Kennesaw State University

College of Computing & Software Engineering

Department of Computer Science

CS4308 Concepts of Programming Languages

CS4308 Course Project

James Bozhkov, Asa Marshall, Funayan Ojiagbaje

zbozhkov@students.kennesaw.edu, amarsh65@students.kennesaw.edu, cojiagb3@students.kennesaw.edu

Initial Problem: The goal of this project is to develop an interpreter for the SCL language. The interpreter consists of a scanner, parser, and executer file. The interpreter will process an SCL program. The parsing algorithm will detect any syntactic or semantic error. The first error discovered decree accounts an appropriate error message to be printed, and then the interpreter would terminate. Run-time errors will also be detected with appropriate error messages printed.

Summary: The scanner program handles a keyword table, which then perceives it towards the parser. The parser handles the identifier table, that includes several attributes for every identifier. Constants are also stored in the identifier table. These two functions return the index value of the entry in the table, that prelates onto the executor. The executer outputs the expected results obtained from both the scanner and parser. The interpreter abides by the scanner, parser and executer. The interpreter will recognize each statement, check for errors and execute the corresponding statement. A parsing algorithm should detect any syntactic or semantic errors obtained from executing the interpreter.

Detailed Description:

Scanner: The scanner essentially reads the source code given from the input file and separates each word or number into a token. Upon scanning each token, the scanner recognizes each lexeme and determines the token type. Our interpreter stores information about each token type in the Constants class by declaring a series of final variables set to a token ID. The scanner also holds a set of keywords, which are used as reserved words with specific functions. If a keyword is used in the source code, the scanner assumes that the keyword is intending to use its preset functionality, meaning that a programmer cannot use a keyword for another purpose like a variable name. The scanner uses a series of switch cases to find the type of each token. When scanning, the Scanner class first checks for an end-of-file value to determine if scanning is necessary. Following this, the scanner compares the current token to the preset keywords. If a keyword or symbol is not found, the scanner will recognize the token as an identifier. By scanning and tokenizing each word in the source code, the scanner delivers the parser a stream of tokens and associated token types.

Parser:

The parser consists of a declaration for the lexical rules for the subset chosen for our subset of SCL language. This Class contains an array of lexical ID numbers which are used conceptually as a dictionary of dictionaries, where each internal array key holds an array of lexical id numbers to test against input to verify whether each statement is abided by takeoff, itself to keep s input from the Scanner, the lexical rules for our SCL grammar. The parser essentially gets fed input from the Scanner class in the form of tokens and defines a list used to store identifiers as they are created. This list also serves as memory during execution time. The parser class utilizes various methods which, based on the tokens inputted, create a complete statement to be executed as well as detect syntactical errors from the input file. The various methods all coincide with one another

to determine if input data may be derived from the start symbol of the SCL grammar. Based on this, the parser class will call on other functions which aid in constructing the execution statements for the executor. The memory in the SCL execution is stored in an Arraylist called IDtable, which holds identifier objects and their ID's for execution during execution time. The Parser interprets each token, and based on the token type, it differentiates each lexical rule to verify whether each word exists within each rule. If there are no matched words, or no concede, the system inclines to throw an error. To execute for loops, the parser class creates new instances of itself to monitor the number of iterations made. As the parser retrieves tokens from the scanner, based on the token read, the parser determines what token types inputted in next will make for a valid statement, if not then there will be an error. The parser parses input files until the end of the file has been reached.

Interpreter: The interpreter consists of multiple helper functions within the program for successfully executing each of the correlated input files, including AssignmentStatement and Constants class. The interpreter constitutes Scanner, Parser, and Executor classes. The identifier class administers the structure for each sub-class corresponding to StringIdentifier, and IntegerIdentifier file(s). The listed classes contribute a dictionary for the scanner to determine the token types converted from the parser class. A file name is transpired as an argument in the first method of the interpreter. Multiple switch statements are insinuated as the parse program parses one complete serve each time. The scanner executes before the end of the file is attained. Next token is reconstructed onto a string type from the convertType function. A new parser is created from the current parser in the third case block. The methods that parse a specific command, located in the input file, are further executable. The corresponding memory of the SCL subset execution is stored in a list that holds a corresponding set of identifier objects. The objects give the correlated values the ability to be stored onto a set of identifier names. Each action which is adequate to be implemented in the SCL subset is competent for materializing through a Java method call. The integrated while loop concludes the remaining parsed functions. The subsequent scheme prints the executed file, preceding the acquired end of file statement. The tester function inserted at the end of the interpreter prints out the analogous IdentifierTable gathered from the parser.

Results:

Prompt	**Input**	**Output**
Please enter your name:	Michael Jordan	
Enter a value for integer x:	6	Your chosen value of x is: 6

How many times should loop run?	1	
Would you like to add 2 to x, or multiply x by 2 2 times? Enter 0 to use addition, enter a non-zero integer to use multiplication:	0	X is: 6 X is: 8 Final value of x is: 8 Thank you Michael Jordan

Conclusion: When collaborating with each team member, every individual had a different approach to developing an interpreter. After brainstorming the hypothesis, each team member came to the same desistance. When implementing the interpreter, the parser and scanner must compile and execute properly with the correlated output. The scanner handles the keyword table, whereas the parser grasps the identifier table. The interpreter coincides elements defined in the grammar for the subset language. The interpreter directly executes instructions that are written in a programming language, without requiring to be translated into a machine learning program previously.

References:

"Compiler Design - Types of Parsing." *Tutorialspoint*, Tutorialspoint, 2020, www.tutorialspoint.com/compiler_design/compiler_design_types_of_parsing.htm.

Dixit, Netra. "Java Scanner : Basic Implementation of the Scanner Class." *Udemy Blog*, Udemy, 20 May 2014, blog.udemy.com/java-scanner/.

Mcmanis, Chuck. "Build an Interpreter in Java -- Implement the Execution Engine." *JavaWorld*, JavaWorld, 1 July 1997, www.javaworld.com/article/2076974/build-an-interpreter-in-java----implement-the-execution-engine.html.

Northwood, Chris. "Computer Science Notes ⇒ Lexical and Syntax Analysis of Programming Languages." *Lexical and Syntax Analysis of Programming Languages*, Pling Org, 2012, www.pling.org.uk/cs/lsa.html.

Sebesta, Robert W. *Concepts of Programming Languages*. 12th ed., E, Pearson, 2019.

Spivak, Ruslan. "Let's Build A Simple Interpreter. Part 1." *Ruslan's Blog*, 15 June 2015, ruslanspivak.com/lsbasi-part1/.

Tomassetti, Gabriele. “Parsing in Python: All the Tools and Libraries You Can Use.” *Federico Tomassetti - Software Architect*, Strumenta Websites, 19 Sept. 2019, tomassetti.me/parsing-in-python/.