Temperature as a Class Assignment

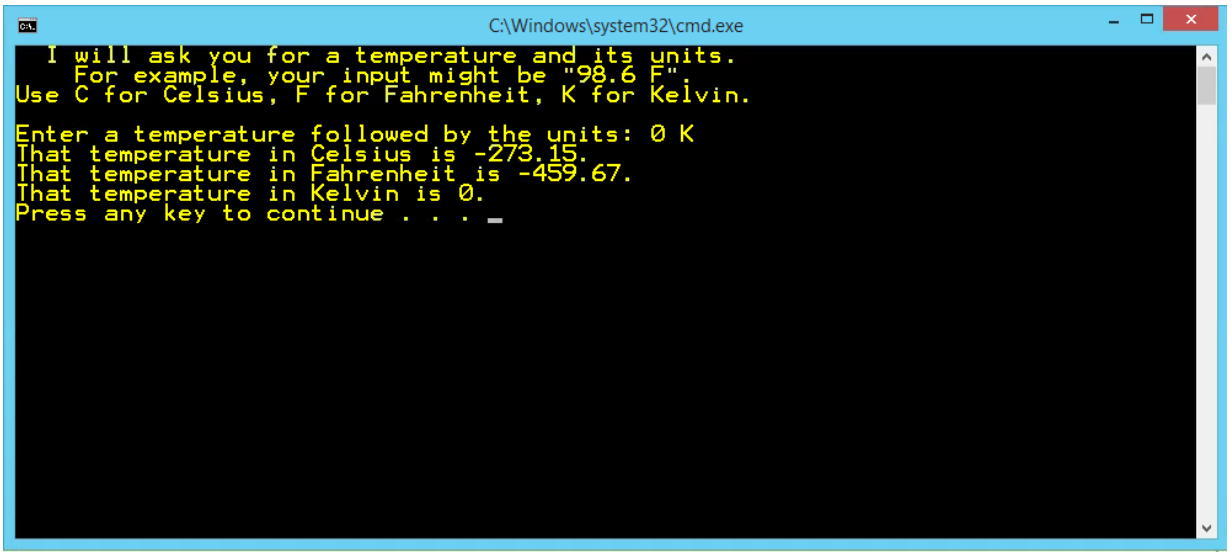
The goal of this assignment is to better acquaint you with using an object in the design of a solution.

Your task is to create an application that asks the user to enter a temperature and the type of units of that temperature. It then reports the temperature in 3 different units.

The application should use a class called Temperature that stores a single temperature, but provides functions to set and get that temperature using any of the three units: Celsius, Fahrenheit, or Kelvin. In the code for `getTemperatureFromUser,` given below, you can see the three set functions being used. Your job is to implement the Temperature class, and then write a `main` function that uses the three get functions to complete this task.

To reduce the work, and demonstrate some coding tricks, I have included a separate function (not part of the Temperature class), called `getTemperatureFromUser(),` that takes a pointer to a Temperature object as its argument. Put this function in the same file where you put main(). The `getTemperatureFromUser` function prompts the user for the temperature and the type of units and reads them in. It also detects if the user gives bad input and provides the user with an opportunity to try again. When the `getTemperatureFromUser` function returns, the temperature object which was passed as a pointer argument should contain a temperature.

In your `main` function, call the `getTemperatureFromUser` function, and then, using the three `get` functions of the temperature object, report the temperature in each of the three units. If written correctly, the output should look exactly as shown in the console window shown below.



Please read the following scenario to make sure you understand what is meant by an "abstract" temperature.

Imagine this scenario. I give my Temperature object to John to take the temperature. He is comfortable using Fahrenheit and his thermometer reads in Fahrenheit. So he sets the temperature in Fahrenheit. Then I give the same Temperature object to Helga. Helga is accustomed to Celsius, and her lab equipment uses Celsius. So she uses the getCelsius to get the

temperature that John just measured. In addition, she wants to compute something that involves the temperature, but the formula that she has is written to use Kelvin. So she gets the same temperature for her formula, but this time she gets it in Kelvin. Now supposing the formula tells her that the temperature they want is a little different, but they only have it in Kelvin. So she sets that temperature in Kelvin and sends it back to John. John looks at what that is in Fahrenheit so he can work on getting the right temperature.

The point is that the object is THE TEMPERATURE and having a Temperature object makes it possible for anybody to see THE TEMPERATURE without having to know anything about who set it or how they set it. We call that an ABSTRACTION. It gives us what we need, while hiding unnecessary details from those who use it.

Yes, there is conversion involved. But that is just part of the job of hiding the detail of which units are used inside the class for storage. In other words, there is no "convert" function. Conversion is just the hidden detail that we may or may not do, in a get or set operation, in order to make it work as an abstraction.

