COMP161 - Lab 6 & Homework 4

Spring 2014

For this lab you'll be gearing up to put together a complete program by working on some key procedures for the program MODEL and VIEW.

Program Overview

We'll be working on the complete design and implementation for a program to convert Fahrenheit temperatures to other temperature scales[1]. For today's lab you'll look at some key state and I/O procedures necessary for the user interface. For homework, you'll write some other procedures that use the same kind of design techniques we've been using for weeks now[2]. In the week to come we'll use all this code to build a Model-View-Controller style program.

To give this week's work some context, let's look at our end goal:

> The *FTo* program will allow the user to convert a Fahrenheit temperature to a temperature on the Celsius, Kelvin, or Rankine scale. Conversions can be carried out using the following formulas:
>
> $$C = \frac{5}{9}(F - 32)$$
>
> $$K = \frac{5}{9}(F + 459.67)$$
>
> $$R = F + 459.67$$
>
> The program provides two user interfaces: command line and interactive. The command line interfaces allows for for a single temperature conversion. In this mode, the user feeds the Fahrenheit temperature and target scale to the program as command line arguments and the resultant conversion is reported. In the interactive mode users begin the program with no command line arguments. They are then guided by a text menu and can do multiple temperatures conversions with one or more Fahrenheit temperatures.

Both interfaces share a lot of functionality and therefore a lot of library code. We'll begin this week with the implementation of this library code and then next week explore how that code comes together to make a complete program.

[1] `http://en.wikipedia.org/wiki/Fahrenheit`

[2] No variable mutation. No I/O

Lab: Interface Procedures

The command line interface requires a mutator method for setting the temperature to the value read from the standard input[3]. Both interfaces require procedures for communicating results of a temperature conversion to the user. The interactive interface requires several I/O procedures:

[3] after it has been validated of course

1. Write the UI Menu text to standard out
2. Read menu option input from the standard input
3. Read a Fahrenheit temperature from the standard input

For lab, you need to write and test a library called *fToUI*[4]. In that library we'll create five procedures:

[4] For FTo User Interface

1. *setFTemp* Used to set a state variable containing the Fahrenheit temperature.
2. *ppConversion* Used to "pretty print" the results of the conversion to standard output
3. *writeMenu* Used to write the interactive UI menu to the standard output
4. *getMenuIn* Used to read a single menu option choice from the standard input
5. *getFTemp* Used to read a single Fahrenheit temperature from the user

Details for each of the methods is given below. *Be sure to follow the design processes laid out in lecture notes 07 for each procedure.* **At the end of lab, submit your source code for the UI library as assignment *lab6* using handin.** If you do not complete this lab in the alloted two hours, you should try to do so by next week. You are not required to, but it is highly recommended that you do.

setFTemp

When the command line interface is used, we'll need to do some validation on the potential number passed from the command line. After we're sure we have a proper Fahrenheit temperature[5], we can assign it to a state variable which maintains the current state of our model[6]. Thus, we need a simple *mutator* to change our state variable from an initial value to a user specified value. This is a very basic MUTATOR method and has been covered in class.

[5] see homework

[6] The Fahrenheit temperature to be converted

Recall that mutators require pass-by-reference variables and that when multiple arguments are passed to a mutator, we prefer by-value arguments to precede by-reference arguments. So, this procedure two take the new temperature by value and then the state variable by reference. We're designing this procedure with the precondition that the new temperature value is a valid Fahrenheit temperature. The postcondition is the mutator standard: the variable's state changed[7].

[7] be more specific in your documentation

ppConversion

We could write three pretty printers, one for each of the conversions[8], but they'd all probably look way to similar. Similar code is always an opportunity for ABSTRACTION. In this case, we can design a procedure that meets all of our pretty printing needs. Our desired output is to first print the Fahrenheit temp with exactly two decimal places, then an F, then an =, then the converted temperature (again with two decimal places), then the conversion unit. Like this:

[8] F→C,F→K, and F→R

```
32.00 F = 0.00 C
32.00 F = 273.15 K
32.00 F = 491.67 R
```

Notice that if we treat both the conversion value and the unit as variables then the pattern is $f\ F = v\ u$ where f is the Fahrenheit value, v is the converted value, and u is the conversion unit letter. This leads to the following signature[9]

[9] less the namespace specifier

```
void ppConversion(double fTemp, double convertValue, char convertUnit)
```

Finish the design and implementation of this procedure under the PRECONDITION that all the values passed to the procedure are logically sound. This means we can assume only the characters 'C', 'R', or 'K' and that the Fahrenheit temperature and the converted temperature are, in fact, equivalent. The CONTROL portion of our program will later ensure that our preconditions are enforced.

writeMenu

The *writeMenu* procedure is simply a procedure to dump some text, namely the menu text, on the screen. Thus, it takes no inputs and produces no outputs; it simply produces the same effect every time. That effect is to write exactly this to the standard output:

```
Select an option (press the corresponding number):
   (1) Enter a Fahrenheit temperature
   (2) Convert current Fahrenheit temperature to Celsius
   (3) Convert current Fahrenheit temperature to Kelvin
```

```
(4) Convert current Fahrenheit temperature to Rankine
(5) Quit the program
```

Be sure that your code for this[10] procedures is written in such a way that it will not wrap when printed[11]. Emacs has modes to enforce this for you.

[10] and all

[11] Not more than 80 characters per line is a good rule of thumb

getMenuIn

In interactive mode, our users will, when confronted with the menu, enter an integer in the interval $[1, 5]$[12]. We'll use a *very* simple input procedure to read their response to a variable. Remember, input procedures are just different forms of MUTATORS. So, they must take a variable by reference and then read a value from an input stream and write it to that variable rather than mutate the variable directly via the assignment operator. As a mutator-like procedure, *getMenuIn* has a clear postcondition. Do not check[13] the user input here, instead we'll save that for the program CONTROLler code that we'll write later. For the time being, we'll work under the assumption that the user will only type numerical[14] literals, no words or other non-number input.

[12] hopefully

[13] or VALIDATE

[14] floating point

getFTemp

Our final I/O procedure is another input procedure. This procedure gets a Fahrenheit temperature from the user. Unlike the menu input procedure, we'll include a prompt with this procedure. The prompt should look like this:

```
Enter a Fahrenheit Temperature:
```

To save space, we'll take the user input on the same line as the prompt[15]. So if the user types 32, the screen should look like:

[15] do not terminate the prompt with a newline

```
Enter a Fahrenheit Temperature: 32
```

Put a little space after the prompt. Because this procedure includes both Input and Output, we should document the expected output in our tests in addition to testing for the correct read effect.

Homework: Procedures for the Model and Control

Submit as assignment *hwk4* via *handin*. **Due by class time on Monday 2/24**

For homework, you need to complete the design and implementation of some other procedures, procedures for managing the Fahrenheit temperature state, UI actions, and for managing the UI and state interaction:

1. *isValidF* Is an accessor for the a Fahrenheit temperature variable that returns true if the value of our variable is in $[-459.67, \text{inf})$ and false otherwise.[16]
2. *isMenuOption* Is an accessor for a user menu choice input variable that returns true if a menu choice variable value is in $[1, 5]$.
3. Three accessors for the Fahrenheit temperature variable that are used for the temperature conversion.
   (a) *toC* which computes the equivalent in Celsius
   (b) *toK* which computes the equivalent in Kelvin
   (c) *toR* which computes the equivalent in Rankine

[16] -459.67 is Absolute Zero