

CSCI 1300 CS1: Starting Computing

Ashraf, Cox, Spring 2020

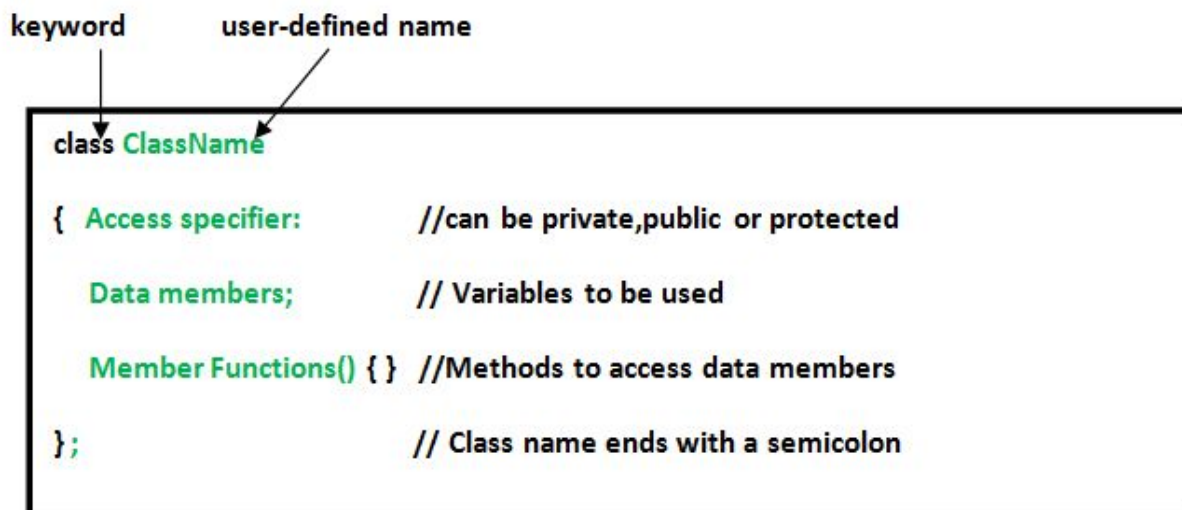
Homework 6

Due: Saturday, March 14, by 6 pm

(5 % bonus on the total score if all three parts are submitted by 11:59 pm March 14)

Classes

When writing complex programs, sometimes the built in data types (such as *int*, *char*, *string*) don't offer developers enough functionality or flexibility to solve their problems. A solution is for the developer (you - yes, **you!**) to create your own custom data types to use, called **classes**. Classes are user-defined data types, which hold their own **data members** and **member functions**, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object, customized for whatever particular problem the programmer is working on. Below is an example of the basic definition of a class in C++.



Let's break down the main components of this diagram:

Class Name

A class is defined in C++ using the keyword **class** followed by the name of class. The body of the class is defined inside the curly brackets and terminated by a semicolon at the end. This class name will be how you reference any *variables* or *objects* you create of that type. For example:

```
ClassName objectName;
```

The above line would create a new object (variable) with name `objectName` and of type `ClassName`, and this object would have all the properties that you have defined within the class `ClassName`.

Access Specifiers

Access specifiers in a class are used to set the accessibility of the class members. That is, they restrict access by outside functions to the class members. Consider the following analogy. Imagine an intelligence agency like the CIA, that has 10 senior members. Such an organization holds various sorts of information, and needs some way of determining who has access to any given piece of information.

Some information concerning classified or dangerous operations may only be accessible to the 10 senior members of the agency, and not directly accessible by any other person. This data would be **private**. In a class, like in our intelligence agency, **private** data is available only to specific data members and member functions and not directly accessible by any object or function outside the class.

