11.2

Overloading Operators

Overloading Operators

- In the Money class, function add was used to add two objects of type Money

- In this section we see how to use the '+' operator to make the following code legal:

```
Money total, cost, tax;
...
total = cost + tax;
  // instead of  total = add(cost, tax);
```

Operators As Functions

- An operator is a function used differently than an ordinary function
  - An ordinary function call enclosed its arguments in parenthesis

    add(cost, tax)

  - With a binary operator, the arguments are on either side of the operator

    cost + tax

Operator Overloading

- Operators can be overloaded
- The definition of operator + for the Money class is nearly the same as member function add
- To overload the + operator for the Money class
  - Use the name + in place of the name add
  - Use keyword operator in front of the +
  - Example:

```
friend Money operator + (const Money& amount1,
                         const Money&
   amount2)
```

Operator Overloading Rules

- At least one argument of an overloaded operator must be of a class type
- An overloaded operator can be a friend of a class
- The number of arguments for an operator cannot be changed
- The precedence of an operator cannot be changed
- `.`, `::`, `*`, and `?` cannot be overloaded

Program Example: Overloading Operators

- The Money class with overloaded operators + and == is demonstrated in

Display 11.5 (1)

Display 11.5 (2)

DISPLAY 11.5 Overloading Operators *(part 1 of 2)*

```
1  //Program to demonstrate the class Money. (This is an improved version of
2  //the class Money that we gave in Display 11.3 and rewrote in Display 11.4.)
3  #include <iostream>
4  #include <cstdlib>
5  #include <cctype>
6  using namespace std;
7
8  //Class for amounts of money in U.S. currency.
9  class Money
10 {
11 public:
12     friend Money operator +(const Money& amount1, const Money& amount2);
13     //Precondition: amount1 and amount2 have been given values.
14     //Returns the sum of the values of amount1 and amount2.

15     friend bool operator ==(const Money& amount1, const Money& amount2);
16     //Precondition: amount1 and amount2 have been given values.
17     //Returns true if amount1 and amount2 have the same value;
18     //otherwise, returns false.

19     Money(long dollars, int cents);

20     Money(long dollars);

21     Money( );

22     double get_value( ) const;

23     void input(istream& ins);

24     void output(ostream& outs) const;
25 private:
26     long all_cents;
27 };
```

Some comments from Display 11.4 have been omitted to save space in this book, but they should be included in a real program.

<Any extra function declarations from Display 11.3 go here.>

```
28 int main( )
29 {
30     Money cost(1, 50), tax(0, 15), total;
31     total = cost + tax;

32     cout << "cost = ";
33     cost.output(cout);
34     cout << endl;
```

Display 11.5 (1/2)

DISPLAY 11.5 Overloading Operators *(part 2 of 2)*

```
35        cout << "tax = ";
36        tax.output(cout);
37        cout << endl;
38        cout << "total bill = ";
39        total.output(cout);
40        cout << endl;
41        if (cost == tax)
42            cout << "Move to another state.\n";
43        else
44            cout << "Things seem normal.\n";
45        return 0;
46    }
47
48    Money operator +(const Money& amount1, const Money& amount2)
49    {
50        Money temp;
51        temp.all_cents = amount1.all_cents + amount2.all_cents;
52        return temp;
53    }
54
55    bool operator ==(const Money& amount1, const Money& amount2)
56    {
57        return (amount1.all_cents == amount2.all_cents);
58    }
59
```

<The definitions of the member functions are the same as in Display 11.3 except that *const* is added to the function headings in various places so that the function headings match the function declarations in the preceding class definition. No other changes are needed in the member function definitions. The bodies of the member function definitions are identical to those in Display 11.3.>

Output

```
cost = $1.50
tax = $0.15
total bill = $1.65
Things seem normal.
```

Display 11.5 (2/2)

Automatic Type Conversion

- With the right **constructors**, the system can do type conversions for your classes
  - The following code (from Display 11.5) actually works

```
Money base_amount(100, 60), full_amount;
full_amount = base_amount + 25;
```

  - The integer 25 is converted to type Money so it can be added to base_amount!
  - How does that happen?

