CSC 402/502 Homework 1 Instructor: Jeff Ward
Covers: F#: Getting started; Values, operators, expressions and functions 30 pts

For this assignment create a file called Program.fs that contains the declaration **`module HW1`** at the top. All of your code for this assignment should be placed in the Program.fs file.

If you are using JetBrains Rider as your IDE then you can create the file (and runnable project) as follows: From the Rider start-up screen select "New Solution". (If you are already in Rider instead of the start-up screen, then go the menu and select File->New...). Select "Console Application" under .NET Core. Give the solution a name (e.g. "HW1" without the quotes). The dialog box will fill in the same name (e.g. "HW1") as the name of the project. Click "Create". Rider will create a file named Program.fs for you. At the top of the file add a line that states **`module HW1`** and save the change (Ctrl+s or File->Save All). Now if you choose Run->Run 'HW1' (or hit Shift+F10) Rider will run the file and you will see the output "Hello World from F#!". You are now ready to begin the homework problems.

Problem 1 (3 points) Section 1.2 (page 3) of the textbook gives a *circleArea* function whose type is *float -> float*. Enter the code for circleArea into your file (after the **`open System`** statement and before the **`[<EntryPoint>]`** declaration):

```
let circleArea r = System.Math.PI * r * r
```

Note that you do not need to put two semicolons (**;;**) at the end of this declaration. In the textbook the authors assume that you will type the declaration into the F# Read-Eval-Print-Loop (REPL). It is common practice (though not always necessary) to terminate with two semicolons an expression that is typed into the REPL: This signals to the REPL that the desired expression is complete. You should not need this terminator token in code that is contained in a source file. In this document I will generally follow the usual custom by terminating expressions at the REPL with *;;*. (I'm quite okay with you leaving out *;;* if you like.)

Next you should test that your function works correctly. Do this in two ways: (1) by calling it from the Read-Eval-Print-Loop (REPL) and (2) by calling it from **`main`**.

(1) Calling your function from the REPL: Click on the line where circleArea is defined. Then hit Ctrl+\ (i.e. hold down the Ctrl key and press the '\' key). This should open up the REPL (also known as F# Interactive), which consists of two small text areas, one above the other. At the > prompt in the bottom text area, evaluate your function on the value r = 3.0 by typing the following and hitting <Enter>:

```
> circleArea 3.0;;
```

You should see the following appear in the text area that is immediately above the prompt (note that here we *do* use the double semicolons):

```
circleArea 3.0;;
val it : float = 28.27433388
```

As you can see, the REPL displays the evaluated expression, followed on the next line by the type and value of the expression. The variable called **`it`** is bound to the value most recently computed by the REPL. You can use `it` in later expressions:

```
> it + 1.0;;
```

```
> it + 1.0;;
val it : float = 29.27433388
```

(2) Calling your function from **`main`**: Insert the following (single) line immediately before the last line in your main function:

```
printfn "The area of a circle with radius %1.1f is %1.8f" 3.0
(circleArea 3.0)
```

Now when you run your program (Shift+F10) you should get the following output:

```
Hello World from F#!
The area of a circle with radius 3.0 is 28.27433388
```

Problem 2 (3 points) Section 2.3 (page 25) shows how to write an **`isLowerCaseVowel`** function that takes a character and returns whether or not it is a lower case vowel. Define this function in your file. (Note: if your definition is split over two lines, as it is in the textbook, if you want to read it into the REPL by using Ctrl+\, then make sure that you select both lines by clicking and dragging over them.)

```
val isLowerCaseVowel : ch:char -> bool

> isLowerCaseVowel 'e';;
val it : bool = true

> isLowerCaseVowel 'f';;
val it : bool = false

> isLowerCaseVowel 'E';;
val it : bool = false
```

Problem 3 (6 points) Define a function **ithCharEquals** of type *string -> int -> char -> bool* where the value of *ithCharEquals str i ch* is true if and only if *ch* is the *i*-th character in *str*. (If *i* is out of range or the *i*-th character is not *ch*, then return false.) Note: You can use the *String.length* function shown in Section 2.3 to get the length of *str*, but strings have a *Length* data field that you can use instead. Regardless of which way you retrieve the length, see if you can define the function without using an if-then-else expression: It should be possible to define it more succintly by using the conjunction (*&&*) of three expressions. Also, note that F# might not be able to infer the type of *str* based on the operations that you use. You can explicitly tell it *str*'s type as follows:

```
let ithCharEquals (str:string) i ch = ...
```

