CSCI-1200 Data Structures — Fall 2021
Lab 2 — C++ Classes

In this lab, we will implement a simple C++ class named `Time`. It represents all possible times in a 24-hour period, including hours, minutes and seconds. An immediate representation issue is how to handle morning (am) and afternoon (pm) times. We could have a separate `bool` indicating whether the time is am or pm. It is easier, however, to represent the hours in *military time*. This means that the hours of the day are numbered from 0 to 23, with 13 being 1 pm, 14 being 2 pm, etc.

Your notes from Lecture 2 with the example `Date` class will be helpful in completing this lab.

Checkpoint 1

estimate: 30-40 minutes

In the first checkpoint you will get started by implementing the initial class design, several member functions, and a simple main program to test your class.

The instructions below describe how to build your executable from the command line using g++ (or clang++) using WSL or UNIX terminal. Even if you plan to use Visual Studio or another IDE for the bulk of your work this semester, you are required to also show that you can successfully build and run this lab using g++ (or clang++) from a terminal on your own machine.

http://www.cs.rpi.edu/academics/courses/fall21/csci1200/labs/02_classes/main.cpp

- Make a subfolder inside of your Data Structures labs directory for Lab 2. We provide basic testing code in `main.cpp`. You'll need to create two new empty code files named `time.h` and `time.cpp`. Note that in C++ the name of the header and implementation file are not required to exactly match the name of the class, but it is good coding style to do so – to help anyone reading your code. *In the lecture examples and provided code for this course, we often follow a convention that capitalizes the name of custom classes or types and lowercase all filenames – but this is not mandatory in C++.*

- Begin work on `time.h`. Within the file, declare a class called `Time`. Follow the form and syntax of the `Date` class from Lecture 2. Read the syntax carefully (such as the semi-colon at the end of the class declaration). Add private member variables for the `hour`, `minute` and `second`.

  In the public area of the class, declare three constructors: 1) the default constructor, which should initialize each of the member variables to zero; 2) a constructor having three arguments, which accepts initial values for the hour, minute and second as function call arguments; and 3) the copy constructor, which takes an existing `Time` object as a pass-by-const-reference argument. *Note: We will implement a copy constructor as an exercise in this lab, but normally we would not write this copy constructor ourselves, since the automatically-created copy constructor would do exactly what we want it to do. We'll discuss copy constructors more in future lectures.*

  Declare member accessor functions to access the values of the hour, the minute and the second (three different member functions). It will be important for Checkpoint 2 to make these accessors `const`. (Recall: a **const** member function can not change the member variables.)

  Don't write the body of any of the functions in the `time.h` file. Save all the implementation for the `time.cpp` file.

- Review the provided `main.cpp`. Note that we must `#include "time.h"` in addition to including `#include <iostream>`. (Note: We use angle brackets for standard library includes and double quotes for our custom header files in the working directory.) The main program creates multiple `Time` objects, using the two different constructors and uses the functions that access the values of hour, minute and second by printing the two times.

  Note: There is a common confusion when creating a new variable using the default constructor:

```
Time t1(5,30,59);  // calls the non-default constructor w/ 3 integer arguments
Time t2();         // COMPILE ERROR - a buggy attempt to call the default constuctor
Time t3;           // the *correct* way to call the default constructor
```

- Now implement all of the class constructors and member functions in the file `time.cpp`. Don't forget to add the line to `#include "time.h"`. Any file that uses or implements `Time` functionality must include the `Time` class header file.

- Put a different descriptive debugging print statement in the body of each of the 3 `Time` constructors so you can confirm and monitor when and *which constructor* is being called.

- Now, compile your program and remove errors. Here's where the difference between compiling and linking matters.

  When **compiling using g++** (or clang++) on the command line, the two separate command lines:

  ```
  g++ -c main.cpp -Wall -Wextra
  g++ -c time.cpp -Wall -Wextra
  ```

  compile the source code to create two object code files called `main.o` and `time.o` separately. The `-c` means "compile only". Compiler errors will appear at this point. If there are errors in `main.cpp` (or `time.cpp`), then the files `main.o` (or `time.o`) will not be created. Use the `ls` command to check.

  Important Note: We only compile `.cpp` *files.* We do not directly compile header files (e.g., we do not directly compile `time.h`). Header files are compiled only indirectly when they are included in a `.cpp` file.

