COMP712 Programming Languages

Programming Portfolio Part 1 (25%)

IMPORTANT: Your code for these programs should not require any functions outside of racket/base.

1. Tail Recursion (3%)

Task: Write and test a Racket procedure called **poly** to compute the value of a polynomial given a value and a list of coefficients. If **coeff** is a list of coefficients **(a0 a1 a2 . . . an)**, then **(poly x coeff)** should compute the value

$a0 + a1*x + a2*x^2 + a3*x^3 + . . . + an*x^n$

You may assume that **coeff** contains at least one item.

2. Higher-Order Functions (3%)

Task: Write a Racket procedure called **apply-all**. When given a list of functions and a number, this procedure will produce a list of the values of the functions when applied to the number.

Usage Example:

```
(apply-all (list sqrt square cube) 4)
⇒ (2 16 64)
```

3. Higher-Order Functions (3%)

Task: Given a predicate that tests a single item, such as **`positive?`**, construct an "all are" version of it. This procedure, to be named **all-are**, takes a predicate and returns a new function that can be applied to a list of elements. It should return **`#t`** if all the elements of the list satisfy the predicate.

Usage Example:

```
((all-are positive?) '(1 2 3 4))
⇒ #t
((all-are even?) '(2 4 5 6 8))
⇒ #f
```

You should test your procedure with more than just **positive?** and **even?**

4. Lexer (16%)

Task: Write a lexer in Racket for a simple language **L1** that can be described in the following example:

```
let x = 4^2;
let y = x * -2.0;
let result = x - y;
print result
```

Here, **`x, y`** and **`result`** are identifiers. Identifiers can include characters like _ (underscore), $, and – (hyphen).
`let` and **`print`** are keywords.
Numbers are either unsigned integers or unsigned floating point numbers.
Arithmetic operators include **`+, *, /,`** and **`^`** . Note that **`–`** should be a different kind of token from numbers and arithmetic operators. It is meant to be left to the parser to determine if it should be interpreted as a minus operator or the negative of a number.

Your lexer should be invoked by

```
(tokenize some-file)
```

where **some-file** is the name of a file with program code written in L1. It should return a list of tokens. Each token should be a structure with a symbol indicating the type of token and the relevant attribute(s).

You are not allowed to use any Racket parser/lexer tools.

Note that you should avoid implementing everything in a single procedure.

Your design should be fully documented by detailed comments.

Submission: Your submission should consist of just the Racket source files. All source codes must have appropriate comments. The file for each question should be named the same as the function. They should be archived (i.e. zipped) into a single file with ***your last name and student number as the filename*** for submission through Canvas.

----------END----------