Constructors and Friend Functions

Modified from Sections 10.2, 11.1

Constructors in Classes

Modified from Section 10.2

Constructors

- A constructor is a special member function that can be used to initialize an object when an object is declared
  - A constructor is usually public
  - A constructor of a class is automatically called when an object of this class is declared
  - A constructor's name must be the name of the class
  - A constructor cannot return a value
    - No return type, not even void, is used in declaring or defining a constructor

Constructor Declaration

- A constructor for the DayOfYear class could be declared as:

```
class DayOfYear
{
      public:
            DayOfYear(int init_month, int init_day);
            //initializes the month to “month”
            //initializes the day to “day”

           …//The rest of the DayOfYear definition
};
```

Constructor Definition

- The constructor for the DayOfYear class could be defined as

```
DayOfYear::DayOfYear(int init_month, int init_day)
{
     month = init_month;
     day = init_day;
}
```

  - Note that the class name and function name are the same

Calling A Constructor

- A constructor is not called like a normal member function:

```
DayOfYear  day1;
day1.DayOfYear(1,26);
```

- A constructor is called in the object declaration

```
DayOfYear day1(1,26);
```

  - Creates a DayOfYear object and calls the constructor to initialize the member variables

Overloading Constructors

- Constructors can be overloaded by defining constructors with different parameter lists
  - Other possible constructors for the DayOfYear class might be

```
DayOfYear (int init_month, int init_day);
DayOfYear (int init_month);
DayOfYear ( );
```

- A constructor uses no parameters is called the default constructor
  - An argument list is not used when calling a default constructor
  - Example: DayOfYear day1;

    ~~DayOfYear day1**()**; // is not legal~~
  - It is a good idea to always include a default constructor even if you do not want to initialize variables

Initialization Sections

- An initialization section in a function definition provides an alternative way to initialize member variables

```
DayOfYear::DayOfYear( ): month(1), day(1)
{
        // No code needed in this example
}
```

  - The values in parenthesis are the initial values for the member variables listed

Parameters and Initialization

- Member functions with parameters can use initialization sections

```
DayOfYear::DayOfYear(int init_month, int init_day):
month(init_month), day(init_day)
{
      // No code needed in this example
}
```

  - Notice that the parameters can be arguments in the initialization

Member Initializers

- C++11 supports a feature called member initialization
  - Simply set member variables in the class
  - Example:

```
class DayOfYear
{
    private:
        int month = 1;
        int day = 1;
    ...
};
```

  - Creating a DayOfYear object will initialize its month variable to 1 and day to 1 (assuming a constructor isn’t called that sets the values to something else)

Constructor Delegation

- C++11 also supports constructor delegation, i.e., you can have a constructor invoke another constructor in the initialization section.
  - For example, make the default constructor call a second constructor that sets month to 1 and day to 1:

```
DayOfYear::DayOfYear(): DayOfYear(1,1)
{
        // No code needed in this example
}
```

Friend Functions

Modified from Section 11.1

Friend Functions

- Class operations are typically implemented as member functions
- Some operations are better implemented as ordinary (nonmember) functions
- In general, private members of a class cannot be accessed outside this class
- A class can grant access to private members to nonmember function by declaring it as a friend

Friend Functions

- Friend functions are not members of a class, but can access private member variables, or call private member functions 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

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
 };
```

Program Example: An Equality Function

- The DayOfYear class 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

An Equality Function - Declaration

- 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

An Equality Function – Definition 1

- The function equal, is not a member function
  - If we define it as a regular nonmember 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.get_month( ) == date2.get_month( )
                  && date1.get_day( ) == date2.get_day( ) );
}
```

An Equality Function – Definition 2

- The function equal can also be 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
};
```

An Equality Function – Definition 2

- After being declared as a friend of the DayOfYear class, the equal function gets direct access of private members
- equal can be defined as:

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

- Notice that a friend function's definition does NOT start with the keyword friend

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

Choosing Friends

- How do you know when a function should be a friend or a member function?
  - Use a member when you can, and a friend when you have to.
  - Member functions and friend functions are equally privileged.
  - The major difference is that a friend function is called like f(x), while a member function is called like x.f().