  Once you have driven out all of the compiler errors, you can "link" the program using the command:

  ```
  g++ main.o time.o -o time_test.exe
  ```

  to create the executable called `time_test.exe`. If you have not implemented all of the necessary member functions in the `Time` class, then you would see "linking" errors at this point. You can combine all three command lines (compiling each of the two `.cpp` files to two object files and linking all object files) with this command:

  ```
  g++ main.cpp time.cpp -o time_test.exe -Wall -Wextra
  ```

  Which is more similar to what we did in Lab 1. Equivalently, if those are the only two `.cpp` files in the current directory, you can compile and link using the command line wildcard:

  ```
  g++ *.cpp -o time_test.exe -Wall -Wextra
  ```

  Note that this will not create the intermediate .o files and will only proceed to the linking step if the two files compile cleanly.

To complete this checkpoint: Show compilation of the program **using g++ (or clang++) within the WSL or UNIX terminal**, with all compiler errors removed and demonstrate correct execution of your program. *Yes, please show us you can compile from the terminal with g++, even if you plan to primarily use Visual Studio or another IDE for the rest of the semester.*

Checkpoint 2

estimate: 30-40 minutes

Create and test a few more member functions. This will require modifications to all three of the files. You should uncomment the provided tests in main.cpp as you work, and *add your own tests.*

- `setHour`, `setMinute`, `setSecond`. Each should take a single integer argument and change the appropriate member variable. For now, do not worry about illegal values of these variables (such as setting the hour to 25 or the minute to -15). Assume whoever calls the functions does the right thing. In general, this is a bad assumption, but we will not worry about it here.

- `PrintAmPm` prints time in terms of am or pm, so that 13:24:39 would be output as 1:24:39 pm. This member function should have no arguments. Note that this requires some care so that 5 minutes and 4 seconds after 2 in the afternoon is output as 2:05:04 pm. The output should be to `std::cout`.

- Finally, let's create a vector of times, sort it, and output the final order. You'll need to create a *non-member* function called `IsEarlierThan` which has the prototype:

  ```
  bool IsEarlierThan(const Time& t1, const Time& t2);
  ```

  It is important that the two time objects are passed by constant reference. Some older compilers may complain if you try to put non-const references or non-const pass-by-value parameters on this function. Even if your compiler does not complain, you should always write your sort comparators with const reference parameters. Why? 1) The two objects we are comparing might be "big" (our guideline in this course is things bigger than 8 bytes), and you don't want to waste memory & time copying them. And 2) the comparator shouldn't modify the objects that it's trying to sort while comparing them!

  The prototype of `IsEarlierThan` should be in `time.h` (in the file, but outside of the class declaration) and the implementation should be in `time.cpp`. It should return true if `t1` is earlier in the day than `t2`. The tough part, from the logic viewpoint, is being able to compare two times that have the same hour or even the same hour and the same minute. Test your function `IsEarlierThan`. What should you do if the two objects are equal? STL sort routines should return false. The sort comparison function is asking "Should I swap the order and place argument 1 in front of argument 2?" To avoid an infinite loop of swapping when the vector contains duplicate values, you should say "no" if the 2 arguments are equal.

  If your `IsEarlierThan` function is correct, sorting becomes very easy. You just need to pass the function to the sorting routine (make sure to `#include <algorithm>`). Be sure to study the output and convince yourself things are debugged before asking a TA/mentor for checkoff.

  ```
  sort(times.begin(), times.end(), IsEarlierThan);
  ```

- **Importance of `const` and reference:** After you have debugged and tested this checkpoint, experiment with `const` and pass-by-reference on the argument types for the function `IsEarlierThan`. Change them from `const` pass-by-reference to pass-by-reference w/o the `const`. Use the `-Wall -Wextra` compiler flags to enable all warnings. You *may* see compiler errors/warnings with some older OS/compilers.

  Next try removing the `const` on the `getHour()`, `getMinute()` and `getSecond()` accessor member functions. Make sure you remove the `const` in both the header and implementation files. What happens if the two arguments of `isEarlierThan` are of type `const` reference? Why is this a problem?

  Also, try `IsEarlierThan` with pass-by-value parameters. Do you see a difference in *which constructors* are called and how often? What is the source of the additional constructor calls?

  Switch `IsEarlierThan` back to const pass-by-reference parameters and add the `const`s back to the accessor member functions before asking for a checkoff.

To complete this checkpoint: Show a TA your tested and debugged extensions. Be prepared to discuss your implementation and your experiments with `const` and pass-by-reference.

Checkpoint 3 will be available at the start of Wednesday's lab.