CSE1121: Structured & OOP Language

Sumaya Kazary
Assistant Professor
Department of Computer Science and Engineering
DUET, Gazipur.

Acknowledgement
Thanks to the authors of all the books and online tutorials used in this slide.

Introducing Operator Overloading

OBJECTIVES

- The basics of operator overloading
- Overloading binary operators
- Overloading the relational and logical operators
- Overloading a unary operator
- Using friend operator functions
- A closer look at the assignment operator
- Overloading the **[ ]** subscript operator

The Basics Of Operator Overloading

- Allows the programmer to define the meaning of the C++ operators relative to programmer defined Classes
- An operator is always overloaded relative to a user-defined type, such as a class
- When overloaded, the operator loses none of its original meaning
- To overload an operator, we create an ***operator function***
- **An operator function can be**
  - A member of the class for which it is defined
  - A friend of the class for which it is defined

The Basics Of Operator Overloading (Contd.)

- General form of a member operator function
  - **return-type class-name::operator#(arg-list) { … }**
- **Restrictions:**
  - *The precedence of the operator cannot be changed*
  - *The number of operands that an operator takes cannot be altered*
- *Except for the =, operator functions are inherited by any derived class.*
  - Operator functions can be further overloaded in the derived classes.
- *Operator functions cannot have default arguments.*

The Basics Of Operator Overloading (Contd.)

- List of Operators That Cannot Be Overloaded in C++
  - Conditional or Ternary Operator (?:)
  - Size of Operator (sizeof)
  - Scope Resolution Operator (::)
  - Class member selector Operator (.)
  - Member pointer selector Operator (.*)
- The operators cannot be overloaded using the friend function but can be overloaded by member functions are as follows:
  - Assignment Operator (=)
  - Function call Operator (( ))
  - Subscript Operator ([ ])
  - Arrow Operator (->)

Overloading Binary Operators

```
class coord {
   int x, y;
public:
   coord(int a = 0, int b = 0)
  {
      x = a; y = b;
   }
   void show()
   {
   cout << x << “, ” << y << endl;
   }
   coord operator+(coord obj);
   coord operator+(int i);
   coord operator-(coord obj);
   coord operator=(coord obj);
};
coord coord::operator=(coord obj)
    {
   x = obj.x;
   y = obj.y;
   return *this;
}
```

```
coord coord::operator+(coord obj)
 {
   coord temp;
   temp.x = x + obj.x;
   temp.y = y + obj.y;
   return temp;
}
coord coord::operator+(int i) {
   coord temp;
   temp.x = x + i;
   temp.y = y + i;
   return temp;
}
coord coord::operator-(coord obj) {
   coord temp;
   temp.x = x - obj.x;
   temp.y = y - obj.y;
   return temp;
}
```

Overloading Binary Operators (Contd.)

```
void main() {
   coord c1(20, 20), c2(10, 10);
   coord c3 = c1 + c2;                  // c1.+(c2)
   c3.show();
   coord c4 = c3 + 5;                   // c3.+(5)
   c4.show();
   coord c5 = c2 – c1;                  // c2.-(c1)
   c5.show();
   coord c6 = c1 + c2 + c3;     // (c1.+(c2)).+(c3)
   c6.show();
   (c6 – c4).show();
   c5 = c6 = c6 – c1;          // c5.=(c6.=(c6.-(c1)))
   c5.show();
   c6.show();}
```

```
30, 30
35, 35
-10, -10
60, 60
25, 25
40, 40
40, 40
```

Overloading The Relational And Logical Operators

```
class coord {
   int x, y;
public:
   coord(int a = 0, int b = 0) {
      x = a; y = b;
   }
   void show() {
      cout << x << “, ” << y << endl;
   }

   int operator ==(coord obj);
   int operator!=(coord obj);
   int operator&&(coord obj);
   int operator||(coord obj);
};
```

```
int coord::operator==(coord obj)
{
   return (x == obj.x) && (y == obj.y);
}
int coord::operator!=(coord obj)
{
   return (x != obj.x) || (y != obj.y);
}
int coord::operator&&(coord obj)
 {
   return (x && obj.x) && (y && obj.y);
}
int coord::operator||(coord obj)
 {
   return (x || obj.x) || (y || obj.y);
}
```

