3 Classes

- A class is a data type whose variables are objects
  - The definition of a class includes
    - Description of the kinds of values of the member variables
    - Description of the member functions
  - A class description is somewhat like a structure definition plus the member variables

3.1 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 4using 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

3.1.1 Class DayOfYear Definition

```
class DayOfYear
{
        public:
                void output( ); //Member Function Declaration
                int month;
                int day;

};
```

- public members are accessible from outside of class.
- private members are not accessible from outside of class.
- Major difference between "class" and "struct" in C++

	class	struct
public		default
private	default	

3.1.2 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

```
void DayOfYear::output()
{
        cout << "month = " << month
                << ",  day = "
                << day
                << endl;
}
```

Member function definition syntax:

```
Returned_Type Class_Name::Function_Name(Parameter_List)
{
        Function Body Statements
}
```

3.1.3 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

3.1.4 '::' and '.'

- '::' used with **classes** to identify a member

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

- '.'used with **variables** to identify a member

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

3.1.5 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

3.1.6 sample23.cpp

```
//
// sample23.cpp
//

#include <iostream>

// class definition
class DayOfYear
{
public:
    void output( ); //Member Function Declaration
    int month;
    int day;

};

// Member Function definition (implementation)
void DayOfYear::output()
{
    std::cout << "month = " << month
            <<  ",  day = " << day
            << std::endl;
}

// main function
int main(int argc, const char * argv[]) {
    DayOfYear today;
    today.month = 2;
    today.day = 17;

    today.output();

    return 0;
}
```

3.2 Encapsulation

- Encapsulation is
  - Combining a number of items, such as variables and functions, into a single package such as an object of a class
  - Information hiding

3.3 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 three characters (JAN, FEB, etc.) instead of an int
  - **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

3.3.1 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**

3.3.2 Fixing DayOfYear

- To fix **DayOfYear**
  - We need to add member functions to use when changing or accessing the member variables
    - If the program never directly references the member variables, changing how the variables are stored will not require changing the program
  - We need to be sure that the program does not ever directly reference the member variables

3.3.3 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

3.3.4 Private Variables

- Private variables cannot be accessed directly by the program
  - Changing their values requires the use of public member functions of the class
  - To set the private month and day variables in a new **DayOfYear** class use a member function such as

```
void DayOfYear::set(int new_month, int new_day)
{
        month = new_month;
        day = new_day;
}
```

3.3.5 Public or Private Members

- The keyword **private** identifies the members of a class that can be accessed only by member functions of the class
  - Members that follow the keyword private are private members of the class
- The keyword **public** identifies the members of a class that can be accessed from outside the class
  - Members that follow the keyword public are public members of the class

3.4 A New DayOfYear

```
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;

};
```

- 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**

3.4.1 sample24.cpp

```
#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;
}
```

3.5 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**

3.6 Summary

3.6.1 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
                :
};
```

3.6.2 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;
```

3.7 The Assignment Operator

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

```
DayOfYear due_date, tomorrow;
tomorrow.set(11, 19);
due_date = tomorrow; // copy values of all member variables
```

3.8 Program Example:BankAccount Class

- This bank account class allows
  - Withdrawal of money at any time
  - All operations normally expected of a bank account(implemented with member functions)
  - Storing an account balance
  - Storing the account's interest rate

3.8.1 Class Design (Definition)

```
class BankAccount
{
public:
    void set(int dollars, int cents, double rate);
    // Postcondition: The account balance has been set $dollars.cents;
    // The interest rate has been set to rate percent.

    void set(int dollars, double rate);
    // Postcondition: The account balance has benn set to $dollars.00.
    // The interest rate has been set to rate percent.

    void update();
    // Postcodition: One year of simple interest has been
    // added to the account balance.

    double get_balance();
    // Returns the current account balance.

    double get_rate();
    // Returns the current account interest rate as a percentage.

    void output(std::ostream& outs);
    // Precondition: If outs is a file output stream, outs has already
    // been connected to a file.
    // Postcondition: Account balance and interest rate have been
    // written to the stream outs.

private:
    double balance;
    double interest_rate;

    double fraction(double percent);
    // Converts a percentage to a fraction.
};
```

3.8.2 Member functions definition (implementation)

- set(int dollars, int cents, double rate)

```
void BankAccount::set(int dollars, int cents, double rate)
{
    if ((dollars < 0) || (cents < 0) || (rate < 0)){
        std::cout << "Illegal values for money or interest rate.\n";
        exit(-1);
    }
    balance = dollars + 0.01 * cents;
    interest_rate = rate;
}
```

- set(int dollars, double rate)

```
void BankAccount::set(int dollars, double rate)
{
    set(dollars,0,rate);
}
```

- update()

```
void BankAccount::update()
{
    balance += fraction(interest_rate) * balance;
}
```

- get_balance()

```
double BankAccount::get_balance()
{
    return balance;
}
```

- get_rate()

```
double BankAccount::get_rate()
{
    return interest_rate;
}
```

- output(std::ostream& outs)

```
void BankAccount::output(std::ostream& outs)
{
    outs.setf(std::ios::fixed);
    outs.setf(std::ios::showpoint);
    outs.precision(2);
    outs << "Account balance $" << balance << std::endl;
    outs << "Interest rate " << interest_rate << "%" << std::endl;
}
```

- fraction(double percent)

```
double BankAccount::fraction(double percent) // this is a private function
{
    return percent / 100.0;
}
```

3.8.3 main

```
// main function
int main(int argc, const char * argv[]) {
    BankAccount account1, account2;
    std::cout << "Start Test:\n";

    account1.set(123, 99, 3.0);
    std::cout << "account1 initial statement:\n";
    account1.output(std::cout);

    account1.set(100, 5.0);
    std::cout << "account1 with new setup:\n";
    account1.output(std::cout);

    account2 = account1;
    std::cout << "account2:\n";
    account2.output(std::cout);

    return 0;
}
```

see "**sample25.cpp**"

3.9 Summary

3.9.1 Calling Public Members

- Recall that if calling a member function from the main function of a program, you must include the the object name:

```
account1.output(std::cout);
```

3.9.2 Calling Private Members

- When a member function calls a private member function, an object name is not used
  - **fraction (double percent);** is a private member of the **BankAccount** class
  - **fraction** is called by member function update

```
void BankAccount::update( )
{
        balance += fraction(interest_rate) * balance;
}
```