Instructors: Matt Fredrikson, Jean Yang TA: Samuel Yeom

Due date: 1/24/2017 at 11:59pm

Assignment 0

Instructions: Complete all the required problems listed below. You may also submit the solutions for the optional problems if you would like our feedback.

When finished, compile your solutions to a pdf, and email a zip of the PDF and relevant code to the TA by the due date. Be sure to add your Andrew ID and full name in the `stulogin` and `stuname` macros at the top of `hw0.tex`.

1 Course startup (5 points)

Part 1 (0 points) If you have not been added to the course's Piazza, please email the instructors (`15316-spring17-staff@cs.cmu.edu`) to be added.

Part 2 (5 Points) Email the course staff (`15316-spring17-staff`) introducing yourself! Tell us your name, your major and year, how the course fits in with your core values, what you hope to get out of the course, and a fun fact about yourself. Feel free to tell us your questions and concerns as well.

2 Getting Started with OCaml (0 points)

The skeleton code for this problem is in `2/exercises.ml`.

Part 1 (0 points) Follow the instructions to install OCaml on your system:

```
http://www.ocaml.org/docs/install.html
```

Install the OPAM package manager and run the following to initalize OPAM, and then to install the OUnit package for unit testing:

```
opam init
opam install ounit
```

Part 2 (0 points) Open an OCaml interactive session (`ocaml`) and use it to determine the types of the following functions. What do the types mean?[1]

1. `let f (x,y) = x ::  y`
2. `let f (g, h) = function x -> g (h x)`
3. `let f g h = function x -> g (h x)`
4. `let f x y z = if x < y then "hello" else z`

Part 3 (0 points) Complete the code and tests from `exercises0.ml` to write the following functions:

1. Fibonacci.
2. List reversal.
3. A function `filter lst f` that takes a list `lst:  'a` and a function `f:  'a -> bool` and returns a list r: 'a of the elements in `f` satisfying `f`.

Run the code using the following command to create the executable `exercises0`:

```
ocamlfind ocamlc -package oUnit -linkpkg -g -o exercises0 exercises0.ml
```

[1] Problem taken from `https://www.cs.rice.edu/~sc40/COMP507/Assignments/assign0.pdf`

3 A Tiny Calculator (0 Points)

For this exercise, we will build a calculator in *reverse polish notation.* Also called *postfix* notation, RPN is a mathematical notation in which every operator follows all of its operands. This notation is often used in stack-based and concatenative programming languages.

Here are some examples of expressions in infix and prefix notation:

Infix	**RPN**
$1 + 2$	$1\ 2 +$
$3 - 4 + 5$	$3\ 4 - 5 +$
$(3 - 4) * 5$	$3\ 4\ 5 * -$

The algorithm for RPN is as follows:

```
while there are input tokens left do
    Read the next token from input;
    if token is a value then
        Push it onto the stack;
    else
        It is already known that the other takes $n$ arguments;
        if there are fewer than $n$ values on the stack then
            Raise an error;
        else
            Pop the top $n$ values from the stack;
        end
        Evaluate the operator with the values as arguments;
        Push the returned results back onto the stack;
    end
    if there is only one value in the stack then
        That value is the result;
    else
        Error: the user input has too many values;
    end
end
```

Algorithm 1: Reverse Polish Notation algorithm.

The skeleton code for this problem is in `3/calc.ml` and is based on a snippet from Rosetta Code.

Part 1 (0 points) Compile `calc.ml`:

```
ocamlc str.cma calc.ml -o calc
```

You should get the `Unimplemented` error when you run `calc`.

Part 2 (0 points) Understand the structure of the code that we have written for you. What do the functions `print_answer`, `rpn_eval`, and `interp_and_show` do?

Part 3 (0 points) Fill in the bodies of `interp` and `binop` so that running `calc` prints the following output:

```
***
3 2 5 + -
Token Action Stack
3 push 3.
2 push 3. 2.
5 push 3. 2. 5.
+ add 3. 7.
-subtr -4.
-4.
***
3 2 + 5 -
Token Action Stack
3 push 3.
2 push 3. 2.
+ add 5.
5 push 5. 5.
-subtr 0.
0.
***
2 3 11 + 5 - *
Token Action Stack
2 push 2.
3 push 2. 3.
11 push 2. 3. 11.
+ add 2. 14.
5 push 2. 14. 5.
-subtr 2. 9.
* mult 18.
18.
***
9 5 3 + 2 4 ^ - +
Token Action Stack
9 push 9.
5 push 9. 5.
3 push 9. 5. 3.
+ add 9. 8.
2 push 9. 8. 2.
4 push 9. 8. 2. 4.
^ exp 9. 8. 16.
-subtr 9. -8.
+ add 1.
1.
```