Recursive Decent Parser and Scanner

Mohamed Salem Ali - Faculty of Science - 3rd year - 2nd semester

Project Proposal

This project is an implementation of a compiler for Proj3 language and this project includes two phases, scanner phase (Lexical analysis) which appears and encounters lexical errors and parser phase (Syntax analysis) which appears and encounters syntax errors. Also, this project includes a counter which counts number of tokens, number of lexical errors and number of syntax error in the test program. This project use Hand Coded (Loop and Switch) method for implementing scanner and use Recursive Descent algorithm for implementing parser and generating many different testing programs and test project thoroughly using different set of proj3.

Proj3 language

<program> ::= program <id> { <states>} .

<states> ::= <state> | <states> ; <state> .

<state> ::= read <id> | write <expr> | <id> <op> <expr> |

if (<expr>) { <stats> } | < id > = (<expr> ? <expr> : <expr>) .

<expr> ::= <expr> <op> <expr> | <expr> [<expr>] | <expr> . length

new <id> [<expr>] | new <id> () | ! <expr> | (<expr>)

<id> | <integer literal>.

<op> ::= += | *= | &&= .

<id> ::= <letter> | <id> <letter> | <id> <digit> .

<integer literal> ::= <digit> | <integer literal> <digit> .

<letter> ::= a | b| c| d| e| f| g| h| i| j| k| l| m| n| o| p| q| r| s| t| u| v| w| x| y| z .

<digit> ::= 0 |1 |2 |3 |4 |5 |6 |7 |8 |9 .

Detailed Project Design Proposal

We will discuss about every theory, definitions, methods, algorithms, functions that used in this project and problems which solved in grammar, write fixed grammar.

---

(1) Scanner Phase

1-Lexical analysis (Scanner): is the first phase of a compiler and it is a process where analysis of the source program by reading the input, character by character, and grouping characters into individual words and symbols (tokens) and correlate error messages (Lexical error) from the compiler with the source program.

2-Token: is a group of characters having a collective meaning to the compiler, for instance identifiers, reserved words, integers, doubles or floats, delimiters, operators, and special symbols.

3-Lexical Error: is an error which detected by the scanner recognizing the lexical tokens of the language such as misspelling an identifier, keyword or operator.

4-Hand Coded Scanner (Loop and Switch): is a method for implementing scanner it consists of a main loop that reads characters one by one from the input file and uses a switch statement to process the character(s) just read. The output is a list of tokens and lexemes (strings) from the source program.

<u>(2) Parser Phase</u>

1-Syntax analysis (Parser): is a second phase of compiler and it is a process where the parser reads tokens and groups them into units as specified by the productions of the **Context Free Grammar** being used. The jobs of parser are verifying that the input program is syntactically correct, in the case of a correct program, beginning to construct a syntax tree (**abstract syntax tree**) for it and in the case of wrong program must correlate error messages (**Syntax error**).

2-Context Free Grammar: G=(N , T , P , S) is defined by N: a finite set non-terminals, T: finite set terminals (each non-terminals stands for sets of strings).
P: finite set of production rules and a set of rules is the core component of a grammar, S: start symbol from non-terminals.

3- Recursive Descent Parser: is a parsing technique traces out a parse tree in top-down order; it is a top-down parser and is built directly from the grammar rules and it consists of a set of functions, one for each terminal and each non-terminal.

4- Each **non-terminal** in the grammar has a subprogram (Function) associated with it; the subprogram parses all sentential forms that the nonterminal can generate.

5- Each **terminal** in the grammar has a subprogram (Function) associated with it; the subprogram compares the grammar terminal with the input token from scanner.

6-Syntax Error: detected by the parser building the parser tree from a program

---

3- Problems Appear in Grammar

1-Common Prefix: a grammar has a common prefix problem if there exist a non-terminal A such that it contains production rule with common prefixes

<A>::= αβ1 | αβ2 | αβ3 | ………. | αβn , where α ≠ β.

2-Left-Recursion: if in a grammar ⅎ A € N $=>^+$ A α for some string α € $(N\ U\ T)^+$

Fixed Grammar

<sample-parser> ::= <program> EOF.

<program> ::= program <id> { <states>} .

<states> ::= <state> | <states> ; <state> .

<state> ::= read <id> | write <expr-seq> | <id> <state-trail>

| if (<expr-seq>) { <states> }.

<state-tail> ::= <op> <expr-seq> | = (<expr-seq> ? <expr-seq> : <expr-seq>).

