C++ Operator Overloading (More Polymorphism)

Dr. Md. Humayun Kabir
CSE Department, BUET

Operator Overloading

- Operator overloading is a type of function overloading
- An operator is always overloaded relative to a class (an user defined data type).
- An overloaded operator gets a special meaning relative to its class. However, the operator does not loose its original meaning relative to other data types
- To overload an operator an operator function is defined for the class
- The operator function can be a member or a friend function of the class

Operator Overloading Restrictions

- You cannot overload :: (scope resolution), . (member selection), .* (member selection through pointer to function), and ? (ternary conditional) operators
- Overloading cannot change the original precedence of the operator
- The number of operands on which the operator would be applicable cannot be changed too.
- Operator functions cannot have default arguments

Operator Overloading General Form

- Prototype definition

```
class class-name{
    ……..
  return-type operator # (arg-list);
};
```

- Function definition

```
return-type class-name :: operator # (arg-list){
  //operation to be performed
}
```

Overloading Binary Operator

```
#include <iostream>
using namespace std;
class CRectangle {
    int width, height;
    public:
        CRectangle () {width=0, height=0;}
        CRectangle (int,int);
        CRectangle operator + (CRectangle rect2);
        CRectangle operator + (int);
        CRectangle operator - (CRectangle rect2);
        CRectangle operator = (CRectangle rect2);
        int area () {return (width*height);}
};
```

Overloading Binary Operator

```
CRectangle::CRectangle (int a, int b) {
    width = a; height = b;
}
CRectangle CRectangle::operator + (CRectangle
    rect2){
    CRectangle temp;
    temp.width=width+rect2.width;
    temp.height=height+rect2.height;
    return temp;
}
```

Overloading Binary Operator

```
CRectangle CRectangle::operator + (int n){
   CRectangle temp;
   temp.width=width+n;
   temp.height=height+n;
   return temp;
}
```

Overloading Binary Operator

```
CRectangle CRectangle::operator - (CRectangle rect2){
    CRectangle temp;
    temp.width=width-rect2.width;
    temp.height=height-rect2.height;
    return temp;
}
CRectangle CRectangle::operator = (CRectangle rect2){
    width=rect2.width;
    height=rect2.height;
    return *this;
}
```

Overloading Binary Operator

```
int main () {
    CRectangle recta (3,4);
    CRectangle rectb (5,6);
    CRectangle rectc;
    cout << "recta area: " << recta.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    rectc=recta+rectb;
    cout << "rectc area: " << rectc.area() << endl;
    rectc=rectb-recta;
    cout << "rectc area: " << rectc.area() << endl;
    rectc=rectb+3;
    cout << "rectc area: " << rectc.area() << endl;
    return 0;
}
```

Overloading Binary Operator

```
#include <iostream>
using namespace std;
class CRectangle {
    int *width, *height;
    public:
        CRectangle ();
        CRectangle (int,int);
        CRectangle(const CRectangle &r);
        CRectangle operator + (const CRectangle &rect2);
        CRectangle operator = (const CRectangle &rect2);
        ~CRectangle ();
        int area () {return (*width * *height);}
};
```

Overloading Binary Operator

```
CRectangle::CRectangle () {
        width = new int;
        height = new int;
        *width = 0;
        *height = 0;
}

CRectangle::CRectangle (int a, int b) {
         width = new int;
        height = new int;
        *width = a;
        *height = b;
}
```

Overloading Binary Operator

```
CRectangle::CRectangle(const CRectangle &r){
    width=new int;
    height=new int;
    *width=*r.width;
    *height=*r.height;
}

CRectangle::~CRectangle () {
    delete width;
    delete height;
}
```

Overloading Binary Operator

```
CRectangle CRectangle::operator + (CRectangle &rect2){
    CRectangle temp;
    *temp.width=*width + *rect2.width;
    *temp.height=*height + *rect2.height;
    return temp;
}
CRectangle CRectangle::operator = (CRectangle &rect2){
    *width=*rect2.width;
    *height=*rect2.height;
    return *this;
}
```

Overloading Binary Operator

```
int main () {
    CRectangle recta (3,4), rectb(5,6);
    CRectangle rectc;
    rectc=recta+rectb;
    cout << "recta area: " << recta.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    cout << "rectc area: " << rectc.area() << endl;
    return 0;
}
```

Overloading Binary Operator

```
CRectangle larger(CRectangle ra, CRectangle rb){
    if(ra.area()>rb.area())
        return ra;
    else
        return rb;
}
```

Overloading Binary Operator

```
int main () {
      CRectangle recta (3,4), rectb(5,6);
      CRectangle rectc=recta; //this will call copy constructor
      //CRectangle rect_larger;
      //CRectangle rectd;
      //rectd=rectb;
      /*Following statement will call copy constructor and cause
      the program to crash
      */
      //rect_larger=larger(recta, rectb);
      cout << "recta area: " << recta.area() << endl;
      cout << "rectb area: " << rectb.area() << endl;
      cout << "rectc area: " << rectc.area() << endl;
      /*this will call both copy constructor and destructor 3 times */
      cout<< “large area: “<<larger(recta,rectb).area();
      //cout << "rect_larger area: " << rect_larger.area() << endl;
      return 0;
}
```

