**Lab07 – Writing an Interpreter**

COMP256 – Computing Abstractions

Dickinson College

Spring 2021

Prof. Grant Braught

**Name(s):**

**Introduction:**

The topic on language abstractions began by drawing a distinction between interpreted and translated (or compiled) languages. In class, we have gotten a little feel for language translation by learning how high-level language instructions can be expressed in assembly language. In this lab you will gain hands-on experience with interpretation by writing an interpreter for a small programming language. Later labs will give you hands-on experience with translation by writing an assembler and possibly a simple compiler as well.

**An Example:**

The following is a short program in the programming language *Silli* – **S**imple **I**nstruction **L**anguage for **L**earning **I**nterpreters that was invented for this lab.

# Make some variables.

LET I=0

LET J=1

LET K=0

# Get the input

READ K

LOOP: PRINT I

I=I+J

IF I<K GOTO LOOP

PRINT I

1. Study the above Silli program.

a. Imagine the user provides the value 3 for the variable K. What output would this program print?

b. Give a short sentence that explains at a high level what this program does. Do not describe the steps it takes, just describe what result it computes.

**The Silli Language Syntax:**

Every high-level language has a syntax that defines the statements that may be used in a program written in that language. Statements in the Silli language will be made up of the following *elements*:



2. Identify the *type* of each of the following *elements*.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Element** | **Element Type** |  |
|  | a. > |  |  |
|  | b. \* |  |  |
|  | c. START |  |  |
|  | d. 8730 |  |  |
|  | e. X |  |  |
|  |  |  |  |

The above *elements* are used to create *statements* in the Silli language. The Silli language has a total of seven different types of statements, each of which is described in the table below:



3. Identify the *type* of each of the following *statements*:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Statement** | **Statement Type** |  |
|  | a. READ X |  |  |
|  | b. Z=M+R |  |  |
|  | c. IF A=B GOTO START |  |  |
|  | d. PRINT Z |  |  |
|  | e. LET K=7 |  |  |
|  |  |  |  |

Notice that Silly is a little peculiar in that arithmetic statements may contain only variables. So, the statement X=Y+Z is valid assuming that Y and Z have been defined. **But X=Y+2 or Z=3+5 are not valid statements.** This complicates programming in Silli a little, bit but it makes writing an interpreter for the language easier. But because we can always put a value into a variable, it doesn’t actually limit what we can do with the language. At the end of the lab there are some optional extensions that will allow arithmetic statements with literal values like X=Y+2 if you are interested.

4. What is the only type of statement ion Silli that is allowed to have an integer value in it?

There are a few other things you need to know about the Silli language before we can begin creating an interpreter for it.

* Silli is *case insensitive*. Thus, the statement LeT R=2 has the same meaning as the statement LET r=2.
* Blank lines and comments are ignored.
* Any leading spaces or tabs (i.e. indentation) are ignored.
* Within statements the *exact spacing* shown in the table above must be used. Thus, the statement X=Y+Z is valid, but the statements X = Y + Z or X=Y + Z or X= Y+ Z, ect. are not valid.

5. Consider the following statements. Rewrite each statement with proper spacing if possible. If a statement cannot be rewritten so that it is valid in Silli, just write *invalid*.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Statement** | **Rewritten Statement** |  |
|  | a. LET X = 7 |  |  |
|  | b. C = X+ Q |  |  |
|  | c. IF X > R GOTO END |  |  |
|  | d. Y=X + 9 |  |  |
|  |  |  |  |

The last piece of Silli syntax that we need to be precise about are the *line labels*. **A line label is defined by starting a line with a <LABEL> element, followed by a colon.**  **Note that given this definition, the : character is not part of the label.** A defined label must then be followed by space and a valid non-blank, non-comment statement. Some examples of lines that define labels are:

INPUT: READ K

LINE1: PRINT I

MATH: I=I+J

COND: IF I<K GOTO LOOP

6. Indicate which of the following are **labels that are defined by** the above statements:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Label** | **Defined Above (Y/N)?** |  |
|  | a. INPUT |  |  |
|  | b. LINE1: |  |  |
|  | c. COND |  |  |
|  | d. LOOP |  |  |
|  | e. PRINT |  |  |
|  |  |  |  |

7. While any non-blank, non-comment line may define a label, only one type of statement makes use of a label. What type of statemen is that?

