Unit 4: Operator Overloading [7hrs]

Fundamentals of Operator overloading

It is one of the important features of C++ language.

→ C++ has the ability to provide the operators with a special meaning for a data type. The mechanism of giving such special meanings to an operator is known as operator overloading.

→ Operator overloading provides a flexible option for the creation of new definitions for most of the C++ operators.

We can overload (give additional meaning to) all the C++ operators except the following:

- class member access operators (.)
- Scope resolution operator (::)
- Size of operator (sizeof)
- Conditional operator (?)

→ Operator overloading is done with the help of special function called “operator” function.

Defining operator overloading:

```
return –type class name :: operator op (arg_list)
{
Function body
}
```

Here,
op = operator being overloaded

1. Overloading unary minus (-) operator:

```
#include<iostream>
using namespace std;
class  space
{
    int x;
    int y;
    int z;
public:
    void getdata (int a, int b, int c);
    void display (void);
    void operator - (); // overloaded unary minus
};
void space:: getdata (int a, int b, int c)
        {
        x = a;
        y = b;
        z = c;
        }
void space :: display (void)
    {
    cout <<x<< "  " ;
    cout << y << "  " ;
    cout<< z << "\n" ;
    }
void space :: operator - () // here operator is a keyword
    {
    x = -x;
    y = -y;
    z = -z;
    }
int main ()
{
        space s;
    s.getdata (10, -20, 30);
    cout << "S:";
    s.display();
    -s;   // activates operator-() function
    cout << "S:";
    s.display ();
}
```

Output:

```
S:     10      -20      30
S:     -10     20       -30
```

2. Overloading binary operator:

Example 1

```
# include<iostream>
using namespace std;
class  complex
{
    float  x;
    float  y;
public:
    complex () {}
    complex (float real, float imag)
    {
    x = real; y = imag;
    }
complex operator + (complex);
void display();
};
complex complex :: operator + (complex c)
    {
    complex temp;
    temp.x = x + c.x;
    temp.y = y + c.y;
    return (temp);
    }
void complex :: display (void)
    {
    cout<< x << "+j" <<y<<"\n";
    }
int main ()
{
    complex C1, C2, C3; // invokes constructor1
    C1 = complex (2.5,3.5); // invokes constructor 2
    C2 = complex (1.6, 2.7);
    C3 = C1 + C2; // activates operator + () function
    cout<<"C1= " ;   C1.display();
    cout<<"C2= " ;    C2.display();
    cout<<"C3= ";    C3.display ();
}
```

Output:

```
C1 = 2.5 + j3.5
C2 = 1.6 + j2.7
C3 = 4.1 + j6.2
```

Example 2

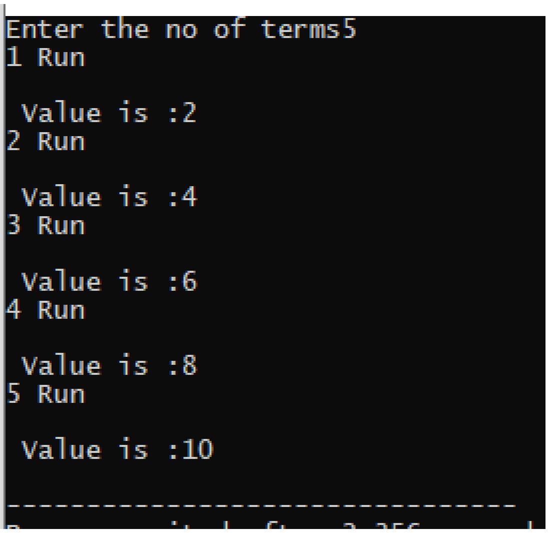
```
#include<iostream>
using namespace std;
class overloading
{
 int value;
 public:
 void setValue(int temp)
 {      value = temp;    }
 overloading operator+(overloading ob)
  {
  overloading t;
 t.value=value+ob.value;
  return(t);
  }
void display()
{
 cout<<"\n Value is :"<<value<<endl;
}
};

int main()
{
   overloading obj1,obj2,result;
              int n;
   cout<<"Enter the no of terms";
   cin>>n;

             for(int i=1;i<=n;i++)
{   cout<<i<<" Run\n";
                              obj1.setValue(i);
                             obj2.setValue(i);
                           result = obj1 + obj2;
                            result.display();
 }

        return 0;
   }
```

Output



```
Enter the no of terms5
1 Run

 Value is :2
2 Run

 Value is :4
3 Run

 Value is :6
4 Run

 Value is :8
5 Run

 Value is :10

--------------------------------
```

Data Conversion

Type conversion refers to changing an entity of one data type, expression, function agreement or return value into another.

