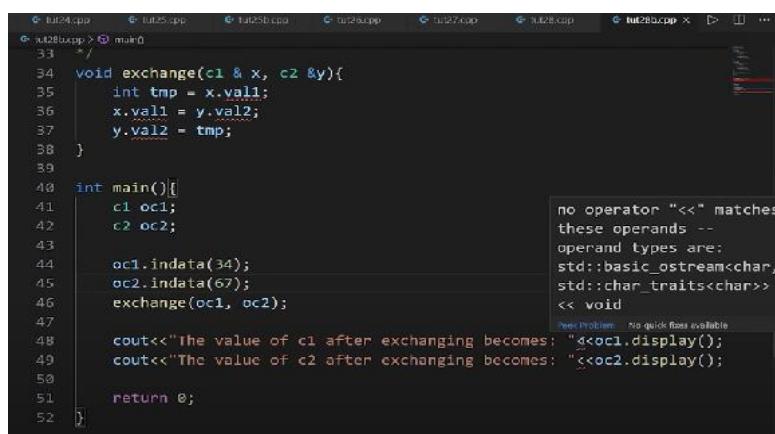
CodeBeauty Subscribe 4.8K Share Download Thanks Transcript & Summary

Also friendship in programming is not mutual and

Friend is not bydefault inherited

Used in operator overloading

All members of struct are public by default and private in class



The screenshot shows a video player interface with a C++ code editor in the foreground. A tooltip provides information about the `<<` operator. A person's arm is visible on the right side of the screen.

```
void exchange(c1 &x, c2 &y){
    int tmp = x.val1;
    x.val1 = y.val2;
    y.val2 = tmp;
}

int main(){
    c1 oc1;
    c2 oc2;

    oc1.indata(34);
    oc2.indata(67);
    exchange(oc1, oc2);

    cout<<"The value of c1 after exchanging becomes: "<<oc1.display();
    cout<<"The value of c2 after exchanging becomes: "<<oc2.display();

    return 0;
}
```

no operator "less than less than" matches these operands  
--  
operand types are:  
`std::basic_ostream<char,>`  
`std::char_traits<char>>`  
`<< void`

STL

Introduction to STL → Competitive Programming

STL → Standard template library

Library of what?

↓

→ Generic Classes and functions.

Why use STL? → Reuse: Well tested Components

→ Time Savings!

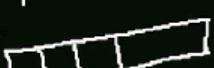
Limited time → Resize  
→ Sort  
→ Search

Stick figure with arrows pointing to 'Resize', 'Sort', and 'Search'.

History: { HP  
→flex.  
→Meng }

Introduction to STL

Components of STL → ① Containers →  
② Algorithms →  
③ Iterators →

Vector: 

→ Stores data  
→ use template classes

Sorting  
Searching  
use template functions

→ Object points to an element in a Container  
→ Handled just like pointers  
→ Connects Algo. with Containers.

STL is used because its a good idea not to Re-invent the wheel.



STL = Containers + Algo + Iterators

Sanje

Object which stores data

Procedure to process data

Object which points to an element of a container

### Containers

- ① → Sequence Containers → Linear fashion      ↗ ② → ④ → ⑩ → ① ↗  
→ Vector  
→ List  
→ Dequeue
- ② → Associative Containers → Direct Access      ↗ ⑨ ↗ Tree ↗  
→ Set / Multiset  
→ Map / Multimap
- ③ → Derived Containers → Real world modelling  
→ Stack ✓  
→ queue ✓  
→ priority\_queue



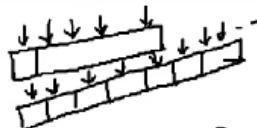
12:31 / 23:22

Screenshot

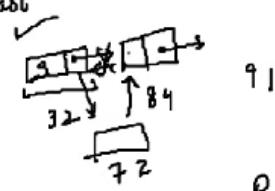
When to use which?

### Sequence Containers

1 → Vector → RA = Fast  
M: Insertion / Del > Slow  
Del / Insertion at the end > fast



2 → List → RA → Slow  
M: Insertion → Fast  
Del / Ins at the end → fast



Data Structure

Associative Containers → All operations fast  
except RA ✓

Rashmi  
Raman  
Rohit

## Functors

The screenshot shows a Visual Studio Code interface. In the top bar, there are tabs for 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. The active tab is 'tut74.cpp'. Below the tabs, there is a code editor with the following content:

```
8     // Function Objects: Function wrapped in a class so that it available like an object
9     int arr[] = {1, 73, 4, 2, 54, 16};
10    sort(arr, arr+6);
11    // sort(arr, arr+6, greater<int>());
12    for (int i = 0; i < 6; i++)
13    {
```

In the terminal below, the command `g++ tut74.cpp -o tut74` is run, followed by `./tut74`. The output is:

```
1
PS D:\MyData\Business\code playground\C++ course> cd "D:\MyData\Business\code playground\C++ course"
4 ) ; lf ($?) { ./tut74 }
1
2
4
16
54
73
this is not a function, this is an object
PS D:\MyData\Business\code playground\C++ course>
```

A red circle highlights the error message 'this is not a function, this is an object'.

Function Objects (Functors) In C++ STL | C++ Tutorials for Beginners #74

A functor (or function object) is a C++ class that acts like a function. Functors are called using the same old function call syntax. To create a functor, we create a object that overloads the operator().

```
The line,
MyFunctor(18);

is same as
MyFunctor.operator()(18);
```

Let's delve deeper and understand how this can actually be used in conjunction with STLs.

CPP

```
// C++ program to demonstrate working of
// functors.
#include <iostream>
using namespace std;

// A Functor
class increment
{
private:
    int num;
public:
    increment(int n) : num(n) { }

    // This operator overloading enables calling
    // operator function () on objects of increment
    int operator () (int arr_num) const {
        return num + arr_num;
    }
};

// Driver code
int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int n = increment(arr)/increment(arr[0]);
    int to_add = 5;

    transform(arr, arr+n, arr, increment(to_add));

    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}
```

Output:

```
6 7 8 9 10
```

Time Complexity: O(n)

Space Complexity: O(n) where n is the size of the array.

Smart pointer aur arrow function

Dangling pointer wagera

Virtual base

<https://www.geeksforgeeks.org/cpp-pointers/>

<https://www.geeksforgeeks.org/cpp-interview-questions/?ref=lbp>

<https://www.geeksforgeeks.org/cpp-stl-interview-questions/?ref=lbp>

<https://www.geeksforgeeks.org/oops-interview-questions/?ref=lbp>

<https://www.geeksforgeeks.org/cpp-interview-questions/?ref=lbp>

<https://www.geeksforgeeks.org/smarter-pointers-cpp/?ref=lbp>