C++ for Science and Engineering COSC3000/6000

2018 Spring Semester

# Part VII
# Friends and const member functions

## 1 Friend Function

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

### 1.1 Program Example : An Equality Function

**sample24.cpp: DayOfYear** class

```
#include <iostream>
// class definition
class DayOfYear{
  public:
    void output( ); //Member Function Declaration
    void set(int new_month, int new_day);
    int get_day();
    int get_month();
  private:
    int month;
    int day;
};
// Member Function definition (implementation)
void DayOfYear::output(){
    std::cout << "month = " << month <<  ", day = " << day << std::endl;
}
void DayOfYear::set(int new_month, int new_day){
    month = new_month;
    day = new_day;
}
int DayOfYear::get_day(){
    return day;
}
int DayOfYear::get_month(){
    return month;
}
```

```
// main function
int main(int argc, const char * argv[]) {
    DayOfYear today;
    today.set(2, 17);
    today.output();
    return 0;
}
```

- The **DayOfYear** class from **"Defining Classes"** 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

### 1.1.1 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 foreach 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**

### 1.1.2 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.get_month( ) == date2.get_month( ) &&
                        date1.get_day( ) == date2.get_day( ) );
}
```

### 1.1.3 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!";
```

### 1.1.4 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)

### 1.1.5 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!**

## 1.2 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

## 1.3 Declaring A Friend

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

```
// class definition
class DayOfYear{
  public:
        DayOfYear(int month, int day);
        // precondition : month and day form a possible date.
        // Initializes the date according to the arguments.
        DayOfYear();
        // Initialize the tade to January first
        friend bool equal(DayOfYear date1, DayOfYear date2);
        // precondition : date1 and date2 have values.
        // returns true if date1 and date2 represents the same date;
        // otherwise,  return false
    void output( ); //Member Function Declaration
    void set(int new_month, int new_day);
    int get_day();
    int get_month();
  private:
    int month;
    int day;
};
```

## 1.4 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

**sample32.cpp**

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

// main function
int main(int argc, const char * argv[]) {
    DayOfYear today(2,29);
    DayOfYear leapday(2, 29);

    if (equal(today,leapday)){
        std::cout << "Today is leap day\n";
    }
    today.output();

    return 0;
}
```

## 1.5 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
};
```

## 1.6 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

## 1.7 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 non-member function if the task performed by the function involves more than one object
    - Choosing to make the non-member function a friend is a decision of efficiency and personal taste