Type Conversion Event 1

- When the compiler sees base_amount + 25,
  it first looks for an overloaded + operator to perform

  Money_object + some-integer

  - If it exists, it might look like this

    ```
    friend Money operator +(const Money& amount1,
                            const int& amount2);
    ```

Type Conversion Event 2

- When the appropriate version of + is not found, the compiler looks for a constructor that takes a single integer
  - The Money constructor that takes a single parameter of type long will work
  - The constructor Money(long dollars) converts 25 to a Money object so the two values can be added!

Type Conversion Again

- Although the compiler was able to find a way to add

  base_amount + 25

  this addition will cause an error

  base_amount + 25.67

  - There is no constructor in the Money class that takes a single argument of type double

A Constructor For double

- To permit base_amount + 25.67, the following constructor should be declared and defined

```
class Money
{
   public:
                …
                Money(double amount);
                // Initialize object so its value is $amount
                …
```

Overloading Unary Operators

- Unary operators take a single argument
- The unary – operator is used to negate a value

x = -y

- ++ and - - are also unary operators
- Unary operators can be overloaded
  - The Money class of Display 11.6 can include
    - A binary – operator
    - A unary – operator

Overloading -

- Overloading the - operator with two parameters allows us to subtract Money objects as in

```
Money  amount1, amount2, amount2;
  ...
amount3 = amount1 - amount2;
```

- Overloading the - operator with one parameter allows us to **negate** a money value like this

```
amount3 =  - amount1;
```

Display 11.6

DISPLAY 11.6 Overloading a Unary Operator

```
1  //Class for amounts of money in U.S. currency.
2  class Money
3  {
4  public:
5      friend Money operator +(const Money& amount1, const Money& amount2);
6      friend Money operator -(const Money& amount1, const Money& amount2);
7      //Precondition: amount1 and amount2 have been given values.
8      //Returns amount 1 minus amount2.
9      friend Money operator -(const Money& amount);
10     //Precondition: amount has been given a value.
11     //Returns the negative of the value of amount.
12     friend bool operator ==(const Money& amount1, const Money& amount2);
13     Money(long dollars, int cents);
14     Money(long dollars);
15     Money( );
16     double get_value( ) const;
17     void input(istream& ins);
18     void output(ostream& outs) const;
19 private:
20     long all_cents;
21 };
```

This is an improved version of the class `Money` *given in Display 11.5.*

We have omitted the `include` *directives and some of the comments, but you should include them in your programs.*

<Any additional function declarations as well as the main part of the program go here.>

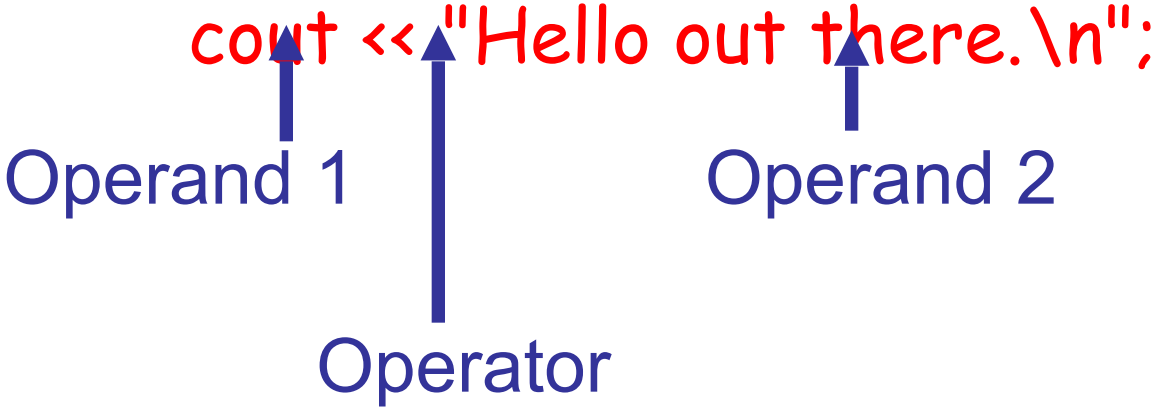
```
22 Money operator -(const Money& amount1, const Money& amount2)
23 {
24     Money temp;
25     temp.all_cents = amount1.all_cents - amount2.all_cents;
26     return temp;
27 }
28 Money operator -(const Money& amount)
29 {
30     Money temp;
31     temp.all_cents = -amount.all_cents;
32     return temp;
33 }
```