**Getting Setup and Familiar:**

Okay we are now ready to begin writing the interpreter that will execute programs written in our language. We’ll start by getting the project setup and configured so that you can compile and run the interpreter. Then you’ll explore the starter code and then finally begin writing the rest of the interpreter.

8. Download and install Eclipse if you don’t already have it running on your machine. See: <https://www.eclipse.org/downloads/>).

9. Download and unzip the Lab07-A-Interpreter.zip file from the course Moodle. What files are contained in this zip?

10. You can use any text editor to write your Silli interpreter. If you would like to use Eclipse to write your Silli interpreter you can follow the brief screen cast linked here to get the starter code up and running as an Eclipse project.

* <https://web.microsoftstream.com/video/a00dcd12-4505-4a00-bcd7-d0f729994e98> (2:54)
  + Note this video references a program file that is no longer part of the lab. Instead you will have a program named FirstProg.si.

*The Interpreter and ProgramStatement Classes:*

The Interpreter class is the main class of the project and controls the reading and interpretation of the programs. Instances of the ProgramStatement class are used to represent each line of the program. This section will help you to become familiar with how these classes work, some of the code that they contain and some of the data structures that they use to create the interpreter.

*The Interpreter Class:*

The main method in the Interpreter class controls the process of reading and executing the program in the file that is specified on the command line.

11. The main method calls the readProgram method. This method reads the file containing the program to be executed, processes it to make it easier to interpret and returns it in a List structure.

a. What is the data type of the program variable in this method?

b. In the while loop, each line of the program is read one-by-one. The call scr.nextLine() reads the line from the program file and returns it as a String. Two additional methods are invoked on this String. What are they and what do they do?

c. After being processed by the methods that you identified in b, the resulting rawLine is passed to what method? In what class is that method defined? What is the data type of the value that that method returns?

d. What does the code do with the value that is returned from the method you identified in c.

*The ProgramStatement Class:*

The ProgramStatement class is an *abstract class* that will be used to define a concrete sub-class for each of the different types of Silli statements (Comment, Declaration, etc…). Instances of these sub-classes are what make up the program that you saw in the Interpreter class above.

12. Use the source code for the ProgramStatement class to answer the following questions:

a. What fields does the ProgramStatement class contain?

b. What 4 accessor methods does the ProgramStatement class defined for these fields?

c. What is the abstract method that all sub-classes will need to implement?

d. For the method that you identified in part c:

i. What two parameters must provided when the method is invoked?

ii. What are their data types?

iii. What does the return value of this method indicate?

13. The ProgramStatement class is organized using the *factory design pattern*. In this design pattern an abstract class provides a static method, called the *factory method*, instead of a public constructor. The factory method accepts parameters and uses them create and return an instance of a sub-class that can represent and execute the statement. In the ProgramStatement class the factory method is the getStatement method.

a. What three types of statements are currently handled by the getStatement method?

b. What condition is used to detect declaration statements?

c. What is the name of the sub-class that is created and returned to represent and execute a declaration statement.

d. To execute a declaration statement, the variable and its value must be added to the Map (i.e. a Dictionary in Python) that is keeping track of the variables. What statement in the execute method does this?

e. What value does the execute method return? Why?

*Printing the Program:*

Returning to the main function in the Interpreter class. If the DEBUG flag is set to true, the next thing that main will do is print out the program. This was included as a debugging step to ensure that the program is read into the List correctly and that all of the ProgramStatement objects are constructed correctly.

