CSE 335 Winter 2017
Homework 7

Suggested reading:
- EOPL: 2.4 (refresh your memory on define-datatype)
- EOPL: B.1-B.3 (about sllgen)
- EOPL: 3.1-3.2 (implementation of the LET language)
- EOPL: 3.3 (implementation of PROC language)
- Course-slides: Apr5-CSE335 lecture notes (for the procedure definition and procedure call part)

In this homework you will be adding new features regarding procedures to the language you implemented for the last homework (hw06).

To refresh your memory on the syntax of the language from hw06:

```
<program> ::=
             <expr> <expr>*  "a-program"

<expr> ::=
           number           "num-expr"
         | up(<expr>)       "up-expr"
         | down(<expr>)     "down-expr"
         | left(<expr>)     "left-expr"
         | right(<expr>)    "right-expr"

         | (<expr> <expr>)   "point-expr"
         | + <expr> <expr>   "add-expr"
         | origin? (<expr>)  "origin-expr"
         | if (<expr>)
            then <expr>
            else <expr>       "if-expr"

         | move (<expr> <expr> <expr>*)  "move-expr"
         | identifier              "iden-expr"
         | { <var-expr>* <expr>*} "block-expr"


<var-expr> ::=
               | val identifier = <expr>         "val"
               | final val identifier = <expr>   "final-val"
```

The implementation code is spread out in 3 files:
- "hw07-lang.rkt" contains the lexical, grammar specification and the sllgen boilerplate code
- "hw07-env-values.rkt" contains the expressed values and the environment
- "hw07-interpreter.rkt" contains the implementation of our interpreter.
+ the tests found in "hw07-tests.rkt"

You can look at the test suite called "original" in hw07-tests.rkt
to refresh your memory on the semantics of the already existing
language as a result of hw06.

Recall that we invoked the interpreter for our language with the
"run" function as follows:
> (run "42")
(num-val 42)
and expression in our language evaluates to one of the "expressed-val"
found in hw07-env-values.rkt

Submission guidelines:
  - please archive all 4 files into one zip file: hw07-LASTNAME.zip
        -hw07-lang.rkt
        -hw07-env-values.rkt
        -hw07-interpreter.rkt
        -hw07-tests.rkt
  - only submit the zip file: hw07-LASTNAME.zip

Since you will be working with an already existing implementation, you
will need to :
  + change hw07-lang.rkt, hw07-interpreter.rkt and hw07-env-values.rkt
    to add the required new functionality described in problem 1 & 2;
  + please comment any changes you make.

When you add these new features, the original semantics of the
language have to remain intact, so whenever you test your
implementation make sure you run the regression tests "original".

;===
1. [50p] Anonymous function definitions

Just like in racket, functions/procedures in our language are first
class citizens and you can pass them as parameters, store them in
variables or return them as the value of the program.

We will implement a feature similar to the (lambda) expression in
racket described by the grammar:

<expr> ::= <original language>
         | fun (identifier*) = <expr>   "fun-expr"
         | call (<expr> <expr>*)   "fun-call-expr"

The "fun-expr" will create a function with 0 or more parameters that
has the body <expr>

The "fun-call-expr" , the first <expr> has to be a function, and the
remainder of the <expr>s will be the parameters passed to the said

function.

You will have to implement support for the concept of closures (see test file for more information) and higher order procedures in the language.

Important steps:
  - add a new type of expressed value
  - extend the grammar
  - implement the new features in the value-of functions.

See the tests for additional clarification on semantics.

;==
==
2. [50p] Named function definitions

Modify the syntax of the language to the following:

```
<program> ::=
             <var-expr>;* <expr> <expr>* "a-program"

<var-expr> ::=
                | val identifier = <expr>                "val"
                | final val identifier = <expr>          "final-val"
                | def identifier (identifier*) = <expr>  "fun-def"
```

The "fun-def" <var-expr> allows the declaration of named functions. It is mostly a syntactic sugar for assignments anonymous functions to variables, so you should be able to use any named function just like you could an anonymous one.

With this new grammar specification, one can write an arbitrary number of variable and function definitions before writing the body of a program.

```
> (run
 "def named-fun(x) = x;

  #program body:
  call(named-fun 42)
 "
)
(num-val 42)
```