IT206 Data Structures Lab with OOP

Rachit Chhaya

DA-IICT

May 4, 2022

Operator Overloading

int a
a++;

- Giving Additional meaning to an operator

Michael Goodrich

Operator overloading

Function Overloading

compile time → run time

Operator Overloading

- Giving Additional meaning to an operator
- One of the ways in C++ to implement polymorphism

Operator Overloading

- Giving Additional meaning to an operator
- One of the ways in C++ to implement polymorphism
- Semantics of operator can be extended but syntax cannot be changed.

+

Operator Overloading

- Giving Additional meaning to an operator
- One of the ways in C++ to implement polymorphism
- Semantics of operator can be extended but syntax cannot be changed.
- Original meaning of operator is preserved.

Operator Overloading

- Giving Additional meaning to an operator
- One of the ways in C++ to implement polymorphism
- Semantics of operator can be extended but syntax cannot be changed.
- Original meaning of operator is preserved.
- Some operators cannot be overloaded.

sizeof(int)

- Class member access operators (., .*).
- Scope resolution operator (::).
- Size operator (**sizeof**).
- Conditional operator (?:).

___ ? ___ : ___

Defining Operator Overloading

add
add(2,3)
add(a1,a2)
a1.add(a2)

```
return type classname :: operator op(arglist)
{
        Function body              // task defined
}
```

- Operator functions - either member functions or Friend functions

Defining Operator Overloading

```
return type classname :: operator op(arglist)
{
        Function body              // task defined
}
```

- Operator functions - either member functions or Friend functions
- Friend function- 1 argument for unary operator, two for binary operator.

Defining Operator Overloading

```
return type classname :: operator op(arglist)
{
        Function body              // task defined
}
```

- Operator functions - either member functions or Friend functions
- Friend function- 1 argument for unary operator, two for binary operator.
- Member function - no argument for unary operator and one for binary operator.

Steps for Operator Overloading

1. Create class for data type to be used in operator overloading.

Steps for Operator Overloading

1. Create class for data type to be used in operator overloading.
2. Declare operator function in public section of class.

Steps for Operator Overloading

1. Create class for data type to be used in operator overloading.
2. Declare operator function in public section of class.
3. Define the operator function

Unary Operator Overloading

-a;

c = -a;

```
// C++ program to show unary operator overloading
#include <iostream>

using namespace std;

class Distance {
public:

    // Member Object
    int feet, inch;

    // Constructor to initialize the object's value
    Distance(int f, int i)
    {
        feet = f;
        inch = i;
    }
```

```
// Overloading(-) operator to perform decrement
// operation of Distance object
void operator-()
{
    feet--;
    inch--;
    cout << "\nFeet & Inches(Decrement): " << feet
    << "'" << inch;
}
};

// Driver Code
int main()
{
    Distance d1(8, 9);
    -d1;
    return 0;
}
```

Operator function

decrement()

d1.feet

d1.feet = 8

d1.inch = 9

≈ d1.decrement();

-d1

Unary Operator Overloading using Friend Function

```
#include <iostream>
using namespace std;
class UnaryFriend
{
    int a=10;
    int b=20;
    int c=30;
    public:
        void getvalues()
        {
            cout<<"Values of A, B & C\n";
            cout<<a<<"\n"<<b<<"\n"<<c<<"\n"<<endl;
        }
        void show()
        {
            cout<<a<<"\n"<<b<<"\n"<<c<<"\n"<<endl;
        }
        void friend operator-(UnaryFriend &x);
```

```
void operator-(UnaryFriend &x)
{
      x.a = -x.a;
      x.b = -x.b;
      x.c = -x.c;
}
int main()
{
      UnaryFriend x1;
      x1.getvalues();
      cout<<"Before Overloading\n";
      x1.show();
      cout<<"After Overloading \n";
      -x1;
       x1.show();
       return 0;
}
```