<The other function definitions are the same as in Display 11.5.>

Display 11.6

Overloading << and >>

- The **insertion** operator << is a binary operator
  - The first operand is the **output stream**
  - The second operand is the value following <<



Replacing Function output

- Overloading the << operator allows us to use << instead of Money's output function
  - Given the declaration: Money amount(100);

```
amount.output( cout );
```

can become

```
cout << amount;
```

What Does << Return?

- Because << is a binary operator

  cout << "I have " << amount << " in my purse.";

  seems as if it could be grouped as

  ((cout << "I have") << amount) << "in my purse.";

- To provide cout as an argument for << amount, (cout << "I have") must return cout

Display 11.7

<< as an Operator

```
cout << "I have " << amount << " in my purse.\n";
```

means the same as

```
((cout << "I have ") << amount) << " in my purse.\n";
```

and is evaluated as follows:

First evaluate `(cout << "I have ")`, which returns `cout`:

```
((cout << "I have ") << amount) << " in my purse.\n";
```

and the string "I have" is output.

```
(cout << amount) << " in my purse.\n";
```

Then evaluate `(cout << amount)`, which returns `cout`:

```
(cout << amount) << " in my purse.\n";
```

and the value of amount is output.

```
cout << " in my purse.\n";
```

Then evaluate `cout << " in my purse.\n"`, which returns `cout`:

```
cout << " in my purse.\n";
```

and the string " in my purse.n" is output.

```
cout;
```

Since there are no more << operators, the process ends.

Display 11.7

Overloaded << Declaration

- Based on the previous example, << should return its first argument, the output stream
  - This leads to a declaration of the overloaded << operator for the Money class:

```
class Money
{
  public:
        ...
    friend ostream& operator << (ostream& outs,
                                 const Money& amount);
        ...
```

Overloaded << Definition

- The following **defines** the << operator

```
ostream operator <<(ostream& outs,
                                const Money& amount)
  {
       <Same as the body of Money::output  in
      Display 11.3 (except all_cents is replaced
       with amount.all_cents) >

       return outs;
   }
```

Return ostream& ?

- The & means a reference is returned
  - So far all our functions have returned values
- The value of a stream object is not so simple to return
  - The value of a stream might be an entire file, the keyboard, or the screen!
- We want to return a reference to the stream , not the value of the stream
- The & means that we want to return a reference to the stream, not its value

Overloading >>

- Overloading the **extraction** >> operator for input is very similar to overloading the << for output
- >> could be defined this way for the Money class

```
istream& operator >>(istream& ins, Money& amount)
{
    <This part is the same as the body of
     Money::input in Display 11.3 (except that
     all_cents is replaced with amount.all_cents)>

    return ins;
}
```

Display 11.8 (1-4)

Display 11.8 (1/4)

DISPLAY 11.8 Overloading << and >> *(part 1 of 4)*

```
1   //Program to demonstrate the class Money.
2   #include <iostream>
3   #include <fstream>
4   #include <cstdlib>
5   #include <cctype>
6   using namespace std;
7
8   //Class for amounts of money in U.S. currency.
9   class Money
10  {
11  public:
12      friend Money operator +(const Money& amount1, const Money& amount2);
13      friend Money operator -(const Money& amount1, const Money& amount2);
14      friend Money operator -(const Money& amount);
15      friend bool operator ==(const Money& amount1, const Money& amount2);
```

This is an improved version of the class **Money** *that we gave in Display 11.6.*

Although we have omitted some of the comments from Displays 11.5 and 11.6, you should include them.

