Kennesaw State University

College of Computer and Software

Department of Computer Science

CS 4308 - Concepts of Programming Languages - Section W01

Project Deliverable 3

Leonard Hayes | lhayes32@students.kennesaw.edu

12/5/2019

**Initial Problem Statement**

We were given test files of the Julia Language, This subset consisted of small functions that do simple tasks. Now that the Scanner and parser are complete, our final task was to make an interpreter, which uses the expressions and statements from the parser to complete the output of the Julia functions.

**Summary and Purpose**

The purpose of this report is to describe the process of creating the Interpreter. The Interpreter takes the contents of the file and outputs based on said file.

**Description**

Several changes has to be made to the Scanner to accommodate for the Interpreter. To ensure the interpreter worked with Identifiers, an **identifiers** list was added to the Scanner to catch all of the identifiers used in the Julia file. From there they are added to a dictionary variable, where they are all initialized to 0 then changed to their corresponding numbers as the interpreter runs through the code.

The design of the Interpreter is pretty simple. Using the **statements** variable from the **decoded** variable given by the parser, we are able to take the statements from the code and look at their types. Depending on their type (if, else, etc.) we are able to interpret line by line. The types we interpret were given to use by the professor: if, else, while, assignment, and print, so the for loop we parsed is not used. If a statement includes an expression, it is done with either the **interpret\_assignment**, **interpret\_arithmetic,**  or **interpret\_boolean** static methods. All of the methods here are static so that they can be called easier with the class rather than needing to create an object for it with instance methods used for the Scanner and Parser. These methods put high focus on the inputs for these methods rather than their operators. If the inputs are numerical, then we just use the value of the Token and don’t have to do anything further. If the input is an Identifier, then it must be reference the **dIdentifiers** dictionary to find the value of that identifier and substitute it in. **interpret\_arithmetic** and **interpret\_boolean** use strings to collect this date and we use the function **eval** to evaluate these functions. The method then returns to its caller and continues cycling through statements until the end. Note: The **interpret\_boolean** is not exactly required, but I added it because it was very similar to **interpret\_arithmetic** and was necessary for the while loop to work.

Most challenges with this project stemmed from time, so I looked to outside help for the general structure. Some classmates gave me some ideas and from there I was able to build a fully functioning interpreter. I also found some issues without using static methods, which would constantly cause fragments of code working and other fragments not working. So I decided to go with another approach with static method which worked much better.

**Input Data and Results**

|  |  |  |
| --- | --- | --- |
| **Input** | **In-Code Translation** | **Result** |
| Function (Program) | <rsvp\_func> | Function id () <block> end |
| Block of code | <block> | <statement> | <statement><block> |
| If | <rsvp\_if> | If <boolean\_expression><block>else<block> end |
| while | <rsvp\_while> | While <boolean\_expression><block> end |
| statement | <statement> | <rsvp\_if> | <assignment\_statement> | <rsvp\_while> | <rsvp\_print> |
| print | <rsvp\_print> | print(<arithmetic\_expression>) |
| <boolean\_expression> | “BOOLEAN” | <arithmetic\_expression><relative\_op><arithmetic\_expression> |
| <relative\_op> | “relative” | Le\_operator | lt\_operator | ge\_operator | gt\_operator | eq\_operator | ne\_operator |
| <arithmetic\_expression> | “ARITHMETIC” | <id> | <literal\_integer> | <arithmetic\_expression><arithmetic\_op><arithmetic\_expression> |
| <arithmetic\_op> | “arithmetic” | Add\_operator | sub\_operator | mul\_operator | div\_operator |

**Syntax Error Detection**

|  |  |  |
| --- | --- | --- |
| **Error** | **Julia File** | **Screen Shot** |
| Double Space detection | function a ( )  for i = 3 : 5  if != i 4  ... |  |
| Incorrect Boolean expressions | function a ( )  for i = 3 : 5  if != i  ... |  |
| Incorrect iteration usage | function a ( )  for i = 3 : j  if i != 4 |  |
| No function detection | functio a ( )  for i = 3 : 9  if i != 4  ... |  |
| Misplaced symbols | function a ( )  for i = 3 9  if i != 4  ... |  |

**Limitations**

Some limitations with this part of the code is that it does not include a for loop. Though not required, it can be done similar to the while loop but would require the use of both the **interpret\_boolean** and **interpret\_assignment** to watch the boolean and keep up the iteration. Another limitation is that it does not recognize local and global variables. In this case it didn’t seem necessary given the test files but was still something that could be implemented at another time.

**Test Inputs and Results**

|  |  |
| --- | --- |
| **Input** | **Result** |
|  |  |
|  |  |
|  |  |

**Conclusion**

After completing these three projects. I feel like I completely understand the process of the compiler. To scan the code, to parse it into readable statements and expressions, and using those to find the solution to the file. This compiler is not complete in the slightest but is a great start and key to understanding more about coding as a whole. Though I had a lot of help along the way structure wise, working on the details by myself was very interesting to say the least. Understanding how the code works and how to implement parts of code that will be shared across the entire compiler was something that took a lot of work and effort. This proved very helpful with the creation of the interpreter.

**References**

Ruslan Spivac. “Let’s Build a Simple Interpreter” Ruslan’s Blog,

https://ruslanspivak.com/lsbasi-part1/. Accessed December 6, 2019

Shivam\_k. “Python | Add new keys to a dictionary” Geeks for Geeks

https://www.geeksforgeeks.org/python-add-new-keys-to-a-dictionary/ Accessed

December 6, 2019

Shubham. “Python Static Method: Journal Dev,

https://www.journaldev.com/18722/python-static-method. Accessed December 6, 2019

Nikhil Kumar. “Eval in Python” Geeks for Geeks,

https://www.geeksforgeeks.org/eval-in-python/. Accessed December 6, 2019