

بسم الله الرحمن الرحيم

برنامه سازی پیشرفته



ابراهیم اردشیر لاریجانی

زمستان ۱۴۰۱

Chapter 11

Friends, Overloaded Operators,
and Arrays in Classes

Overview

1. Friend Functions
2. Overloading Operators
3. Arrays and Classes
4. Classes and Dynamic Arrays

11.1

Friend Functions

Friend Function

- Class operations are typically implemented as member functions
- Some operations are better implemented as ordinary (nonmember) functions

Program Example: An Equality Function

- The DayOfYear class from Chapter 10 can be enhanced to include an equality function
 - An equality function tests two objects of type DayOfYear to see if their values represent the same date
 - Two dates are equal if they represent the same day and month

Declaration of The equality Function

- We want the equality function to return a value of type bool that is true if the dates are the same
- The equality function requires a parameter for each of the two dates to compare
- The declaration is

```
bool equal(DayOfYear date1, DayOfYear date2);
```

- Notice that equal is not a member of the class DayOfYear

Defining Function equal

- The function equal, is not a member function
 - It must use public accessor functions to obtain the day and month from a DayOfYear object
- equal can be defined in this way:

```
bool equal(DayOfYear date1, DayOfYear date2)
{
    return ( date1.getMonth( ) == date2.getMonth( )
            &&
            date1.getDay( ) == date2.getDay( ) );
}
```


Using The Function equal

- The equal function can be used to compare dates in this manner

```
if ( equal( today, bach_birthday) )  
    cout << "It's Bach's birthday!";
```

- A complete program using function equal is found in

Display 11.1 (1)

Display 11.1 (2)

Display 11.1 (3)

```

1 //Program to demonstrate the function equal. The class DayOfYear
2 //is the same as in Self-Test Exercises 23-24 in Chapter 10.
3 #include <iostream>
4 using namespace std;

5 class DayOfYear
6 {
7 public:
8     DayOfYear(int theMonth, int theDay);
9         //Precondition: theMonth and theDay form a
10        //possible date. Initializes the date according
11        //to the arguments.

12     DayOfYear( );
13        //Initializes the date to January first.

14     void input( );

15     void output( );

16     int getMonth( );
17        //Returns the month, 1 for January, 2 for February, etc.

18     int getDay( );
19        //Returns the day of the month.
20 private:
21     void checkDate( );
22     int month;
23     int day;
24 };

25
26 bool equal(DayOfYear date1, DayOfYear date2);
27 //Precondition: date1 and date2 have values.
28 //Returns true if date1 and date2 represent the same date;
29 //otherwise, returns false.
30
31 int main( )
32 {
33     DayOfYear today, bachBirthday(3, 21);
34
35     cout << "Enter today's date:\n";
36     today.input( );
37     cout << "Today's date is ";
38     today.output( );
39
40     cout << "J. S. Bach's birthday is ";

```

Display 11.1 (1/3)



```

41     bachBirthday.output( );
42
43     if (equal(today, bachBirthday))
44         cout << "Happy Birthday Johann Sebastian!\n";
45     else
46         cout << "Happy Unbirthday Johann Sebastian!\n";
47     return 0;
48 }
49
50 bool equal(DayOfYear date1, DayOfYear date2)
51 {
52     return ( date1.getMonth( ) == date2.getMonth( ) &&
53             date1.getDay( ) == date2.getDay( ) );
54 }
55
56 DayOfYear::DayOfYear(int theMonth, int theDay)
57     : month(theMonth), day(theDay)
58 {
59     checkDate();
60 }
61
62 int DayOfYear::getMonth( )
63 {
64     return month;
65 }
66
67 int DayOfYear::getDay( )
68 {
69     return day;
70 }
71
72 //Uses iostream:
73 void DayOfYear::input( )
74 {
75     cout << "Enter the month as a number: ";
76     cin >> month;
77     cout << "Enter the day of the month: ";
78     cin >> day;
79 }
80
81 //Uses iostream:
82 void DayOfYear::output( )
83 {
84     cout << "month = " << month
85           << ", day = " << day << endl;
86 }

```

Omitted function and constructor definitions are as in Chapter 10, Self-Test Exercises 14 and 24, but those details are not needed for what we are doing here.

