CSC 402/502 Homework 2 Instructor: Jeff Ward
Covers: F#: Tuples, records and tagged values 20 pts

For this assignment create a file called Program.fs that contains the declaration **`module HW2`** 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 you did for HW1: Hit File->New... . Select "Console Application" under .NET Core. Give the solution a name (e.g. "HW2" without the quotes). The dialog box will fill in the same name (e.g. "HW2") as the name of the project. Click "Create". Rider will create a file named Program.fs. At the top of the file add the declaration **`module HW2`**.

Problem 1 (Tuples - 3 points) In F# you can use parentheses to group expressions just as you do in mathematics and in other programming languages. However, if you put commas inside those parentheses then you are creating a tuple value, as discussed in Section 3.1. So, for instance, if you have a function f and you call it like this:

f (2, 3, 5)

then, technically, are passing f a single value: a 3-tuple of ints (i.e. a value of type int*int*int).

In the above example, f might have type int*int*int -> int and be defined by:

```
let f (x, y, z) =
    x + y + z
```

The *usual* way to pass a function three parameters in F# is to pass them to the function without any commas. So if you want to pass to function g the values 2, 3, and 5, you typically call it like this:

g 2 3 5

Here g might have type int -> int -> int -> int and be defined by:

```
let g x y z =
    x + y + z
```

For this problem, write a function `min3` that returns the smallest element of a 3-tuple. Assume that the elements of the tuple can be compared to each other. You may use ML's `min` function if you like. You may want to compare your solution to this problem to the `min3` function(s) that you wrote for HW1.

```
val min3 : x:'a * y:'a * z:'a -> 'a when 'a : comparison

min3 (2.1, -1.2, 7.);;
val it : float = -1.2

min3('A','B','C');;
val it : char = 'A'
```

Problem 2 (Records - 4 points) Records are explained in Section 3.4.

(a) Define a record type *Dinosaur*. Every Dinosaur has a *name* (a string), a *weight* (a float), and a *height* (also a float).

(b) Create a record named *tyranno*, of type Dinosaur, that represents that Tyrannosaurus weighed 7 metric tons and was 3.66 meters tall.

```
> tyranno.name;;
val it : string = "Tyrannosaurus"

> tyranno.height;;
val it : float = 3.66
```

(c) Create a record named brachio, of type Dinosaur, that represents that Brachiosaurus weighed 35 metric tons and was 9.4 meters tall.

```
> brachio.weight;;
val it : float = 35.0
```

(d) Write a function *nameOfHeavier* that takes two Dinosaurs as arguments and returns the name of the heavier one.

```
val nameOfHeavier : x:Dinosaur -> y:Dinosaur -> string

> nameOfHeavier tyranno brachio;;
val it : string = "Brachiosaurus"
```

Problem 3 (Locally declared identifiers - 3 points)

Write a function `threeDDist` of type

```
x1:float * y1:float * z1:float -> x2:float * y2:float * z2:float -> float
```

It should compute the distance in three-dimensional space from (x_1,y_1,z_1) to (x_2,y_2,z_2). Recall that the Pythagorean Theorem in 3-D says that this is

$$\text{sqrt}((x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2)$$

Perform the squaring by using the multiplication (*) operator. Use at least three locally declared identifiers (via let expressions, as in Section 3.6) in the body of your function definition so that the expression under the square root operator is simple to read. You may need to include a declaration to help the compiler understand that the individual numbers are of type float. See if you can accomplish this with only a single *:float* declaration within one of the parameters.

```
val threeDDist :
  x1:float * y1:float * z1:float -> x2:float * y2:float * z2:float -> float

> threeDDist (1.1,2.5,3.7) (-1.4,1.5,-4.3);;
val it : float = 8.440971508
```

Problem 4 (Tagged values - 5 points)