14. In this question you will complete the printProgram method.

a. Run the interpreter using FirstProg.si as the input (java Interpreter FirstProg.si) what output is generated?

b. What is the data type of the value passed to the printProgam method?

c. Complete the printProgram method so that it will **display the lines of the program using the format indicated in the documentation for the method**. Hint: You’ll need a loop to go through each line of the program and you’ll need to use methods in the ProgramStatement class to get the line number and the label (if the line has one). Also, don’t print out the extra “filler” line that was added. **You will turn in the Java code that you write at the end of the lab, so no answers will be required for questions that ask you to add to the Java program.**

d. Run the interpreter using FirstProg.si as the input (java Interpreter FirstProg.si) to check that your printProgram is displaying the lines of the program with their line numbers. Copy and paste the output as your answer to this question.

*Interpreting a Program:*

Now that we know that the interpreter can successfully read in a program (at least one containing blank, comment and declaration statements), we can now write the bit of code that interprets those statements.

The process of interpreting a program requires only that we go to the first statement, execute it, then go to the next statement and execute it and so on. The design of the Silli interpreter ensures that every ProgramStatement object will contain an execute method that implements the execution of the statement (Note: This is an example of what is called the *Strategy* design pattern). For example, recall from question 13 that the execute method in the DECLARATION class adds a new variable to the Map. That execute function then returns the line number of the next statement in the program.

15. Examine the interpretProgram method in the Interpreter class. As provided this method will execute just the first line of the program.

a. What line of code retrieves the programStatement to be executed from the program?

b. What line of code executes the programStatement?

c. What variable keeps track of the instruction that is to be interpreted?

d. How is this variable updated?

Now, to interpret the entire program, you start with the first statement, execute it, obtain the line number of the next statement to be interpreted, and then execute that statement. You simply repeat this process over and over and over again until the line number of the next instruction is larger than the line number of last line of the program (i.e. we reach the end of the program.)

16. Complete the interpretProgram method so that it will interpret the entire program.

17. Okay, so you have the interpretProgram method implemented, how can we tell if it works? Well, after interpreting the program the main method in the Interpreter calls the printVariables method to display the values of all of the variables that have been declared in the program.

a. Examine the FirstProg.si program. What variables are declared and what are their values?

b. Run the interpreter on the FirstProg.si program (See question 14a for the command). What output is generated by the printVariables method?

c. Are the values of the variables correctly displayed? If so great! If not, revisit questions 16 and 17 to correct your implementation of the interpretProgram method.

*Output Statements:*

The next logical step in implementing the interpreter is to add output statements so that programs can print the values of variables. To do this you will need to add a sub-class of ProgramStatement that can represent and execute output statements and then add code to the factory method that recognizes them when they appear in the program.

18. Add a sub-class to the ProgramStatement class that represents and executes *output statements*. To execute an output statement you will need to obtain the value of the variable to be output from the variable Map and then print it. Your method should also display appropriate syntax error messages if the output statement is not properly formatted. Use the DECLARATION sub-class as a model.

19. Add code to the getStatement method in the ProgramStatement class that recognizes OUTPUT statements and adds them to the program. You will turn in the Java code that you write at the end of the lab, so no answer is required here.

20. Write a small Silli program named Test1.si that uses both declaration and output statements to test that your implementation works. **Save the source code for your test program in the directory with your .java files.**

21. Set the DEBUG flag to false and run your Test1.si program from question 20 with the interpreter. If the program works as expected, great! If not, revisit to questions 18-20 until it does.

*Input Statement:*

22. Add to the ProgramStatement class so that it can recognize and interpret *input statements*. The execution of these statements will require that you update the values in the variable Map. Your method should also display appropriate syntax error messages if the input statement is not properly formatted. Use the DECLARATION sub-class as a model.

You can use a Scanner object to read input from the user. See the link below for an example:

* <http://www.learningaboutelectronics.com/Articles/How-to-read-input-with-the-Scanner-class-in-Java.php>
  + Note: Declaring the Scanner as a static field in the ProgramStatement class will make things easier due to some technicalities in how Scanner works.