Display 11.1 (2/3)



Display 11.1

(3/3)



DISPLAY 11.1 Equality Function *(part 3 of 3)*

Sample Dialogue

```
Enter today's date:  
Enter the month as a number: 3  
Enter the day of the month: 21  
Today's date is month = 3, day = 21  
J. S. Bach's birthday is month = 3, day = 21  
Happy Birthday Johann Sebastian!
```

Is equal Efficient?

- Function equal could be made more efficient
 - Equal uses member function calls to obtain the private data values
 - Direct access of the member variables would be more efficient (faster)

A More Efficient equal

- As defined here, equal is more efficient, but not legal

```
bool equal(DayOfYear date1, DayOfYear date2)
{
    return (date1.month == date2.month
            &&
            date1.day == date2.day );
}
```

- The code is simpler and more efficient
- Direct access of private member variables is not legal!

Friend Functions

- Friend functions are not members of a class, but can access private member variables of the class
 - A friend function is declared using the keyword friend in the class definition
 - A friend function is not a member function
 - A friend function is an ordinary function
 - A friend function has extraordinary access to data members of the class
 - As a friend function, the more efficient version of equal is legal

Declaring A Friend

- The function equal is declared a friend in the abbreviated class definition here

```
class DayOfYear
{
    public:
        friend bool equal(DayOfYear date1,
                          DayOfYear date2);
        // The rest of the public members
    private:
        // the private members
};
```


Using A Friend Function

- A friend function is declared as a friend in the class definition
- A friend function is defined as a nonmember function without using the "::" operator
- A friend function is called without using the '.' operator

Display 11.2

Display 11.2



```
1 //Demonstrates the function equal.
2 //In this version equal is a friend of the class DayOfYear.
3 #include <iostream>
4 using namespace std;
5
6 class DayOfYear
7 {
8 public:
9     friend bool equal(DayOfYear date1, DayOfYear date2);
10    //Precondition: date1 and date2 have values.
11    //Returns true if date1 and date2 represent the same date;
12    //otherwise, returns false.
13
14    DayOfYear(int theMonth, int theDay);
15    //Precondition: theMonth and theDay form a
16    //possible date. Initializes the date according
17    //to the arguments.
18
19    DayOfYear( );
20    //Initializes the date to January first.
21
22    void input( );
23
24    void output( );
25
26    int getMonth( );
27    //Returns the month, 1 for January, 2 for February, etc.
28
29    int getDay( );
30    //Returns the day of the month.
31 private:
32     void checkDate( );
33     int month;
34     int day;
35 };
36
37 int main( )
38 {
```

<The main part of the program is the same as in Display 11.1.>

```
33 }
34
35 bool equal(DayOfYear date1, DayOfYear date2)
36 {
37     return (date1.month == date2.month &&
38            date1.day == date2.day);
39 }
40
```

Note that the private
member variables
month and day can
be accessed by name.

<The rest of this display, including the Sample Dialogue, is the same as in Display 11.1.>

Friend Declaration Syntax

- The syntax for declaring friend function is

```
class class_name
{
    public:
        friend Declaration_for_Friend_Function_1
        friend Declaration_for_Friend_Function_2
        ...
        Member_Function_Declarations
    private:
        Private_Member_Declarations
};
```

Are Friends Needed?

- Friend functions can be written as non-friend functions using the normal accessor and mutator functions that should be part of the class
- The code of a friend function is simpler and it is more efficient

Choosing Friends

- How do you know when a function should be a friend or a member function?
 - In general, use a member function if the task performed by the function involves only one object
 - In general, use a nonmember function if the task performed by the function involves more than one object
 - Choosing to make the nonmember function a friend is a decision of efficiency and personal taste

Program Example: The Money Class (version 1)

- Display 11.3 demonstrates a class called Money
 - U.S. currency is represented
 - Value is implemented as an integer representing the value as if converted to pennies
 - An integer allows exact representation of the value
 - Type long is used to allow larger values
 - Two friend functions, equal and add, are used

