Homework 1
COMP 302 Programming Languages and Paradigms

Francisco Ferreira and Brigitte Pientka
McGill University: School of Computer Science

Due Date: 22 September 2017

Your homework is due at the beginning of class on Sept 22, 2017. All code files must be submitted electronically using my courses. **Your program must compile.** Solutions should correct (i.e. produce the correct result), be elegant and compact, and take advantage of OCaml's pattern matching. Please consult the style guides posted on the course website to get more information regarding what constitutes good style.

The answer to Q0 should be submitted as a txt-file with the name `Q0.txt`; the answer to Q1 should be submitted as a txt-file with the name `Q1.txt`. For the remaining questions, please fill in the template code given on mycourses and submit the files without changing their names.

Remember: Assginments can be done individually or in groups of up to two students. If you choose to do an assignment with another student, both students have to hand in the assignment on mycourses, but you should clearly indicate the person (first and last name and student ID) you have collaborated in the header of each submitted file.

Q0. Copyright and Collaboration policy [3 points]

Read the copyright and collaboration policy of this course. For each of the three scenarios below, write 2-3 sentences explaining who is in violation with the copyright and collaboration policy of the course and why.

Scenario 1: Bob and Tom are working on question Q3 of the homework and have already spent 1h trying to get the code to compile and the deadline is drawing near. They decide to ask CS wiz Hanna who already completed the assignment for help. They explain to her that they get a type error when trying to add two numbers using the operation +. She asks: "Are you sure you are adding two numbers of the same type? You can use "+" to add two integers or "`+.`" to add two floating point numbers". Sure enough, Bob and Tom used the wrong symbol to add two integers, change their code and submit their fixed code.

Scenario 2: Bob and Anne work together on all their homework and early on decide to use github, a web-based Git repository hosting service, to better collaborate. Tom finds their repository after searching for material for COMP302 on the web and re-uses their solution for HW2 and submits it with minor modifications.

Scenario 3: John and Matt agreed to be in one team and have started to work already on question 1 of the homework. Matt talks over skype with Abbey who is also in his class. He mentions that he and John are really stuck on Q4 of the COMP302 homework. Abbey wants to help and sends him her solution in OCaml. A week later, John and Matt submit the solution to Q4; Abbey also submits her homework.

Q1. Parsing, Type-checking and Variable Binding [7 points]

Consider the programs in the file `hw1-fixme.ml`. Try to compile the program either by typing into the caml-toplevel `#use ''hw1-fixme.ml'';;` (note `#use` is a command in OCaml) or in a shell `ocaml hw1-fixme.ml`.

Your task is: Explain step-by-step what errors arise when trying to compile the file `hw1-fixme.ml`; write down the error message you encounter by copy and pasting your errors in your file `Q1.txt` and state below each error how you fix it to proceed to the next error until your program is error free. Classify your error messages into "Syntax Error" (caught before typing and evaluating a program), "Type Error" (caught before executing the program), and "Run-time Error" (caught during evaluation of the program).

Q2. Triangles are the best[45 points]

In this question we explore data representation options, and the importance of choosing the right representation. We motivate this by using everyone's favourite geometric figure, triangles!

We can classify triangles into three categories:

- A *scalene* triangle is a triangle where all its sides of different lengths.
- An *isosceles* triangle is a triangle where two of its sides of equal length and the other side of different length[1].
- An *equilateral* triangle is a triangle where all its sides of the same length.

We define a data-type to distinguish these triangles:

```
type tr_kind
  = Scalene
  | Equilateral
  | Isosceles
```

To represent triangles, we will use triples of floating point numbers. Where each number represents the length of a side of the angle. We define the type `tr_by_sides` to represent triangles.

```
type side = float

type tr_by_sides = side * side * side
```

Q2.1 (15 points) Not all triples form a valid triangle. The Triangle Inequality Theorem states that the sum of two sides of a triangle must be greater than the third side. If this is true for all three combinations, then you will have a valid triangle.

Write a function `well_formed_by_sides: tr_by_sides -> bool =` ... that takes a triple and returns true, if it describes a valid triangle and it returns false otherwise. The function needs to check that all sides are positive and that the sides indeed form a valid triangle.

