CS 4308 – Concepts of Programming Languages

Course Project Deliverables

The CS4308 project consists of the development of an interpreter for a subset of the SCL language. The interpreter can be implemented in any of the following programming languages: **C , C + + , C # , Java, Python, or Ada**.

This project consists of developing a complete language processor (interpreter) for a **subset** of the **SCL language**.

The language processor will process an SCL program. All tokens in this language are separated by white spaces. The parsing algorithm should detect any syntactical error. Each error discovered should cause an appropriate error message to be printed. Run-time errors should also be detected with appropriate error messages being printed.

You are required to apply a complete software development process. In your implementation, the source code (of the scanner, parser, and complete interpreter):

- The source code must be well-structured and be easy to understand, comments should help in clarifying your implementation.
- Do not 'hard-code' input data in your source programs, use appropriate input statements. Otherwise it beats the purpose of software development.

In your submission, do not include compiled files and/or IDE project files.

Your complete submission must include: a well written report (see 'submission report.pdf'), the subset of the SCL grammar used (in BNF or EBNF notation), source code files of the implementation, input SCL program file used to test your project. Submit all your files in a single archive.

Your report must document the work done. Include explanation of how to run your program, the input and the output produced when running your program.

Deliverables (*see course Module's schedule for due dates*):

1. Module_3 – 1st Deliverable

Create an interpreter for a subset of SCL language (*is an experimental system programming language that can be used as: http://ksuweb.kennesaw.edu/~jgarrido/sysplm/*). The scanner implementation must include an array of the keywords used in the subset of SCL or other selected programming Language, an array (or list) of the identifiers (variables), and other tokens *(such as operators, keywords, constants, and/or special characters.)*. Another word, you need to implement a scanner that reads source code and create a list of tokens, including to define the grammar of the subset of SCL that you are using in BNF/EBNF. *I encourage you to use Python.*

The program works best by taking command line input for a file and generating a JSON file with a list of tokens, as well as printing those tokens to the console. Example, the following command will run the program on the given source SCL file:

python scl_scanner.py hello_world.scl

You must submit a short report describing the work performed.

- You must also include the grammar of the subset of SCL, source code files of the scanner program, the input and output files.
- The report must show the execution of this scanner program by using appropriate input files, the program must show a list of the tokens scanned.

2. Module_5 – 2nd Deliverable

Develop a complete parser for the subset of the SCL language. This parser program must execute with the scanner. The report must show the execution of this parser program by using one or more relevant input files, the program must show the corresponding statements recognized. The report must describe the work performed. Include the parser source program, input, and output files.

On the 1st deliverable report included a description of this subset, which has been slightly modified since then to more accurately demonstrate how the parser works. A parser may be initialized with an input file string. Upon initialization, the parser initializes the scanner with the same input file, which is necessary for returning the following tokens and lexemes in the input file.

The purpose of this 2nd deliverable is to demonstrate understanding of the parsing process of compilation through creation of a parser which is based off a subset of SCL written in BNF. *Note: you can write your own SCL.*

Detailed Description Of Solution

The code for both the scanner (Scanner.cpp) and the parser (Parser.cpp) are included needs to be included in this report where the compilation steps are based off a subset of SCL.

The previous project deliverable report included a description of this subset, which has been slightly modified since then to more accurately demonstrate how the parser works. A parser may be initialized with an input file string. Upon initialization, the parser initializes the scanner with the same input file, which is necessary for returning the following tokens and lexemes in the input file.

Example, there are three (3) public functions in this class, *getNextToken(), identifierExists(string identifier)*, and *begin()*.

I. The first function, *getNextToken()*, that will return the next token that is not a comment. This function allows comments to be fully ignored by the parser.
II. The second function, *identifierExists(string identifier)*, returns true if an identifier has already been declared. This is useful for two reasons: making sure when we are defining variables that identifiers are not declared multiple times and making sure when we are calling an action that the variable in question has been defined.
III. The final public function, *begin()*, calls upon the private *start()* function, which is the first nonterminal in the grammar subset. Every function following *start()* in the parser is based on the format of the grammar subset, however in some instances easier methods were found for achieving the same outcome and the grammar subset was changed to better represent what is actually happening in the code. These functions are all private as they are only useful if we begin at *start()*, and should never be called outside of the class.

Remember, your report should have the code for both the scanner (Scanner.cpp) and the parser (Parser.cpp) in the file/report is found Your output may look like below.

3. Module_3rd Deliverable

In the past two deliverables you created a scanner for a subset of the SCL language which feature an array of keywords used in the subset of SCL and to show understanding of the grammar of the subset of SCL.

The 3rd deliverable you need to develop a complete interpreter or a translator to intermediate code and an abstract machine that includes the scanner, parser, and executer. The report must show the execution of this interpreter program by using one or more input files, the program must show the results of executing every statement

recognized by the parser using the programming language Ada, Python, or language that you did use in other two deliverables.

Here break down the code into lexemes and identify what the lexeme was (keyword, string literals, real constants, and even integer constants). The parse could be implemented using one of the programming languages. This implementation employed Java.

There are four (4) classes comprising the lexical analyzer, eleven classes for the parser, and eleven classes for the interpreter. The scanner would also identify the token code appropriate for the lexeme in question, and the scanner will even find the next token code for the parser to use. While we recognize lexemes at this stage, it is also a great opportunity to filter out certain parts. These parts include commented out lines of code and sections.

In another word, an interpreter has three fundamental components being a program, statement, and expression class. The program class defines two principal methods, load and run. The load method parses the SCL program into a series of statements. The statements are defined in the Statement class and can be thought of as a sentence in SCL. *For example*, a statement would be "set x = 45.95". The various statements in the SCL program are added to an ArrayList during the parsing phase.

Therefore, develop a complete interpreter or a translator to intermediate code and an abstract machine. that includes the scanner, parser, and executer. ***You need to use Python, Ada, or language that you haven't used in other two deliverable for this deliverable.***

Detailed Description Of Solution

You can program the interpreter in Python. This last part, the executer, takes input in the form of parse trees from the parser, which are made from scanned tokens from the scanner. The parse tree contains a hierarchical list of the functions, variables, and actions in the program, so the executer can step through this parse tree to run the program. A Python dictionary can be created to represent the program's memory.

First, at the function level, a key in the memory dictionary is created for every variable definition. Then, the function's actions are iterated through, with different action types being handled in different ways. Display actions are printed to the console and report log, set actions assign the value of an expression to a key in memory, input actions use Python's input functionality to prompt the user and assign their input value to a variable

in memory, exit/return actions stop the program execution, and if statements have their nested actions executed if their condition is evaluated to be true.

You can develop several helper functions that the executer uses, all for evaluating different levels of the parse tree. There can be an evaluation function for expressions, terms, and conditions. You can also separate actions execution into its own function so that it can be called recursively for if statement blocks. Lastly, you have a utility function that allows for printing to the console and writing to a file at the same time.

The **repor**t must show the execution of this interpreter program describing the work performed. Include the source code of the program, by using one or more input files. The results of program executing every statement recognized by the parser.

Please write a report that describing the work performed. Include the source code of the program, input, and output files.

Deliverable:

1. Please submit ONE submission per team for each Deliverable.
2. Please make sure to have all the team members Full name in each submission.
3. Submit a short video .mp4 and all the necessary documents/report (*Refer to course and/or attached resources for report format)* and source codes/Screenshot within a Compressed .Zip file on or before due date via d2l Assignment dropbox.