Display 11.3 (1 – 5)

```
1 //Program to demonstrate the class Money.
2 #include <iostream>
3 #include <cstdlib>
4 #include <cctype>
5 using namespace std;

6 //Class for amounts of money in U.S. currency.
7 class Money
8 {
9 public:
10     friend Money add(Money amount1, Money amount2);
11     //Precondition: amount1 and amount2 have been given values.
12     //Returns the sum of the values of amount1 and amount2.

13     friend bool equal(Money amount1, Money amount2);
14     //Precondition: amount1 and amount2 have been given values.
15     //Returns true if the amount1 and amount2 have the same value;
16     //otherwise, returns false.

17     Money(long dollars, int cents);
18     //Initializes the object so its value represents an amount with the
19     //dollars and cents given by the arguments. If the amount is negative,
20     //then both dollars and cents must be negative.

21     Money(long dollars);
22     //Initializes the object so its value represents $dollars.00.

23     Money( );
24     //Initializes the object so its value represents $0.00.

25     double getValue( );
26     //Precondition: The calling object has been given a value.
27     //Returns the amount of money recorded in the data of the calling object.

28     void input(istream& ins);
29     //Precondition: If ins is a file input stream, then ins has already been
30     //connected to a file. An amount of money, including a dollar sign, has been
31     //entered in the input stream ins. Notation for negative amounts is -$100.00.
32     //Postcondition: The value of the calling object has been set to
33     //the amount of money read from the input stream ins.

34     void output(ostream& outs);
35     //Precondition: If outs is a file output stream, then outs has already been
36     //connected to a file.
37     //Postcondition: A dollar sign and the amount of money recorded
38     //in the calling object have been sent to the output stream outs.
39 private:
40     long allCents;
41 };
```

Display 11.3 (1/5)



```

42  int digitToInt(char c);
43  //Function declaration for function used in the definition of Money::input:
44  //Precondition: c is one of the digits '0' through '9'.
45  //Returns the integer for the digit; for example, digitToInt ('3') returns 3.
46  int main( )
47  {
48      Money yourAmount, myAmount(10, 9), ourAmount;
49      cout << "Enter an amount of money: ";
50      yourAmount.input(cin);
51      cout << "Your amount is ";
52      yourAmount.output(cout);
53      cout << endl;
54      cout << "My amount is ";
55      myAmount.output(cout);
56      cout << endl;
57
58      if (equal(yourAmount, myAmount))
59          cout << "We have the same amounts.\n";
60      else
61          cout << "One of us is richer.\n";
62      ourAmount = add(yourAmount, myAmount);
63      yourAmount.output(cout);
64      cout << " + ";
65      myAmount.output(cout);
66      cout << " equals ";
67      ourAmount.output(cout);
68      cout << endl;
69      return 0;
70  }
71  Money add(Money amount1, Money amount2)
72  {
73      Money temp;
74
75      temp.allCents = amount1.allCents + amount2.allCents;
76      return temp;
77  }
78  bool equal(Money amount1, Money amount2)
79  {
80      return (amount1.allCents == amount2.allCents);
81  }
82
83  Money::Money(long dollars, int cents)
84  {
85      if (dollars * cents < 0) //If one is negative and one is positive

```

Display 11.3 (2/5)