<expr-seq> ::= <term> <expr-0> | new <id> <expr-2> | ! <expr-seq>

| (<expr-seq>).

<expr-0> ::= { <expr-1> }.

<expr-1> ::= <op> <expr-seq> | [<expr-seq>] | . length.

<expr-2> ::= [<expr-seq>] | ().

<term> ::= <id> | <integer-literal>. **<op> ::= += | *= | &&=.**

<id> ::= <letter> | <id> <letter> | <id> <digit>.

<letter> ::= a | b| c| d| e| f| g| h| i| j| k| l| m| n| o| p| q| r| s| t| u| v| w| x| y| z .

<digit> ::= 0 |1 |2 |3 |4 |5 |6 |7 |8 |9 .

Main functions in Program

1- get_token(void): this is a main function which used in the two phases scanner and parser, it reads text file character by character and output one token at a time.

If they match, continue by return equivalent token, else there is a lexical_error.

2- check_reserved(string s): used in the two phases scanner and parser, it take string as a parameter this function compare between the string and the reserved word in the grammar and return the equivalent token of the string.

3- Match(Token t): this is a main function which used in parser phase, compares the grammar terminal (t) with the current input token (current_token) from scanner. If they match, continue by calling get_token, else there is a syntax_error.

4- Name(Token t): used in the two phases scanner and parser, it takes token as a parameter and return the equivalent string of this token, use it also in display.

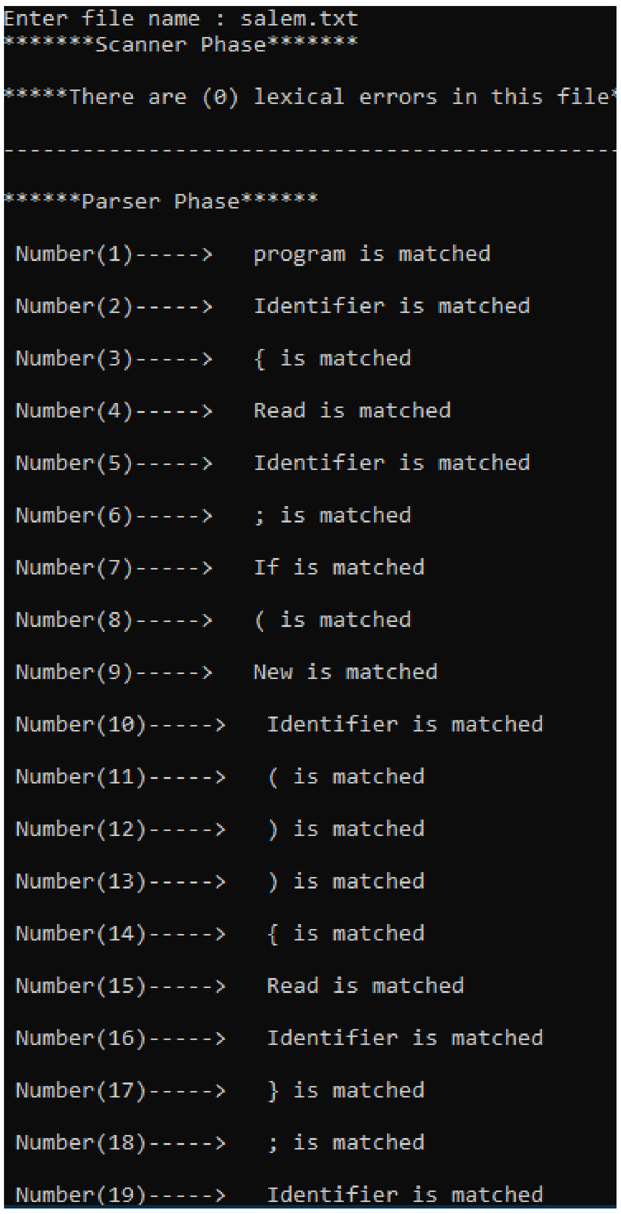
5- ScannerCheck(void): used in the scanner phase, this function checks the current token and if it error_sy token return lexical error and return name of error.

6- Syntax_Error(Token t): used in the parser phase, it takes token as a parameter. This function is called when there are syntax errors in the tested code and this function return syntax error and the number of this error and the name of the error.

