Assignment 2: Temperature Class

Assigned: September 23, 2016
Due: October 03, 2016, 11:59:59pm

Purpose

The purpose of this project is to provide you more practice with implementing classes. Here you will gain familiarity with stream objects, error checking, and converting between different types of logical objects.

Project Task

You are to write a class called `Temperature` using filenames `temperature.h` and `temperature.cpp`. This class will handle the storing and manipulating of temperatures.

An object of type Temperature should represent a temperature in terms of degrees and scale. Degrees should allow decimal precision (type `double`). The possible scales are Celsius, Fahrenheit, and Kelvin. You may store the scale however you like inside the object, but keyboard input and function parameters will come in as type char. The passed in values for scale will be 'c' or 'C' for Celsius, 'f' or 'F' for Fahrenheit, and 'k' or 'K' for Kelvin.

Your object must always represent a valid temperature (e.g. 0 Kelvin represents the lowest possible temperature in existence). Your object should also store a format setting to be used for the display of temperatures on the screen. There will be more than one possible format.

For conversions between scales, use the following formulas:

- Celsius = (Fahrenheit – 32) x (5/9)
- Celsius = Kelvin – 273.15

The class' public interface should work exactly as specified regardless of what program might be using Temperature objects. You can use whatever private functions you need, but the public interface needs to include the prototypes listed below.

Make sure your implementation works with `g++` on `linprog.cs.fsu.edu` before submitting. You should also add your own tests to the provided driver to fully test the functionality of your implementation.

Documentation

In addition to the project implementation, you will also need to provide a `README` file. Here you will need to document at least 5 errors you encountered and how you solved them. The errors can be some combination of compile errors, linker errors, runtime, and runtime errors. However, you can not use the same errors you used for Assignment 1. The format needs to be a numbered list where each item describes the error type, the error description, and the steps taken to fix the error. For the description, you can just describe the g++ message for compile/linker errors or the crash / wrong output for runtime errors.

You can include documentation for places where your implementation is unfinished. If you document buggy or unfinished code and describe what the problem is and and general approach to fixing it, then point deductions will be less severe. This shows me that you are aware of problems in your code and that you at least have an idea of how to fix them. This is completely optional, however, it can only help you as I'm not going to change my test code based on your documentation. That is, if you report an error I would have otherwise missed, you won't lose any points for it.

Part 1: Constructor(s)

The Temperature class should have a constructor that allows the user to specify the values for the degrees and scale, using types `double` and `char`, respectively. If any of the values would result in an invalid temperature, the constructor should throw out the erroneous information and initialize the object to represent 0 Celsius, by default. This default value should also be used if the Temperature object is declared without the specified parameter values.

Examples:

```
• Temperature t1;            // initializes to 0 Celsius
• Temperature t2(23.5, 'F'); // initializes to 23.5 Fahrenheit
• Temperature t3(12.6, 'Z'); // invalid scale, inits to 0 Celsius
• Temperature t4(-300, 'c'); // below 0 Kelvin, inits to 0 Celsius
• Temperature t5(15, 'k');   // initializes to 15 Kelvin
```

Part 2: Accessor Methods

There needs to be two accessor methods:

> `double GetDegrees()` returns the degrees to the caller.
> `char GetScale()` returns the scale to the caller.

Part 3: Mutator Methods

> `bool Set(double deg, char s)`

This function should set the temperature to the specified values (degrees and scale respectively). If the resulting temperature is invalid, the operation should abort/fail and not change the object. This function returns `true` for success and `false` for failure.

> `bool SetFormat(char f)`

This function allows the caller to change the format setting. The setting should be adjusted inside the object based on the character code passed in. This means the future uses of the Show function will display in the given format until the format is changed. The valid setting codes that can be passed in are:

- 'D' = Default format
- 'P' = Precision-1 format
- 'L' = Long format

Part 4: Conversion Routines

> `bool Convert(char sc)`

This function should convert the current temperatures (i.e. the calling object) so that it is now represented in the new scale given in the parameter. You'll need to use the temperature conversion factors for this. If the scale provided is invalid, abort the operation and do not change the current temperature. Otherwise, convert the temperature to the new scale so that it is equivlent to the previous representation. Return true for success and false for failure. Examples:

```
• Temperature t1(68.9, 'F');  // 68.9 Fahrenheit
• t1.Convert('T');            // invalid. no change, returns false
• t1.Convert('c');            // t1 is now 20.5 Celsius
• t2.Convert('K');            // t1 is now 293.65
```

