Gebze Technical University
Computer Engineering

CSE 341
Programming Languages
2020 Fall

HOMEWORK 3 REPORT

Yusuf Can Kan
161044007

Course Assistant: Muhammet Ali Dede

1 INTRODUCTION

1.1 Problem Definition

In this problem we will build and interpreter for G++ language (which is a language created for Programming Language Courses).We will implement our interpreter using lexer analyzer which we had implemented our previous homework. We will implement our tree in 2 way. First is, using yacc/bison (is is a tool to implement syntax analysis given rules in a meta-grammar sucs as CFG.) Yacc generates the C code for syntax analysis. In first part of the homework we will use out flex files from previous homework and write a new yacc file to combine them together. In second part of the homework we will write our parser in lisp language. Just like first part, in this part we will use our lexer from previous homework. Both of the analyses will start the interpreter and checks if syntax correct while generating parse tree.

Since gpp_lexer.l and gpp_intepreter.y files are easy to read and easy to understand, you can check my DFA and Backus-Nour(BNF) rules structures in these files.

Implemented and Unimplemented Parts

- I implemented all the expression in yacc part.
- In lisp part I implemented everything except load keyword. In order to implement this we need to implement string type token to out lexer and we didn't implement that. For exaple int this input;

  "1 2 a b 12sa"

  Out lexer creates this tokens;

  OP_OP VALUE VALUE IDENTIFIER IDENTIFIER ERROR

  Since "12sa" does not represent anything in our dfa, it is not possible to solve this from parser implementation.

1.2 Problem Usage

1.2.1 Part 1

```
yck@ubuntu:~/Desktop/asd$ make
lex lexer.l
yacc -d gpp_interpreter.y -o gpp_interpreter.c
gcc helperfunctions.c lex.yy.c gpp_interpreter.c -o gpp_interpreter.out
yck@ubuntu:~/Desktop/asd$ ./gpp_interpreter.out input.txt
yck@ubuntu:~/Desktop/asd$ ./gpp_interpreter.out
(and true false)
(exit)
yck@ubuntu:~/Desktop/asd$
```

1.2.2 Part 2

```
yck@ubuntu:~/Desktop/asd$ clisp gpp_interpreter.lisp input.txt
yck@ubuntu:~/Desktop/asd$ clisp gpp_interpreter.lisp
(and true false)
(exit)
yck@ubuntu:~/Desktop/asd$
```

1.3 Assumptions, Ambiguities and Problem Solution Approach

1.3.1 Part1

One of the main problem in this part was parce-reduce errors. Some tokens might go to diferent non terminal or terminal symbols. For example VALUE symol both have VALUES and EXPI way to go. In order to prevent this the EXPI symbol had put under the VALUES symbol. So structure become this;

VALUES:
  EXPI | VALUES EXPI

EXPI:
  VALUE

Besides this, the union created for every token to pass variables inside the parse tree which union has 3 different data type in it. In order to pass integers one integer variable created, for strings 1 char array that has 256 size, and for lists one int array pointer. In order to manipulate this array some helpful functions implemented, which are provided in “helperfunctions.c/.h files.

In order to represent IDLIST for function implementation a new symbol created which is IDLIST.

IDLIST:
  IDLIST IDENTIFIER | IDENTIFIER

This structure represents identifier lists as IDLIST.

In order to calculate append and concatenation procedures in interpreter, append function implemented. Besides this implementation, in order to storing and reaching defined variables, set_identifier and get_identifier functions created. Variables are storing in one global array, so in this way user can be reach every time any variable with using created functions.
I implemented all the writing file and final calculations satements in INPUT nonterminal symbol. This symbol in final connects to START start symbol. So this structure is following;

START: | START INPUT {};

INPUT:
  EXPI | EXPB | EXPLISTI |
  NEW_LINE |
  COMMENT |
  OP_OP KW_EXIT OP_CP ;

Part2

In this part I tried to create same structure with yacc implementation. I created a function that "calculateParseTree" which takes the tokens and inputs, checks tokens left to roght and creates a parse tree. It checks if token is EXPI,EXPB,EXPLISTI,COMMENT or EXIT. When it finds token structure, it startes to create parser tree with proper recursive calls.

I store the tokens which created by lexer in global variable and passed it to the parser file. I implemented parser tree in left to right appproach. So when interpreter first started it started to look from left and started to build parse tree automatically. Since I implemented all functions and calculations recursively I implemented same BNF in part 1.

