

**Computer Science Department**

**CS4308 Concepts of Programming Languages Section 01**

**Project - 3rd Deliverable - Interpreter - Submission 1/1**

**Souleymane Camara (scmara3@students.kennesaw.edu)**

**April 30, 2021**

**Initial Problem Statement**

The assigned task for this class (CS 4308) was to develop an Interpreter or a Compiler which translates or processes a language for the subset of the Basic Programming Language. This project could be done either in Python, C, C++, or Java. Then, write a short report describing the work done, and to provide the source program of the Interpreter/Compiler, input and output files.

**Summary and Purpose of the Assignment**

As the lecturer mentioned earlier in the semester, the purpose of this course, most specifically this project is to train students how to think as a designer of the programming languages instead of thinking as a user. In order to achieve this goal, we were given an assignment where to apply the concepts and fundamentals we have learned so far from the course in order to gain hands-on experience about the overall process involve in language processing.

**Detailed Description of the Solution**

In order to get the job done, we (team’s members) choose Java as the language of implementation of the language processor because of its conveniency and also most of us were knowledgeable at it. We used Visual Studio (Visual Basic Program) and NetBeans (Java Program for Interpreter) as tools or IDEs to write interpreter’s input program and the interpreter itself respectively.

**The Three Important Steps to Develop an Interpreter**

1. The scanner for the language processor. It accepts a file path given in a parameter, then the scanner takes in the text of a Basic program or raw source code as a series of characters and groups it into a series of chunks we call **tokens**. These are the meaningful “words” and “punctuation” that make up the language’s grammar. Finally, the scanner grouped each string into its respective token type according to token’s lexeme and others.
2. The next step is parsing, this takes place immediately after the scanning. During this phase, the parser begins checking each line for errors. As it is checking, the parser is also determining what type of statement is on each single line. Then, each token is split up, stored, checked for errors, and well organized in its line into a statement with a defined function.
3. The last step is the easiest step because the interpreter just has to go through each Basic statement, perform the equivalent Java code, and process the output.

Compare to the other two first steps that are, scanning and parsing, the interpreter is designed to manage less tasks because of the way the project was structured and how we defined the requirements from previous deliverables.

The diagram below shows a visualization of the flow of data during the entire process.

Basic file → String → List of Strings → List of Tokens → List of Statements → Output String

**List of Input Data and Results**

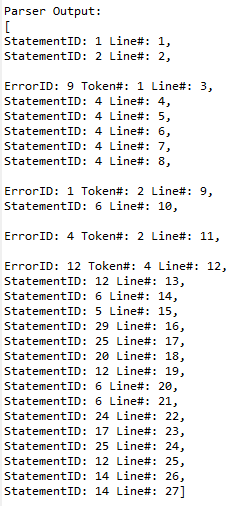
The input data or Basic program file used in this report was collected from the sample program provided for this project. Statements that are outside the interpreter’s scope are simply ignored and do not affect the interpreter’s output. Below is a table of certain lines of code from the input file and their interpretations.

|  |  |  |  |
| --- | --- | --- | --- |
| Original line of Basic code | Statement ID (given by parser) | Affect on memory/output | Translated code |
| Dim n as Integer | 4 (Assignment Statement) | n is declared as a new name of variable, variable at this index is initialized with a value of zero | varNames.add(curr.tokens.get(1).lexeme);  varValues.add(0); |
| n = 2 | 6 (Assignment w/ Integer Literal) | Value of variable with index given by the variable name on the left hand side is changed to match the value of the integer literal | recipientIndex = varNames.indexOf(curr.tokens.get(0).lexeme);  rightValue = Integer.parseInt(curr.tokens.get(2).lexeme);  varValues.set(recipientIndex, rightValue); |
| s = s + n | 17 (Addition reassignment w/ two variables) | Value of variable with index given by the variable name on the left hand side is changed to match the value of the sum of the two variables | recipientIndex = varNames.indexOf(curr.tokens.get(0).lexeme);  leftIndex = varNames.indexOf(curr.tokens.get(2).lexeme);  leftValue = varValues.get(leftIndex);  rightIndex = varNames.indexOf(curr.tokens.get(4).lexeme);  rightValue = varValues.get(rightIndex);  varValues.set(recipientIndex, leftValue + rightValue); |
| Console.WriteLine(n) | 12 (Console call w/ WriteLine) | Value of variable given by variable name inside the WriteLine call is concatenated with the output String | recipientIndex = varNames.indexOf(curr.tokens.get(4).lexeme);  output += varValues.get(recipientIndex) + "\n"; |

