Lexical Analyzer and a Syntax Recognizer

CSIE35460-Programming Languages and Compilers

Programming Assignment 1

February 20th, 2023

Clara Choc 411021326

Samuel Chavarria 411021334

The Problem Description:

Use lex (or flex) and yacc (or bison) to implement a front end (including a lexical analyzer and a syntax recognizer) of the compiler for the P programming language.

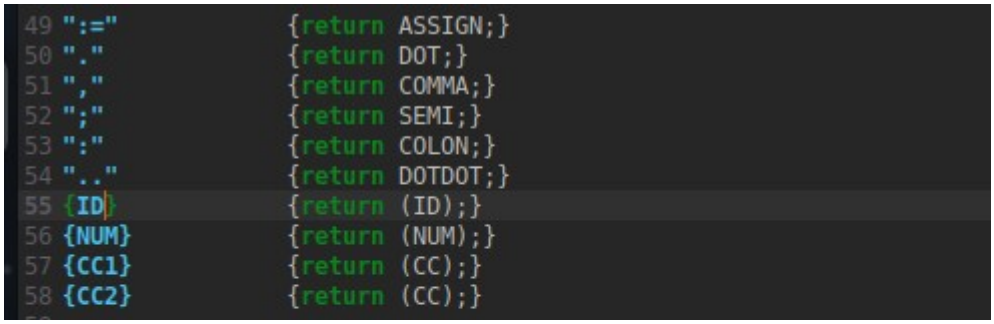
- See an attached document for the lexical rules and grammar rules in details.
- You are requested to separate the C code, the Lex specification, the Yacc specification into separated files.

Highlight of the way you write the program:

In order to successfully parse both bison and flex files (p2c_lex.l and p2c_yacc.y) , we needed to modified them in this assignment.

Starting off with the p2c_lex.l file, we are required to write the appropriate lexical rules according to the description given.

The following is the modified subsection of p2c_lex.l file after finalizing it:



In the p2c_lex.l file, we had to understand the rules and write the corresponding tokens and return appropriately. For the ID, NUM, CC1 and CC2, since they are character sets, they required a curly brackets.

For the p2c_yacc.y file, we needed to declare our tokens first. Then, understanding the grammar rules was the next step to write and define the grammar based on the P programming language lexical rules provided. There were some minor errors/mistakes in the given files that we needed to correct. For instance, line 70, file p2c_yacc.y:

```
|    {printf("morestm => Null \n")}
```

was updated to

```
|    {printf("morestm => Null \n");}
```

Writing the grammar consisted of:

1. Defining the grammar rules

2. Declaring necessary tokens with non-reserved words.

3. Defining Kleene closures with bases case to avoid redundancy

4. Additionally, defining the constant grammar.

5. Confirming if the expected values are correct . If the expected values is not correct, when using bison, there will be warnings after executing, therefore, making the compilation unsuccessful.

The program listing:

Link to the code file on github:

https://github.com/samchavita/Programming-Languages-and-Compilers/tree/main/P2C_A1

Test Run Results:

1. Use bison to compile p2c_yacc.y into p2c_yacc.c. The -d switch produces a header file p2c_yacc.h.
2. Use flex to compile p2c_lex.l into p2c_lex.c.
3. Use gcc to compile p2c_lex.c into p2c_lex.o, p2c_yacc.c into p2c_yacc.o, and p2c.c into p2c.o.
4. Use gcc to link p2c.o, p2c_lex.o, and p2c_yacc.o into parse.

Alternatively use the command $ make that will accomplish the same as the above three steps.

