Assignment 2
COMP 302 Programming Languages and Paradigms

Brigitte Pientka and Francisco Ferreira
MCGILL UNIVERSITY: School of Computer Science

Due Date: 8 Oct 2015

Q1. Proof we need! Proof! [25 points]

Hand in your proofs as a pdf file q1.pdf

Consider the following program that concatenates strings in a given list to one string.

```
let rec concat l = match l with
  | [] -> ""
  | x::xs -> x ^ (concat xs)
```

Your friend implements the following version of this program and claims it is doing the same thing. In fact, he claims it is better!

```
let concat' l =
let rec conc l acc = match l with
  | [] -> acc
  | x::xs -> conc xs (acc ^ x)
in
   conc l ""
```

20 points Prove that indeed `concat l` produces the same result as `concat' l`.

5 points Is your friend right in claiming `concat'` is better? In what sense is it better?

Q2. Removing Duplicates [10 points]

Implement a function `remove_duplicates:('a * 'a -> bool)-> 'a list -> 'a list`. This function accepts an function `eq:'a * 'a -> bool` which compares two elements and a list `l`. It returns a list `l'` which is obtained by removing duplicates from the list `l`.

Q3. Convolution [15 points]

Write a function conv which computes the following integral: $\int_0^x f(y) * g(x+y)\, dy$

The function conv takes in as arguments a function f of type floats -> floats and a function g of type floats -> floats, and a small value dy. It will return a function of type floats -> floats. In other words:

```
conv:(floats -> floats)-> (floats -> floats)-> floats -> (floats -> floats)
```

Use any function you have written before or we have discussed in class. Your answer should be very short, and not more than one line.

Q4. Functional Parsing [50 points]

In the functional parsing lecture we learned how to write a parser for a small language of arithmetic expressions. In this assignment we will extend the language of that parser so it can parse a richer set of arithmetic expressions. You will work on the file parser.ml and write all your answers in it. We will discuss how to build a functional parser in class. This question builds on this discussion.

We revisit the pocket calculator from assignment 1, question 3. The idea of this assignment is to implement a parser for a similar language.

```
type exp =
| Plus of exp * exp
| Minus of exp * exp
| Times of exp * exp
| Div of exp * exp
| Sin of exp
| Cos of exp
| Exp of exp * exp
| Neg of exp
| Num of int
```

There are some small changes: instead of floats we will use natural numbers, and we will add the unary negation of expressions.

The syntax your parser needs to understand is the following:

$$
\begin{array}{lll}
\text{Expressions} & E & ::= \; N \mid E+E \mid E-E \mid E*E \mid E/E \mid sinE \mid cosE \mid E\hat{\ }E \mid -E \\
\text{Numbers} & N & ::= \; 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid N*
\end{array}
$$

Note how the parser for numbers is the same that appears in the functional parsing notes.

Q4.1 [20 points] In order to implement this parser you will have to factor the grammar, so that the exponential binds tighter, then product and division, and addition and subtraction bind the loosest. Not that the unary operators have the highest precedence. To do this proceed in a similar way as in the example expression parser from the notes. Write your solution in the comment section of the file `parser.ml`.

Q4.2 [15 points] Implement a parser for these expressions using the parsing library developed in the notes on functional parsing. To simplify the implementation the expressions will not contain any whitespace. The task is to implement the function `expr_no_parens` which takes as input a string, **in**p, and returns an expression.

Some example expressions:

$$1 + 1$$

$$1 + 2 * 3$$

$$1 * 2 + 3$$

$$1 + 2\text{\textasciicircum}3$$

$$1/2 - 3$$

$$\mathtt{sin}1 + 2 * 3$$

$$1 + \mathtt{cos}2\text{\textasciicircum}3$$

$$-2 * -3$$

Q4.3 [5 points] Parenthesis are an important aspect of arithmetic expressions. Add parenthesis support to your grammar. Start from the grammar definition in Q4.1 and extend it to support parenthesis. Write your solution in the comments provided in the file `parser.ml`

Q4.4 [10 points] Implement the parser using the grammar from Q4.3. Note that you should NOT change the definition of the type of expressions.

Some example expressions with parenthesis:

$$((1) + (1))$$

$$(1 + 2) * 3$$

$$(1 * 2) + 3$$

$$1 + 2\text{\textasciicircum}3$$

$$1/2 - 3$$

$$\mathtt{sin}(1 + 2) * 3$$

$1 + \cos(2\hat{}3)$

$-2 * (-3)$

Pro tip: You can adapt your evaluation function from assignment 1 to work with these expressions and validate that your parser produces the correct tree.