**Error Examples**

The table below shows examples of syntax errors caught by the parser.

|  |  |  |  |
| --- | --- | --- | --- |
| Original line of code | Line with added error | Parser Output | Error info on lookup table (shortened to fit in the table) |
| Dim n as Integer | Dim n as Int | ErrorID: 9 Token#: 1 Line#: 3 | 9: Missing expected syntax in assignment statement |
| n = 2 | n = + | ErrorID: 1 Token#: 2 Line#: 9 | 1: Missing expected syntax in reassignment statement using variable/literal |
| s = s + n | s = s + Integer | ErrorID: 4 Token#: 2 Line#: 11 | 4: Missing expected syntax in reassignment statement using addition |
| Console.WriteLine(n) | Console.WriteLine(n | ErrorID: 12 Token#: 4 Line#: 12 | 12: Missing expected syntax in Sub header |



**Limitations and Design**

Despite the fact that the project almost does what was asked to, but it still needs some improvement as it is in the world of the new technology. There are always things to be done. By starting by the code organization, it is far from being the best practice in coding because it is always good to make your language more readable and reliable in order to attract more customers, and these two aspects are definitely linked to the overall performance of the language processor including the cost, of course. The time is also among the factors which did not permit us to deliver an outstanding work by making use of pertinent abstract data structures that would be suitable and make the language more efficient.

**Sample Program Output**

The output shown below is copied from the console output window when using the code with the Input file. A full screenshot of the output can be found in the submission.