Bison

$ bison -d -o p2c_yacc.c p2c_yacc.y

Flex

$ flex -op2c_lex.c p2c_lex.l

GCC

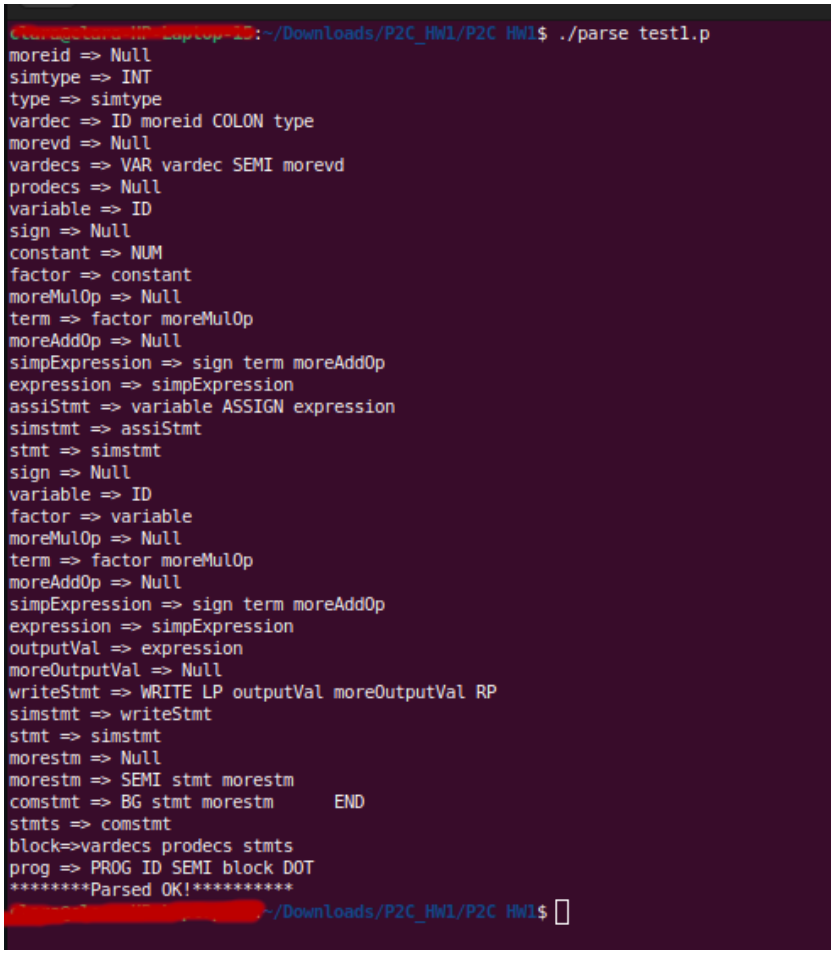
$ gcc -c p2cc_lex.c

$ gcc -c p2c_yacc.c

$ gcc -c p2c.c

$ gcc -o parse p2c_lex.o p2c.o p2c_yacc.o

Test result 1:

