CSE 465/565
Spring 2019
Homework #1

Instructions: Submit an electronic copy of all questions and programs. The electronic copies should be placed into the course's Canvas site.

Submit to Canvas a single zip file that contains the following directory structure:

uniqueidHW1 ; top-level directory containing all of your stuff
HW1.pdf ; which contains your Z+- report and answers to non-coding questions
ZPM ; directory containing your source code,
Main.java, Helper.java, etc ; your source Java source code. NO packages

1. (20/20) Vandelay Consultants are designing a new language based on Java and their language designers have been debating whether their language should allow variable declarations to occur at arbitrary positions in the source code. For example, allowing arbitrary declarations would allow:

```
public static int f(int x) {
     int a = 0;
     for (int i=1; i<=x; i++) {
          int tmp = 2 *  i;
          a += tmp;
     }
     int b = 2 * a;
     return a;
}
```

If they disallow this feature, all variable declarations will need to occur at the top of the function definition, as shown here:

```
public static int f(int x) {
     int a = 0;
     int b;
     int tmp;
     for (int i=1; i<=X; i++) {
          tmp = 2 *i;
          a += tmp;
     }
     b = 2 * a;
     return a;
```

```
}
```

You have been hired as an expert by to write 300 words (at most) describing the pros and cons of allowing declarations to occur at arbitrary positions. Your report should focus on readability and writeability. When appropriate, you should provide code examples (which are not part of your 300 word budget) to support your claims. The last sentence of your write up should provide your recommendation for the proposal on a scale of 1 to 10, where 1 represents definitely oppose; 5 represents a neutral; and 10 represents definitely support.

Scoring will be based on the following criteria:

- All relevant pros and cons identified
- Examples provided to illustrate issues
- Statements and supporting evidence are accurate
- Writing is well-organized, clearly written, and free from grammatical and spelling errors.
- Conclusions and statements logically follow the supporting evidence

2. (10/10) Log into `ceclnx01.csi.miamioh.edu` and create the following Java file - `Sample.java`.

```
import java.util.*;
public class Sample {
    public static void main(String [] args) {
        Scanner input = new Scanner(System.in);
        String user = System.getProperty("user.name");
        System.out.println(user);
        int printed = 0;
        while (input.hasNext() && printed < 10) {
            String line = input.nextLine();
            if (line.indexOf("zmudam") == 0 ||
                line.indexOf(user) == 0) {
                    System.out.println(line);
                    printed++;
            }
        }
        input.close();
    }
```

```
}
```

Compile and run the program with the following commands. Insert the output into your PDF.

```
bash> javac Sample.java
bash> last | java Sample
```

2. (70/70) Consider the simple programming language named `Z+-`. The `Z+-` programming language has the following features:

- `Z+-` variables can store a string or integer value. Each variable can switch between integer and string values during program execution, which is known as dynamic typing. Assigning a value to a variable creates that variable for future use. A runtime error occurs if a variable is used before it is given a value.
- Variables are case sensitive and consist only of upper and lower-case letters.
- The following are `Z+-` reserved words: `PRINT, FOR, ENDFOR, PROC, ENDPROC, CALL`
- You may assume that the `Z+-` program is syntactically correct.
- The `PRINT` statement displays one particular variable's value. This is done as:

```
PRINT numCookies ;
```

- The right-hand side of a simple assignment statement (i.e., =) is either a variable name (which must have a value), signed integer, or string literal. For example, the following are valid:

```
A = 12 ;
A = B ;
A = "hello" ;
```

- There are three compound assignment statements: `+=`, `*=`, and `-=`. The meaning of these operators depends on data type of the left and right hand side of the operator.

```
<string var> += <string>   concat right string onto end of
left string
<integer var> += <integer> increment left integer with
value on right
<integer var> *= <integer> multiply left integer by value
on right
<integer var> -= <integer> subtract right integer from
value on left
```

```
A += 34 ;
```

```
A *= B ;
A += "hello world" ;
```

All other combinations are illegal and cause a runtime error.

- Every statement is terminated by a semi-colon.
- There is a loop statement – `FOR` – whose body contains at least one simple statement (i.e., no nested loops), which are on presented on one line. The keyword `FOR` is followed by an integer constant, which indicates the number of times to execute the loop. Following this number is a sequence of statements defining the loop's body, followed by the word `ENDFOR`, as done here:

  ```
  FOR 5 B += A ; A *= 2 ; ENDFOR
  ```

- `Z+-` for loops can be nested and must appear on one line:

  ```
  FOR 5 B += A ; A *= 2 ; FOR 10 A += B ; ENDFOR A += 2 ;
  ENDFOR
  ```

- `Z+-` programs must have at least one space separating all lexical elements.
- Here is an example `Z+-` program:

```
A = 1 ;
B = 0 ;
FOR 5 B += A ; A *= 2 ; ENDFOR
A += 1000 ;
PRINT A ;
PRINT B ;
```

This program's output would be:

```
A=1032
B=31
```

Here is a second `Z+-` program:

```
A = 10 ;
A += A ;
PRINT A ;
A = "hello" ;
A += A ;
PRINT A ;
A += 123 ;
```

```
PRINT A ;
```

The output to this second program would yield an error. Your program should display the line number of this error and then stop processing:

```
A=20
A=hellohello
RUNTIME ERROR: line 7
```

- Requirement for graduate students only: Graduate students must support parameterless procedures. Any procedures must be defined prior to the "main program statements". The procedures must have unique names and contain a list of legal Z+- statements. Recursion is not allowed. All variables created/used are those at the global level. Here is an example:

```
PROC F
A *= 2 ;
B *= 2 ;
ENDPROC

PROC G
C += 1 ;
CALL F ;
ENDPROC

A = 1 ;
B = 1 ;
C = 0 ;
CALL F ;
FOR 10 CALL G ; ENDFOR
```

Notes.

- You may assume that the programs are syntactically correct but may have runtime errors (e.g., add integer and string).
- Your program must run on ceclnx01 using the standard compile command. The name of the ZPM file will be passed to your program using a command line argument:

```
c++ -std=c++11 -o myapp *.cpp
myapp prog.zpm
```

```
javac *.java
java Main prog.zpm
```

Tasks and scoring.

- (70/70 points) Write an interpreter in Java to execute `Z+-` programs
  - (30/15 points) Basic structure, integer variables only
  - (15/10 points) Basic structure, integer AND string variables
  - (10/10 points) For loops
  - (10/15 points) Nested for loops
  - (5/5 points) Detection of runtime errors
  - (0/15) Parameterless procedures
- Provide a report that lists the following:
  - The report should not include a narrative about your experiences writing the interpreter. Simple list the items that do/do not work.
  - Provide a list of the features that your program correctly processes. This could be as simple as: “everything works correctly.”
  - Provide a list of the features that your program does not correctly processes. This could be as simple as: “there are no unimplemented requirements.”
  - Provides the output of your program when run on all sample `Z+-` programs provided in the Google Docs HW folder. Identify any incorrect output that your program produces.
  - Compares the runtime speed of `Z+-` programs versus an equivalent program in some compiled language (e.g., C++). Provide timings to support your estimate. This should be done with simple and complex `Z+-` programs. Are your findings typical of interpreted languages?