**Scanner Output:**

[Module(KEYWORD), BasicProgram(MODULE\_IDENTIFIER),

(

NEW\_LINE), Sub(KEYWORD), Main(SUB\_IDENTIFIER), ((LEFT\_PARENTHESES), )(RIGHT\_PARENTHESES),

(

NEW\_LINE), Dim(KEYWORD), n(INTEGER\_IDENTIFIER), as(KEYWORD), Integer(KEYWORD),

(

NEW\_LINE), Dim(KEYWORD), s(INTEGER\_IDENTIFIER), as(KEYWORD), Integer(KEYWORD),

(

NEW\_LINE), Dim(KEYWORD), a(INTEGER\_IDENTIFIER), as(KEYWORD), Integer(KEYWORD),

(

NEW\_LINE), Dim(KEYWORD), b(INTEGER\_IDENTIFIER), as(KEYWORD), Integer(KEYWORD),

(

NEW\_LINE), Dim(KEYWORD), x(INTEGER\_IDENTIFIER), as(KEYWORD), Integer(KEYWORD),

(

NEW\_LINE), Dim(KEYWORD), p(INTEGER\_IDENTIFIER), as(KEYWORD), Integer(KEYWORD),

(

NEW\_LINE), n(IDENTIFIER), =(EQUALS), 2(INTEGER\_CONSTANT),

(

NEW\_LINE), s(IDENTIFIER), =(EQUALS), 0(INTEGER\_CONSTANT),

(

NEW\_LINE), s(IDENTIFIER), =(EQUALS), s(IDENTIFIER), +(PLUS), n(IDENTIFIER),

(

NEW\_LINE), Console(KEYWORD), .(DOT), WriteLine(KEYWORD), ((LEFT\_PARENTHESES), n(IDENTIFIER), )(RIGHT\_PARENTHESES),

(

NEW\_LINE), Console(KEYWORD), .(DOT), WriteLine(KEYWORD), ((LEFT\_PARENTHESES), s(IDENTIFIER), )(RIGHT\_PARENTHESES),

(

NEW\_LINE), a(IDENTIFIER), =(EQUALS), 5(INTEGER\_CONSTANT),

(

NEW\_LINE), b(IDENTIFIER), =(EQUALS), a(IDENTIFIER),

(

NEW\_LINE), b(IDENTIFIER), =(EQUALS), b(IDENTIFIER), /(SLASH), 2(INTEGER\_CONSTANT),

(

NEW\_LINE), b(IDENTIFIER), =(EQUALS), b(IDENTIFIER), \*(STAR), 2(INTEGER\_CONSTANT),

(

NEW\_LINE), b(IDENTIFIER), =(EQUALS), a(IDENTIFIER), -(MINUS), b(IDENTIFIER),

(

NEW\_LINE), Console(KEYWORD), .(DOT), WriteLine(KEYWORD), ((LEFT\_PARENTHESES), b(IDENTIFIER), )(RIGHT\_PARENTHESES),

(

NEW\_LINE), x(IDENTIFIER), =(EQUALS), 0(INTEGER\_CONSTANT),

(

NEW\_LINE), p(IDENTIFIER), =(EQUALS), 0(INTEGER\_CONSTANT),

(

NEW\_LINE), x(IDENTIFIER), =(EQUALS), b(IDENTIFIER), \*(STAR), p(IDENTIFIER),

(

NEW\_LINE), x(IDENTIFIER), =(EQUALS), x(IDENTIFIER), +(PLUS), x(IDENTIFIER),

(

NEW\_LINE), p(IDENTIFIER), =(EQUALS), p(IDENTIFIER), \*(STAR), 10(INTEGER\_CONSTANT),

(

NEW\_LINE), Console(KEYWORD), .(DOT), WriteLine(KEYWORD), ((LEFT\_PARENTHESES), p(IDENTIFIER), )(RIGHT\_PARENTHESES),

(

NEW\_LINE), End(KEYWORD), Main(IDENTIFIER),

(

NEW\_LINE), End(KEYWORD), Module(KEYWORD)]

Parser Output:

[

StatementID: 1 Line#: 1,

StatementID: 2 Line#: 2,

StatementID: 4 Line#: 3,

StatementID: 4 Line#: 4,

StatementID: 4 Line#: 5,

StatementID: 4 Line#: 6,

StatementID: 4 Line#: 7,

StatementID: 4 Line#: 8,

StatementID: 6 Line#: 9,

StatementID: 6 Line#: 10,

StatementID: 17 Line#: 11,

StatementID: 12 Line#: 12,

StatementID: 12 Line#: 13,

StatementID: 6 Line#: 14,

StatementID: 5 Line#: 15,

StatementID: 29 Line#: 16,

StatementID: 25 Line#: 17,

StatementID: 20 Line#: 18,

StatementID: 12 Line#: 19,

StatementID: 6 Line#: 20,

StatementID: 6 Line#: 21,

StatementID: 24 Line#: 22,

StatementID: 17 Line#: 23,

StatementID: 25 Line#: 24,

StatementID: 12 Line#: 25,

StatementID: 14 Line#: 26,

StatementID: 14 Line#: 27]

Interpreter Output:

2

2

1

0

**Conclusion**

Frankly speaking, this project was among the most difficult assessments in Computer Science I ever worked on due to its complexities, ambiguous requirements definitions, and mostly having hard time to choose among plenty available options that was appropriate and best for the project. But at the same time, I felt challenge overwhelmed, and worried by the project which therefore allow me to get closed to my teammates and people who have experienced in the field in order to learn more and acquire experience to complete project.

In sum, I found the project relevant, interesting, career-oriented assessment.

**References or Works Cited (APA citation)**

Sebesta, R. W. (2016). Concepts of Programming Languages (11th ed.). Pearson Education

Limited.

Sebesta, R. W. (2016). Concepts of Programming Languages (12th ed.). Pearson Education

Limited.