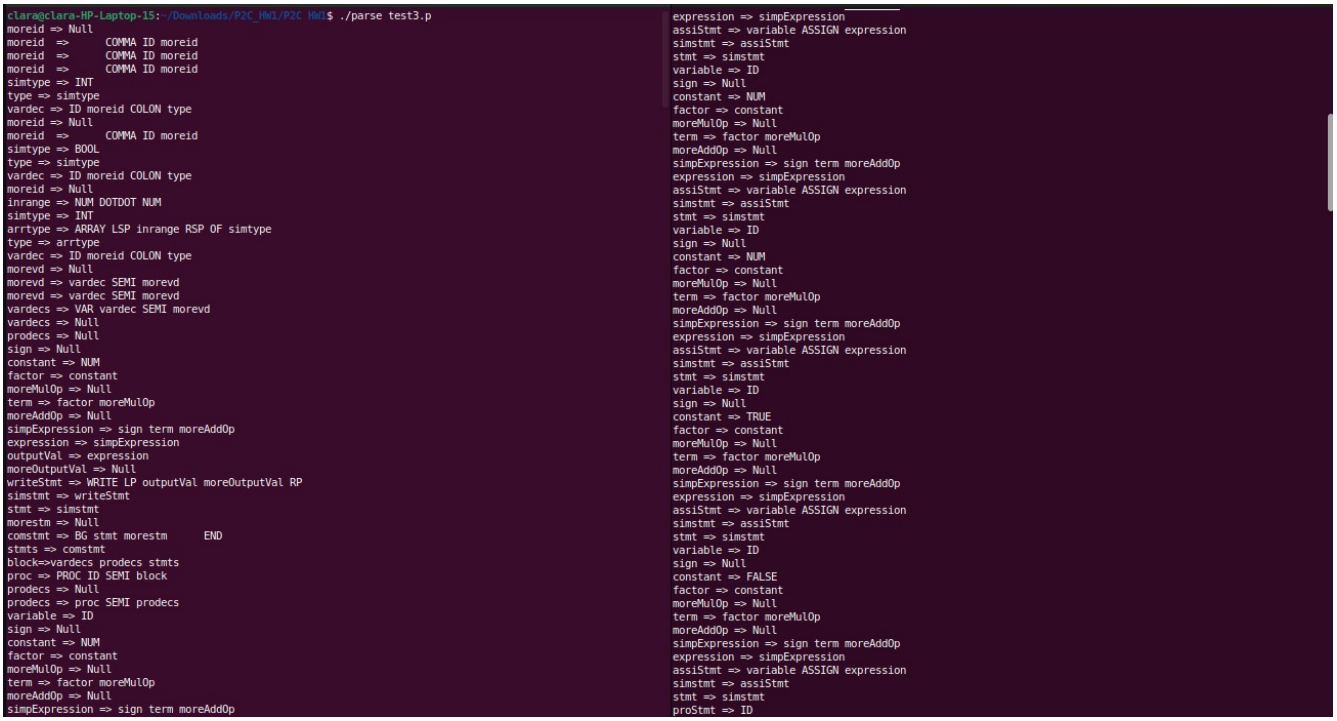


Test results 2:

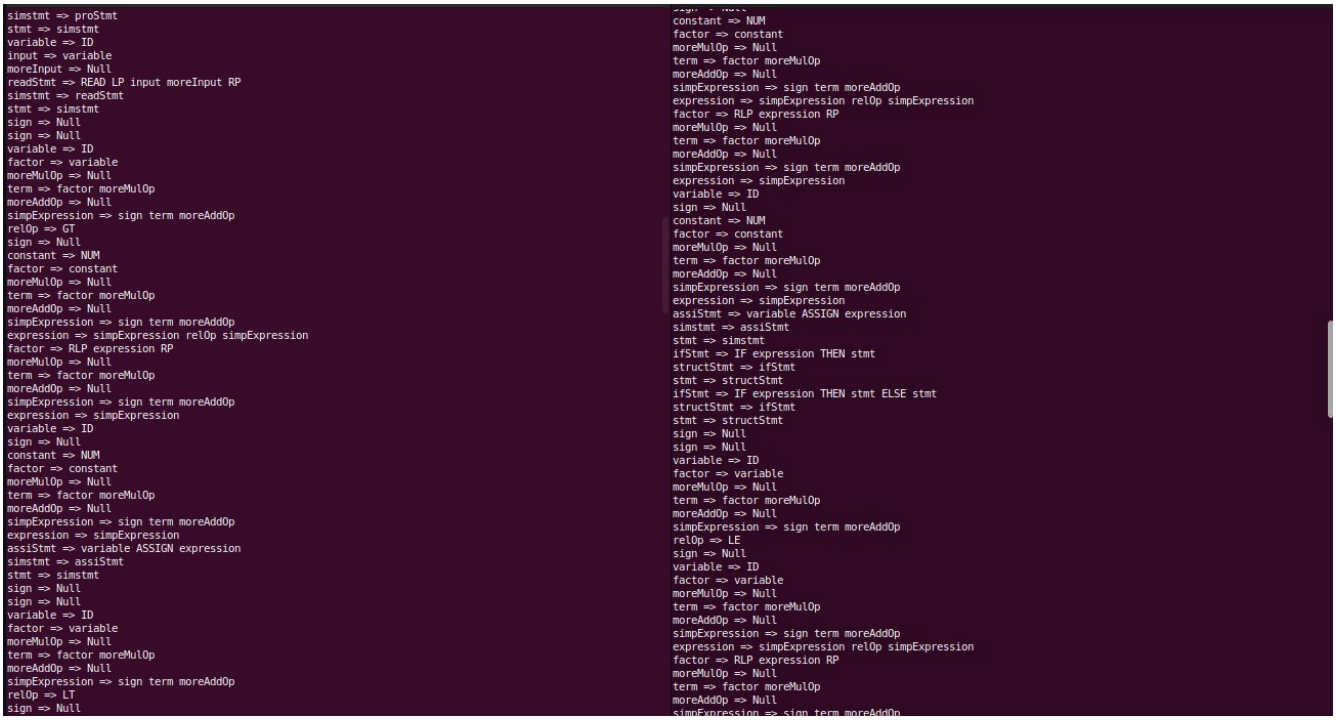
```
[redacted]:~/Downloads/P2C_HW1/P2C HW1$ ./parse test2.p
moreid => Null
moreid  =>      COMMA ID moreid
moreid  =>      COMMA ID moreid
simtype => INT
type => simtype
vardec => ID moreid COLON type
morevd => Null
vardecs => VAR vardec SEMI morevd
prodecs => Null
variable => ID
sign => Null
constant => NUM
factor => constant
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
variable => ID
sign => Null
constant => NUM
factor => constant
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
variable => ID
sign => Null
constant => NUM
factor => constant
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
sign => Null
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
relOp => LE
sign => Null
constant => NUM
factor => constant
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression relOp simpExpression
factor => RLP expression RP
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
variable => ID
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
addOp => ADD
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
moreAddOp => addOp term moreAddOp
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
variable => ID
sign => Null
variable => ID
factor => variable
mulOp => TIMES
variable => ID
factor => variable
moreMulOp => Null
moreMulOp => mulOp factor moreMulOp
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
variable => ID
sign => Null
```

```
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
addOp => ADD
constant => NUM
factor => constant
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
moreAddOp => addOp term moreAddOp
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
morestm => Null
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
comstmt => BG stmt morestm      END
structStmt => comstmt
stmt => structStmt
whileStmt => WHILE expression DO stmt
structStmt => whileStmt
stmt => structStmt
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
outputVal => expression
moreOutputVal => Null
writeStmt => WRITE LP outputVal moreOutputVal RP
simstmt => writeStmt
stmt => simstmt
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
outputVal => expression
moreOutputVal => Null
writeStmt => WRITE LP outputVal moreOutputVal RP
simstmt => writeStmt
stmt => simstmt
morestm => Null
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
comstmt => BG stmt morestm      END
stmts => comstmt
block=>vardecs prodecs stmts
prog => PROG ID SEMI block DOT
********Parsed OK!**********
```

