Classes Introduction

Modified from Sections 10.2 and 11.1

What Is a Class?

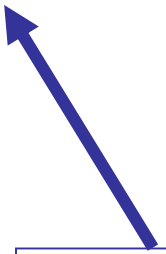
- A class is a user-defined data type
  - A pre-defined class (type) you have used is string
- A class can include
  - Member variables
  - Member functions
  - Descriptions need to be given for both types of members when defining a class
- Users can define their own classes

A Class Example

- To create a new type named DayOfYear as a class definition
  - Decide on the values to represent
  - This example's values are dates such as July 4 using an integer for the number of the month
    - Member variable month is an int (Jan = 1, Feb = 2, etc.)
    - Member variable day is an int
  - Decide on the member functions needed
  - We use just one member function named output

Class DayOfYear Definition

```
class DayOfYear
{
    public:
        void output( );
        int month;
        int day;
};
```



Member Function **Declaration**

Defining a Member Function

- Member functions are declared in the class declaration
- Member function definitions identify the class in which the function is a member
  - Member function definition syntax:
    Returned_Type Class_Name::Function_Name(Parameter_List)
    {
        Function Body Statements
    }
  - void DayOfYear::output()
    {
        cout << “month = “ << month
            << “, day = “ << day << endl;
    }

The ‘::’ Operator

- ‘::’ is the scope resolution operator
  - Tells the class a member function is a member of
  - void DayOfYear::output( ) indicates that function output is a member of the DayOfYear class
  - The class name that precedes ‘::’ is a type qualifier

Calling Member Functions

- Calling the DayOfYear member function output is done in this way:

```
DayOfYear today, birthday;
today.output( );
birthday.output( );
```

  - Note that today and birthday have their own versions of the month and day variables for use by the output function

Encapsulation

- Encapsulation is
  - Combining a number of items, such as variables and functions, into a single package such as an object of a class
- Encapsulation hides implementation details from the users of a class
- Encapsulation makes a program easier to maintain and reuse

Problems With DayOfYear

- Changing how the month is stored in the class DayOfYear requires changes to the program
- If we decide to store the month as an array of three characters ({‘J’, ‘A’, ‘N’}, etc.) instead of an integer
  - cin >> today.month will no longer work because we now have three character variables to read
  - if (today.month == birthday.month) will no longer work to compare months
  - The member function “output” no longer works

Ideal Class Definitions

- Changing the implementation of DayOfYear requires changes to the program that uses DayOfYear
- An ideal class definition of DayOfYear could be changed without requiring changes to the program that uses DayOfYear
- We can make the variables inaccessible to the users of this class, and add (accessible) member functions to use for changing or accessing the member variables

A New DayOfYear

- The new DayOfYear class demonstrated in Display 10.4
  - Uses all private member variables
  - Uses member functions to do all manipulation of the private member variables
    - Member variables and member function definitions can be changed without changes to the program that uses DayOfYear

Display 10.4 (1)

Display 10.4 (2)

Display 10.4 (1/2)

DISPLAY 10.4 Class with Private Members *(part 1 of 2)*

```
1  //Program to demonstrate the class DayOfYear.
2  #include <iostream>
3  using namespace std;
4  class DayOfYear
5  {
6  public:
7      void input( );
8      void output( );
9      void set(int new_month, int new_day);
10     //Precondition: new_month and new_day form a possible date.
11     //Postcondition: The date is reset according to the arguments.
12     int get_month( );
13     //Returns the month, 1 for January, 2 for February, etc.
14     int get_day( );
15     //Returns the day of the month.
16 private:
17     void check_date( );
18     int month;
19     int day;
20 };
21 int main( )
22 {
23     DayOfYear today, bach_birthday;
24     cout << "Enter today's date:\n";
25     today.input( );
26     cout << "Today's date is ";
27     today.output( );
28     bach_birthday.set(3, 21);
29     cout << "J. S. Bach's birthday is ";
30     bach_birthday.output( );
31     if ( today.get_month( ) == bach_birthday.get_month( ) &&
32               today.get_day( ) == bach_birthday.get_day( ) )
33         cout << "Happy Birthday Johann Sebastian!\n";
34     else
35         cout << "Happy Unbirthday Johann Sebastian!\n";
36     return 0;
37 }
38 //Uses iostream:
39 void DayOfYear::input( )
40 {
41     cout << "Enter the month as a number: ";
```

