Chapter 6

Objective

- Operator Overloading
- Overloading Binary Operators
- Overloading an Operator Relative To A Class
- Using Friend Operator Functions
- Overloading the Relation and Logical Operators
- Overloading assignment operator
- Overloading A Unary Operator
- Function Pointer
- Function Pointer as a Parameter
- Custom Manipulators

Unit 1

Operator Overloading

•Built-in operators are used for the function names.

•Operator functions extend the meaning by overloading the built-in operators.

•Operator functions have same syntax as regular C/C++ functions.

•Operator in an expression calls the corresponding operator function.

•Multiple operator functions can be defined for a single operator (overloading)

Restrictions on Operator Functions

•New operator symbols cannot be defined.

•Operators cannot be combined to create new operators.

•The precedence of an operator cannot be changed.

Operators That Cannot Be Overloaded

.	Member selection
.*	Pointer-to-member selection
::	Scope resolution
:>	Base operator
?:	Conditional
#	Preprocessor symbol
##	Preprocessor symbol

Unit 2

Overloading Binary Operators

•For class member binary operator function left operand must always be an object.

```
1   #include <iostream>    // Example 6-1
2   using namespace std;
3
4   class CScreen {
5       public:
6           CScreen() { m_nRow = 0; m_nColumn = 0; }
7           CScreen(int r, int c) { m_nRow = r; m_nColumn = c; }
8           void GetPosition(int &r, int &c);
9           CScreen operator+(CScreen win2);
10       private:
11           int m_nRow, m_nColumn; // coordinate values
12   };
13
```

Unit 2

```
14   // Overload plus (+) relative to CScreen class object.
15   CScreen CScreen::operator+(CScreen win2)
16   {
17       CScreen temp;
18   // the left operand is passed implicitly to the function
19       temp.m_nRow = m_nRow + win2.m_nRow;
20
21   // The right operand is passed as an argument
22       temp.m_nColumn = m_nColumn + win2.m_nColumn;
23       return(temp);
24   }
25
26   void CScreen:: GetPosition(int &r, int &c)
27   {
28      r = m_nRow;
29      c = m_nColumn;
30   }
31
```

Unit 2

```
32   int main()
33   {
34       CScreen w1(10, 10), w2(5, 3), w3;
35       int x, y;
36
37   // add two objects, this calls operator+()
38       w3 = w1 + w2;    // w3 = w1.operator+(w2);
39       w3.GetPosition(x, y);
40      cout << "(w1+w2) X: " << x << ", Y: " << y << "\n";
41       }
```

OUTPUT: (w1+w2) X: 15, Y: 13

Unit 3

Operator Overloading to Support Built-in Data Type

•The operand on the right side of binary operator is a built-in type, instead of a class object

```
1   #include <iostream>    // Example 6-2
2   using namespace std;
3
4   class CScreen {
5       public:
6           CScreen()    { m_nRow = 0; m_nColumn = 0; }
7           CScreen(int r, int c) { m_nRow = r; m_nColumn = c; }
8           void GetPosition(int &r, int &c) { r = m_nRow; c = m_nColumn;}
9
10           CScreen operator+(CScreen &win2); // winObj1 + winObj2
11           CScreen operator+(int i);    // winObj + integer
12       private:
13           int m_nRow, m_nColumn;    // coordinate values
14   };
15
```

Unit 3

```
16   // Overload "+" that is objOne + objTwo
17   CScreen CScreen::operator+(CScreen &win2)
18   {
19       CScreen temp;
20       temp.m_nRow = m_nRow + win2.m_nRow;
21       temp.m_nColumn = m_nColumn + win2.m_nColumn;
22       return(temp);
23   }
24
25   // Overload "+" that is object + integer
26   CScreen CScreen::operator+(int i)
27   {
28       CScreen temp;
29    // built-in type must be on the right side of the operator
30       temp.m_nRow = m_nRow + i;
31    // object on the left that generates the call to the operator function
32       temp.m_nColumn = m_nColumn + i;
33
34       return(temp);
35   }
```

Unit 3

```
36
37
38     int main()
39    {
40        CScreen w1(10, 10), w2(5, 3), w3;
41        int x, y;
42
43        // add two objects - this calls operator+(CScreen)
44        w3 = w1 + w2;    // w3 = w1.operator+(w2);
45        w3.GetPosition(x,y);
46        cout << "(w1+w2) X: " << x << ", Y: " << y << "\n";
47
48        // add object + int this calls operator+(int)
49        w3 = w1 + 100; // w3 = w1.operator+(100);
50        cout << "(w1+100) X: " << x << ", Y: " << y << "\n";
51    }
52
```

OUTPUT:

(w1+w2) X: 15, Y: 13

(w1+100) X: 15, Y: 13

Unit 4

Using Friend Operator Functions

•For non-member binary operator function at least one operand must be an object.

