Kennesaw State University

College of Computing and Software Engineering

Department of Computer Science

CS 4308 Concepts of Programming Languages - Section W01

3nd Deliverable - CPL Project

Bryson Phillip bphill91@students.kennesaw.edu

4/25/2022

**Problem Statement:**

This writeup details the third stage of the CPL project where the full interpreter is initialized. Interpreting is the final stage of processing occurring after the lexical analyzer has identified the tokens within a file and the parser has traced the parse tree. The full process of interpreting involves translating program statements into their corresponding operations within the CPU. Formal operational or denotational semantics are not included in this writeup, but the basic ideas still hold. Simulated RAM will be generated during the parsing process and the outcome of various state changes within the code will be sequentially applied until the program finishes.

**Detailed Description:**

**EBNF Grammar:**

<procedure> → procedure <title> is <varlist> begin <statementlist> end <title> ;

<title> → (a b c … z A B C … Z 0 1 2 3 4 5 6 7 8 9) {(a b c … z A B C … Z 0 1 2 3 4 5 6 7 8 9)}

<varlist> → <declaration> <varlist>

| <initialization> <varlist>

| ε

<declaration> → <title> : Integer;

<initialization> → <title> := <value>;

<value> → (0 1 2 3 4 5 6 7 8 9) {(0 1 2 3 4 5 6 7 8 9)}

<statementlist> -> <varlist> <statementlist>

|<conditional> <statementlist>

| ε

<conditional> → if <condition> then <statementlist> {elsif <condition> then <statementlist>} [else <statementlist>] end if;

<condition> → <conditionB> {or <conditionB>}

<conditionB> → <comparison> {and <comparison>}

<comparison> → (<condition>)

| !(<condition>)

| <num> <operator> <num>

<num> → <title>

| <value>

<operator> → (< > <= >= /=)

Note: the ellipsis is not a symbol

No changes have been made to the grammar after the second project deliverable.

Restrictions:

- Variables must be integers
- Variables cannot be declared and initialized in the same line
- Names are simplified
- Complex expressions cannot be assigned to variables or used in conditionals
- Functions are basic and do not include arguments
- IO is not implemented
- All of other functionalities are not present

**Source Code Description:**

The ‘Interpreter’ class has been implemented to interface with the parsing algorithm in real time. In particular the variable declaration/assignment statements and all of the recursive functions involved in processing conditional statements activate functions within the interpreter. Two arraylists, one indexing variable names (‘RAM_indentifier’), and another storing the values of the simulated memory addresses

(‘RAM_value’), have been created as members of ‘interpreter’. The declare and initialize functions add indexes of new variables to the arraylists and initialize their values respectively. Within the ‘CONDITIONAL_check’ function of the parser, ‘enter_conditional’ is called from the interpreter which evaluates a boolean variable to determine if the following variable manipulations will be processed. This function adds to a stack of boolean variables, where the top element determines if the current nested statement should be performed. The flag variable ‘contiguous’ determines if one of the if clauses has already been activated, and prevents further statements from being performed only within the current if-else.

A ‘wrapped_boolean’ object has been implemented primarily to allow for passing boolean values by reference. Recursive parsing of conditional statements allows for passing wrapped_boolean objects between them. The truthfulness of any sub-conditional can be stored in the object that was passed to it and referenced later on in the functions that called it. This information is processed according to the logical operation performed at each function, and the parent caller is passed the final result, which is then forwarded to the interpreter. Separate temporary integer variables are included within the wrapped_boolean object and are used to evaluate relational operators at the lowest level.

```
Exit <conditional>
Enter <statementlist>
Enter <conditional>
Enter terminal
Exit terminal
Exit <conditional>
Enter <varlist>
Enter terminal
Exit terminal
Exit <varlist>
Exit <statementlist>
Exit <statementlist>
Enter terminal
END was found: end
Exit terminal
Enter terminal
NAME was found: carson
Exit terminal
Enter terminal
SEMI was found: ;
Exit terminal
bar: 5
foo: 0
```

The final output for interpreting input1.txt shows that the if clause was activated, as variable bar has its value updated to 5 but foo is still uninitialized. As both variables default to 0 when they are declared, they are both equal and the condition (bar = foo) was correct

```
Exit <conditional>
Enter <statementlist>
Enter <conditional>
Enter terminal
Exit terminal
Exit <conditional>
Enter <varlist>
Enter terminal
Exit terminal
Exit <varlist>
Exit <statementlist>
Exit <statementlist>
Enter terminal
END was found: end
Exit terminal
Enter terminal
NAME was found: carson
Exit terminal
Enter terminal
SEMI was found: ;
Exit terminal
bar: 0
foo: 5
```

The final output for interpreting input2.txt shows that the else clause was activated as variable foo has its value updated to 5 but bar is still uninitialized. As both variables default to 0 when they are declared, they are both equal and the condition (bar /= foo) was false.

```
Exit <conditional>
Enter <statementlist>
Enter <conditional>
Enter terminal
Exit terminal
Exit <conditional>
Enter <varlist>
Enter terminal
Exit terminal
Exit <varlist>
Exit <statementlist>
Exit <statementlist>
Enter terminal
END was found: end
Exit terminal
Enter terminal
NAME was found: carson
Exit terminal
Enter terminal
SEMI was found: ;
Exit terminal
bar: 0
foo: 5
```

The final output for interpreting input3.txt shows that the else clause was activated as variable foo has its value updated to 5 but bar is still uninitialized. The if clause included an impossible statement of bar and foo both being equal and not equal. This is obviously false so the if clause was not activated.