Preston Phan

Luan Nguyen

Calvin Nguyen

CPSC 323 – Section 2

CPSC 323 Project 3 Documentation

1. Problem Statement

For project 3, we created a top-down predictive recursive descent parser. The parser will generate an instruction table that will print the assembly operations, and addresses. Then the symbol table will print the identifiers, memory-locations, and identifier types.

2. How to use the program

The program can be run in Windows 10 by using the terminal, navigate to the directory containing all the files. Then type in “py syntax2.py” or “python syntax2.py” to compile the code. The output.txt file will contain all the assembly code instruction and symbol table.

3. Design of your program

The design of the program starts off by importing the lexical analyzer code onto the project. The lexical analyzer from project 1 will help us identify the tokens and lexemes to be analyze. Then we declare an empty stack to store memory address.

The lexemes, token, and memory address stack list will be passed into our helper function **statement(),** which includes two other helper functions **declarative()** and **assignment().** The **declarative()** helper function is used to store the declarative statements and assigning the memory address into the memory stack list and assembly instruction into the dirstack. After every iteration, it will check whether the top of the stack is a PUSHI or POPM action and process accordingly. Next the **assignment()** helper function will be used to determine the assembly

instructions to be used for the code from the input text. The assignment function will start by populating the uniquelist and uniquevalues list with elements from the memory address stack list. Next the table list will be populated by using the append() function, loading the elements for the table one by one. Then the function will grab the indexes of where the assignment (=) operator is and identify the operations. After all these steps, the directionfunction() helper function will be passed with the 3 parameters, table, tokens, and values. The function will perform a for loop to traverse through the token list being passed through. Identifying the tokens and inserting and appending these tokens and values to their respective list. Once the function has completely pass through the token list. It will print out the list from the rulestack onto the output text file to complete the compilation.

4. Any Limitation

A minor limitation for the code is that the lexemes of the text file must be spaced out with at least one blank space. For example, the semicolon at the end of the statement must be separated from the lexeme.

5. Any Shortcomings

The program is currently only able to parse through addition operations.

Code Example:

```
int num nu2m nu2m sum;
num = 10 ;
nu2m = 15 ;
su2m = 0 ;
sum = 20 ;
sum = nu2m + num ;
```

Address	**Op**	**Oprnd**
1	PUSHI	10
2	POPM	2000
3	PUSHI	15
4	POPM	2001
5	PUSHI	0
6	POPM	2002
7	PUSHI	20
8	POPM	2003
9	PUSHI	2000
10	PUSHI	2001
11	ADD	nil
12	POPM	2003