Some other information may be available to anyone who wants to know about it. This is **public** data. Even people outside the CIA can know about it, and the agency might release this information through press releases or other means. In terms of our class, this **public** data can be accessed by any member function of the class, as well as outside functions and objects, even other classes. The **public** members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

Data Members and Member Functions

Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behavior of the objects in a Class. The data members declared in any class definition are fundamental data types (like int, char, float etc). For example, for strings objects, the data member is a `char array[]` that holds all of the individual letters of your string. Some of a string's member functions are functions like `.substr()`, `.find()`, and `.length()`.

Accessing Data Members

Keeping with our intelligence agency example, the code below defines a class that holds information for any general agency. In main, we then create a new `intelligenceAgency` object called `CIA`, and we set its `organizationName` to “CIA” by using the access operator (`.`) and the corresponding data member’s name. However, cannot access the `classifiedIntelligence` data member in the same way. Not everyone has access to that private data. Instead, we need to use a member function of the agency `getClassifiedIntelligence()` in order to see that information. However, we may write this function remove sensitive information from the string. This allows us to control the release of private information by our `intelligenceAgency`.

```
class intelligenceAgency
{
    public:
        intelligenceAgency();           // Default constructor
        intelligenceAgency(string s);   // Parameterized constructor
        string organizationName;
        string getClassifiedIntelligence();
        void setClassifiedIntelligence(string s);

    private:
        string classifiedIntelligence;
};

int main()
{
    intelligenceAgency CIA;
    CIA.organizationName = "CIA";
    cout << CIA.classifiedIntelligence; //gives an error
    cout << CIA.getClassifiedIntelligence();
}
```

Defining Member Functions

You may have noticed that we *declared* various member functions in our class definition, but that we didn’t specify how they will work when called. The body of the function still needs to be written. The solution is to define a function, such as `getClassifiedIntelligence()`, corresponding to our class’s functions. But how does our program know that these functions are the same as the ones we declared in our class? You as the developer need to explicitly tell the computer that these functions you are defining are the same ones you declared earlier.

```
string intelligenceAgency::getClassifiedIntelligence()
{
    return classifiedIntelligence;
}

void intelligenceAgency::setClassifiedIntelligence(string s)
{
    classifiedIntelligence = s;
}
```

We start the definition as we do any function, with the return type. Then, we have to add a specifier `intelligenceAgency::` that lets our program know that this function and the one we declared inside the class are the same. We can see that this function returns the class's `classifiedIntelligence` to the user. Functions like `getClassifiedIntelligence()` are called accessor, or **getter**, functions. This is because they retrieve or 'get' the private data members of a class object and return it to the program so that these values may be used elsewhere. The second function, `setClassifiedIntelligence(string s)`, is called a mutator, or a **setter**, function. These allow functions from other parts of our program to modify or 'set' the private data members of our class objects. Getters and setters are the functions that our program uses to interact with the private data members of our class objects.

Header and Source files

Creating our own classes with various data members and functions increases the complexity of our program. Putting all of the code for our classes as well as the main functionality of our program into one `.cpp` file can become confusing for you as a programmer, and we need ways of reducing the visual clutter that this creates. This is why, as we increase the complexity of a program, we might need to create multiple files: **header** and **source** files.

Header file

Header files have `“.h”` as their filename extensions. In a header file, we *declare* one or more of the complex structures (classes) we want to develop. In a class, we define member functions and member attributes. These functions and attributes are the building blocks of the class.

className.h

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

class className
{
    public:
        .
        .
        .
    private:
        .
        .
        .
};
```

Source file

Source files are recognizable by the “.cpp” extension. In a source file we *implement* the complex structures (class) defined in the header file. Since we are splitting the development of actual code for the class into a definition (header file) and an implementation (source file), we need to link the two somehow.