Some Solution Approachs

(and true (or true false))

In this statement my code first read the "OP_OP" and "KW_AND" token and jumped to EXPI statement for calculating and operation. After that it looks the first statement and recursively calls EXPI calculation statement again and returns true and does the same operations for or statement and calculates value creating parse tree. Basically this statement corresponds to below BNF representation;
EXPB : OP_OP KW_AND EXPB EXPB OP_CP

Besides this one of the one of the different calculations that I implemented is in less and equal statement. Because in this statements sometimes we need to calculate EXPI expressins so in this statements I called EXPI statements in proper positions and it has created this structure;

EXPB; OP_OP KW_EQUAL EXPB EXPB OP_CP
EXPB; OP_OP KW_EQUAL EXPI EXPI OP_CP
EXPB; OP_OP KW_LESS EXPI EXPI OP_CP

The example for this might be below statements;

(if (equal 3 4) (list 1 2 3))

In this statement, first tree builds in if expression and calculates equal statement (EXPB), and in order to do that it goes to the EXPB part and inside of this it calls the EXPI for obtaining 3 and 4.

In VALUES and IDLIST statements I created list recursively and returned it.

Functions and Assignment Operations

For asssignment operator I store a global list, so in this way whenever user tries to reach any variable my interpreter directly takes the input from this list and returns it.
In function definition I implemented the same logic. I created a global list for functions that stores the name of the function, variables of the function and body part of the function. When user calls a defined function, first function takes variables, assignes to the proper places in function body part (such as if user gives variable i as 4, function rewrites i+5 as 4+5) and calls parse tree recursivelt for this subprocess. In this process interpreter stores the changed body in temporarily. After function returned last expression this body part becomes unavailable to user.

Note
Since EXPLISI created in list syntax, statements like for,if,etc. Does not takes directly expressions. They accepts the list expressions as defined in “GppSyntax.pdf” file.

Since I implemented recursive-right to left approach I didn’t faced any ambiguity in my arithmetic and boolean implementations. Thanks to lisp like syntax of G++ language, interpreter always calculates inside the paranthesis.

One of the ambiguity that i faced is VALUE symbol. In pdf file both EXPI and VALUES symbols both represents VALUE. It created a problem as part1. I solved this problem same as part1, I represent integer value as EXPI in VALUES.

1.4 Test Cases

1.4.1 Part 1

Input1

input

```
1
2   ;; fda fgdasfgdasfdas
3
4
5   (disp (list 7 8 (** 3 2)))
6
7
8   (if (and true (not true))
9       (list 1 (+ 1 2) (* 3 4)))
10
11  (if (or false true)
12      (list 1 (+ 1 2) (* 3 4)))
13
14  ;; Returns loop output as a list
15  (for (i 4 8) (list (+ 6 5) (* 2 3)))
16
17
18
```

Output1

parsed_cpp.txt

```
1   7 8 9
2   12
3   6
4
```

Inputt2

input

```
1
2
3   ;; You must provide function body as a list
4   (deffun a (x y z) (list 1 6))
5
6
7   ;; FUNCTIONM CALL
8   ;; You must provide variables as a list
9   (a (list 3 4 5))
10
11
12  ;; Two display side by side
13  (disp (list 7 8 (** 3 2))) (disp 66)
14
15  ;; variable definition
16  (defvar y 5)
17  (defvar z 6)
18  (defvar x 7)
19  (set a 12)
20  (set b 11)
21  (set c 37)
22  (/ a z)
23  (list 1 2 (+ 1 3) a c 5 b b (/ a b))
24
25
```

Output2

parsed_cpp.txt

```
0
5
7 8 9
66
5
6
7
12
11
37
2
1 2 4 12 37 5 11 11 1

```

Inputt3

input

```
(and true true)
(or false true)
(+ 1 2) (+ (/ 30 5)  (* (+ 5 4) (- 5 3)))




(if (and true (not true)) (list 1 (+ 1 2) (* 3 4)))
(and true true)
(or false true)
(+ 1 2) (+ (/ 30 5)  (* (+ 5 4) (- 5 3)))
```