(continued)

DISPLAY 11.8 Overloading << and >> *(part 2 of 4)*

```
16      Money(long dollars, int cents);
17      Money(long dollars);
18      Money( );
19      double get_value( ) const;
20      friend istream& operator >>(istream& ins, Money& amount);
21      //Overloads the >> operator so it can be used to input values of type Money.
22      //Notation for inputting negative amounts is as in -$100.00.
23      //Precondition: If ins is a file input stream, then ins has already been
24      //connected to a file.
25      friend ostream& operator <<(ostream& outs, const Money& amount);
26      //Overloads the << operator so it can be used to output values of type Money.
27      //Precedes each output value of type Money with a dollar sign.
28      //Precondition: If outs is a file output stream,
29      //then outs has already been connected to a file.
30  private:
31      long all_cents;
32  };
33  int digit_to_int(char c);
34  //Used in the definition of the overloaded input operator >>.
35  //Precondition: c is one of the digits '0' through '9'.
36  //Returns the integer for the digit; for example, digit_to_int('3') returns 3.
37
38  int main( )
39  {
40      Money amount;
41      ifstream in_stream;
42      ofstream out_stream;
43
44      in_stream.open("infile.dat");
45      if (in_stream.fail( ))
46      {
47          cout << "Input file opening failed.\n";
48          exit(1);
49      }
50
51      out_stream.open("outfile.dat");
52      if (out_stream.fail( ))
53      {
54          cout << "Output file opening failed.\n";
55          exit(1);
56      }
57
```

(continued)

Display 11.8(2/4)

DISPLAY 11.8 Overloading << and >> *(part 3 of 4)*

```
58      in_stream >> amount;
59      out_stream << amount
60                 << " copied from the file infile.dat.\n";
61      cout << amount
62           << " copied from the file infile.dat.\n";
63
64      in_stream.close( );
65      out_stream.close( );
66
67      return 0;
68  }
69  //Uses iostream, cctype, cstdlib:
70  istream& operator >>(istream& ins, Money& amount)
71  {
72      char one_char, decimal_point,
73           digit1, digit2; //digits for the amount of cents
74      long dollars;
75      int cents;
76      bool negative;//set to true if input is negative.

77      ins >> one_char;
78      if (one_char == '-')
79      {
80          negative = true;
81          ins >> one_char; //read '$'
82      }
83      else
84          negative = false;
85      //if input is legal, then one_char == '$'

86      ins >> dollars >> decimal_point >> digit1 >> digit2;

87      if ( one_char != '$' || decimal_point != '.'
88           || !isdigit(digit1) || !isdigit(digit2) )
89      {
90          cout << "Error illegal form for money input\n";
91          exit(1);
92      }

93      cents = digit_to_int(digit1)*10 + digit_to_int(digit2);

94      amount.all_cents = dollars*100 + cents;
95      if (negative)
96          amount.all_cents = -amount.all_cents;
```

(continued)

Display 11.8 (3/4)

DISPLAY 11.8 Overloading << and >> *(part 4 of 4)*

```
97      return ins;
98  }
99
100 int digit_to_int(char c)
101 {
102     return ( static_cast<int>(c) - static_cast<int>('0') );
103 }
104 //Uses cstdlib and iostream:
105 ostream& operator <<(ostream& outs, const Money& amount)
106 {
107     long positive_cents, dollars, cents;
108     positive_cents = labs(amount.all_cents);
109     dollars = positive_cents/100;
110     cents = positive_cents%100;
111
112     if (amount.all_cents < 0)
113         outs << "-$" << dollars << '.';
114     else
115         outs << "$" << dollars << '.';
116
117     if (cents < 10)
118         outs << '0';
119     outs << cents;
120
121     return outs;
122 }
123
```

<The definitions of the member functions and other overloaded operators go here.
See Display 11.3, 11.4, 11.5, and 11.6 for the definitions.>

infile.dat
(Not changed by program.)

```
$1.11 $2.22
$3.33
```

outfile.dat
(After program is run.)

```
$1.11 copied from the file infile.dat.
```

Screen Output

```
$1.11 copied from the file infile.dat.
```

Display 11.8 (4/4)

File input and output will be discussed soon.

