



# C++ Programming

Trainer : Pradnyaa S Dindorkar

Email: [pradnya@sunbeaminfo.com](mailto:pradnya@sunbeaminfo.com)



# We did.....

---

1. Deep copy
2. Static
3. Friend



- 
1. Operator overloading
  2. OOP and its pillars
  3. Association



# Operator Overloading

- operator is token in C/C++.
- It is used to generate expression.
- operator is keyword in C++.
- Types of operator:
  - Unary operator ( ++,--,&!,~,sizeof())
  - Binary Operator (Arithmetic, relational, logical , bitwise, assignment)
  - Ternary operator (conditional)
- In C++, also we can not use operator with objects of user defined type directly.
- If we want to use operator with objects of user defined type then we should overload operator.
- To overload operator, we should define **operator function**.
- **We can define operator function using 2 ways:**
  - Using member function
  - Using non member function



# Need Of Operator Overloading

- we extend the meaning of the operator.
- If we want to use operator with the object of use defined type, then we need to overload operator.
- To overload operator, we need to define operator function.
- In C++, operator is a keyword
  - Suppose we want to use plus(+) operator with objects then we need to define operator+( ) function.

We define operator function either inside class (as a member function) or outside class (as a non-member function).

```
Point pt1(10,20), pt2(30,40 ), pt3;
```

```
pt3 = pt1 + pt2; //pt3 = pt1.operator+( pt2); //using member function
```

```
//or
```

```
pt3 = pt1 + pt2; //pt3 = operator+( pt1, pt2); //using non member function
```



# Operator Overloading

## using member function

- **operator function must be member function**
- If we want to overload, binary operator using member function then **operator function should take only one parameter.**
  - Example :  $c3 = c1 + c2$ ; //will be called as -  
-----  $c3 = c1.operator+( c2 )$

Example :

```
Point operator+( Point &other ) //Member Function
```

```
{  
    Point temp;  
    temp.xPos = this->xPos + other.xPos;  
    temp.yPos = this->yPos + other.yPos;  
    return temp;  
}
```

## using non member function

- **Operator function must be global function**
- If we want to overload binary operator using non member function then **operator function should take two parameters.**
  - **Example :**  $c3 = c1 + c2$ ; //will be called as -  
----- $c3 = operator+(c1,c2)$ ;

Example:

```
Point operator+( Point &pt1, Point &pt2 ) //Non Member  
Function
```

```
{  
    Point temp;  
    temp.xPos = pt1.xPos + pt2.xPos;  
    temp.yPos = pt1.yPos + pt2.yPos;  
    return temp;  
}
```



# We can not overloading following operator using member as well as non member function:

1. dot/member selection operator( . )
2. Pointer to member selection operator(.\*)
3. Scope resolution operator( :: )
4. Ternary/conditional operator( ? : )
5. sizeof() operator
6. typeid() operator
7. static\_cast operator
8. dynamic\_cast operator
9. const\_cast operator
10. reinterpret\_cast operator



# We can not overload following operators using non member function:

- Assignment operator( = )
- Subscript / Index operator( [] )
- Function Call operator[ ( ) ]
- Arrow / Dereferencing operator( → )





# Object Oriented programming structure(oops) :-

-> It is a programming methodology to organise complex program into simple program in terms of class and objects such methodology is called as "Object Oriented programming structure"

-> It is a programming methodology to organise complex program into simple program by using the concept of Abstraction, Encapsulation and Inheritance, modularity.

->so the language which supports Abstraction, Encapsulation and Inheritance is called as Object Oriented programming language.



# Major pillars of oops

- **Abstraction**

- getting only essential things and hiding unnecessary details is called as abstraction.
- Abstraction always describe outer behavior of object.
- In console application when we give call to function in to the main function , it represents the abstraction

- **Encapsulation**

- binding of data and code together is called as encapsulation.
- Implementation of abstraction is called encapsulation.
- Encapsulation always describe inner behavior of object
- Function call is abstraction
- Function definition is encapsulation.
- Information hiding
  - Data : unprocessed raw material is called as data.
  - Process data is called as information.
  - Hiding information from user is called information hiding.
  - In c++ we used access Specifier to provide information hiding.

- **Modularity**

- Dividing programs into small modules for the purpose of simplicity is called modularity.

- **Hierarchy (Inheritance [is-a] , Composition [has-a] , Aggregation[has-a], Dependency)**

- Hierarchy is ranking or ordering of abstractions.
- Main purpose of hierarchy is to achieve re-usability.



# Minor pillars of oops

- **Polymorphism (Typing)**

- One interface having multiple forms is called as polymorphism.
- Polymorphism have two types

1. **Compile time polymorphism**

when the call to the function resolved at compile time it is called as compile time polymorphism. And it is achieved by using function overloading and operator overloading

2. **Runtime polymorphism.**

when the call to the function resolved at run time it is called as run time polymorphism. And it is achieved by using function overriding.

- Compile time / Static polymorphism / Static binding / Early binding / Weak typing / False Polymorphism
- Run time / Dynamic polymorphism / Dynamic binding / Late binding / Strong typing / True polymorphism

- **Concurrency**

- The concurrency problem arises when multiple threads simultaneously access same object.
- You need to take care of object synchronization when concurrency is introduced in the system.

- **Persistence**

- It is property by which object maintains its state across time and space.
- It talks about concept of serialization and also about transferring object across network.



# Association

- If has-a relationship exist between two types then we should use association.
- Example : Car has-a engine (OR engine is part-of car)
- If object is part-of / component of another object then it is called association.
- If we declare object of a class as a data member inside another class then it represents association.
- Example Association:

```
class Engine
```

```
{   int cc, fuel;  };
```

```
class Car
```

```
{           private:
```

```
    Engine e;  //Association
```

```
};
```

```
int main( void )
```

```
{   Car car;
```

```
    return 0;
```

```
}
```

```
//Dependant Object : Car Object
```

```
//Dependency Object : Engine Object
```



# Composition and aggregation are specialized form of association

## Composition

- If dependency object do not exist without Dependant object then it represents composition.
- Composition represents tight coupling.
- Example: Human has-a heart.

```
class Heart
```

```
{ };
```

```
class Human
```

```
{ Heart hrt; //Association->Composition
```

```
};
```

```
int main( void )
```

```
{ Human h;
```

```
    return 0;
```

```
}
```

- //Dependant Object : Human Object
- //Dependency Object : Heart Object

## Aggregation

- If dependency object exist without Dependant object then it represents Aggregation.
- Aggregation represents loose coupling.

```
class Faculty
```

```
{ };
```

```
class Department
```

```
{
```

```
    Faculty f; //Association->Aggregation
```

```
};
```

```
int main( void )
```

```
{
```

```
    Department d;
```

```
    return 0;
```

```
}
```

- //Dependant Object : Department Object
- //Dependency Object : Faculty Object



---

# Thank You