Output3

parsed_cpp.txt

```
True
True
3
24
True
True
3
24

```

Terminal Test

```
(and true false)
(deffun foo (a b c) (list (+ 1 2) (* (+ 3 4) (- 10 5))))
(if (not (or false false)) (list 1 2 3 4))
(for (i 1 4) (list i (* 3 3) (+ 45 2)))
(defvar gtu 5)
gtu
(concat (list 1 2 3) (list 4 5 6))
```

parsed_cpp.txt

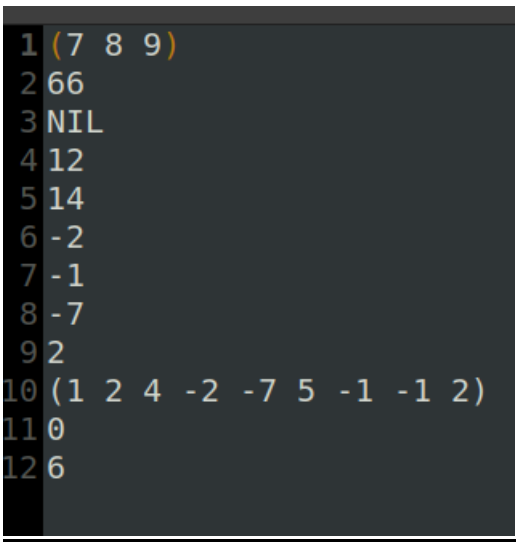
```
1  False
2  0
3  4
4  47
5  5
6  5
7  1 2 3 4 5 6
8  5 6 6 6 6 6 6
9
```

1.4.2 Part 2

Input 1

```
;; fda fgdasfgdasfdas


(disp (list 7 8 (** 3 2))) (disp 66)

(if (and true (not true))
    (list 1 (+ 1 2) (* 3 4)))

(if (or false true)
    (list 1 (+ 1 2) (* 3 4)))

;; Returns loop output as a list
(for (i 4 8) (list (+ i 5) (* 2 i)))


;; Assignment
;; Returns assigned values
(set y -2)
(set z -1)
(set x -7)
(/ y z)
(list 1 2 (+ 1 3) y x 5 z z (/ y z))


;; You must provide function body as a list
(deffun a (x y z) (list 1 (+ x y) 3 (** y (+ 2 z)) 6))


;; FUNCTIONM CALL
;; You must provide variables as a list
(a (list 3 4 5))
```

Output 1



```
(7 8 9)
66
NIL
12
14
-2
-1
-7
2
(1 2 4 -2 -7 5 -1 -1 2)
0
6
```

Input 2

```
;; Two display side by side
(disp (list 7 8 (** 3 2))) (disp 66)

;; variable definition
(set y -2)
(set z -1)
(set x -7)
(/ y z)
(list 1 2 (+ 1 3) y x 5 z z (/ y z))



(and true true)
(or false true)
(+ 1 2) (+ (/ 30 5)  (* (+ 5 4) (- 5 3)))




(if (and true (not true)) (list 1 (+ 1 2) (* 3 4)))
(and true true)
(or false true)
(+ 1 2) (+ (/ 30 5)  (* (+ 5 4) (- 5 3)))


(and true true)
(or false true)
(+ 1 2) (+ (/ 30 5) (* (+ 5 4) (- 5 3)))

(if (and true (not false)) (list 1 (+ 1 2) (* 3 4)))
(if (and true (not true)) (list 1 (+ 1 2) (* 3 4)))
(if true (list 1 3))
```

Output 2

```
(7 8 9)
66
-2
-1
-7
2
(1 2 4 -2 -7 5 -1 -1 2)
T
T
3
24
NIL
T
T
3
24
T
T
3
24
12
NIL
3
```

Input 3

```
(and true false)
(deffun foo (a b c) (list (+ a b) (- c 2)))
(if (not (or false false)) (list 1 2 3 4))
(for (i 1 4) (list i (* i i) (+ i 5)))
(defvar gtu 5)
gtu
(set gtu 5)
(set gtu 6)
gtu
(concat (list 1 2 3) (list 4 5 6))
(append 5 (list 6 6 6 6 6))

(for (i 1 4) (list i (* i i) (+ i 5)))

(for (i 1 4) (list 5))
```

Output 3

```
NIL
0
4
8
5
5
5
6
6
(1 2 3 4 5 6)
(5 6 6 6 6 6)
8
5
```

Terminal Test

```
yck@ubuntu:~/Desktop/asd$ clisp gpp_interpreter.lisp
(and true false)
(deffun foo (a b c) (list (+ a b) (- c 2)))
(foo (list 2 1 3))
(if (not (or false false)) (list 1 2 3 4))
(for (i 1 4) (list i (* i i) (+ 1 5)))
(defvar gtu 5)
gtu
(set gtu 6)
gtu
(concat (list 1 2 3 4) (list 8 9 10))
(append 5 (list 6 6 6 6 6))
(exit)
```

```
NIL
0
1
4
6
5
5
6
6
(1 2 3 4 8 9 10)
(5 6 6 6 6 6)
```