Test result 3:



```
clara@clara-HP-Laptop-15:~/Downloads/P2C_HW1/P2C_HW1$ ./parse test3.p
moreid => Null
moreid  =>      COMMA ID moreid
moreid  =>      COMMA ID moreid
moreid  =>      COMMA ID moreid
simtype => INT
type => simtype
vardec => ID moreid COLON type
moreid => Null
moreid  =>      COMMA ID moreid
simtype => BOOL
type => simtype
vardec => ID moreid COLON type
moreid => Null
inrange => NUM DOTDOT NUM
simtype => INT
arrtype => ARRAY LSP inrange RSP OF simtype
type => arrtype
vardec => ID moreid COLON type
morevd => Null
morevd => vardec SEMI morevd
morevd => vardec SEMI morevd
vardecs => VAR vardec SEMI morevd
vardecs => Null
prodecs => Null
sign => Null
constant => NUM
factor => constant
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
outputVal => expression
moreOutputVal => Null
writeStmt => WRITE LP outputVal moreOutputVal RP
simstmt => writeStmt
stmt => simstmt
morestm => Null
comstmt => BG stmt morestm      END
stmts => comstmt
block=>vardecs prodecs stmts
proc => PROC ID SEMI block
prodecs => Null
prodecs => proc SEMI prodecs
variable => ID
sign => Null
constant => NUM
factor => constant
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
variable => ID
sign => Null
constant => NUM
factor => constant
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
variable => ID
sign => Null
constant => TRUE
factor => constant
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
variable => ID
sign => Null
constant => FALSE
factor => constant
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
proStmt => ID
```



```
simstmt => proStmt
stmt => simstmt
variable => ID
input => variable
moreInput => Null
readStmt => READ LP input moreInput RP
simstmt => readStmt
stmt => simstmt
sign => Null
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
relOp => GT
sign => Null
constant => NUM
factor => constant
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression relOp simpExpression
factor => RLP expression RP
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
variable => ID
sign => Null
constant => NUM
factor => constant
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
sign => Null
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
relOp => LT
sign => Null
constant => NUM
factor => constant
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
ifStmt => IF expression THEN stmt
structStmt => ifStmt
stmt => structStmt
ifStmt => IF expression THEN stmt ELSE stmt
structStmt => ifStmt
stmt => structStmt
sign => Null
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
relOp => LE
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression relOp simpExpression
factor => RLP expression RP
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
```

```
expression => simpExpression
variable => ID
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
addOp => ADD
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
moreAddOp => addOp term moreAddOp
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
variable => ID
sign => Null
variable => ID
factor => variable
mulOp => TIMES
variable => ID
factor => variable
moreMulOp => Null
moreMulOp => mulOp factor moreMulOp
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
indexedVar => expression
variable => indexedVar
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
addOp => ADD
variable => ID
factor => variable
```

```
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
moreAddOp => addOp term moreAddOp
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
sign => Null
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
indexedVar => expression
variable => indexedVar
factor => variable
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
outputVal => expression
moreOutputVal => Null
writeStmt => WRITE LP outputVal moreOutputVal RP
simstmt => writeStmt
stmt => simstmt
variable => ID
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
addOp => ADD
constant => NUM
factor => constant
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
moreAddOp => addOp term moreAddOp
simpExpression => sign term moreAddOp
expression => simpExpression
assiStmt => variable ASSIGN expression
simstmt => assiStmt
stmt => simstmt
morestm => Null
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
```

```
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
comstmt => BG stmt morestm      END
structStmt => comstmt
stmt => structStmt
whileStmt => WHILE expression DO stmt
structStmt => whileStmt
stmt => structStmt
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
outputVal => expression
sign => Null
variable => ID
factor => variable
moreMulOp => Null
term => factor moreMulOp
moreAddOp => Null
simpExpression => sign term moreAddOp
expression => simpExpression
outputVal => expression
moreOutputVal => Null
moreOutputVal => COMMA outputVal moreOutputVal
writeStmt => WRITE LP outputVal moreOutputVal RP
simstmt => writeStmt
stmt => simstmt
morestm => Null
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
morestm => SEMI stmt morestm
comstmt => BG stmt morestm      END
stmts => comstmt
block=>vardecs prodecs stmts
prog => PROG ID SEMI block DOT
********Parsed OK!**********
```

MakeFIle:

```
parse: p2c_yacc.o p2c_lex.o p2c.o
	gcc -o parse p2c_lex.o p2c.o p2c_yacc.o

p2c_lex.o: p2c_lex.c p2c.h p2c_yacc.h
	gcc -c p2c_lex.c

p2c_lex.c: p2c_lex.l
	flex -op2c_lex.c p2c_lex.l

p2c_yacc.o: p2c_yacc.c p2c.h p2c_yacc.h
	gcc -c p2c_yacc.c

p2c_yacc.c: p2c_yacc.y p2c.h p2c_yacc.h
	bison -d -o p2c_yacc.c p2c_yacc.y

p2c_yacc.h: p2c_yacc.y
	bison -d -o p2c_yacc.c p2c_yacc.y

p2c.o: p2c.c p2c.h
	gcc -c p2c.c

clean:
	rm *.o p2c_lex.c parse p2c_yacc.c p2c_yacc.h

# hidden rules
.c:	.l
	flex -o$< $@
```

Discussion:

We think there are still some room for improvement, assume that we want to run more complicate test code in the future, we need to declare more variable in the p2c_lex.l file, such as float, double, char, long, extends, etc., we can also modify our grammar rules in p2c_yacc.y to make it more flexible and stricter, by adding more %token and priority rules using %left or %right.