The getTemperatureFromUser function given below has two new features which we haven't used before (do-while and switch-case). They are generally used only in special cases. It just so happens that those special cases occur here.

```
#include <iostream>
#include <string>
#include "Temperature.h"
using std::cout;
using std::endl;
using std::cin;
using std::string;

// Function:  getTemperatureFromUser
// Purpose:   Prompt user for a temperature and put it in the argument object
// Parameter: temperatureObject is a pointer to a Temperature object
// Returns:   nothing
//
void getTemperatureFromUser(Temperature* temperatureObject) {
    double amount = 0.0;     // this is the user's input for the temperature
    string units = " ";      // this is a string, but we only use the first letter
    bool goodInput = false;  // this is a flag to stay in the "try-again" loop

    // Space the text over so it looks kind of centered
    cout << "  I will ask you for a temperature and its units.\n";
    cout << "    For example, your input might be \"98.6 F\".\n";
    // Use new line characters instead of endl since the last cout has endl.
    cout << "Use C for Celsius, F for Fahrenheit, K for Kelvin.\n" << endl;
    do {
        cout << "Enter a temperature followed by the units: ";
        cin >> amount >> units;
        if (cin.fail()) {
            // clear all artifacts of the failed input
            cin.clear();
```

```
            cin.ignore(1000, '\n');
            cout << "I'm sorry. Your input failed. Please try again." << endl;
            // "continue" means skip rest of code block for the containing loop
            goodInput = false;
        } else
            goodInput = true;

        if (goodInput) {
            // Use a switch statement to select among one letter responses
            // C++ string operator [] returns the letter in that position
            switch (units[0]) {
            // The choices are character constants (not strings), hence '
            case 'C':
            case 'c':
                temperatureObject->setTemperatureAsCelsius(amount);
                // "case" represents a starting point for execution
                // If you don't want to continue further, use "break"
                break;
            case 'F':
            case 'f':
                temperatureObject->setTemperatureAsFahrenheit(amount);
                break;
            case 'K':
            case 'k':
                temperatureObject->setTemperatureAsKelvin(amount);
                break;
            default:
                cout << "The type was not recognizable as C, F, or K." << endl;
                goodInput = false;
            }
        }
        // If we make it to here without changing notDone, we are done.
    } while (!goodInput);
}
```

For starters, the outer loop is a `do/while` loop that says, do this loop at least the first time, and if the while condition is still met, loop again. The while condition is that the user has not completed entering a temperature, which we represent with a Boolean variable called `goodInput`. So the condition for looping is `while(!goodInput)`.

The `cin.fail()` indicates that cin did not get a number followed by a string. Cin leaves input in the "input buffer" when it doesn't know what to do. So to try again, you have to both clear the error flags and empty the input buffer of any garbage that was left over from the failed attempt. That's the `clear` and `ignore`.

Once we know we have a failure, we want to skip the rest of the work inside the loop by jumping to the end. In C++, and in Java, the word "`continue`" does just that.

The other item that we haven't talked about is the `switch/case` control structure. Switch case is sometimes used in place of a series `if/else if/else` statements. But it does not work the same way. Switch/case works as follows. It compares the value in the parens of the `switch` statement with each value in a `case` statement. If it finds a match, it resumes execution immediately after that case statement. After that it does not check any other case statements.

Since all it does is resume execution, it would in theory execute every executable statement from that point on. If you want it to only do what is in that case, and not the next case, you have to include a break statement at the point where you want it to stop executing code within the switch. If you don't include the break, it will just continue and do the code under the next case as well. (That's why we can put several case statements next to each other and all have the same affect.) A break statement is like a continue statement, except that when used in a loop, it exits the loop entirely. Here is exits the switch. The continue tests the loop condition and if true continues looping.

In `switch/case`, the values after the `case` must be primitive constants. That means they must be either a literal integer or a literal character. They cannot be strings or anything else that involves computation. In this case, we use the first character of the user's response for the case statements. A literal character is indicated with single quotes instead of double quotes, and represents an ASCII value that occupies 1 or 2 bytes. If you were to declare a variable of that type, the type name is `char`.

In the switch statement, the `default` case is the starting point for anything that hasn't matched one of the other cases.

Because of the strange way in which it works, switch/case is seldom used. But for some situations, like here where we are matching on one of a set of characters, it works well.

Copy this function onto the end of your main file (the one where you have main()). Then add a prototype to the beginning of the file so you can call it from main. In your main, declare an object of your `Temperature` class, call `getTemperatureFromUser` with a pointer to your object, and then write three lines with cout to output the lines shown in the console output above.

For the `Temperature` class, you have to decide which type of units to use for the private storage, and then provide getters and setters with conversions to and from that value for the other choices. In this case, because the conversions are well known, you don't need to use constants. You can find the conversion formulas by Googling them (e.g. "formula for fahrenheit to celsius conversion").