Overloading Unary Operator

```
class CRectangle {
    int width, height;
    public:
        CRectangle () {width=0, height=0;}
        CRectangle (int,int);
        CRectangle operator ++ ();
        CRectangle operator ++ (int notused);

            ……..
};
```

Overloading Unary Operator

```
CRectangle CRectangle::operator ++ (){
        width++;
        height++;
        return *this;
}
CRectangle CRectangle::operator ++ (int notused){
        CRectangle temp=*this;
        width++;
        height++;
        return temp;
}
```

Overloading Unary Operator

```
int main () {
      CRectangle recta (3,4), rectb(5,6), rectc;
      cout << "recta area: " << recta.area() << endl;
      recta++;
      cout << "recta area: " << recta.area() << endl;
      ++recta;
      cout << "recta area: " << recta.area() << endl;
      cout << "rectb area: " << rectb.area() << endl;
      rectc=rectb++;
      cout << "rectb area: " << rectb.area() << endl;
      cout << "rectc area: " << rectb.area() << endl;

      rectc=++rectb;
      cout << "rectb area: " << rectb.area() << endl;
      cout << "rectc area: " << rectb.area() << endl;
      return 0;
}
```

Overloading Unary Operator

```
class CRectangle {
    int width, height;
    public:
        CRectangle () {width=0, height=0;}
        CRectangle (int,int);
        CRectangle operator - (CRectangle rect2);
        CRectangle operator - ();
        void getDimensions(int &w, int &h);

            ……..……..
    };
```

Overloading Unary Operator

```
CRectangle CRectangle::operator - (CRectangle rect2){
        CRectangle temp;
        temp.width=width-rect2.width;
        temp.height=height-rect2.height;
        return temp;
}
CRectangle CRectangle::operator - (){
        width= - width;
        height= - height;
        return *this;
}
```

Overloading Unary Operator

```
void CRectangle::getDimensions(int &w, int &h){
    w=width;
    h=height;
}
```

Overloading Unary Operator

```
int main () {
    int width, height;
    CRectangle recta (3,4), rectb(5,6), rectc;
    cout << "recta area: " << recta.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    rectc=rectb-recta;
    cout << "rectc area: " << rectc.area() << endl;
    recta.getDimensions(width, height);
    cout << “width: " << width << “, height: “<<height<<endl;
    recta=-recta;
    recta.getDimensions(width, height);
    cout << “width: " << width << “, height: “<<height<<endl;
    return 0;
}
```

Overloading Logical Operator

```
class CRectangle {
    int width, height;
    public:
        CRectangle () {width=0, height=0;}
        CRectangle (int,int);
        bool operator == (CRectangle rect2);
        bool operator > (CRectangle rect2);
        int area () {return (width * height);}

};
```

Overloading Logical Operator

```
bool CRectangle::operator == (CRectangle rect2){
    return this->area()==rect2.area();
}
bool CRectangle::operator > (CRectangle rect2){
    return this->area()>rect2.area();
}
```

Overloading Logical Operator

```
int main () {
    CRectangle recta (3,4), rectb(2,6), rectc(8,9);
    if(recta==rectb){
        cout<<“two rectangles are of equal size”<<endl;
    }
    if(rectc>rectb){
        cout<<“the first rectangle is of larger size”<<endl;
    }

        return 0;
}
```

Friend Function to Overload Operator

```
class CRectangle {
    int width, height;
    public:
        CRectangle () {width=0, height=0;}
        CRectangle (int w,int h){width=w, height=h;}
        friend CRectangle operator + (CRectangle rect1, CRectangle rect2);
        friend CRectangle operator + (CRectangle rect1, int n);
        friend CRectangle operator + (int n, CRectangle rect2);
        int area () {return (width*height);}
};
```

Friend Function to Overload Operator

```
CRectangle operator + (CRectangle rect1, CRectangle rect2){
    CRectangle temp;
    temp.width=rect1.width+rect2.width;
    temp.height=rect1.height+rect2.height;
    return temp;
}

CRectangle operator + (CRectangle rect1, int n){
    CRectangle temp;
    temp.width=rect1.width+n;
    temp.height=rect1.height+n;
    return temp;
}
```

Friend Function to Overload Operator

```
CRectangle operator + (int n, CRectangle rect2){
    CRectangle temp;
    temp.width=rect2.width+n;
    temp.height=rect2.height+n;
    return temp;
}
```

Friend Function to Overload Operator

```
int main () {
    CRectangle recta (3,4);
    CRectangle rectb (5,6);
    CRectangle rectc;
    cout << "recta area: " << recta.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    rectc=recta+rectb;
    cout << "rectc area: " << rectc.area() << endl;
    rectc=recta+2;
    cout << "rectc area: " << rectc.area() << endl;
    rectc=3+rectb;
    cout << "rectc area: " << rectc.area() << endl;
    return 0;
}
```

cout<<“Thank You”<<endl;

cout<<“**Have a Good Day**”<<endl;