Classes

CS A150 - C++ Programming 1

Principles of OOP

- **Information Hiding**
  - Details of how operations work **not** known to "user" of class
- **Data Abstraction**
  - Details of how data is manipulated within ADT/class **not** known to "user" of class
- **Encapsulation**
  - Bring together data and operations,
    but keep "details" hidden

Classes

- Integral to object-oriented programming
- A class is a **type**
  - Just like **`int`**, **`double`**, etc.
- Can have variables of class type
  - We simply call them "**objects**"
- Naming a class:
  - Use ***CamelCase*** convention, *but* ***capitalize*** the ***first initial***

SEPARATE COMPILATION

- "User" of class need **not** see details of how class is implemented
  - Principle of OOP → **encapsulation**
- We will have two files
  - **Header file** (also called **interface**)
    - **.h** extension
    - Contains the **class definition** (***function prototypes***)
  - **Implementation file**
    - **.cpp** extension
    - Contains the class implementation
    - Implementation is hidden

INTERFACE

- To define a class,

  first specify its **public** ***interface***
- The *public interface* consists of all member functions we want to apply to objects of that type
- We are describing the object's ***behavior***
- Example:
  - Rectangle class member functions:
    - Make a new rectangle
    - Set the height and the width of the rectangle

Interface (cont.)

- The ***public interface*** can be divided logically into 3 parts:
  - **Constructors**
    - Initialize new objects
    - Same name as class
    - If **no** parameters is called the ***default constructor***
  - **Mutators**
    - Modify an object (modifies member variables)
  - **Accessors**
    - Simply query an object, without modifying it
    - Tagged as a `const` function

Syntax: Class Definition

```
class ClassName
{
public:
      constructor declarations
      member function declarations
      destructor
private:
      member variables
};
```

- In the **header** (**.h**) file
- **Purpose:** Define the interface and member variables of a class

Avoiding Multiple Inclusions

- Since other files can use the same class, you need to let the compiler know that there is **only** one class that has that name
- You need to frame your class definition:

```
#ifndef NAMEOFCLASS_H
#define NAMEOFCLASS_H
//class definition follows
class NameOfClass
{
   …
};
#endif
```

Note the semicolon

Avoiding Multiple Inclusions(cont.)

- Another option:
  - Using **#pragma once** in the ***header*** of the class
    - **BUT** not all compilers will recognize this command
    - So, we will adopt **#ifndef**, **#define**, **#endif**

Example 1

- File: Rectangle.h (in project rectangle_class)

Purpose of Constructors

- The purpose of a **constructor** is to:
  - Initialize member variables
  - Validate member variables
    - Ensure only appropriate data is assigned to class private member variables
    - Powerful **OOP** principle

Default Constructors

- A **default constructor**
  - Initializes ***all*** data fields of an object
  - ***Always*** has the **same name** as the **class**
  - Does **not** take any arguments
  - *Generally* set fields to a default value (if one makes sense)
  - You ***always*** include the default constructor in C++
- Each class needs *at least* 1 constructor

Constructors with Parameters

- Classes may have *multiple* constructors
- **Overloaded constructors**
  - ***All*** constructors have the **same name** as the **class** but have ***different*** *parameters*

- Declaration:

```
//default constructor
Rectangle();

//overloaded constructor
Rectangle(double newHeight, double newWidth);
```

Member Functions

- Functions that are part of the class are called **member functions**
- If a member function does ***not*** modify any of its own member variables, it should be tagged as **`const`**

Member vs. Non-member Functions

- Member functions
  - Belong to a **class**
  - Can access **private** members of the class
  - Can use **`const`** modifier
  - Need an **object** to call the function
- Non-member functions
  - Do **not** belong to a class
  - **Cannot** access any private members of any class
  - **Cannot** use **`const`** modifiers
  - Does **not** need an object to call a function

Destructor

- A **destructor** is automatically called when a value is destroyed
  - At the end of a block for any local variable
  - At the end of a function for any arguments
  - When the main function terminates for all static variables in the class
- We will leave the destructor **empty** for now
  - We will return to this when we cover **dynamic variables**
    - Dynamic variables are not delete automatically

