Classes in C++

CS 16: Solving Problems with Computers I
Lecture #15 PRE-RECORDED

Ziad Matni
Dept. of Computer Science, UCSB

```
int main(int argc, char *argv[])
{
    if (argc > 1)
        filename = argv[1];
    ifstream setIn(filename);
    ifstream vecIn(filename);
    set<string> wordSet = getWordSet(setIn);
    vector<string> wordVec = getWordVec(vecIn);
    map<string, string> wordMap = generateMap(wordVec);

    string name = filename.substr(0, filename.size() - 4);
    string setFilename = name + "_set.txt";
    string vecFilename = name + "_vec.txt";
    string mapFilename = name + "_1_1.txt";

    // Writes set file
    ofstream setOut(setFilename);
    for (set<string>::iterator it = wordSet.begin(); it != wordSet.end(); it++)
    {
        setOut << *it << endl;
    }
    setOut.close();

    // Writes vector file
    ofstream vecOut(vecFilename);
    for (int i = 0; i < wordVec.size(); ++i)
    {
        vecOut << wordVec[i] << endl;
    }
    // Writes to map
    ofstream mapOut(mapFilename);
    printMap(wordMap, mapOut);
    mapOut.close();

    // Generate and print random string
    string str = "";
    for (int i = 0; i < 100; i++)
    {
        cout << wordMap[str] << " ";
        str = wordMap[str];
    }
    cout << endl << endl << endl;

    // Generate more intelligent map
    map<string, vector<string>> wordVecMap;
    str = "";
    for (int i = 0; i < wordVec.size(); i++)
    {
        wordVecMap[str].push_back(wordVec[i]);
        str = wordVec[i];
    }
```

Part 1 of 2

Lecture Outline

- Class Definition and Declaration
- Member Functions
- Public and Private Members

- Declarations and Assignments
- Constructors

- Follows textbook section **10.2**

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**

- **Class** descriptions look a lot like **struct** descriptions

A Class Example

- Let's create a new type of variable named **`DayOfYear`** as a class definition

- Member variables?
  - Member variable `month` is an **int** (Jan = 1, Feb = 2, etc.)
  - Member variable `day` is an **int**

- Member functions?
  - We use just one member function named `output`

Class Definition and Declaration

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

Member Function **Declaration**

Member Function **Definition**

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

- Member functions are **declared** in the class declaration
- Member function ***definitions*** identify the class in which the function is a member using the **::** operator

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**
  - **NOTE:** Different classes can have same-named member functions!

- Whereas **.** is used with **variables** to identify a member

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

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

Display 10.3
From textbook

```
3   #include <iostream>
4   using namespace std;

5   class DayOfYear
6   {
7   public:
8       void output( );
9       int month;
10      int day;
11  };

12  int main( )
13  {
14      DayOfYear today, birthday;

15      cout << "Enter today's date:\n";
16      cout << "Enter month as a number: ";
17      cin >> today.month;
18      cout << "Enter the day of the month: ";
19      cin >> today.day;
20      cout << "Enter your birthday:\n";
21      cout << "Enter month as a number: ";
22      cin >> birthday.month;
23      cout << "Enter the day of the month: ";
24      cin >> birthday.day;

25      cout << "Today's date is ";
26      today.output( );
27      cout << "Your birthday is ";
28      birthday.output( );

29      if (today.month == birthday.month
30          && today.day == birthday.day)
31          cout << "Happy Birthday!\n";
32      else
33          cout << "Happy Unbirthday!\n";

34      return 0;
35  }

36  //Uses iostream:
37  void DayOfYear::output( )
38  {
39      cout << "month = " << month
40           << ", day = " << day << endl;
41  }
```

Member function declaration

Calls to the member function **output**

Member function definition

Limitations With **DayOfYear**

- Changing how the month is stored in the class **DayOfYear** requires changes to the program

- Example: if we decide to store the month as 3 **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

Ideal Class Definitions

- Changing the implementation of **`DayOfYear`** requires changes to the program that uses **`DayOfYear`**
  - Not ideal

- An ideal class definition of **`DayOfYear`** could be changed without requiring changes to the program that uses **`DayOfYear`**

Fixing DayOfYear

- To fix this issue with **DayOfYear** we will need to do 2 things:
  - add member functions to use when changing or accessing the member variables
    - If the program never directly references the member variables, then changing how the variables are stored will not require changing the program!
  - be sure that the **program** does not directly reference member variables

Public Or Private?

