ECE 71 – Introduction to Computational Programming

Professor Kriehn – Fall 2017

Code Due By: Midnight on Tuesday, October 03, 2017
Write-up Due By: Class on Wed/Thu, October 05/06, 2017

HOMEWORK #13 – Time Conversion

Write a program that converts from 24-hour time to 12-hour time. For example, 14:25 is 2:25 PM.

Specifications:
The input should read in integer values for the hours, read in the colon character and discard it, and read in the integer values for the minutes. Three user defined functions should be used with the following function prototypes:

```
void getInput(int &hrs, int &min);
void convertTime(int &hrs, string &ampm);
void displayOutput(int hrs, int min, string ampm);
```

The function **`getInput()`** should be used to store the hours and minutes input from the keyboard using reference parameters.

The function **`convertTime()`** should be used to convert the hours from 24-hour format to 12-hour format and determine the AM/PM information using reference parameters.

The function **`displayOutput()`** should print out the results of the time conversion in 12-hour format using call by value parameters.

NOTE: The `string` data type is defined in the `std` namespace. Therefore, "**`using namespace std;`** "must be declared globally before your function prototypes since the `displayOutput()` function uses the `string` data type. Otherwise, use **`std::string`** to refer to all string data types.

The **`setw()`** function will be needed to ensure that two spaces are always used for the minutes. To force a leading '`0`' to be printed in the case of "`08`" minutes, the **`setfill()`** function should be used as well. Any blank spaces that are present due to the `setw()` function will be filled by a character passed to the `setfill()` function as an argument. Since we require 2 spaces to print out the minutes, and we always want to print the leading '`0`' if necessary, `setw(2)` and `setfill('0')` can be used to format the output just before printing the minutes to the screen. Note that printing a leading '`0`' for the hours is not necessary. These functions are present in the I/O Manipulation library `<iomanip>`.

As an example, if you execute the program with the following underlined inputs, the output will be:

```
~> main.o
Enter a time in 24 hour format: 24:00
Time: 12:00 AM
~>
```

```
~> main.o
Enter a time in 24 hour format: 12:00
Time: 12:00 PM
~>

~> main.o
Enter a time in 24 hour format: 00:08
Time: 12:08 AM
~>

~> main.o
Enter a time in 24 hour format: 03:09
Time: 3:09 AM
~>

~> main.o
Enter a time in 24 hour format: 18:50
Time: 6:50 PM
~>
```

Develop your I/O diagram and pseudocode, debug your code, and submit to the Grader Program.

ECE 71 – Introduction to Computational Programming

Professor Kriehn – Fall 2017

HOMEWORK #14 – Day of the Week Calculator

Write a program that inputs a date (for example, "July 4, 2008") and outputs date in `mm/dd/yy` format as well as the day of the week that corresponds to that date. The problem can be solved by implementing the algorithm described by:

http://en.wikipedia.org/wiki/Calculating_the_day_of_the_week

Specifications:
The solution will require several user-defined functions:

```
void getInput(int &month, int &day, int &year);
bool isLeapYear(int year);
int getCenturyValue(int year);
int getYearValue(int year);
int getMonthValue(int month, int year);
string dayOfWeek(int month, int day, int year);
```

The **`getInput()`** function should read in the month, day, and year from the keyboard. Assume, however, that the month is entered in via text in either full or abbreviated form. For instance, February could be input as either "**`Feb`**" or "**`February`**". Therefore, the function must also convert the string for the month into an appropriate integer value since the call by reference value to store the month uses an integer (not a string) reference. Finally, the function should also be able to read in the comma after the day and discard it.

The **`isLeapYear()`** function returns a boolean value that is `true` if the year is a leap year and `false` if it is not a leap year. The pseudocode to determine a leap year is given by:

```
        leapYear = (year divisible by 400) OR
(year divisible by 4 AND year not divisible by 100)
```

The **`getCenturyValue()`** function should take the first two digits of the year variable (that is, the century), divide by 4, and save the remainder. The remainder should then be subtracted from 3 and the subsequent value multiplied by 2. For example, the year 2008 becomes: (20/4) = 5 with a remainder of 0. 3 – 0 = 3. Return 3 * 2 = 6.

The **`getYearValue()`** function computes a value based on the years since the beginning of the century. First, extract the last two digits of the year. For example, 08 is extracted for 2008. Save this value. Next, divide the value from the previous step by 4 and discard the remainder. Add the two results together and return the result. For example, from 2008 we extract 08. Then (8/4) = 2 with a remainder of 0. Return 2 + 8 = 10.

The **`getMonthValue()`** function returns a value based on the table below, and will require invoking the **`isLeapYear()`** function:

Month	Return Value
January	0 (6 if year is a leap year)
Feburary	3 (2 if year is a leap year)
March	3
April	6
May	1
June	4
July	6
August	2
September	5
October	0
November	3
December	5

Finally, the **`dayOfWeek()`** function should be used to compute the day of the week. The weekday is computed by the sum of the date's day plus the values returned by the `getMonthValue()`, `getYearValue()`, and `getCenturyValue()` functions. Divide the sum by 7 and compute the remainder. A remainder of 0 corresponds to Sunday, 1 corresponds to Monday, etc., up to 6, which corresponds to Saturday. For example, the date July 4, 2008 should be computed as (day of month) + (getMonthValue) + (getYearValue) + (getCenturyValue) = 4 + 6 + 10 + 6 = 26. 26/7 = 3 with a remainder of 5. The 5th day of the week corresponds to Friday. Return this value as a string.

Again, note the requirement of declaring "**`using namespace std;`**" before your function prototypes, otherwise **`std::string`** will be required to access the string data types. Likewise, the **`setw()`** and **`setfill()`** functions will have to be used as well when printing the date in **`mm/dd/yy`** format.

As an example, if you execute the program with the following underlined inputs, the output will be:

```
~> main.o
Enter a date: September 28, 2017
09/28/17 is a Thursday.
~>

~> main.o
Enter a date: Jan 1, 2000
01/01/00 is a Saturday.
~>

~> main.o
Enter a date: July 4, 2008
07/04/08 is a Friday.
~>
```

Develop your I/O diagram and pseudocode, debug your code, and submit to the Grader Program.