Kennesaw State University

College of Computing and Software Engineering

Department of Computer Science

CS 4308-W02: Concepts of Programming Languages

Deliverable 1 - Scanner

Jesse Schultheis [redacted]

Katlin Scott [redacted]

William Stigall [redacted]

Hailey Walker [redacted]

September 19, 2023

Initial Problem Statement

The problem is to create a Python-based interpreter for a subset of the SCL language, involving the development of a scanner to parse source code into tokens (including keywords, identifiers, operators, constants, and special characters), specifying the grammar using BNF/EBNF, and providing both console output and a JSON file as output when processing a designated SCL source file via a command-line input.

Summary/Purpose

The purpose of this deliverable is to implement a Scanner file that reads the source code of a given SCL file and then creates a list of tokens in another file.

Detailed Description/Work Done

Python code defines the keywords ("DISPLAY", "IF", "THEN", "ENDIF", "FUNCTION", "IS", "ENDFUN", "PARAMETERS", "INTEGER", "FLOAT", "CHAR", "NOT") and token types (“IDENTIFIER”, “UNSIGNICON”, “SIGNICON”, “PLUS”, “MINUS”, “STAR”, “DIVOP”, “EQUOP”, “RELOP”, “LB”, “RB”, “LP”, “RP”, “COMMA”, “OF”), takes in six SCL files and reads them. It then tokenizes all of the files and puts each token with its correct token type in the output.json file.

Input Files

arduino_ex1.scl

arrayex1b.scl

bitops1.scl → Used in the screenshots in this report

datablistp.scl

linkedg.scl

welcome.scl

Output Files

output.json

tokens.json → Used in the screenshots in this report

Grammar

<program> ::= <statements>

<statements> ::= <statement> | <statement> <statements>

<statement> ::= <var_declaration> | <expression> | <print_statement> | <if_statement> | <function_declaration> | <function_call>

<var_declaration> ::= IDENTIFIER EQUOP <expression> | IDENTIFIER <array_def>

<array_def> ::= LB <expression> RB

<expression> ::= <term> | <term> PLUS <term> | <term> MINUS <term> | <term> STAR <term> | <term> DIVOP <term>

<term> ::= IDENTIFIER | UNSIGNICON | SIGNICON | <expression> | <function_call>

<print_statement> ::= DISPLAY <expression_list>

<expression_list> ::= <expression> | <expression> COMMA <expression_list>

<if_statement> ::= IF <condition> THEN <statements> ENDIF

<condition> ::= <expression> RELOP <expression> | NOT <condition>

<function_declaration> ::= FUNCTION IDENTIFIER parameters IS <statements> ENDFUN IDENTIFIER

<parameters> ::= PARAMETERS param_list | ε

<param_list> ::= IDENTIFIER OF <data_type> | IDENTIFIER OF <data_type> COMMA <param_list>

<data_type> ::= INTEGER | FLOAT | CHAR

<function_call> ::= IDENTIFIER parguments

<parguments> ::= LP <expression_list> RP

Limitations of the above specification and design of the system

For our scanner, we only used a subset of the grammar rather than the entire grammar because we were given that option. So, because of that, there will be limitations on that end.

Discussion of how the solution can be improved and extended

One way the solution could be improved would be to include all the grammar instead of just the subset. However, since we were given the option, we only did a subset to save us some time on that part.