Recall the following declarations from Section 3.8:

```
type Shape = | Circle of float
             | Square of float
             | Triangle of float*float*float;;

let isShape = function
    | Circle r        -> r > 0.0
    | Square a        -> a > 0.0
    | Triangle(a,b,c) ->
        a > 0.0 && b > 0.0 && c > 0.0
        && a < b + c && b < c + a && c < a + b;;

let area x =
    if not (isShape x)
    then failwith "not a legal shape"
    else match x with
         | Circle r        -> System.Math.PI * r * r
         | Square a        -> a * a
         | Triangle(a,b,c) ->
             let s = (a + b + c)/2.0
             sqrt(s*(s-a)*(s-b)*(s-c));;
```

Extend these declarations by adding a `Rectangle` shape and a `perimeter` function. The functions `isShape`, `area`, and `perimeter` should work on any kind of Shape. Also in your source file define `circ` to be a Circle of radius 2.1, `sq` to be a Square with side 3.6, `tri` to be a triangle with sides 5, 12, and 13, and `rect` to be 3 by 4 Rectangle.

```
val isShape : _arg1:Shape -> bool

val area : x:Shape -> float

val perimeter : x:Shape -> float

> area rect;;
val it : float = 12.0

> isShape tri;;
val it : bool = true

> perimeter circ;;
val it : float = 13.19468915

> perimeter sq;;
val it : float = 14.4

> perimeter tri;;
val it : float = 30.0

> perimeter rect;;
val it : float = 14.0
```

Problem 5 (The option type - 5 points)

Define a function `makeShape` that takes a string, a float, and two float options, and returns a Shape option. If the parameters describe a valid Shape, then that Shape is returned in the option. Otherwise, the function returns None. The third and fourth parameters should be None if they are not needed to define the Shape. (For instance, a Circle or Square can be defined using only one float. Defining a Rectangle requires only two floats.) Use the equality comparison operator (=) to check for the appropriate None values in the parameters. Also, use the isShape function from Problem 4 to verify that the specified Shape is valid. Make sure that your function has the type indicated below, and that it behaves as indicated in the following examples. For each example where the function returns None, I have inserted comments explaining why it returns None.

Use constructs such as local variables, nested-if expressions, and the && and || operators (as covered in the text and as used on some of the earlier exercises in HW 1 & 2) to make your function clear and concise. The instructor's sample solution is fourteen lines lines of easy to read code.

```
val makeShape :
    s:string -> a:float -> b:float option -> c:float option -> Shape option

makeShape "Circle" 3.1 None None;;
val it : Shape option = Some (Circle 3.1)

makeShape "Circle" 3.1 (Some 2.2) None;; // The 2.2 value is superfluous.
val it : Shape option = None             // so the function returns None.

makeShape "Circle" -2.0 None None;;      // The Shape fails the isShape test.
val it : Shape option = None

makeShape "Rectangle" 2.1 (Some 5.4) None;;
val it : Shape option = Some (Rectangle (2.1, 5.4))

makeShape "Rectangle" 3.1 None None;; // Did not provide enough dimensions.
val it : Shape option = None

makeShape "Triangle" 3.0 (Some 4.0) (Some 6.2);;
val it : Shape option = Some (Triangle (3.0, 4.0, 6.2))

makeShape "Triangle" 3.0 (Some 4.0) (Some 7.2);; // Fails the isShape test.
val it : Shape option = None

makeShape "Triangle" 3.0 (Some 0.0) (Some 3.0);; // Fails the isShape test.
val it : Shape option = None

makeShape "Rhombus" 1.2 None None;;       // Not a recognized shape.
val it : Shape option = None
```

Suggestion: There are many possible ways to do this problem, but here is one approach that you may use if you like. First use a `let` expression to initialize a local variable that will be a Shape option, i.e. either Some Circle, Some Square, Some Rectangle, Some Triangle, or None. If an appropriate string and an appropriate combination of numeric parameters are passed in then make the variable some shape. Otherwise, make it None.

For example, if the string is “Rectangle” and we label the other three parameters a, b, and c, as shown in the preceding type expression for makeShape, then b should not be None and c should be None. You can express this logic with something like the following:

```
elif (s = "Rectangle") && b <> None && c = None
then …
```

Finally, after the local variable has been initialized, if it is None or if it fails the isShape test then return None. Otherwise return the variable.

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.