Computer Science II — CSci 1200
Lab 3, September 14-15, 2004

Overview

This lab explores defining a relatively simple C++ class. Having the notes from Lecture 4 handy, especially the example of the `Date` class, will make this lab **much** easier. Still, there's a lot to do in this lab, so please work efficiently.

The class you will implement is called `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.

As a reminder, you can now insert the line

```
using namespace std;
```

at the start of your program (after the `#includes`), so that you don't need to type

```
using std::cin;
using std::vector;
```

etc.

Checkpoints

1. In the first checkpoint you will get started by implementing the initial class design, several member functions, and a "driver" main program.

   (a) Create a new project. Eventually you will create three files within the project. These files will be called `Time.h`, `Time.cpp` and `main.cpp`.

   (b) Begin work on `Time.h`. Create this file by selecting the "Add New Item..." option in the Project Menu (or use Ctrl+Shift+A). Select which type of new file you want to add (in this case a Header file) and give it the name `Time`. You do not need to include the extension (in this case .h) in the name because it will be added for you. Adding the extension though will have the same effect.

(Aside: when there is more than one project in the Solution you need to make sure that the correct project is highlighted in the Solution Explorer before carrying out the above.) Now click OK. Within the file, declare a class called `Time`. Follow the form and syntax of the `Date` class which was distributed during lecture. 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 two constructors: one, the default constructor, should initialize each of the member variables to 0; the other, having three arguments, accepts initial values for the hour, minute and second as function call arguments. Declare member functions to access the values of the hour, the minute and the second (three different member functions). It will be crucial for checkpoint 3 to make these `const`. (Recall: a **const** member function can not change the member variables.)

(c) Switch to working on `main.cpp`. Create a new file within your project, as above. Be sure to add code to `#include Time.h` in addition to including `iostream`. (Warning: the syntax is different — see the `Class` example.) Have the main program create two `Time` objects, one using each constructor. Show use of the functions that access the values of hour, minute and second by printing the two times.

(d) Switch to working on `Time.cpp`. Create a new file within your project as above. Don't forget to add the line to `#include Time.h`. Implement the constructors and member functions.

(e) Now, compile your program and remove errors. Here's where the difference between compiling and linking matters. You can compile each of the two .cpp files individually (the .h file isn't compiled separately) by typing Ctrl-F7 when the pane containing that file is active. This will allow you to see and remove compiler errors for each file individually. (Note that errors caused by the code in the .h file, `Time.h` will appear when compiling either .cpp file.) You can also compile and link multiple files at once using "`Build->Build {project name}`" menu. You will have to do this even after compiling each .cpp file individually because the Ctrl-F7 command does not link to create an executable program.

(f) To complete this checkpoint, remove all compiler errors and execute your program.

2. Create and test a few more member functions. This will require modifications to all three of your files.

   - `setHour`, `setMinute`, `setSecond`. Each should take a single integer argument and change the appropriate member variable. For now, don't 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. To do this, simply note that if the minute is less than 10 you will need to add the character '0' to the output. The same is true for seconds. The output should be to `cout`.

3. The last checkpoint involves the creation of a vector of times and then sorting it. First, create a *non-member* function called `IsEarlierThan` which has the prototype

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

   (It is very important that the two time objects are passed by constant reference.) The prototype should be in `Time.h` 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`.

   Now, create a vector of times in the main program. Just create a few time values and push them onto the back of the vector. (The following discussion assumes that the vector is called `times`, but you can use any other appropriate name.) Now, if your `IsEarlierThan` function is correct, sorting becomes very easy. You just need to pass the function to the sorting routine:

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

   That's all there is to it ! (Don't forget to `#include` the `algorithm` header file.) Complete this checkpoint and the whole lab by outputting

the times after sorting using a `for` loop and the `PrintAmPm` function you wrote in Checkpoint 2.

Final Note: After you have completed the lab, go back and change the argument types for the function `IsEarlierThan`. Make the parameters passed by reference and take a look at the compiler errors that result. Do the same thing with parameters that are passed by value. The problem is that the compiler expects the parameters of the comparison function (the 3rd argument to the `sort` function) to be in a certain form and complains that it can't find the function when the parameters aren't in this form.