Source Code Screenshots

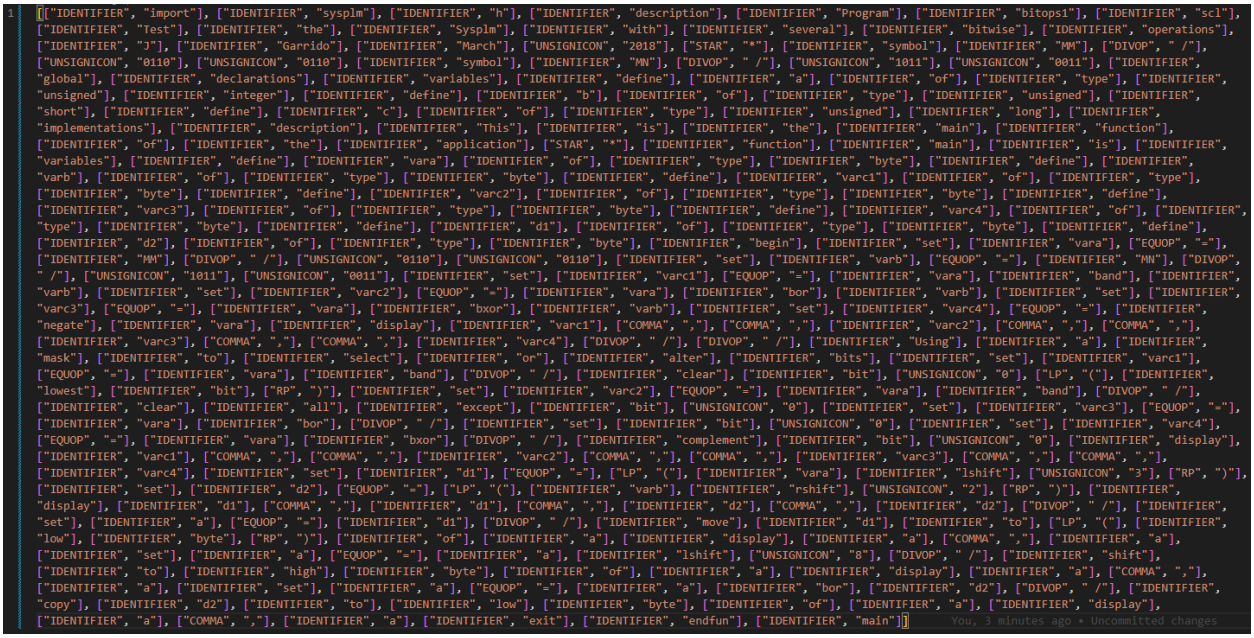
scl_scanner.py Source Code:

```python
import re
import json

#Define Keywords and token types
KEYWORDS = ["DISPLAY", "IF", "THEN", "ENDIF", "FUNCTION", "IS", "ENDFUN", "PARAMETERS", "INTEGER", "FLOAT", "CHAR", "NOT"]
TOKEN_TYPES = [
    ("IDENTIFIER", r"\b[A-Za-z_][A-Za-z0-9_]*\b"),          #Alphanumeric Identifier
    ("UNSIGNICON", r"\b\d+\b"),                             #Unsigned integers
    ("SIGNICON", r"[-+]?\b\d+\b"),                          #Signed integers
    ("PLUS", r"\+"),                                        #Addition Operator
    ("MINUS", r"\-"),                                       #Subtraction Operator
    ("STAR", r"\*"),                                        #Multiplication Operator
    ("DIVOP", r"[^(//)\"][0-9A-Za-z]*/[0-9A-Za-z]*"),       #Division Operator
    ("EQUOP", r"="),                                        #Assignment Operator
    ("RELOP", r"(==|!=|<=|>=|<|>)"),                        #Relational Operators
    ("LB", r"\["),                                          #Left Bracket
    ("RB", r"\]"),                                          #Right Bracket
    ("LP", r"\("),                                          #Left Parenthesis
    ("RP", r"\)"),                                          #Right Parenthesis
    ("COMMA", r","),                                        #Comma
    ("OF", r"OF")                                           #Of Keyword
]

#Function to tokenize the source code
def tokenize(source_code):
    tokens = [] #list to store tokens
    position = 0 #Initialize Position in source code
    #loop through the source code for tokenization
    while position < len(source_code):
        match = None
        #loop through each token type and its corresponding regex pattern
        for token_type, pattern in TOKEN_TYPES:
            regex = re.compile(pattern)
            match = regex.match(source_code, position)
            #if match append to tokens list
            if match:
                token_value = match.group(0)
                if token_type == "IDENTIFIER" and token_value in KEYWORDS:
                    tokens.append((token_value, token_value))
                else:
                    tokens.append((token_type, token_value))
                position = match.end()
                break
        #if no match is found increment position
        if not match:
            position += 1

    return tokens #returns the list of tokens
```

```python

#Main function to execute the program
def main():
    import sys
    if len(sys.argv) != 2: #check for the correct number of command line arguments
        print("Usage: python scl_scanner.py <filename>")
        return
    #read source code from file
    with open(sys.argv[1], 'r') as f:
        source_code = f.read()
    #tokenize source code
    tokens = tokenize(source_code)
    #print each token
    for token in tokens:
        print(token)
    #Save tokens to JSON
    with open("tokens.json", "w") as outfile:
        json.dump(tokens, outfile)

if __name__ == "__main__":
    main()

```

Source Code Execution Screenshot

tokens.json File Source Code (is also printed in the Terminal):



Comments and Conclusion

This program successfully tokenizes each SCL file and uses the type definitions to correctly categorize each token. They were all stored in the tokens.json, and the categorized tokens were stored in outputs.json. We learned about the basic inner workings of a scanner, and we met every requirement of Deliverable 1. In future deliverables, we will continue our education on how scanners work and develop a fully functional one.

References

Sebesta, R. W. (2012). *Concepts of Programming Languages* (10th ed.). Pearson.