[1]Normally having two equal sides is enough, but it will make our lives easier to be more strict

```
# well_formed_by_sides (2.0, 3.0, 4.0);;
- : bool = true
# well_formed_by_sides (2.0, 4.0, 6.0);;
- : bool = false
```

Q2.2 **(20 points)** For this question we want to generate triangles that have a specific area. Remember that the area of a triangle is half of the product of its base by its height.

Write a function `create_triangle: tr-kind -> float -> tr_by_sides = ...` that given a kind and a float describing its area, it generates a triangle of that kind and size. Notice that there might be many possible triangles for some combinations, any triangle that satisfies the constraints is a correct response.

For example:

```
# create_triangle Scalene 5.0;;
- : tr_by_sides = (2.23..., 4.47.., 5.)

# create_triangle Isosceles 5.0;;
- : tr_by_sides =
(3.16..., 3.16..., 4.47..)

# create_triangle Equilateral 5.0;;
- : tr_by_sides =
(3.39..., 3.39..., 3.39...)
```

Q2.3 **(10 points)** In any program, the choice of data representation is a very important one. as we saw in Q2.1, the function to check that a value actually represents a triangle is rather convoluted. Let's consider how we could improve the representation to see that it is easier to see that a value represents a triangle.

One such option is to use, the length of two sides and the angle (in radians) between them. In this case, the triangle exists as long as the angle is not a multiple of π.

The representation and the function that checks values are just:

```
type angle = float

type tr_by_angle = side * side * angle
```

```
let well_formed_by_angle (a, b, gamma) : bool =
  (positive a && positive b && positive gamma) &&
    (not (is_multiple_of gamma pi))
```

Notice how the function is almost trivial (using some helper functions provided in the provided file `hw1.ml`).

In this question we ask you to write two function that convert from one representation to another:

- `sides_to_angle: tr_by_sides -> tr_by_angle option`
- `angle_to_sides: tr_by_angle -> tr_by_sides option`

Notice that we use option types to represent that the function fails when provided invalid triangles.

Note:
To solve this question you will need to use some trigonometry. If you do not remember *Pythagoras's theorem* and *the cosine law*, check the wikipedia pages for them, it should be all the math that you need.

Q3. Flexing recursion and lists [45 points]

Recursive functions are the bread and butter of the functional programmer. In this question we explore some functions over lists, a simple recursive data structure.

Q3.1 **(5 points)** Consider a list of integers, write a function `evens_first: int list -> int list` that reorders its elements putting all the even elements in the beginning, followed by the odd elements. This function should preserve the relative order of the even and odd elements.

For example:

```
# evens_first [7 ; 5 ; 2; 4; 6; 3; 4; 2; 1];;
- : int list = [2; 4; 6; 4; 2; 7; 5; 3; 1]
```

Q3.2 **(10 points)** Again, consider a list of integers, but this time write a function `even_streak: int list -> int` that returns the length of longest uninterrupted sequence of even numbers in the list.

For example:

```
# even_streak [7; 2; 4; 6; 3; 4; 2; 1];;
- : int = 3
```

Q3.2 **(30 points)** DNA is the building block of life. It is formed by finite sequences of nucleotides:

- **C**ytosine
- **G**uanine
- **A**denine
- **T**hymine

The sequences can get very long, and contain lots of redundant information.

In this question we will develop a data compressor to save some space in very repetitive sequences. So if we have a sequence like:

```
AAAAGGATTTCTC
```

The compression function will replace repeating nucleotides by a pair of the number of times it appears and that particular nucleotide. Such as the previous sequence results in:

```
(4,A) (2, G) (1, A) (3, T) (1, C) (1, T) (1, C)
```

We represent nucleotides as:

type `nucleobase = A | G | C | T`

Write two functions to compress and decompress these DNA sequences such as decompressing a compressed sequence produces the original sequence.

- `compress: nucleobase list -> (int * nucleobase)list`
- `decompress: (int * nucleobase)list -> nucleobase`

For example this needs to produce:

```
# compress [A; A; A; A; G; G; A; T; T; T; C; T; C];;
- : (int * nucleobase) list =
[(4, A); (2, G); (1, A); (3, T); (1, C); (1, T); (1, C)]
# decompress [(4, A); (2, G); (1, A); (3, T); (1, C); (1,
   T); (1, C)];;
- : nucleobase list = [A; A; A; A; G; G; A; T; T; T; C;
   T; C]
```