CAP444
OBJECT ORIENTED PROGRAMMING USING C++

Unit- 2

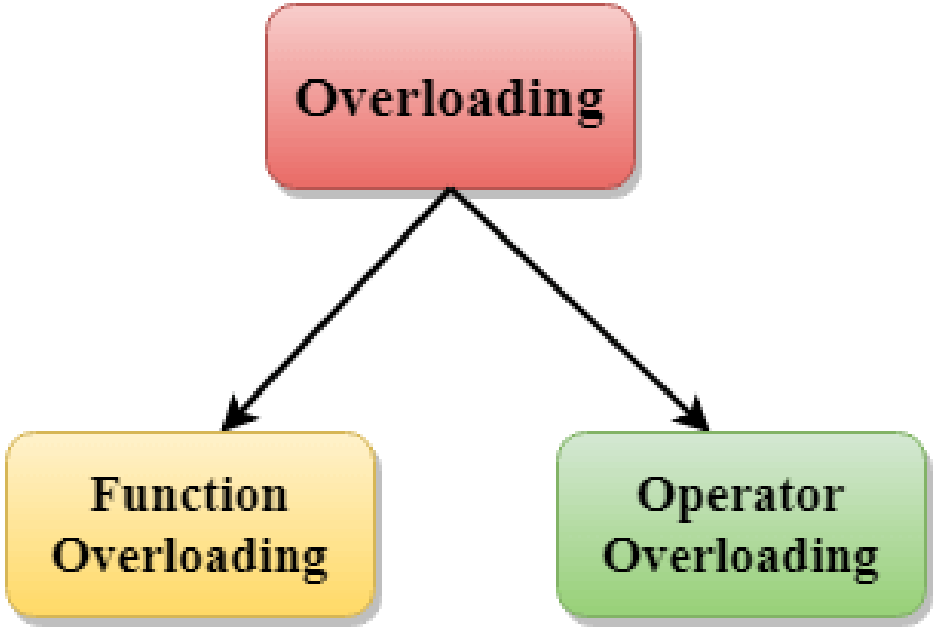
Created By:
Kumar Vishal
(SCA), LPU

Topics Covered:

Operator overloading and type conversions :

- rules for operator overloading,
- overloading unary operators,
- overloading binary operators,
- overloading binary operators using friend function,
- type conversions: basic to class type,
- class to basic type,
- one class to another class type

Polymorphism



```
#include <iostream>
 using namespace std;
class Teacher
 {
    public:
    string name;
     Teacher()
     {
        name="kumar";
      }
     Teacher(string str1)
     {
        name=str1;
     }
   void getName()   {
      cout<<name;   }
```

```
int main()
 {
  Teacher T1("vishal"),T2("Ajay"),T3;
  T3=T1+T2;
  T3.getName();
  return 0;
 }
```

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.

- You can redefine built in operators except few:
  - Scope operator (::)
  - Sizeof
  - member selector(.)
  - member pointer selector(.*)
  - ternary operator(?:)

Operators which can overload

+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

Operator Overloading Syntax:

Return_type operator operator_Symbol(parameters)

{

}

Example:

```
Teacher operator+(Teacher &t)
   {
      return name+t.name;
    }
```

- Unary operators operate on only one operand

Ex:

i++

- Binary operators work on two operands

Ex:

+

Which of the following operator cannot be overloaded?

a) +

b) ?:

c) –

d) %

What is a binary operator?
a) Operator that performs its action on a single operand
b) Operator that performs its action on two operand
c) Operator that performs its action on three operand
d) Operator that performs its action on any number of operands

Operator overloading for Unary operators:

The unary operators operate on a single operand :

- The increment (++) and decrement (--) operators.
- The unary minus (-) operator.
- The logical not (!) operator.

Rules for Operator Overloading

- Existing operators can only be overloaded.
- The overloaded operator contains at least one operand of the user-defined data type.
- When unary operators are overloaded through a member function take no explicit arguments, but, if they are overloaded by a friend function, takes one argument.
- When binary operators are overloaded through a member function takes one explicit argument, and if they are overloaded through a friend function takes two explicit arguments.

Friend function

- It can access all private and protected member of a class
- It can be call without object of the class
- It can define out side of the class scope

Rule:

Prototypes of friend function must be declare inside the class

It can be declared either in the private or the public part.