Display 11.3 (3/5)



```
86     {
87         cout << "Illegal values for dollars and cents.\n";
88         exit(1);
89     }
90     allCents = dollars * 100 + cents;
91 }
92
93 Money::Money(long dollars) : allCents(dollars * 100)
94 {
95     //Body intentionally blank. 96     }
97
98 Money::Money( ) : allCents(0)
99 {
100     //Body intentionally blank. 101     }
102
103 double Money::getValue( )
104 {
105     return (allCents * 0.01);
106 }
107 //Uses iostream, ctype, cstdlib:
108 void Money::input(istream& ins)
109 {
110     char oneChar, decimalPoint, digit1, digit2;
111     //digits for the amount of cents
112     long dollars;
113     int cents;
114     bool negative; //set to true if input is negative.
115
116     ins >> oneChar;
117     if (oneChar == ' ')
118     {
119         negative = true;
120         ins >> oneChar; //read '$'
121     }
122     else
123         negative = false;
124     //if input is legal, then oneChar == '$'
125
126     ins >> dollars >> decimalPoint >> digit1 >> digit2;
127
128     if (oneChar != '$' || decimalPoint != '.'
129         || !isdigit(digit1) || !isdigit(digit2))
```

```

130     {
131         cout << "Error illegal form for money input\n";
132         exit(1);
133     }
134     cents = digitToInt(digit1) * 10 + digitToInt(digit2);
135
136     allCents = dollars * 100 + cents;
137     if (negative)
138         allCents = -allCents;
139 }
140
141 //Uses cstdlib and iostream:
142 void Money::output(ostream& outs)
143 {
144     long positiveCents, dollars, cents;
145     positiveCents = labs(allCents);
146     dollars = positiveCents / 100;
147     cents = positiveCents % 100;
148
149     if (allCents < 0)
150         outs << "-$" << dollars << '.';
151     else
152         outs << "$" << dollars << '.';
153
154     if (cents < 10)
155         outs << '0';
156     outs << cents;
157 }
158
159 int digitToInt(char c)
160 {
161     return ( static_cast<int>( c ) - static_cast<int>( '0' ) );
162 }
163

```

Display 11.3 (4/5)

Back

Next

Display 11.3 (5/5)



Sample Dialogue

```
Enter an amount of money: $123.45  
Your amount is $123.45  
My amount is $10.09  
One of us is richer.  
$123.45 + $10.09 equals $133.54
```

Characters to Integers

- Notice how function input (Display 11.3) processes the dollar values entered
 - First read the character that is a \$ or a –
 - If it is the -, set the value of negative to true and read the \$ sign which should be next
 - Next read the dollar amount as a long
 - Next read the decimal point and cents as three characters
 - digitToInt is then used to convert the cents characters to integers

digitToInt (optional)

- digitToInt is defined as

```
int digitToInt(char c)
{
    return ( static_cast<int> ( c ) – static_cast<int>( '0' ) );
}
```

- A digit, such as '3' is parameter c
 - This is the character '3' not the number 3
- The type cast static_cast<int>(c) returns the number that implements the character stored in c
- The type cast static_cast<int>('0') returns the number that implements the character '0'

$\text{int}(c) - \text{int}('0')$?

- The numbers implementing the digits are in order
 - $\text{int}('0') + 1$ is equivalent to $\text{int}('1')$
 - $\text{int}('1') + 1$ is equivalent to $\text{int}('2')$
- If c is '0'
 - $\text{int}(c) - \text{int}('0')$ returns integer 0
- If c is '1'
 - $\text{int}(c) - \text{int}('0')$ returns integer 1

Leading Zeros

- Some compilers interpret a number with a leading zero as a base 8 number
 - Base 8 uses digits 0 – 7
- Using 09 to represent 9 cents could cause an error
 - the digit 9 is not allowed in a base 8 number
- The ANSI C++ standard is that input should be interpreted as base 10 regardless of a leading zero

Parameter Passing Efficiency

- A call-by-value parameter less efficient than a call-by-reference parameter
 - The parameter is a local variable initialized to the value of the argument
 - This results in two copies of the argument
- A call-by-reference parameter is more efficient
 - The parameter is a placeholder replaced by the argument
 - There is only one copy of the argument

Class Parameters

- It can be much more efficient to use call-by-reference parameters when the parameter is of a class type
- When using a call-by-reference parameter
 - If the function does not change the value of the parameter, mark the parameter so the compiler knows it should not be changed

const Parameter Modifier

- To mark a call-by-reference parameter so it cannot be changed:
 - Use the modifier const before the parameter type
 - The parameter becomes a constant parameter
 - const used in the function declaration and definition

const Parameter Example

- Example (from the Money class of Display 11.3):
 - A function declaration with constant parameters
 - `friend Money add(const Money& amount1,
const Money& amount2);`
 - A function definition with constant parameters
 - `Money add(const Money& amount1,
const Money& amount2)
{
...
}`

const Considerations

- When a function has a constant parameter, the compiler will make certain the parameter cannot be changed by the function
 - What if the parameter calls a member function?

```
Money add(const Money& amount1,  
          const Money& amount2)  
{ ...  
  amount1.input( cin );  
}
```