• Overload the plus "+" operator relative to CScreen class using a friend.

```
1   #include <iostream>    // Example 6-3
2   using namespace std;
3
4   class CScreen {
5       public:
6           CScreen() { m_nRow = 0; m_nColumn =0; }
7           CScreen(int r, int c) { m_nRow = r; m_nColumn = c; }
8           void GetPosition(int &r, int &c) { r = m_nRow; c = m_nColumn; }
9
11           friend CScreen operator+(CScreen &win1, CScreen &win2);
10           friend CScreen operator+(CScreen &win1, int i);
11
12          // Using friend Operator Function to add Flexibility
13           friend CScreen operator+(int i, CScreen &win1);
14       private:
15           int m_nRow, m_nColumn;
16   };
```

Unit 4

Overload + using a friend.

```
17   CScreen operator+(CScreen &win1, CScreen &win2)
18   {
19       CScreen temp;
20       temp.m_nRow = win1.m_nRow + win2.m_nRow;
21       temp.m_nColumn = win1.m_nColumn + win2.m_nColumn;
22
23       return(temp);
24   }
25
```

Unit 4

```
26   // Overload for object + int.
27   CScreen operator+(CScreen &win1, int i)
28   {
29       CScreen temp;
30
31       temp.m_nRow = win1.m_nRow + i;
32       temp.m_nColumn = win1.m_nColumn + i;
33
34       return(temp);
35   }
36
37   // Overload for int + object.
38   CScreen operator+(int i, CScreen &win1)
39   {
40       CScreen temp;
41       temp.m_nRow = win1.m_nRow + i;
42       temp.m_nColumn = win1.m_nColumn + i;
43
44       return(temp);
45   }
```

Unit 4

```
46
47   int main()
48   {
49       CScreen w1(10,10), w2(5,3), w3;;
50       int x, y;
51       // add two objects - this calls operator+()
52       w3 = w1 + w2;    // w3 = operator+(w1,w2)
53       w3.GetPosition(x,y);
54       cout << "(w1+w2) X: " << x << ", Y: " << y << "\n";
55
56       w1 = w1 + 10;    // object + integer
57       w1.GetPosition(x,y);
58       cout << "w1+10) X: " << x << ", Y: " << y << "\n";
59
60       w1 = 99 + w1;    // integer + object
61       w1.GetPosition(x,y);
62       cout << "(99+w1) X: " << x << ", Y: " << y << "\n";
63   }
64
```

OUTPUT:
(w1+w2) X: 15, Y: 13

Unit 5

Overloading the Relation and Logical Operators

•Operator function returns an integer indicating true or false

•Allows the operators to be integrated into larger relational and logical expressions

```
1   // Overload the == and && relative to CScreen class
2   #include <iostream>    // Example 6-4
3   using namespace std;
4
5   class CScreen {
6       public:
7           CScreen() {m_nRow = 0; m_nColumn = 0; }
8           CScreen(int r, int c) { m_nRow = r; m_nColumn = c; }
9           int operator==(CScreen &win2);
10           int operator&&(CScreen &win2);
11       private:
12           int m_nRow, m_nColumn;
13   };
14
```

Unit 5

```
15   // Overload the == operator for CScreen.
16   int CScreen::operator==(CScreen &win2)
17   {
18     if (m_nRow == win2.m_nRow && m_nColumn == win2.m_nColumn) return(1);
19     else return(0);
20   }
21
22   // Overload the && operator for CScreen.
23   int CScreen::operator&&(CScreen &win2)
24   {
25       return((m_nRow && win2.m_nRow) && (m_nColumn && win2.m_nColumn));
26   }
27
```

