Classes, Objects, Separation, and Constructors

Sections 7.12, 7.13, 7.14, and 7.15

A Time Class

```
class Time
{
  private:
    int hour;     // 0 - 23 (24-hour clock format)
    int minute;   // 0 - 59
    int second;   // 0 - 59

  public:
    void setTime(int h, int m, int s);
    void printUniversal();             // 13:27:06
    void printStandard();              // 1:27:06 PM

}; // end class Time
```


Member Function Definitions

- How would we write the definitions of the member functions?
- Where would they be written?

Objects

- How do we create objects of the class time?
  - A regular object
  - An array of objects
- How would you use the objects to call the member functions?

Separating Classes into Files (7.13)

- Every program we have written so far has been in one file (projectName.cpp)
- One of the principles of Software Engineering is to separate the interface from the implementation
- We will be storing class declarations and member functions in their own separate files

Separation

- The class declaration in a header file (.h). The name of the file is usually the same as the class name (e.g. Time.h)
- The definitions of the class member functions in a source file (.cpp). The name of the file is usually the same as the class name (e.g. Time.cpp)
- The main program is stored in its own source file (.cpp)

Splitting the Time Program

- How would we split the Time program into different files?

Notes on Separating into Files

- The class declaration should contain an *include guard* to prevent the header file from being included more than once

```
#ifndef TIME_H
#define TIME_H
Class declaration
#endif
```


Notes on Separating into Files

- The file containing the member function definitions (e.g. time.cpp) needs to include the class header
  - `#include "time.h"`
- The " indicate that the file is located in the current project directory
- Note: Only header files are ever included
  - `#include "time.cpp"  // ERROR!`

Constructor (7.14)

- Special member function to initialize data members
- It has the same name as the class
- It does not have a return value
- The constructor is called whenever an object of that class is created
- `Time();`

Constructor

- What would the implementation of the constructor look like?

```
Time::Time()
{
  hour = minute = second = 0;
}
```


Examples

- Which of the following statements is invalid and why?
  - `Time &cTimeRef;`
  - `cTimeRef.printUniversal;`
  -
```
Time cTimeArray[5];
for(int i = 0; i < 5; i++)
{
  cout << cTimeArray[i].printStandard();
}
```
  -
```
Time cTime;
cTime.hour = 14;
```

Object-Oriented Features

- Information hiding
  - Separate the implementation from the interface
  - Objects are concerned with the interface, for example what functions are available to manipulate the data
  - Objects are not concerned with the implementation. They do not care how the functions do what they do, as long as they do it correctly

Overloading Constructors (7.15)

- Recall from last semester that it is possible to create multiple functions with the same name
- How?

Overloaded Constructors

- Overloaded constructors are the same as overloaded functions
- We could have multiple constructors in the Time class, each of which accepts a different number of arguments
- The appropriate constructor will be chosen based on the number of arguments used when creating the object
- Create multiple constructors for Time

Default Constructor

- The default constructor is the constructor with no arguments
- If you do not create any constructors in your class, then the default constructor will be created for you
- If you have a constructor that takes arguments, then the default constructor will be created for you
- It is good programming practice to always create a default constructor, why?

Default Arguments (7.15)

- You can set default arguments to constructors
- In the class definition, the constructor prototype will be
  - `Time(int = 0, int = 0, int = 0);`
- The function definition will be

```
Time::Time(int hr, min, int sec)
{
   setTime(hr, min, sec);
}
```


Constructors

- By having default arguments in the constructor, we can now create objects of the Time class as follows:

```
Time cT1;
Time cT2(9);
Time cT3(9, 25);
Time cT4(9, 25, 30);
Time cT5(45, 90, 72);
```

Homework

- For next time, you are to implement the Time class that we have used in this lecture
- This class should have four constructors
  - With no arguments
  - With 1 argument
  - With 2 arguments
  - With 3 arguments
- You should create three files for your project
  - Time.h
  - Time.cpp
  - Main.cpp