CSE 465/565
Homework #1: 55 points

Outcomes:

- Warming up by writing a java code
- Understand how an interpreter works
- Understand some of the difficulties involved in designing a new programming language.

Scoring:

- (10 pts) Successful upload on GIT. If your code is submitted on GIT successfully, the following aspects of your program will be graded
- (45 pts) Write an interpreter in Java to execute Z+- programs
    - (10 pts) Basic structure, integer variables only
    - (10 pts) Basic structure, integer and string variables
    - (10 pts for Graduate students-20 pts for Undergraduate students) For loops
    - (10 pts) Nested for loops (only for graduate students)
    - (5 pts) Detection of runtime errors

Requirements:

- Java version 17 or older. If you haven't installed java before on your laptop, download JDK from here: https://www.oracle.com/java/technologies/downloads/ .You can install the latest version which is JDK21 (if you already don't have any java version 17 or older on your laptop), reset your computer, and it should be done.
- Although you can use any IDE of your choice, Eclipse is recommended. Follow the instructions for installing and configuring Eclipse here: Install and Configure Eclipse
- You **must** name your main class **Zpm**. You can create additional classes as needed, but the **Zpm** class should be the one containing the main method.
- Your program must take a Z+- code as a file with **.zpm** extensions through the command line argument. which means your program must run on a terminal (windows terminal or CMD on windows, and terminal on mac) using the standard compile command. Naming the main class **Zpm**, and assuming prog.zpm is a file with Z+- codes in it:

```
javac *.java
java Zpm prog.zpm
```

- So, your program does not prompt the user to enter a file or anything like that. Instead, it receives a file as an argument when the program is run through the terminal (as shown above), not during compilation.

Instructions:

(55 points) Consider a very simple programming language named Z+-. The Z+- programming language has the following features:

1. Call your project on GitLab/GitHub **Homework1**. This folder/package should be saved inside your "**CSE465_565**" project on your GIT.
2. Z+- variables are case-sensitive and consist of one letter [A-Z]. No numbers, no special characters.
3. Z+- variables can store a string or integer value. **<u>A single variable can switch between integer and string values during program execution</u>**. Assigning a value to a variable creates that variable for future use.
4. The only runtime error occurs if a variable is used before it is given a value. No need to check for other types of errors: syntax or runtime errors.
5. The PRINT statement displays a particular variable's value. This is done as:

   ```
   PRINT numCookies ;
   ```

6. 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 ;    (B must have values)
   A = "hello" ;
   ```

7. There are three compound assignment statements: +=, *=, and -=. The meaning of these operators depends on the 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 ;    (B must have values)
   A += "hello world" ;    (Assuming A is already holding a
   String value)
   ```

   All other combinations are illegal and cause a runtime error.
8. Every statement is terminated by a semi-colon.
9. There is a loop statement – **FOR** - whose body contains at least one simple statement (i.e., no nested loops), which are 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** (no semicolon after ENDFOR).

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

10. **Graduate Students:** Graduate students should also make the Z+- language to work with nested loops:

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

this could be more than one loop inside another one.

11. Z+- programs must have at least <u>one space separating all elements.</u>
12. You also need to throw an exception if the given file is not a .zpm file or no file is given.
13. A general rule: Any line that ends with a value (e.g., "hello", 3434) or a variable (e.g., A) should conclude with a semicolon. Therefore, assignment statements and print statements must end with a semicolon. However, for loops should conclude with 'ENDFOR'.

Sample Run:

1. 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 is (**pay attention that there is no space in between)**:

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

2. 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 be (**pay attention that there is no space in between)**:

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

You may assume that the programs are syntactically correct but may have runtime errors (e.g., add integer and string, or doing operator on a variable that hasn't been initialized).

When an error happens, your program should print the runtime error, and stop the program.

Test your program:

Come up with Z+- codes similar to the ones shown above, and test your program comprehensively.