- The call to input will change the value of amount1!

const And Accessor Functions

- Will the compiler accept an accessor function call from the constant parameter?

```
Money add(const Money& amount1,  
          const Money& amount2)  
{  
    ...  
    amount1.output(cout);  
}
```

- The compiler will not accept this code
 - There is no guarantee that output will not change the value of the parameter

const Modifies Functions

- If a constant parameter makes a member function call...
 - The member function called must be marked so the compiler knows it will not change the parameter
 - `const` is used to mark functions that will not change the value of an object
 - `const` is used in the function declaration and the function definition

Function Declarations With const

- To declare a function that will not change the value of any member variables:
 - Use const after the parameter list and just before the semicolon

```
class Money
{
    public:
        ...
        void output (ostream& outs) const ;
        ...
}
```

Function Definitions

With const

- To define a function that will not change the value of any member variables:
 - Use const in the same location as the function declaration

```
void Money::output(ostream& outs) const
{
    // output statements
}
```


const Problem Solved

- Now that output is declared and defined using the const modifier, the compiler will accept this code
- ```
Money add(const Money& amount1,
 const Money& amount2)
{
 ...
 amount1.output(cout);
}
```

# const Wrapup

- Using const to modify parameters of class types improves program efficiency
  - const is typed in front of the parameter's type
- Member functions called by constant parameters must also use const to let the compiler know they do not change the value of the parameter
  - const is typed following the parameter list in the declaration and definition

**Display 11.4**

#### DISPLAY 11.4 The Class Money with Constant Parameters

```
1 //Class for amounts of money in U.S. currency.
2 class Money
3 {
4 public:
5 friend Money add(const Money& amount1, const Money& amount2);
6 //Precondition: amount1 and amount2 have been given values.
7 //Returns the sum of the values of amount1 and amount2.
8
9 friend bool equal(const Money& amount1, const Money& amount2);
10 //Precondition: amount1 and amount2 have been given values.
11 //Returns true if amount1 and amount2 have the same value;
12 //otherwise, returns false.
13
14 Money(long dollars, int cents);
15
16 //Initializes the object so its value represents an amount with the
17 //dollars and cents given by the arguments. If the amount is negative,
18 //then both dollars and cents must be negative.
19
20 Money(long dollars);
21 //Initializes the object so its value represents $dollars.00.
22
23 Money();
24 //Initializes the object so its value represents $0.00.
25
26 double getValue() const;
27 //Precondition: The calling object has been given a value.
28 //Returns the amount of money recorded in the data of the calling object.
29
30 void input(istream& ins);
31 //Precondition: If ins is a file input stream, then ins has already been
32 //connected to a file. An amount of money, including a dollar sign, has been
33 //entered in the input stream ins. Notation for negative amounts is -$100.00.
34 //Postcondition: The value of the calling object has been set to
35 //the amount of money read from the input stream ins.
36
37 void output(ostream& outs) const;
38 //Precondition: If outs is a file output stream, then outs has already been
39 //connected to a file.
40 //Postcondition: A dollar sign and the amount of money recorded
41 //in the calling object have been sent to the output stream outs.
42
43 private:
44 long allCents;
45 };
```

## Display 11.4



# Use const Consistently

- Once a parameter is modified by using const to make it a constant parameter
  - Any member functions that are called by the parameter must also be modified using const to tell the compiler they will not change the parameter
- It is a good idea to modify, with const, every member function that does not change a member variable

# Section 11.1 Conclusion

- Can you
  - Describe the promise that you make to the compiler when you modify a parameter with `const`?
  - Explain why this declaration is probably not correct?

```
class Money
{ ...
 public:
 void input(istream& ins) const;
 ...
};
```

# Chapter 11 -- End

