CPSC 326: Homework Assignment 6
Due Thursday, March 9

This homework assignment has two parts. The first part is to finish off the MyPL language implementation. The second part is essentially a simple getting-started in Haskell assignment.

PART I.

The goal of this part is to write an interpreter for MyPL. You will need to create a new class Interpreter in the file mypl_interpreter.py. Your Interpreter class must implement the methods defined in the Visitor class from mypl_ast.py and you must use the SymbolTable class in your implementation. You will also need to call your main program hw6.py to run the interpreter (which is also a slightly modified/simplified version from HW5):

```python
... 
stmt_list = the_parser.parse()
stmt_list.accept(type_checker.TypeChecker())
stmt_list.accept(interpreter.Interpreter())
```

As a simple example of what your program should do, given the MyPL program test.txt:

```plaintext
x = 3 + 4;
msg = "The value is: "
print(msg);
println(x);
```

your program should output the following.

```bash
$ python hw6.py test.txt
The value is: 7
```

As before, make sure to give yourself enough time to ensure you finish the assignment. Like in HW5:

1. To receive **full points**, you only need to implement your evaluator for *non-list types*.

2. If you are able to finish (1) and would like an additional challenge, you can receive extra **credit** by implementing the interpreter for list types. (Note, however, you should also ensure type checking works for list types. This is really worth doing to go deeper into the approaches and also to finish off the language.) For lists, you must do index out-of-range checking (both within expressions and when assigning into a list).

Note that we will go over strategies for implementing the interpreter in class.

On the due date, **hand in** a cover sheet together with hard copy of your implementation (all new or modified source code files), hard copy of your test program, print outs of tests showing your parser works properly, and a design document. In addition, submit all files required to run your program as a single zip file to the online dropoff site [https://www.cs.gonzaga.edu/dropoff/](https://www.cs.gonzaga.edu/dropoff/). Note that in your hard copy, your tests must include both the input files and the output produced after running your program over the files.
PART II.

The goal of this part is to introduce you to basic Haskell syntax and to help you become familiar with the GHC interpreter (ghci). If you want to install and use GHC on your personal computer or laptop (highly recommended), you can download it from http://www.haskell.org. General information about Haskell is also available on the HaskellWiki (http://www.haskell.org).^1^ Note that GHC version

**Instructions.** Provide answers to the following steps and turn in your assignment by the due date. Be sure your name is clearly marked on your assignment. Also, clearly mark the answers to each question (i.e., the question number and letter as appropriate).

1. Start ghci and issue the help command

   Prelude> :help

   Note that you can also use :? for help within ghci. Now issue the command

   Prelude> :info round

   What does this return?^2^

2. Enter each of the following commands into ghci and list what they return and whether the result was what you expected. If the command gives an error, take a guess at what the problem is.

   (a) 5 + 8
   (b) 3 * 5 + 8
   (c) (+) 5 8
   (d) 2^1000
   (e) even 9
   (f) succ 6
   (g) succ (pred 6)
   (h) succ pred 6
   (i) gcd 21 27
   (j) gcd(21, 27)
   (k) [1,2,3,4]
   (l) [1..4]
   (m) [1..4] ++ [5..10]
   (n) 1 : [2,4..8] ++ [9,11..16]
   (o) [1, 2, 3.14]
   (p) [1, "2"]

---

1^To install on Ubuntu, from a terminal type sudo apt-get install ghc

2^Note that round is a function defined within the Haskell “Standard Prelude” (the default Haskell library), which is documented here (assuming you are using ghc 7.0.4 or higher): http://www.haskell.org/ghc/docs/7.0.4/html/libraries/base-4.3.1.0/Prelude.html
3. List the types for each valid expression in Question 2 (i.e., those that don’t result in an error). In ghci you can obtain the type for an expression \( e \) by issuing the special “:type” command. For this part, store the result of the expression in a variable, then obtain the type of the variable using :type. For example,

\[
\text{Prelude>} \text{let } x = 5 + 8 \\
\text{Prelude>} \text{:type } x
\]

The type of an expression is everything after the :: symbol (read as “has type”). Be sure to list the type returned as it is given.

4. Answer the following using ghci:

(a) Define \( \text{zp} \) to be the infinite list of positive integers (i.e., \( \mathbb{Z}^+ \)). *Hint 1:* Use the \textbf{let} \( v = e \) syntax to assign the result of expression \( e \) to \( v \). *Hint 2:* Use the list enumeration syntax \([n..]\) to create an infinite list starting at number \( n \).

(b) What is the type of \( \text{zp} \)?

(c) What happens when you enter

\[
\text{Prelude>} \text{zp}
\]

at the command prompt? *Hint:* to stop printing hit Ctrl-C.

(d) What is the result of executing

\[
\text{Prelude>} \text{head zp}
\]

(e) What is the result of executing

\[
\text{Prelude>} \text{head (tail zp)}
\]

(f) What is the result of executing

\[
\text{Prelude>} \text{head (drop 100 zp)}
\]

5. An often used function in Haskell (and functional programming in general) is \textit{filter}.

(a) What is the result of executing

\[
\text{Prelude>} \text{filter even [1..10]}
\]

(b) What are the types of the two arguments being passed to \textit{filter} in this call? (Again, to obtain the types use the :type command.)

(c) What is the type of the \textit{filter} function itself (i.e., :type filter)?

(d) In Haskell, we can create “partially defined” functions (we’ll talk more about how this works later). For example, execute the following:

\[
\text{Prelude>} \text{let evens = filter even}
\]

Now, what is the result of executing:

\[
\text{Prelude>} \text{evens [1..10]}
\]

(e) What is the type of your new \textbf{evens} function? Make a guess as to why \textbf{evens} has this type.