OVERLOADING A UNARY OPERATOR

```
class coord {
   int x, y;
public:
   coord(int a = 0, int b = 0) {
      x = a; y = b;
   }
   void show() {
      cout << x << “, ” << y << endl;
   }
   coord operator ++();   //prefix
   coord operator ++(int unused);
    //postfix
   coord operator-(); /unary minus
   coord operator-(coord obj);
};
```

```
coord coord::operator++() {
   ++x; ++y;
  return *this;
}

coord coord::operator++(int unused) {
   coord duplicate(*this);
   x += 1;   y+ =1;
   return duplicate;
} // postfix version

coord coord::operator-() {
   coord temp;
   temp.x = -x; temp.y = -y; return temp;
}

coord coord::operator-(coord obj) {
   coord temp;
   temp.x = x-obj.x; temp.y = y-obj.y;
   return temp;
}
```

Overloading A Unary Operator (Contd.)

(Test the codes and try to understand the effects)

```
void main() {
   coord c1(10, 10), c2(10, 10);
   coord c3 = ++c1;
   coord c4 = c2++;
   c1.show();
   c2.show();
   c3.show();
   c4.show();
   coord c5 = -c1;
   c1.show();
   c5.show();
   coord c6 = c3 – c4;
                        // c3.-(c4)
   c6.show();
}
```

OBJECT COPY ISSUES

- Whenever possible we should use reference parameters while passing objects to or returning objects from a function.
  - coord coord::operator+(coord& obj) { … }
  - coord& coord::operator=(coord& obj) { … }
  - coord& coord::operator++() { … }
- Otherwise should use copy constructors to overcome object copy problems.

Using Friend Operator Functions

- It is possible to overload an operator relative to a class by using a friend rather than a member function.
- As a friend function does not have a ***this*** pointer –
  - For binary operators, both operands must be passed explicitly
  - For unary operators, the single operand must be passed explicitly
- Allows us to perform operations like -
  - c2 = 10 + c1; Assume: coord c1(10, 10), c2;
    - We cannot perform this using member operator functions as the left argument of '+' is not an object of class "coord"
- **We cannot use a friend to overload the assignment operator (=)**
  - It can be overloaded only by a member operator function

Using Friend Operator Functions (Contd.)

```
class coord {
   int x, y;
public:
   coord(int a = 0, int b = 0) {
      x = a; y = b;
   }
   void show() {
      cout << x << ", " << y << endl;
   }
   friend coord operator +(coord &ob1,
                           coord &ob2);
   friend coord operator +(int i, coord
    &ob);
   friend coord& operator++(coord
    &ob);
};
```

```
coord operator+(coord &ob1,
    coord &ob2) {
   coord temp;
   temp.x = ob1.x + ob2.x;
   temp.y = ob1.y + ob2.y;
   return temp;
}
coord operator+(int i, coord &ob)
    {
   coord temp;
   temp.x = ob.x + i;
   temp.y = ob.y + i;
   return temp;
}
```

Using Friend Operator Functions (Contd.)

```
coord& operator++(coord & ob)
   {
   ob.x++;
   ob.y++;
   return ob;
}
```

Here, in case of "++" we must use reference parameter.

Otherwise changes made inside the function will not be visible outside and the original object will remain unchanged.

```
void main() {
   coord c1(20, 20), c2(10, 10);
   coord c3 = c1 + c2;  // +(c1, c2)
   c3.show();          // 30, 30
   coord c4 = 5 + c3;      // +(5, c3)
   c4.show();                // 35, 35
   ++c4;                    // ++(c4)
   c4.show();                // 36, 36
}
```

A Closer Look At The Assignment Operator

- By default, "ob1 = ob2" places a bitwise copy of "ob2" into "ob1"
  - This causes problem when class members point to dynamically allocated memory
- Copy constructor is of no use in this case as it is an ***assignment***, not an initialization
- So, we need to overload '=' to overcome such problems