```
#include "className.h"
```

In the source file we will include the header file that defines the class so that the source file is “aware” of where we can retrieve the definition of the class. We must define the class definition in every source that wants to use our user defined data type (our class). When implementing each member function, our source files must tell the compiler that these functions are actually the methods defined in our class definition using the syntax that we showed earlier.

How to compile multiple .cpp and .h files

In this homework, it will be necessary to write multiple files (.h and .cpp) and test them before submitting them. You need to compile and execute your code **using the command line**. This means you need to type commands into a **bash** window instead of pressing the *Run* button as you may have been doing.

Make sure that you start by changing directories so that you are in the folder where your solution's files are stored. In this example, our folder will be called hmwk6. To change to this directory, use:

```
cd hmwk7/
```

When compiling from the command line, you need to specify each of the .cpp files in your project. This means that when you call the g++ compiler, you need to explicitly name the files you're compiling:

```
g++ -std=c++11 file1.cpp file2.cpp main.cpp
```

The compiling command results in the creation of an executable file. If you did not specify a name for this executable, it will be named a.out by default. To execute this file, use the command:

```
./a.out
```

You can add the -o flag to your compiling command to give the output file a name:

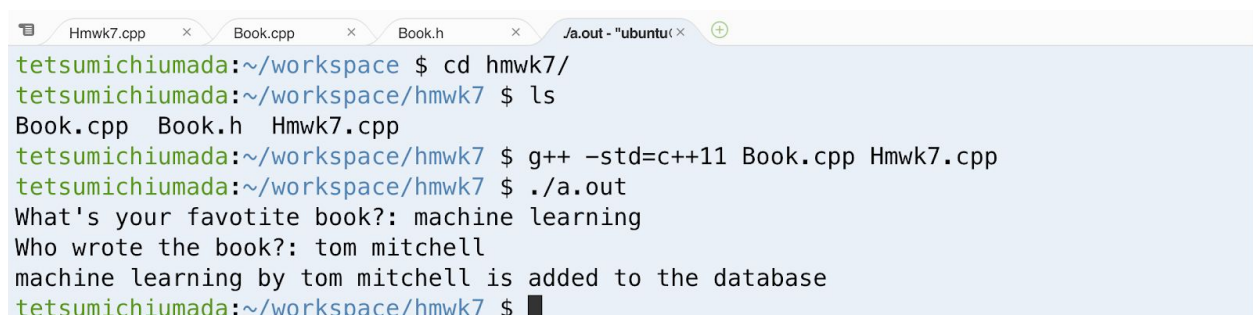
```
g++ -o myName.out -std=c++11 file1.cpp file2.cpp main.cpp
```

And then run the file with

```
./myName.out
```

Example 1:

Compiling book.cpp and Hmwk7.cpp. The picture below shows a **bash** window.



```
Hmwk7.cpp x Book.cpp x Book.h x /a.out - "ubuntu" x +
tetsumichiumada:~/workspace $ cd hmwk7/
tetsumichiumada:~/workspace/hmwk7 $ ls
Book.cpp Book.h Hmwk7.cpp
tetsumichiumada:~/workspace/hmwk7 $ g++ -std=c++11 Book.cpp Hmwk7.cpp
tetsumichiumada:~/workspace/hmwk7 $ ./a.out
What's your favorite book?: machine learning
Who wrote the book?: tom mitchell
machine learning by tom mitchell is added to the database
tetsumichiumada:~/workspace/hmwk7 $
```

Example 2:

Here, we have named the executable hmwk7

```
Hmwk7.cpp x Book.cpp x Book.h x /hmk7 - "ubunl x User.cpp x User.h x
tetsumichiumada:~/workspace $ cd hmk7/
tetsumichiumada:~/workspace/hmk7 $ ls
Book.cpp Book.h Hmwk7.cpp User.cpp User.h
tetsumichiumada:~/workspace/hmk7 $ g++ -std=c++11 Book.cpp User.cpp Hmwk7.cpp -o hmk7
tetsumichiumada:~/workspace/hmk7 $ ./hmk7
Select a numerical option:
=====Main Menu=====
1. Read book file
2. Read user file
3. Print book list
4. Find number of books user rated
5. Get average rating
6. Quit
6
good bye!
tetsumichiumada:~/workspace/hmk7 $ █
```