Simple example : friend function

```
#include <iostream>
using namespace std;
class A
{
private:
   int x;
public:
   A()
   {
      x=10;
   }
private:
   friend void newfriend(A &a);
};
void newfriend(A &a)
{
   a.x=20;
   cout<<a.x;
}
int main()
{
A a1;
newfriend(a1);
return 0;
}
```

Overloading binary operators using friend function

Friend function takes two parameters in case when we want to overload binary operators using friend function

Ex:

friend A operator +(A &x, A &y);

Example:

What will be output for following code?

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

class Sub
{
private:
   int a;
   int b;
public:
   Sub()
   {
      a=10;
      b=20;
   }
friend Sub operator -(Sub &x, Sub &y);
   void getresult()
   {
      cout<<a<<endl;
      cout<<b<<endl;
   }
};
```

```
Sub operator -(Sub &x,Sub &y)
{
      Sub z;
      z.a=x.b-x.a;
      z.b=y.b-y.a;
      return z;
}
int main()
{
  Sub a1,a2,a3;
  a3=a1-a2;
  a3.getresult();
  return 0;
}
```

A.
10
10

B.
-10
-10

C.
0
0

D. None

Situation??

Type Conversion

- Basic data types conversion done automatic by compiler
- User define data type conversion not done automatically
- User define data type conversion done by using either constructor or by using casting operator

What will be output?

```
#include <iostream>
 using namespace std;
 int main()
 {
    double a = 21.09399;
    float b = 10.20;
    int c ;
    c = a;
    cout << c ;
    c = b;
    cout << c ;
    return 0;
 }
```

A) 2110
B) 1210
C) 21
D) 121

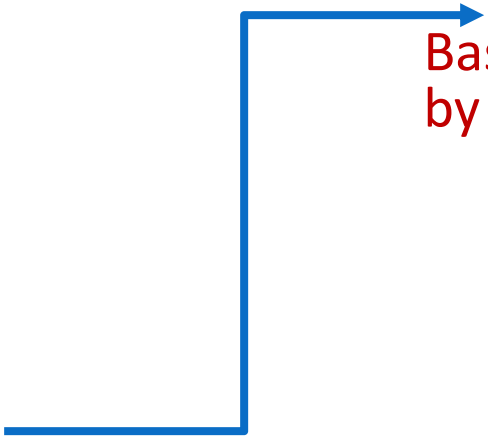
Three type of situation occurs during user define type conversion:

- 1. basic type to class type(using constructor)
- 2. class type to basic type(using casting operator function)
- 3. class type to class type (using constructor and casting operator function both)

basic type to class type(using constructor)

```
#include <iostream>
using namespace std;
class A
{

};
int main()
{
A a1;
int  x=8;
a1=x ;//basic to class type
   return 0;
}
```



Basic type to class type achieved by using constructor.

class type to basic type(using casting operator function)

Class type to basic type done by using casting operator function

1. It must be a define inside in class.
2. It must not specify a return type in function signature.
3. **It must not have any arguments.**

```
class A
{};
A a1;
int  x;
x=a1 //class type to basic type
```

casting operator function

Syntax:

```
operator dest_typename()
{
  return type;
}
```

```
operator int()
{
  return a;
}
```

class type to class type (using constructor and casting operator function both)

Ex: A obj1; B obj2;

obj1 = obj2 ; // obj1 and obj2 are objects of different classes

- First approach using Constructor:-

Left side of assignment operator(=) which is class object we have to create constructor in that class here in Class A.

- Second approach using casting operator function:

Right side of assignment operator(=) which is class object we have to create casting operator function in that class here class B.

What will be out put for the following code?

```
 #include <iostream>
using namespace std;
class Circle
{
   int radius;
    Circle(){}

   Circle(int radius)
   {
    this->radius=radius;
    cout<<this->radius;
   }
};
```

```
int main()
{
   Circle a1, b1(5);
   Circle b2 = Circle(8);
   return 0;
}
```

A. 55
B. 88
C. 58
D. Error

What will be out put for the following code?

```
 #include <iostream>
using namespace std;
class Circle
{
   int radius;
public:
    Circle(){}

   Circle(int radius)
   {
    this->radius=radius;
    cout<<this->radius;
   }
};
```

```
int main()
{
   Circle a1, b1(5);
   Circle b2 = Circle(8);
   return 0;
}
```

A. 55
B. 88
C. 58
D. Error

Any Query?