- C++ helps us restrict the program from directly referencing member variables

- By default, all members of a class are **Public**
  - But we can make some of them "**Private**"

- **Private** members of a class can only be referenced *within the definitions of member functions, not by the program!*
  - If the program tries to access a private member, the compiler gives an error message
  - Private members can be variables or functions

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;
}
```

Public or Private Members

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

```
class DayOfYear {
    public:
        void input( );
        void output( );
        void set(int new_month, int new_day);
        int get_month( );
        int get_day( );
    private:
        void check_date( );
        int month;
        int day;
};
```

A New DayOfYear

- Take a **close look** at the new **DayOfYear** class demonstrated in Display 10.4 in the textbook
- 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**

```
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: ";
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  }
```

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

Private member function

Private member variables

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!
```

Using Private Variables

- It is conventionally normal to **make all member variables private**
- **Private** variables **require member functions** to perform all changing/retrieving of values

- *Accessor* functions allow you to obtain the values of member variables
  - Example: **`get_day`** in class **`DayOfYear`**

```
int DayOfYear::get_month( )
{
    return month;
}

int DayOfYear::get_day( )
{
    return day;
}
```

- *Mutator* functions allow you to change the values of member variables
  - Example: **`set`** in class **`DayOfYear`**

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

END OF PART 1

Classes in C++

CS 16: Solving Problems with Computers I
Lecture #15 PRE-RECORDED

Ziad Matni
Dept. of Computer Science, UCSB

```
int main(int argc, char *argv[])
{
    if (argc > 1)
        filename = argv[1];
    ifstream setIn(filename);
    ifstream vecIn(filename);
    set<string> wordSet = getWordSet(setIn);
    vector<string> wordVec = getWordVec(vecIn);
    map<string, string> wordMap = generateMap(wordVec);

    string name = filename.substr(0, filename.size() - 4);
    string setFilename = name + "_set.txt";
    string vecFilename = name + "_vec.txt";
    string mapFilename = name + "_1_1.txt";

    // Writes set file
    ofstream setOut(setFilename);
    for (set<string>::iterator it = wordSet.begin(); it != wordSet.end(); it++)
    {
        setOut << *it << endl;
    }
    setOut.close();

    // Writes vector file
    ofstream vecOut(vecFilename);
    for (int i = 0; i < wordVec.size(); ++i)
    {
        vecOut << wordVec[i] << endl;
    }
    // Writes to map
    ofstream mapOut(mapFilename);
    printMap(wordMap, mapOut);
    mapOut.close();

    // Generate and print random string
    string str = "";
    for (int i = 0; i < 100; i++)
    {
        cout << wordMap[str] << " ";
        str = wordMap[str];
    }
    cout << endl << endl << endl;

    // Generate more intelligent map
    map<string, vector<string>> wordVecMap;
    str = "";
    for (int i = 0; i < wordVec.size(); i++)
    {
        wordVecMap[str].push_back(wordVec[i]);
        str = wordVec[i];
    }
```

Part 2 of 2

Class Definition Syntax

```
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
  - i.e. Just like with **`structs`**

- Example: To create two objects of type Bicycle:

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

 Bicycle my_bike, your_bike;
```

The Assignment Operator

- Class objects (again, like structs) can be assigned values with the assignment operator (=)

- Example:

```
DayOfYear  due_date, tomorrow;

tomorrow.set(11, 19);

due_date = tomorrow;
```

