CSE341 – Programming Languages
Homework #3 REPORT
Canberk Arıcı - 171044062

In this homework, all parts work correctly.

PART 1:

In this part, I have rules in lexer that gives tokens to parser.
I also do error check for starting with 0 addition to rules in lexer.
I declared values above in gpp_interpreter.y that I use in parser, also there are tokens that I take from lexer.
In rules part, I implemented Concrete Syntax of G++, EXPI(printing), EXPLISTI, EXPB, LISTVALUE, VALUES, EXPI, EXPB, BinaryVal, IDList, EXPLISTI, IDLIST respectively according given G++ Syntax.

In main, I implemented that if no parameter is given then take input from user and process. If file name is given as parameter in argv, then I read contents from file and process.

* I implemented assignment operator, you can use it as **setq**
* I implemented deffun and it works
* I implemented just the first case **EXPI -> (if EXPB EXPLISTI)**
* I dealt with errors and future errors by using many control flags, according to flags' values, program understands what to print. I implemented 2 functions that are required by some operations, functions are printArr and mAppend. printArr prints array and mAppend appends a value to array. Functions are in gpp_interpreter.h

*** Like in a normal programming language, I exit when there is any syntax error.**

*** There is a makefile for part1. After running make command, please run ./gpp_interpreter.out if you want interpreter mode, if you want to read from file and process its contents then please run ./gpp_interpreter.out with a filename, for example : ./gpp_interpreter.out input.g++**

*** If you click enter without entering anything, program terminates in interpreter mode.**

TEST of PART1:

From File

input.txt :

```
; a.g++
'( 1 2 3 4 )
(-  20  (/ 20 (* 1 2)))
( +  (* (+ 3 4 ) 2) ( /  ( + 3 3 ) 2))
(* (- 10 5)( + 3 ( - 9 8)))
( /  (+ 6 (* 2 3)) 2)
(and  true (or true (equal 1 1 )))
(not (equal  true false))
(and false (or true (and false true)))
(or true (equal true false))
(equal (+ 2 2) ( + 1 3))
(equal true (and true false))
(less 1 3)
(less 13 12)
(concat '(1 2 3 )  '(4 5 6 ))
(append (+ 1 3 ) '(6 5 7 8 ) )
(set aa (* (+ 1 3 ) (- 5 2)))
(setq aa (* (+ 1 3 ) (- 5 2)))
(aa (append 9 '(3 5 7)))
(if (and true true ) '(2 4 6))
(if true (concat '(1 3 5 ) '(7 8 9)))
(if true (append ( / 4 2 )  '(1 3 5 7)))
(while (and true true) (concat '(4 5 6) '(7 6 5)))
(for (aa  (+ 1 9 ) (* 1 3)) (append 2 '(9 8 7)))
(defvar aa (- 7 5))
(deffun aa ( hello welcome ) '( 12 13 14))
(load "file.txt")
(disp "canberk")
(list 12 3 4 )
(if true '( 6 1 7 ) '( 2 5 4))
```

OUTPUT:

```
(base) canberk@canberk-Aspire:~/Desktop/Yacc_Interpreter$ ./gpp_interpreter.out input.g++
SYNTAX OK.


SYNTAX OK.
Result: (1 2 3 4)

SYNTAX OK.

Result: 10


SYNTAX OK.

Result: 17


SYNTAX OK.

Result: 20


SYNTAX OK.

Result: 6


SYNTAX OK.
Result: True

SYNTAX OK.
Result: True

SYNTAX OK.
Result: False

SYNTAX OK.
Result: True
```

```
SYNTAX OK.
Result: True

SYNTAX OK.
Result: False

SYNTAX OK.
Result: True

SYNTAX OK.
Result: False

SYNTAX OK.
Result: (1 2 3 4 5 6)

SYNTAX OK.
Result: (4 6 5 7 8)

SYNTAX OK.

Result: 12


SYNTAX OK.

Result: (9 3 5 7)

SYNTAX OK.

Result: (2 4 6)

SYNTAX OK.

Result: (1 3 5 7 8 9)

SYNTAX OK.

Result: (2 1 3 5 7)
```

```
SYNTAX OK.

Result: (4 5 6 7 6 5)

SYNTAX OK.

Result: (12 13 14)

SYNTAX OK.

Result: (2 9 8 7)

SYNTAX OK.


SYNTAX OK.

Result: 2


SYNTAX OK.


SYNTAX OK.

Result: (12 3 4)


SYNTAX_ERROR: Expression not recognized!
```