Section 11.2 Exercises

- Can you
  - Describe the purpose of a making a function a friend?
  - Describe the use of constant parameters?
  - Identify the return type of the overloaded operators << and >>?

11.3

Arrays and Classes

Arrays and Classes

- Arrays can use structures or classes as their base types
  - Example:

```
struct WindInfo
{
    double velocity;
    char direction;
}

WindInfo  data_point[10];
```

Accessing Members

- When an array's base type is a structure or a class…
  - Use the dot operator to access the members of an indexed variable
  - Example:

```
for (i = 0; i < 10; i++)
{
        cout << "Enter velocity: ";
        cin >> data_point[i].velocity;
         …
 }
```

An Array of Money

- The Money class of Chapter 11 can be the base type for an array

- When an array of classes is declared
  - The default constructor is called to initialize the indexed variables
- An array of class Money is demonstrated in

Display 11.9 (1-3)

DISPLAY 11.9 Program Using an Array of Money Objects *(part 1 of 3)*

```
1  //This is the definition for the class Money.
2  //Values of this type are amounts of money in U.S. currency.
3  #include <iostream>
4  using namespace std;
5    class Money
6    {
7    public:
8  friend Money operator +(const Money& amount1, const Money& amount2);
9  //Returns the sum of the values of amount1 and amount2.
10 friend Money operator -(const Money& amount1, const Money& amount2);
11 //Returns amount 1 minus amount2.
12 friend Money operator -(const Money& amount);
13 //Returns the negative of the value of amount.
14 friend bool operator ==(const Money& amount1, const Money& amount2);
15 //Returns true if amount1 and amount2 have the same value; false otherwise.
16 friend bool operator < (const Money& amount1, const Money& amount2);
17 //Returns true if amount1 is less than amount2; false otherwise.
18 Money(long dollars, int cents);
19 //Initializes the object so its value represents an amount with
20 //the dollars and cents given by the arguments. If the amount
21 //is negative, then both dollars and cents should be negative.
22 Money(long dollars);
23 //Initializes the object so its value represents $dollars.00.
24 Money( );
25 //Initializes the object so its value represents $0.00.
26 double get_value( ) const;
27 //Returns the amount of money recorded in the data portion of the calling
28 //object.
29 friend istream& operator >>(istream& ins, Money& amount);
30 //Overloads the >> operator so it can be used to input values of type
31 //Money. Notation for inputting negative amounts is as in -$100.00.
32 //Precondition: If ins is a file input stream, then ins has already been
33 //connected to a file.
34
35 friend ostream& operator <<(ostream& outs, const Money& amount);
36 //Overloads the << operator so it can be used to output values of type
37 //Money. Precedes each output value of type Money with a dollar sign.
38 //Precondition: If outs is a file output stream, then outs has already been
39 //connected to a file.
```

(continued)

Display 11.9 (1/3)

DISPLAY 11.9 Program Using an Array of Money Objects *(part 2 of 3)*

```
40      private:
41      long all_cents;
42  };
43
```

<The definitions of the member functions and the overloaded operators goes here.>

```
44  //Reads in 5 amounts of money and shows how much each
45  //amount differs from the largest amount.
46  int main( )
47  {
48      Money amount[5], max;
49      int i;
50      cout << "Enter 5 amounts of money:\n";
51      cin >> amount[0];
52      max = amount[0];
53      for (i = 1; i < 5; i++)
54      {
55          cin >> amount[i];
56          if (max < amount[i])
57              max = amount[i];
58          //max is the largest of amount[0],..., amount[i].
59      }
60      Money difference[5];
61      for (i = 0; i < 5; i++)
62          difference[i] = max - amount[i];
63      cout << "The highest amount is " << max << endl;
64      cout << "The amounts and their\n"
65           << "differences from the
66  largest are:\n";
67       for (i = 0; i < 5; i++)
68      {
69          cout << amount[i] << " off by "
70               << difference[i] << endl;
71      }
72      return 0;
73  }
```