23. Write a small Silli program named Test2.si that reads values for at least 2 variables from the user and then prints them to test that your implementation works. **Save the source code for your test program in the directory with your .java files.**

24. Run your program from question 21 with the interpreter. If the program works as expected, great! If not, revisit to questions 23 and 24 until it does. Copy and paste the output of the program as your answer to this question.

*Arithmetic Statements:*

25. Add to the ProgramStatement class so that it can recognize and interpret *arithmetic statements*. The execution of these statements will require that you update the values in the variable Map. Your method should also display appropriate syntax error messages if the arithmetic statement is not properly formatted. Use the DECLARATION sub-class as a model.

26. Write a small Silli program named Test3.si that reads values from the user, **performs each of the arithmetic operations** at least once and then prints values to test that they work.

27. Run your Test3.si program from question 26 with the interpreter. If the program works as expected, great! If not, revisit to questions 25 and 26 until it does.

*Conditional Statements:*

Conditional statements are necessary for a full-featured interpreter that can run interesting programs. Implementing them will give you a more complete understanding of how interpreters work. *However, if you feel it has taken you too much time to reach this point, skip 28-30 and turn in the lab at this point.*

28. Add to the ProgramStatement class so that it can recognize and interpret *conditional statements*. The execution of these statements will require that you use the values in the variable Map to perform the comparison. You will also need to use the statements in the program List to determine which line to branch to if the condition holds. Your method should display appropriate syntax error messages if the conditional statement is not properly formatted. Use the DECLARATION sub-class as a model.

29. Write a small Silli program named Test4.si that reads 2 values from the user and prints out the largest value that was read.

30. Run your Test4.si program from question 29 with the interpreter. If the program works as expected, great! If not, revisit to questions 28 and 29 until it does.

Congratulations!! You have just written your first programming language interpreter. What you have built can now run the FirstProg.si program from the start of the lab. Run it and be proud of what you have accomplished!

**Submitting:**

31. Submit this lab as follows:

a. Place this completed activity sheet into the Lab07-A-Interpreter folder containing your code.

b. Compress the Lab07-A-Interpreter folder into a zip file.

c. Submit your Lab07-A-Interpreter.zip file to the L07 assignment in Moodle.

**Optional Extensions:**

All of the following are optional extensions to this lab for those who are interested in learning more and creating a more full-featured interpreter. Do none, do one, do all, do them in any order that works. Some are more challenging than others, but all are rewarding!

*Programming in Our Language:*

32. Write a Silli program named Exponent.si that reads two values (e.g. X, Y) and computes XY. Be sure to run your program in your interpreter to make sure that it works.

*Improving the Interpreter:*

Each of the following improvements are independent of the others. Thus, you may do any number of them in any order.

33. Extend the interpretation of output statement so that they can print either a variable or a string literal. For example:

PRINT “The result is: “

PRINT X

34. Extend the syntax of all types of instructions to allow variable names to have more than one character.

35. Extend the syntax of the conditional statement to allow for the additional relational operators: >=, <=, !=.

36. Extend the syntax of arithmetic statements to allow either variables or integer literals (e.g. accept X = Y + 1 or Z = 2 \* 3).

37. Extend the syntax of conditional statements to allow either variables or integer values in conditional statements (e.g. accept IF X>0 GOTO START).

38. Extend the syntax of conditional statements to allow compound conditions using AND or OR (e.g. accept IF X > Y AND Y > Z GOTO XMAX).

39. Make the processing of the statements less picky about the spacing (e.g. accept X = Y + 2 or LET X =2 as valid statements).

40. EXTRA CHALLENGING: Extend the syntax of arithmetic expressions to allow for any infix expression on the right-hand side (e.g. accept expressions like X=R\*X\*(1-X)). Hint: You’ll need to use a stack data structure. Here is a resource to get you started if you are interested:

* <https://www.geeksforgeeks.org/expression-evaluation/>

Optional: To help me improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.