10 => -10
20 => -20
30 => -30

-x1

x1.decrement( ) - 30

- 10
- 20

Binary Operator Overloading

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

class Complex {
   private:
    float real;
    float imag;

   public:
    // Constructor to initialize real and imag to 0
    Complex() : real(0), imag(0) {}

    void input() {
        cout << "Enter real and imaginary parts: ";
        cin >> real;
        cin >> imag;
    }
```

complex Friend operator
+ (complex &a,
complex &b);

add

complex &obj

```
    // Overload the + operator
    Complex operator + (const Complex& obj) {
        Complex temp;
        temp.real = real + obj.real;
        temp.imag = imag + obj.imag;
        return temp;
    }

void output() {
    if (imag < 0)
     cout << " Complex number: " << real << imag << "i";
    else
    cout << " Complex number: " << real << "+" << imag << "
    }
};
```

```
int main() {
Complex complex1, complex2, result;

cout << "Enter first complex number:\n";
complex1.input();

cout << "Enter second complex number:\n";
complex2.input();

// complex1 calls the operator function
// complex2 is passed as an argument to the function
result = complex1 + complex2;
result.output();

return 0;
}
```

=> result = add (complex1, complex2)

overloaded

How to do using Friend Function?

Rules For Operator Overloading

- Only existing operators overloaded.

Rules For Operator Overloading

- Only existing operators overloaded.
- One operand must be of user defined type ← you need a class

Rules For Operator Overloading

Type conversion

Type cast

- Only existing operators overloaded.
- One operand must be of user defined type
- Some operators can not be overloaded and some cannot use friend function (= ,[ ],(), ->)

assignment

a = float(b) + c;

float a;
int b = 5;
float c = 7.3;

a = b + c;

12.3
12.0

a, c ← int
b ← float

Type Conversions

Basic to Class type

Type Conversions

Basic to Class type

```
class time
{
    int hrs;
     int mins;
  public:
     ....
     ....
     time(int t)                    // constructor
     {
          hours = t/60;             // t in minutes
          mins  = t%60;
     }
};
```

Class time
{ int hrs;
mins

using constructor

time(int t)
{
hours = t/60;
mins = t%60
}

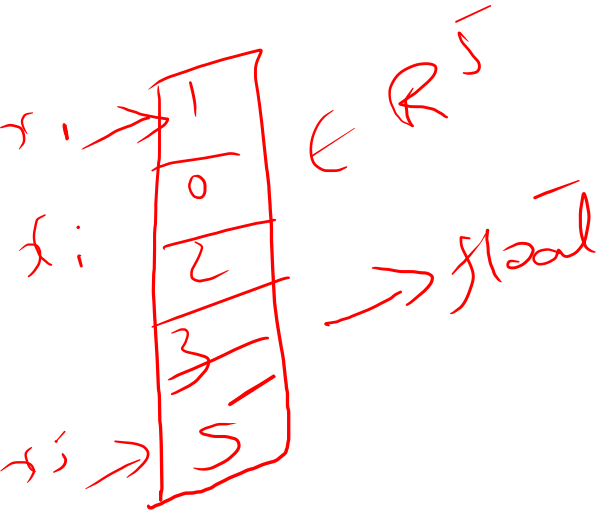
The following conversion statements can be used in a function:

```
time T1;                            // object T1 created
int duration = 85;
T1 = duration;                      // int to class type
```

Class to Basic

```
operator typename()
{
        .....
        ..... (Function statements)
        .....
}
```

Class to Basic



$$\sqrt{\sum_{i=1}^{n} x_i^2}$$

```
operator typename()
{
        .....
        ..... (Function statements)
        .....
}
```

```
vector :: operator double()
{
  double sum = 0;
  for(int i=0; i<size; i++)
        sum = sum + v[i] * v[i];
  return sqrt(sum);
}
```

Class to Basic

```
operator typename()
{
        .....
        ..... (Function statements)
        .....
}
```

```
vector :: operator double()
{
   double sum = 0;
   for(int i=0; i<size; i++)
          sum = sum + v[i] * v[i];
   return sqrt(sum);
}
```

Casting Operator Function must be: class member, no return type, no arguments