This is an improved version of the class `DayOfYear` *that we gave in Display 10.3.*

Private member function

Private member variables

(continued)

Display 10.4 (2/2)

DISPLAY 10.4 Class with Private Members *(part 2 of 2)*

```
42      cin >> month;
43      cout << "Enter the day of the month: ";
44      cin >> day;
45      check_date( );
46  }
47
48  void DayOfYear::output( )
     <The rest of the definition of DayOfYear::output is given in Display 10.3.>
49
50  void DayOfYear::set(int new_month, int new_day)
51  {
52      month = new_month;
53      day = new_day;
54      check_date();
55  }
56
57  void DayOfYear::check_date( )
58  {
59      if ((month < 1) || (month > 12) || (day < 1) || (day > 31))
60      {
61          cout << "Illegal date. Aborting program.\n";
62          exit(1);
63      }
64  }
65
66  int DayOfYear::get_month( )
67  {
68      return month;
69  }
70
71  int DayOfYear::get_day( )
72  {
73      return day;
74  }
```

Private members may be used in member function definitions (but not elsewhere).

A better definition of the member function `input` *would ask the user to reenter the date if the user enters an incorrect date.*

The member function `check_date` *does not check for all illegal dates, but it would be easy to make the check complete by making it longer. See Self-Test Exercise 14.*

The function `exit` *is discussed in Chapter 6. It ends the program.*

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!
```

Public Or Private?

- C++ helps us restrict the program from directly referencing member variables
  - Private members of a class can only be referenced within the definitions of member functions
    - If the program tries to access a private member, the compiler gives an error message
  - Private members can be variables or functions

Using Private Variables

- It is normal to make all member variables private
- Private variables require member functions to perform all changing and retrieving of values
- Accessor functions allow you to obtain the values of member variables
  - Example: get_day in class DayOfYear
- Mutator functions allow you to change the values of member variables
  - Example: set in class DayOfYear

General Class Definitions

- The syntax for a class definition is
  - class Class_Name
    {
        public:
            Member_Specification_1
            Member_Specification_2
            …
            Member_Specification_3
        private:
            Member_Specification_n+1
            Member_Specification_n+2
            …
    };

Declaring an Object

- Once a class is defined, an object of the class is declared just as variables of any other type
  - Example: To create two objects of type Bicycle:

```
class Bicycle
{
    // class definition lines
};

Bicycle my_bike,  your_bike;
```

The Assignment Operator

- Objects and structures can be assigned values with the assignment operator (=)
  - Example:

```
DayOfYear  due_date, next_friday;
next_friday.set(1, 31);
due_date = next_friday;
```

Class Objects as Function Parameters

- A call-by-value parameter less efficient than a call-by-reference parameter
  - a local variable is created with the argument's value
- It can be much more efficient to use call-by-reference parameters when the parameter is of a class type (size is large or unknown)
- Use the modifier const before the parameter type to mark a call-by-reference parameter so it cannot be changed

const Parameter Example

- A function definition with constant parameters
  - bool equal(const DayOfYear &day1,
    const DayOfYear &day2)
    {
    …
    }
  - day1 and day2's data members cannot be modified within the function body

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?

```
bool equal(const DayOfYear &day1,
           const DayOfYear &day2)
{
    day1.output();
    …
}
```

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

Constant Member Function

- The member function called must be marked with the const modifier, so the compiler knows it will not change the parameter
- const is used in the function declaration and definition
  - Declaration in class:

```
class DayOfYear{
    public:
        void output () const ;
        …
};
```

  - Definition:

```
void DayOfYear::output() const{
    // …function body
}
```

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 every member function that does not change a member variable with const