Once you're loaded the function in the REPL you should be able to have the following interactions:

```
val ithCharEquals : str:string -> i:int -> ch:char -> bool

> ithCharEquals "Grace" 1 'r';;
val it : bool = true

> ithCharEquals "Grace" 5 'e';;
val it : bool = false
```

Problem 4 (6 points) Write a higher-order function **ithCharSatisfies** of type *string -> int -> (char ->bool) -> bool* where the value of *ithCharSatisfies str i f* is true if and only if *f* returns true on the *i*-th character in *str*. (If the *i*-th character does not satisfy *f*, or if *i* is out of range, then return false.) Using your isLowerCaseVowel function from Problem 2, you should be able to generate the following interactions in the REPL:

```
val ithCharSatisfies : str:string -> i:int -> f:(char -> bool) -
> bool
```

```
> ithCharSatisfies "Grace" 3 isLowerCaseVowel;;
val it : bool = false

> ithCharSatisfies "Grace" 4 isLowerCaseVowel;;
val it : bool = true

> ithCharSatisfies "Grace" 5 isLowerCaseVowel;;
val it : bool = false
```

Note: The = operator in F# is the binary infix operator for equality comparison:

```
> 3 = 4;;
val it : bool = false

> 3 = 3;;
val it : bool = true
```

You can use = as a prefix operator if you put it inside parentheses:

```
> (=);;
val it : ('a -> 'a -> bool) when 'a : equality

> (=) 3 4;;
val it : bool = false

> (=) 3 3;;
val it : bool = true
```

Due to "currying", you can give *(=)* one parameter and you will get back a function that receives one parameter:

```
> (=) 3;;
val it : (int -> bool) = <fun:it@3-12>

> ((=) 3) 4;;
val it : bool = false

> ((=) 3) 3;;
val it : bool = true
```

Consequently, you should be able to use your *ithCharSatisfies* function as follows

```
> ithCharSatisfies "Grace" 0 ((=) 'G');;
val it : bool = true

> ithCharSatisfies "Grace" 0 ((=) 'H');;
val it : bool = false
```

Problem 5 (6 points) Section 2.4 shows how to write *if-then-else* expressions. Such expressions can, of course, be nested. In your Program.fs file use nested if-then-else expressions and the < operator to write a function **min3** of type `int -> int -> int -> int` that returns the smallest of three values. Do not use any other operators. (For example, do not use F#'s `min` operator.)

Source code:

```
let min3 (a:int) b c =
    ...
```

REPL inteaction:

```
val min3 : a:int -> b:int -> c:int -> int

> min3 1 2 3;;
val it : int = 1

> min3 4 2 3;;
val it : int = 2

> min3 4 5 3;;
val it : int = 3

> min3 2 2 2;;
val it : int = 2
```

Once you get your min3 function working correctly, change the parameter `(a:int)` to simply `a`. You will get a more function with the more general type `'a -> 'a -> 'a -> 'a when 'a : comparison`. Now you will be able to call min3 on integers, but also on floats or strings, or any other type that is compatible with the < operator (more technically, that supports comparison).

```
val min3 : a:'a -> b:'a -> c:'a -> 'a when 'a : comparison

> min3 10 -2 8;;
val it : int = -2

> min3 10.0 -2.1 8.7;;
val it : float = -2.1

> min3 "Abe" "Betty" "Carl";;
val it : string = "Abe"
```

Problem 6 (6 points) Write a function called **min3WithMin** that performs the same task as **min3**. This time do use F#'s *min* operator in defining your function. Do not use if-then-else expressions or the < operator. The type of the function should be `'a -> 'a -> 'a -> 'a when a : comparison`. (F#'s *min* function is not covered in the textbook but you can easily guess what it does.)

```
val min3WithMin : a:'a -> b:'a -> c:'a -> 'a when 'a :
comparison
```

```
> min3WithMin 5 2 3;;
val it : int = 2
```

```
> min3WithMin "Melanie" "Zeke" "Angie";;
val it : string = "Angie"
```

What to turn in:

Submit on Canvas your Program.fs file. You are not required to include any comments in the file, but feel free to include them if you so prefer.