Implementation

- The **implementation** of the class is
  - Where the **constructor(s)**, **destructor**, and **member functions** are defined
  - Each member's name needs to be preceded by:

    ```
    NameOfClass::
    ```

    - ***Scope resolution operator*** **" :: "**
      - Specifies what class the function definition comes from
    - ***Class qualifier***
      - The **type** name that precedes the scope resolution operator is called **class qualifier** → **Rectangle::**
  - Example:

    ```
    void Rectangle::calculateArea() const;
    ```

Implementation of Constructors

```
//default constructor
Rectangle::Rectangle()
{
  length = 0.0;     //set to default values
  width = 0.0;
}

//overloaded constructor
Rectangle::Rectangle(double newHeight, double newWidth)
{
  length = newLength;  //values were given
  width = newWidth;
}
```

Implementation of Member Functions

```
//member function
double Rectangle::getWidth( ) const
{
  return width;  //has direct access to private members
}

//member function
void Rectangle::setWidth( double newWidth )
{
  width = newWidth; //has direct access to private members
}
```

EXAMPLE 2

- File: Rectangle.cpp (in project rectangle_class)

Using the Class

- To use our Rectangle class, we need to include it in the file that will use the class

```
#include "Rectangle.h"
```

- We need to create an object of the class

```
Rectangle r1;    //the default constructor is used

Rectangle r2(3.0, 4.5);  //the overloaded constructor
                         //  is used
```

- Note that declaring an object using the **default constructor** does **not** require ()

Using the Class (cont.)

- Once you have your objects set, you can call any functions by using
  - object + dot operator + name of function + parameters

```
r1.setWidth(1.0);    //resets width and height for rectangle 1
r1.setHeight(2.0);
r2.setWidth(3.0);    //resets width and height for rectangle 2
r2.setHeight(4.0);

cout << r1.getWidth(); //outputs width of rectangle 1
```

Example 3

- Project: Rectangle_class

More Member Functions

- The member functions seen previously are the basic functions needed for any class
- Of course, we can add more functions to make our class more useful
- For example:
  - calculateArea( )
  - calculatePerimeter( )

Example 4

- Project: rectangle_class_modified

Terminology

- ***Scope resolution operator*** “ :: ”
  - Specifies what class the function definition comes from
- ***Class qualifier***
  - The type name that precedes the scope resolution operator is called class qualifier → Rectangle::
- ***Public members***
  - Accessible to anyone
- ***Private members***
  - Accessible only to the class
  - Member variables are always private (or ***protected***)

Terminology (cont.)

- ***Accessor functions***
  - Allow to read the member variables of a class
- ***Mutator functions***
  - Allow to change the values of the member variables of a class
- ***Dot operator*** “ **.** ”
  - Specifies member of particular object

Encapsulation

- **Encapsulation**
  - Means "bringing together as one"
- Declare a class → get an object
- Object is an "encapsulation" of
  - Data values
  - Operations on the data (*member functions*)

Most Common Errors

- Forgetting the ***semicolon*** at the end of the class definition
- Forgetting to write the class qualifier in the implementation

```
void Rectangle::print() const
```

- Not using `#include ClassName.h` in the class implementation file and the **drive file**
  - Note that the *drive file* (the file containing the main function) is also called the **application file** or the **main file**

Most Common Errors (cont.)

- Forgetting to initialize ***all*** member variables in a constructor
- Trying to ***reset*** an object by re-calling a constructor
  - A constructor creates an object for the first time
  - If you want to change parameters,

    call a mutator function

```
Rectangle r(3.5, 2.7);
r.setWidth(5.0);  //changes the width
                  //rectangle has now dimensions
                  //   3.5 and 5.0
```

Errors That Can Make You Lose Points…

- Forgetting the `const` modifier for functions
- Not writing comments for
  - Constructors
  - Member functions
  - Member variables

Files

- File: Cpp_Separate_Compilation

QUESTIONS?

(Classes)