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

- Review this in Display 10.5 in textbook

Calling Public Members

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

```
BankAccount account1;
…
account1.update();
```

Calling Private Members

- When a **member function** calls a **private member function**,
  an object name is not used
- Example:

`fraction (double percent);`
is a private member of the BankAccount class

and

`fraction` is called by member function update

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

Constructors

- A constructor can be used to **initialize** **member variables** when an object is declared
- A constructor is a **member function** **that is usually** **public**
- A constructor is **automatically called** when an object of the class is declared
- **A constructor's name must be the same 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 BankAccount class could be declared as:

```
class BankAccount
{
   public:
      BankAccount(int dollars, int cents, double rate);
      //initializes the balance to $dollars.cents
      //initializes the interest rate to “rate” percent

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

Constructor Definition

- The constructor for the BankAccount class could be defined like this (this example is in your textbook):

```
BankAccount::BankAccount(int dollars, int cents, double rate)
 {
    if ((dollars < 0) || (cents < 0) || ( rate < 0 ))
    {
         cout << “Illegal values for money or rate\n”;
         exit(1);
    }
    balance = dollars + 0.01 * cents;
    interest_rate = rate;
}
```

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

Calling A Constructor

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

```
BankAccount  account1;
account1.BankAccount(10, 50, 2.0);
```

- Instead, a constructor is called in the object declaration

```
BankAccount account1(10, 50, 2.0);
```

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

Overloading Constructors

- You can have **multiple constructors** for the same class!

- Constructors can be **overloaded** by defining constructors with different parameter lists

- Other possible constructors for the `BankAccount` class might be

```
BankAccount(double balance, double interest_rate);
BankAccount(double balance);
BankAccount(double interest_rate);
BankAccount( ); // Default constructor
```

Overloading Constructors

Another Example:

- The output of this prog. will be:

```
Person1 Age = 20
Person2 Age = 45
```

- Because obj. person1 was initialized with the 1st constructor…
  - Person()
  - Called the **default constructor**
- …while obj. person2 was initialized with the 2nd constructor
  - Person(int a)

```
class Person {
    private:
        int age;
    public:
    // 1. Constructor with no arguments (default constr.)
    Person() { age = 20; }
    // 2. Constructor with an argument
    Person(int a) {
        age = a;
    }
    int getAge() {
        return age;
    }
};
int main() {
    Person person1, person2(45);
    cout << "Person1 Age = " << person1.getAge() << endl;
    cout << "Person2 Age = " << person2.getAge() << endl;
    return 0;
}
```


Code from https://www.programiz.com/

Initialization Sections

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

- Example:

```
BankAccount::BankAccount( ): balance(0), interest_rate(0.0)
{
   // 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

```
BankAccount::BankAccount(int dollars, int cents, double rate)
                    : balance(dollars + 0.01 * cents), interest_rate(rate)
{
      if (( dollars < 0) || (cents < 0) || (rate < 0))
      {
            cout << “Illegal values for money or rate\n”;
            exit(1);
       }
}
```

- Notice that the parameters can be arguments in the initialization

Member Initializers

- C++11 (and beyond) supports a feature called member initialization
- Simply set member variables in the class to some initial value!
- Example:

```
class Coordinate {
      private:
         int x=1;
         int y=2;
   ...
   };
```

- Just creating a **`Coordinate`** object will initialize its **x** variable to 1 and **y** to 2 (assuming a constructor isn't called that sets the values to something else)

Constructor Delegation

- C++11 also supports **constructor delegation**.
- Constructor invokes *another constructor* in the initialization section.

- For example, you can make the **default constructor** call a **second constructor** that sets variable **x** to 99 and **y** to 99:

```
Coordinate::Coordinate() : Coordinate(99,99) { }
```

You Can See Examples of Constructors in Use…

… by checking out the demo codes on our web page:

constructorExamples1.cpp

constructorExamples2.cpp

constructorExamples3.cpp

And don't forget ALL examples in the relevant textbook section (**Section 10.2**)


</LECTURE>