Unit 5

```
28   int main()
29   {
30       CScreen w1(10,10), w2(5, 3), w3(10,10), w4(0,0);
31
32   // (w1.operator==(w2))
33       if (w1 == w2)    cout << "w1 same as w2\n";
34       else cout << "w1 and w2 differ\n";
35
36       if (w1 == w3)    cout << "w1 same as w3\n";
37       else    cout << "w1 and w3 differ\n";
38
39       if (w1 && w2)    cout << "w1 && w2 is true\n";
40       else    cout << "w1 && w2 is false\n";
41
42       if (w1 && w4)    cout << "w1 && w4 is true\n";
43       else    cout << "w1 && w4 is false\n";
44   }
45
```

Unit 6

Overloading assignment operator

```
1   // Overload the = relative to CScreen class
2   #include <iostream>    // Example 6-5
3   using namespace std;
4
5   class CScreen {
6       public:
7           CScreen() { m_nRow = 0; m_nColumn = 0; }
8           CScreen(int r, int c) { m_nRow = r; m_nColumn = c; }
9           void GetPosition(int &r, int &c) { r =m_nRow; c = m_nColumn; }
10           CScreen operator = (CScreen &win2);
11       private:
12           int m_nRow, m_nColumn;
13   };
14
```

Unit 6

```
15   // Overload = relative to CScreen.
16   CScreen CScreen::operator=(CScreen &win2)
17   {
18       m_nRow = win2.m_nRow;
19       m_nColumn = win2.m_nColumn;
20
21       return(*this);    // return the object that is assigned
22   }
23
24   int main()
25   {
26       CScreen w1(10,10), w2;
27       int x, y;
28
29       w2 = w1;    // assign an object
30       w2.GetPosition(x,y);
31
32       cout << "(w2 = w1) X: " << x << ", Y: " << y << "\n";
33   }
34
```

OUTPUT:
(w2 = w1) X: 10, Y: 10

Unit 7

Overloading A Unary Operator

- The unary operators operate on a single operand.

```
1   // Overload ++ relative to CClock class
2   #include <iostream>    // Example 6-6
3   #include <iomanip>
4   using namespace std;
5
6   class CClock {
7       public:
8           CClock();
9           CClock Tick();
10           void PrintTime();
11           CClock operator++();
12           CClock operator++(int); // postfix
13       private:
14           int m_Hour;
15           int m_Minutes;
16           int m_TimeType;
17   };
18
```

Unit 7

```
19   CClock::CClock()
20   {
21       m_Hour = 12;
22       m_Minutes = 0;
23       m_TimeType = 0;
24}
25
26   CClock CClock::Tick()
27   {
28       ++m_Minutes;
29       if (m_Minutes == 60) {
30           m_Hour++;
31           m_Minutes = 0;
32       }
33       if (m_Hour == 13)
34           m_Hour = 1;
35
36       if (m_Hour == 12 && m_Minutes == 0)
37           m_TimeType = !m_TimeType;
38       return *this;
39   }
```

Unit 7

```
40    CClock CClock::operator++()
41    {
42        return Tick();
43    }
44
45    CClock CClock::operator++(int)
46    {
47        CClock c = *this;
48        Tick();
49
50        return c;
51    }
52
53    void CClock::PrintTime()
54    {
55        cout << setfill('0') << setw(2) << m_Hour;
56        cout << ':' << setw(2) << m_Minutes << setfill(' ');
57
58        cout << (m_TimeType ? " PM" : " AM") << endl;
59    }
```

Unit 7

```
60   int main()
61   {
62       CClock clkObj1, clkObj2;
63
64       clkObj1 = clkObj2++;
65
66       cout << "Clock clkObj1 = ";
67       clkObj1.PrintTime();
68
69       cout << "Clock clkObj2 = ";
70       clkObj2.PrintTime();
71
72   }
```

OutPut:

Clock clkobj1 = 12:00 AM
Clock clkobj2 = 12:01 AM

Unit 8

Overloading [ ] Operator

```
1   #include <iostream>    // Example 6-7
2   using namespace std;
3
4   class CVector {
5       public:
6           CVector(int n);
7           ~CVector();
8           int &operator[](int i);
9
10       private:
11           int *m_Data;
12           int m_Size;
13   };
14
15   CVector::CVector(int n)
16   {
17       m_Data = new int[n];
18       m_Size = n;
19   }
```