Sample Dialogue

```
Enter 5 amounts of money:
$5.00 $10.00 $19.99 $20.00 $12.79
The highest amount is $20.00
The amounts and their
```

(continued)

Display 11.9 (2/3)

Display 11.9 (3/3)

DISPLAY 11.9 Program Using an Array of Money Objects *(part 3 of 3)*

```
differences from the largest are:
$5.00 off by $15.00
$10.00 off by $10.00
$19.99 off by $0.01
$20.00 off by $0.00
$12.79 off by $7.21
```

Arrays as Structure Members

- A structure can contain an array as a member
  - Example:

```
struct Data
{
        double time[10];
        int distance;
}

Data my_best;
```

  - my_best contains an array of type double

Accessing Array Elements

- To access the array elements within a structure
  - Use the dot operator to identify the array within the structure
  - Use the [ ]'s to identify the indexed variable desired
  - Example: `my_best.time[i]` references the i-th indexed variable of the variable time in the structure my_best

Arrays as Class Members

- Class TemperatureList includes an array
  - The array, named list, contains temperatures
  - Member variable size is the number of items stored

```
class TemperatureList
{
public:
  TemperatureList( );
  //Member functions
private:
   double   list [MAX_LIST_SIZE];
                // the allocated memory??
  int size;
}
```

Overview of TemperatureList

- To create an object of type TemperatureList:

  ```
  TemperatureList  my_data;
  ```

- To add a temperature to the list:

  ```
  My_data.add_temperature(77);
  ```

  - A check is made to see if the array is full

- << is overloaded so output of the list is

  ```
  cout << my_data;
  ```

Display 11.10 (1-2)

DISPLAY 11.10 Program for a Class with an Array Member *(part 1 of 2)*

```
1  //This is a definition for the class
2  //TemperatureList. Values of this type are lists of Fahrenheit temperatures.
3
4  #include <iostream>
5  #include <cstdlib>
6  using namespace std;
7
8  const int MAX_LIST_SIZE = 50;
9
10 class TemperatureList
11 {
12   public:
13     TemperatureList( );
14     //Initializes the object to an empty list.
15
16     void add_temperature(double temperature);
17     //Precondition: The list is not full.
18     //Postcondition: The temperature has been added to the list.
19
20     bool full( ) const;
21     //Returns true if the list is full; false otherwise.
22
23     friend ostream& operator <<(ostream& outs,
24                                 const TemperatureList& the_object);
25     //Overloads the << operator so it can be used to output values of
26     //type TemperatureList. Temperatures are output one per line.
27     //Precondition: If outs is a file output stream, then outs
28     //has already been connected to a file.
29   private:
30     double list[MAX_LIST_SIZE]; //of temperatures in Fahrenheit
31     int size; //number of array positions filled
32 };
33
34 //This is the implementation for the class TemperatureList.
35
36 TemperatureList::TemperatureList( ) : size(0)
37 {
38    //Body intentionally empty.
39 }
```

(continued)

Display 11.10 (1/2)

size is also used for next potentially available position in the array.

Display 11.10 (2/2)

DISPLAY 11.10 Program for a Class with an Array Member *(part 2 of 2)*

```
40  void TemperatureList::add_temperature(double temperature)
41  {//Uses iostream and cstdlib:
42      if ( full( ) )
43      {
44          cout << "Error: adding to a full list.\n";
45          exit(1);
46      }
47      else
48      {
49          list[size] = temperature;
50          size = size + 1;
51      }
52  }
53  bool TemperatureList::full( ) const
54  {
55       return (size == MAX_LIST_SIZE);
56  }
57  //Uses iostream:
58  ostream& operator <<(ostream& outs, const TemperatureList& the_object)
59  {
60     for (int i = 0; i < the_object.size; i++)
61          outs << the_object.list[i] << " F\n";
62      return outs;
63  }
```

Section 11.3 Conclusion

- Can you
  - Declare an array as a member of a class?
  - Declare an array of objects of a class?
  - Write code to call a member function of an element in an array of objects of a class?
  - Write code to access an element of an array of integers that is a member of a class?