1) Basic to Basic

```
#include<iostream>
```

```
using namespace std;
        int main()
        {
        float b = 4.4;
        int c;
        c = (int)b;
        cout<<c;
        return 0;
        }
```

2) Basic to user-defined

```
#include<iostream>
using namespace std;
    class X
    {
        int z;
        char y;
        public:
                X() { }
                X (char p)
                {
                z = (int)p;
                y = p;
                }
        void show()
        {
        cout<<z<<y;
        }
    };

    int main ()
    {

        char s = 'a';
        X x1;
         x1 = s;           // calls parameterized constructor. 's' is basic  type and x1 is class type.
         x1.show();
    return 0;
    }
```

3) User-defined to Basic

```
#include<iostream>
```

```
#include<math.h>
using namespace std;
class Hour
{
        int hr;
        public:
        Hour()  { }
        operator int()
        {
                int minute;
                minute= hr * 60;
                return (minute);
        }
        void getdata()
        {
                cout<<"Enter Hours";
                cin>>hr;
        }

};
int main()
{
        Hour h1;
        float min;
        h1.getdata();
        min = h1; //user defined type to basic
        cout<<"Minutes = "<<min;
}
```

4) User -defined to User-defined

a) Class type to Class type conversion using constructor in the destination class: Rectangle to Polar

```
#include<iostream>
#include<math.h>
using namespace std;
class rectangle
{       float x,y;
        public:
        rectangle(float a, float b)
        {       x=a;
```

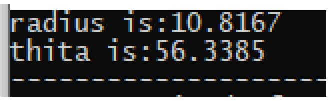
```
                y=b;
        }
        float get_x()
        {  return(x);
        }
        float get_y()
        { return(y);
        }
        };
class polar
        { float radius,thita;
        public:
        void show();
        polar(){ }
        polar(rectangle r)
        {    float tempx=r.get_x();
                float tempy=r.get_y();
                radius = sqrt(tempx*tempx + tempy*tempy);
                thita = atan(tempy/tempx);
         }
         };
         void polar :: show()
         { cout<<"radius is:"<<radius<<endl;
         cout<<"thita is:"<<thita*(180/3.14);
         }
         int main()
         {
         rectangle r(6,9);

         polar p(r);

         p.show();

         return 0;

         }
```



```
radius is:10.8167
thita is:56.3385
--------------------
```

b) Polar to rectangle conversion using casting operator in source class.

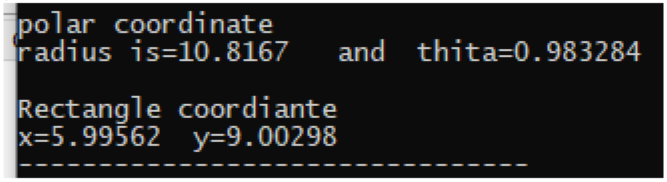
```
/* Polar to rectangle using casting operator  */
#include<iostream>
#include<math.h>
#define PI 3.141592654
using namespace std;
class rectangle   //destination class
{
float x;
float y;
public:
rectangle(){ }
rectangle(float a, float b)
{          x=a;
                              y=b;
}
void show()
           {
cout<<"x="<<x<<"  "<<"y="<<y;
           }

};

class polar   //source class
{          float radius;
           float thita;
           public:
           polar(){ radius =0.0,thita=0.0;}
           polar(float r,float t)
           {  radius=  r;
              thita= t;
```

```
        }
operator rectangle()      {

                    double a= radius * cos(thita);
                     double b= radius * sin(thita);
                     return(rectangle(a,b));
          }
          void show()
          {
                cout<<"radius is="<<radius<<"   and  "<<"thita="<<thita;
          }
};
int  main()
{
                rectangle r1;
                polar p1(10.8167,56.338*PI/180);
                r1=p1;
                cout<<"\npolar coordinate"<<endl;
                p1.show();
                cout<<"\n\nRectangle coordiante "<<endl;
                r1.show();
                return 0;
}
```



```
polar coordinate
radius is=10.8167   and  thita=0.983284

Rectangle coordiante
x=5.99562  y=9.00298
--------------------------------
```