Part 5: Compare Routines

> `int Compare(const Temperature& d)`

This function should compare two Temperature objects (the calling object and the parameter). It returns one of three values and does not change either object:

- -1 if the calling object is the lower temperature
- 0 if the objects represent the same temperature
- 1 if the parameter object is the lower temperature

Examples:

```
• Temperature t1(0, 'c');     // 0 Celsius
• Temperature t2(31.5, 'F');  // 31.5 Fahrenheit
• t1.Compare(t2);             // returns  1 since t1 > t2
• t2.Compare(t1);             // returns -1 since t2 < t1
```

Part 6: Standard Input

> `void Input()`

This function should prompt the user to enter a temperature, and then allow the user to input a temperature from the keyboard. User input is expected to be in the format *degrees scale*, where *degrees* allows values with decimal precision, and *scale* is entered as a character. Whenever the user attempts to enter an invalid temperature, the Input function should display an appropriate error message (like "Invalid temperature. Try again:") and make the user reenter the whole temperature. A few examples of both good and bad inputs:

- Legal:
  - `43.6 K`
  - `53.5 C`
  - `100 F`
  - `-273.15 c`
- Illegal:
  - `12.3 q`
  - `-5 K`
  - `-278 C`
  - `-500 F`

You may assume that the user entry will always be of the form `<D S>` where `D` is a numeric value, `S` is

a character, and there is a white space between them.

Part 7: Standard Output

```
> void Show()
```

This function should simply output the temperature to the screen. Since there there will be more than one possible format for this output, you will need to store a format setting. The Show function should use the format setting to determine the output. When a Temperature object is created, the format setting should start out at the default setting. The possible formats are shown below:

Name	Format	Example	Explanation
Default	`D S`	`50.4316 C`	This will look mostly like the input from the Input function. Print the degrees and scale as a double and char, with default precision on the degrees, and the scale as an uppercase letter
Precision-1	`D.d S`	`50.4 C`	Degrees printed to 1 place after the decimal, fixed format, and scale printed as an uppercase letter. This output will need to make sure to put the output stream BACK to its original format settings when you are done (so that the output from the main program isn't now set to 1 decimal place for the caller)
Long	`D scale`	`50.1436 Celsius`	This display format should show the degrees in default precision, and the scale as the full word: “Celsius”, “Fahrenheit”, or “Kelvin”

General Requirements

- All member data of the Temperature class must be private.
- The const qualifier should be used on any member functions where it is appropriate.
- The only libraries that may be used in these class files are <iostream>, <iomanip>, <cctype>, and <string>. Note that you may use the string class to store the words like “Celsius”
- Do not use language or library features that are C++11-only
- You only need to do error-checking that is specified in the descriptions above. If something is not specified (e.g. user entering a letter where a number is expected), you may assume that part of the input will be appropriate. You must always maintain a valid temperature object, but for keyboard input, you may assume that if you ask for a double, you will get a number for example.
- User input and/or screen output should only be done where described (i.e. do not add in extraneous input/output)
- No global variables other than constants

Extra Credit

Add the following member function to your class:

- `> void Increment(int deg, char sc)`

This function should move the temperature forward by the number of degrees IN the scale given. However, the resulting object should remain in the original scale representation. Examples:

```
• Temperature t1(20.5, 'C'); // 20.5 Celsius
• t1.Increment(3, 'C');      // add 3 degrees Celsius to the
                             // current temperature. t1 is
                             // sill in Celsius.
• t1.Increment(6 'F');       // add 6 degrees Fahrenheit to the
                             // current temperature. T1 is
                             // still represented in Celsius.
```

Testing

You will need to test your class, which means you will need to write one or more main programs that will call upon the functionality (i.e. public member functions) of the class and exercise all of the different cases for each function. You do not need to turn any test programs in, but you should write them to verify your class' features. There will be a sample driver on the website to help you get started.

Submitting

- Remove all binary files, executables, and driver code
- Use the tar utility to archive your `temperature.h`, `temperature.cpp`, and `README` files
- Name the tar file according to the scheme: a2_<*last_name*>_<*first_name*>.tar
- Submit the tar file to the appropriate blackboard assignment link