Unit 8

```
20   CVector::~CVector()
21   {
22       delete [] m_Data;
23   }
24
25   int& CVector::operator[](int i)
26   {
27       return m_Data[i];
28   }
29
30   int main()
31   {
32       CVector Obj(8);   // integer array
33
34       for (int i = 0; i < 8; i++)    Obj[i] = i * 3;
35            // OR Obj.operator[](i) = i * 3;
36
37      for (i = 0; i < 8; i++)       cout << Obj[i] << ' ';
38          // OR cout << Obj.operator[](i) << ' ';
39       cout << '\n';
40   }
```

Unit 9

Overloading Function Call ( ) Operator

- Function call operator supports multiple parameter in its parameter list.

```
1   #include <iostream>    // Example 6-9
2   using namespace std;
3
4   class CMatrix {
5       public:
6           CMatrix(unsigned int row, unsigned int col);
7           ~CMatrix() { delete [] m_pData; }
8           int& operator()(unsigned int row, unsigned int col);
9       private:
10           unsigned int m_nRows, m_nCols;
11           int *m_pData;
12  };
13
14  CMatrix::CMatrix(unsigned int row, unsigned int col)
15  {
16      m_nRows = row;
17      m_nCols = col;
18      m_pData = new int [row * col];
19  }
```

Unit 9

```
20   int& CMatrix::operator ()(unsigned int row, unsigned int col)
21   {
22       return m_pData[m_nCols * row + col];
23   }
24
25   int main()
26   {
27       CMatrix Array(10,10);
28       for (int i=0, r = 0; r < 10; r++) {
29           for (int c=0; c < 10; c++) {
30               Array(r,c) = i++;   // Array.operator()(r,c).
31           }
32       }
33       for (r = 0; r < 10; r++) {
34           for (int c = 0; c < 10; c++) {
35               if (c == 9)    cout << "\n";
36               else cout << " " << Array(r,c) << " ";
37           }
38       }
39
40   }
```

Unit 10

Function Pointer

```
1   #include <iostream>    // Example 6-10
2   using namespace std;
4   int square(int x);
5   int cube(int x);
7   int main()
8   {
9       int y, x = 5;
10       int (*func)(int x); /* pointer to function */
11
12       func = square;      /* not a call to square */
13       y = func(x);        /* indirect call on squre */
14       cout << "Square of " << x << " is " << y << "\n";
15
16       func = cube;
17       y = func(x);
18       cout << "Cube of " << x << " is " << y << "\n";
19
20       return 0;
21   }
```

Unit 10

Function Pointer

```
22   int square(int x)
23   {
24       return(x * x);
25   }
26
27   int cube(int x)
28   {
29       return(x * x * x);
30   }
```

Unit 10

Function Pointer as a Parameter

```
1   #include <iostream>    // Example 6-11
2   using namespace std;
3
4   double square(double y);
5
6   void tabulate(
7             double (*func)(double y), // function address
8             double lower,
9             double upper,
10             double increment);
11
12   int main()
13   {
14       tabulate(square,0.0, 2.0, 0.1);
15
16       return 0;
17   }
```

Unit 10

Function Pointer as a Parameter

```
18   void tabulate(double (*func)(double y),
19                 double lower,
20                 double upper,
21                 double increment)
22   {
23       double x;
24
25       for (x = lower;
26       x <= upper + 0.5 * increment;
27       x += increment) {
28          cout << x << " " << func(x);
29       } // end of for loop
30   }
31
32   double square(double y)
33   {
34       return(y * y);
35   }
36
```

Unit 11

Custom Manipulators

- Custom manipulators must have a parameter of type ostream&.
- Return value of the custom manipulator must be ostream reference type
- Ostream class has an overloaded operator<< function whose parameter is of a pointer to a function type.

Definition of overloaded operator<< function:

```
ostream& operator<<(ostream& (*func)(ostream& str))
{
  return ((*func)(*this));
}
```

Unit 11

```
1  #include <iostream>    // Example 6-12
2  #include <iomanip>
3  using namespace std;
4
5  ostream &Space(ostream& Obj)
6  {
7        Obj << " ";
8        return Obj;
9  }
10
11  // Doller:
12  ostream &Dollar(ostream& Obj)
13  {
14       Obj << "$";
15       return Obj;
16  }
```

Unit 11

```
17  //At:
18  ostream &At(ostream& Obj)
19  {
20       Obj << "@";
21       return Obj;
22  }
23
24  int main()
25  {
26     cout << "Quantity:“;
27     cout << 120;
28     cout << Space;
29      cout << At;
30      cout << Space;
31      cout << Dollar;
32      cout << 90;
33  }
```