TESTING PART1 BY USING IT AS INTERPRETER:

```
(base) canberk@canberk-Aspire:~/Desktop/Yacc_Interpreter$ ./gpp_interpreter.out ; a.g++
SYNTAX OK.


'( 1 2 3 4 )
SYNTAX OK.
Result: (1 2 3 4)

(-  20  (/ 20 (* 1 2)))
SYNTAX OK.

Result: 10


( +  (* (+ 3 4 ) 2) ( /  ( + 3 3 ) 2))
SYNTAX OK.

Result: 17


(* (- 10 5)( + 3 ( - 9 8)))
SYNTAX OK.

Result: 20


( /  (+ 6 (* 2 3)) 2)
SYNTAX OK.

Result: 6


(and  true (or true (equal 1 1 )))
SYNTAX OK.
Result: True

(not (equal  true false))
SYNTAX OK.
Result: True

(and false (or true (and false true)))
SYNTAX OK.
Result: False

(or true (equal true false))
SYNTAX OK.
Result: True

(equal (+ 2 2) ( + 1 3))
SYNTAX OK.
Result: True
```

```
(equal true (and true false))
SYNTAX OK.
Result: False

(less 1 3)
SYNTAX OK.
Result: True

(less 13 12)
SYNTAX OK.
Result: False

(concat '(1 2 3 )  '(4 5 6 ))
SYNTAX OK.
Result: (1 2 3 4 5 6)

(append (+ 1 3 ) '(6 5 7 8 ) )
SYNTAX OK.
Result: (4 6 5 7 8)

(set aa (* (+ 1 3 ) (- 5 2)))
SYNTAX OK.

Result: 12


(setq aa (* (+ 1 3 ) (- 5 2)))
SYNTAX OK.

Result: 12


(aa (append 9 '(3 5 7)))
SYNTAX OK.

Result: (9 3 5 7)

(if (and true true ) '(2 4 6))
SYNTAX OK.

Result: (2 4 6)

(if true (concat '(1 3 5 ) '(7 8 9)))
SYNTAX OK.

Result: (1 3 5 7 8 9)

(if true (append ( / 4 2 )  '(1 3 5 7)))
SYNTAX OK.

Result: (2 1 3 5 7)
```

```
(while (and true true) (concat '(4 5 6) '(7 6 5)))
SYNTAX OK.

Result: (4 5 6 7 6 5)

(for (aa  (+ 1 9 ) (* 1 3)) (append 2 '(9 8 7)))
SYNTAX OK.

Result: (2 9 8 7)

(defvar aa (- 7 5))
SYNTAX OK.

Result: 2


(deffun aa ( hello welcome ) '( 12 13 14))
SYNTAX OK.

Result: (12 13 14)

(load "input.txt")
SYNTAX OK.


(disp "canberk")
SYNTAX OK.


(list 12 3 4 )
SYNTAX OK.

Result: (12 3 4)

(if true '( 6 1 7 ) '( 2 5 4))

SYNTAX_ERROR: Expression not recognized!
```

PART 2:

In this part, I created constants for keywords,comments,values,identifiers,operators.
If there is no input given while running then interpreter mode starts else contents are read from file line by line and process its content, I get tokens of line from lexer and then give it to parser then I print result of parser if there is no error.

Algorithm of lexer function:
1- Determine given character lexeme is what
2- Append its token to token list (tokenL)
Token list is used in parser
3- Append character lexeme to character list (chLst)
4- If there is any syntax error, give error

Parser function's algorithm is complicated, I gave its algorithm in detail in gpp_interpreter.lisp file but I will write algorithms of some parts:

* tokens is token list and chars is character lexeme list

OP_DIV
*** nVal is set to 0 at the beginning and syntaxError is set to t at the beginning**
If tokens are OP_OP and OP_DIV, set errorVal to nil
if tokens's ith element is IntegerValue and i=2 then set nVal to ith element of chars
else divide nVal by ith element of chars
and set syntaxError to nil

OP_MULT
*** nVal is set to 0 at the beginning and syntaxError is set to t at the beginning**
If tokens are OP_OP and OP_PLUS, set errorVal to nil
if tokens's ith element is IntegerValue then increment nVal by ith element of chars and set syntaxError to nil

OP_MINUS
*** nVal is set to 0 at the beginning and syntaxError is set to t at the beginning**
If tokens are OP_OP and OP_MINUS, set errorVal to nil
if tokens's ith element is IntegerValue and i=2 then set nVal to ith element of chars
else decrement nVal by ith element of chars
and set syntaxError to nil

* You can use **set** as assignment operator, implementing **setq** in this part was compelling so there is no **setq** in this part
* I implemented deffun and it works
* I dealt with errors and future errors by detailed conditions&cases.
***If you click enter without entering anything, program terminates in interpreter mode.**

Please run clisp gpp_interpreter.lisp if you want interpreter mode, if you want to read from file and process its contents then please run clisp gpp_interpreter.lisp with a filename, for example : clisp gpp_interpreter.lisp input.g++

TEST of PART2:

From File

input.g++ :

```
; a.g++
'( 1 2 3 4 )
(-  20  (/ 20 (* 1 2)))
( +  (* (+ 3 4 ) 2) ( /  ( + 3 3 ) 2))
(* (- 10 5)( + 3 ( - 9 8)))
( /  (+ 6 (* 2 3)) 2)
(and  true (or true (equal 1 1 )))
(not (equal  true false))
(and false (or true (and false true)))
(or true (equal true false))
(equal (+ 2 2) ( + 1 3))
(equal true (and true false))
(less 1 3)
(less 13 12)
(concat '(1 2 3 )  '(4 5 6 ))
(append (+ 1 3 ) '(6 5 7 8 ) )
(set aa (* (+ 1 3 ) (- 5 2)))
(aa (append 9 '(3 5 7)))
(if (and true true ) '(2 4 6))
(if true (concat '(1 3 5 ) '(7 8 9)))
(if true (append ( / 4 2 )  '(1 3 5 7)))
(while (and true true) (concat '(4 5 6) '(7 6 5)))
(deffun aa ( hello welcome ) '( 12 13 14))
(for (aa  (+ 1 9 ) (* 1 3)) (append 2 '(9 8 7)))
(disp "canberk")
(defvar aa (- 7 5))
(load "input.txt")
(list 12 3 4 )
```

Output:

```
(base) canberk@canberk-Aspire:~/Desktop/Lisp_Interpreter$ clisp gpp_interpreter.lisp input.g++
Syntax OK.
Result:T

Syntax OK.
Result:(1 2 3 4)

Syntax OK.
Result:10

Syntax OK.
Result:17

Syntax OK.
Result:20

Syntax OK.
Result:6

Syntax OK.
Result:T

Syntax OK.
Result:T

Syntax OK.
Result:NIL

Syntax OK.
Result:T

Syntax OK.
Result:T

Syntax OK.
Result:NIL

Syntax OK.
Result:T
```

```
Syntax OK.
Result:NIL

Syntax OK.
Result:(1 2 3 4 5 6)

Syntax OK.
Result:(4 6 5 7 8)

Syntax OK.
Result:12

ID:aa
Syntax OK.
Result:(9 3 5 7)

Syntax OK.
Result:(2 4 6)

Syntax OK.
Result:(1 3 5 7 8 9)

Syntax OK.
Result:(2 1 3 5 7)

Syntax OK.
Result:(4 5 6 7 6 5)

Syntax OK.
Result:aa

Syntax OK.
Result:(2 9 8 7)

Syntax OK.
Result:canberk

Syntax OK.
Result:2
Syntax OK.
Result:input.txt

Syntax OK.
Result:(12 3 4)
```

TESTING PART2 BY USING IT AS INTERPRETER:

```
(base) canberk@canberk-Aspire:~/Desktop/Lisp_Interpreter$ clisp gpp_interpreter.lisp
; a.g++
Syntax OK.
Result:T

'( 1 2 3 4 )
Syntax OK.
Result:(1 2 3 4)

(-  20  (/ 20 (* 1 2)))
Syntax OK.
Result:10

( +  (* (+ 3 4 ) 2) ( /  ( + 3 3 ) 2))
Syntax OK.
Result:17

(* (- 10 5)( + 3 ( - 9 8)))
Syntax OK.
Result:20

( /  (+ 6 (* 2 3)) 2)
Syntax OK.
Result:6

(and  true (or true (equal 1 1 )))
Syntax OK.
Result:T

(not (equal  true false))
Syntax OK.
Result:T

(and false (or true (and false true)))
Syntax OK.
Result:NIL

(or true (equal true false))
Syntax OK.
Result:T
```

```
(equal (+ 2 2) ( + 1 3))
Syntax OK.
Result:T

(equal true (and true false))
Syntax OK.
Result:NIL

(less 1 3)
Syntax OK.
Result:T

(less 13 12)
Syntax OK.
Result:NIL

(concat '(1 2 3 )  '(4 5 6 ))
Syntax OK.
Result:(1 2 3 4 5 6)

(append (+ 1 3 ) '(6 5 7 8 ) )
Syntax OK.
Result:(4 6 5 7 8)

(set aa (* (+ 1 3 ) (- 5 2)))
Syntax OK.
Result:12

(aa (append 9 '(3 5 7)))
Syntax OK.
Result:(9 3 5 7)

(if (and true true ) '(2 4 6))
Syntax OK.
Result:(2 4 6)

(if true (concat '(1 3 5 ) '(7 8 9)))
Syntax OK.
Result:(1 3 5 7 8 9)
```

```
(if true (append ( / 4 2 )  '(1 3 5 7)))
Syntax OK.
Result:(2 1 3 5 7)

(while (and true true) (concat '(4 5 6) '(7 6 5)))
Syntax OK.
Result:(4 5 6 7 6 5)

(deffun aa ( hello welcome ) '( 12 13 14))
Syntax OK.
Result:aa

(for (aa  (+ 1 9 ) (* 1 3)) (append 2 '(9 8 7)))
Syntax OK.
Result:(2 9 8 7)

(disp "canberk")
Syntax OK.
Result:canberk

(defvar aa (- 7 5))
Syntax OK.
Result:2

(load "input.txt")
Syntax OK.
Result:input.txt

(list 12 3 4 )
Syntax OK.
Result:(12 3 4)
```