A Closer Look At The Assignment Operator

```
class strtype {
   char *p;
   int len;
public:
   strtype(char *s) {
      len = strlen(s) + 1;
      p = new char[len];
      strcpy(p, s);
   }
   ~strtype() {
      delete [ ] p;
   }
   strtype &operator=(strtype
    &ob);
};
```

```
strtype &strtype::operator=(strtype
    &ob) {
   if(len < ob.len) {
      delete [ ] p;
      p = new char[ob.len];
   }
   len = ob.len;
   strcpy(p, ob.p);
   return *this;
}
void main() {
   strtype s1("DUET"), s2("CSE");
   s1 = s2; // no problem
}
```

A Closer Look At The Assignment Operator

- The overloaded '=' operator must return *this to allow chains of assignments
  - ob1 = ob2 = ob3 = ob4;
- If the overloaded '=' operator returns nothing (void) then
  - ob1 = ob2; is possible, but
  - ob1 = ob2 = ob3; produces compiler error
    - ob3 can be assigned to ob2, but then it becomes "ob1 = (void)"
    - So, the compiler detects it early and flags it as an error
- Whenever possible we should use references while passing objects to functions
  - Copy constructors can also help in this regard but using references is more efficient as no copy is performed

Overloading The [ ] Subscript Operator

- In C++, the [ ] is considered a binary operator for the purposes of overloading
- The [ ] can be overloaded only by a member function
- O[9] is interpreted as O.operator[ ](9)
- General syntax
  - **ret-type class-name::operator[ ](int index) {…}**
    - "index" does not have to be of type "int"
    - "index" can be of any other type
      - ret-type class-name::operator[ ](char *index) {…}
- It is useful when the class has some array like behavior

Overloading The [ ] Subscript Operator (Example - 1)

```
class array {
        int a[3];
public:
  array() {
    for(int i=0; i<3; i++)
      a[i] = i;
  }
  int operator[ ](int i) {
        return a[i];
  }
  int operator[ ](char *s);
};
```

```
int array::operator[ ](char *s) {
  if(strcmp(s, "zero")==0)
    return a[0];
  else if(strcmp(s, "one")==0)
    return a[1];
  else if(strcmp(s, "two")==0)
    return a[2];
  return -1;
}
void main() {
  array ob;
  cout << ob[1];    //ob.operator[](1)->
   1
  cout << ob["two"]; // 2
  ob[0] = 5; // compiler error
    // ob[i] is not an l-value in this
       example
}
```

Overloading The [ ] Subscript Operator (Example - 2)

```
class array {
   int a[3];
public:
   array() {
      for(int i=0; i<3; i++)
         a[i] = i;
   }
   int& operator[ ](int i) {
      return a[i];
   }
   int& operator[ ](char *s);
};
```

```
int& array::operator[ ](char *s) {
   if(strcmp(s, "zero")==0)
      return a[0];
  else if(strcmp(s, "one")==0)
      return a[1];
   else if(strcmp(s, "two")==0)
      return a[2];
   return a[0];
}
void main() {
   array ob;
   cout << ob[1]; // 1
   cout << ob["two"]; // 2
   ob[0] = 5; // no problem
      // ob[i] is now both an l-value and r-value
   cout << ob["zero"]; // 5
}
```

Overloading the Input & Output Operator

```
class coord {
   int x, y;
public:
   coord(int a = 0, int b = 0) {
      x = a; y = b;
   }
   friend istream  &operator>>(istream &in, Coord &c);
                                   //Overloading >> Operator

   friend ostream &operator<<(ostream &out, Coord &c);
                                    //Overloading << Operator
};
```

Overloading the Input & Output Operator

```
istream &operator>>(istream &in, Coord &c)  {
       cout<<“\n Enter X coordinate   :  ”;
      in>>c.x;
       cout<<“\n Enter Y coordinate :  ”;
      in>>c.y;
      return in;
      }

  ostream &operator<<(ostream &out, Coord &c){
       out<<"\n X coordinate :  "<<c.x;
       out<<"\n Y coordinate :  "<<c.y;
        return out;
   }
```

Lecture Contents

- Teach Yourself C++
  - Chapter 6 (Full, with exercises)
  - Study all the examples from the book carefully