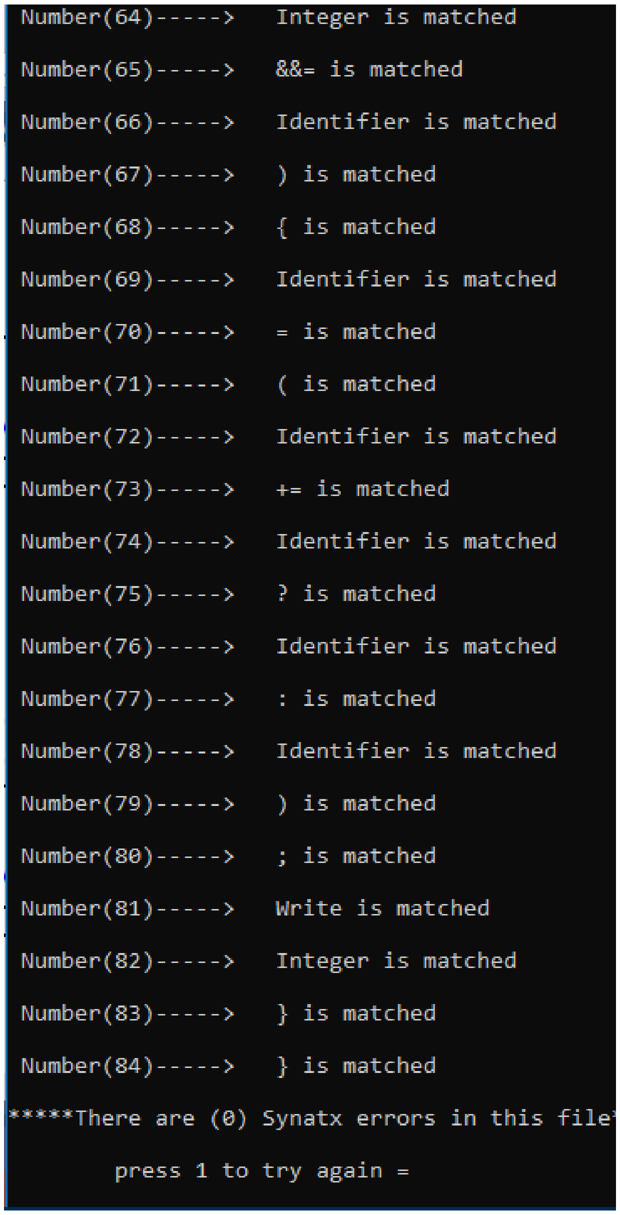
Project Report

Test (1)

```
program vvvv
{
read xxx ;
if (new jjjj () ) { read iii };
sss=( gggg ? jjj : oqqooo ) ;
write kkkk [ffff] ;
if (! ccccc += 9090 ) { read wwww ; rrrr &&= ! llll };
write ( new oooo () );
if ( 7777 += 4747 &&= xxxxx ) { ddodd = (9009 *= loaa ? uuuuu : nnnnn); write 8484 }
}
$
```

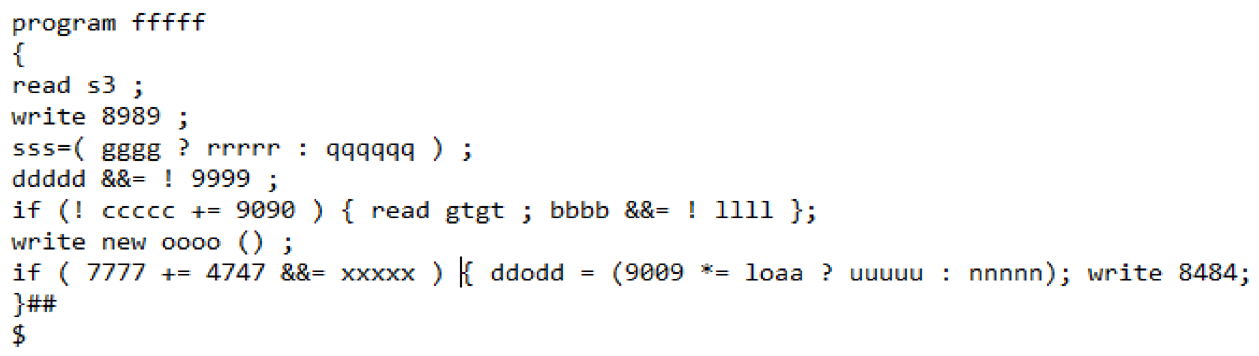


```
Enter file name : salem.txt
*******Scanner Phase*******

*****There are (0) lexical errors in this file*
----------------------------------------------

******Parser Phase******

 Number(1)----->   program is matched

 Number(2)----->   Identifier is matched

 Number(3)----->   { is matched

 Number(4)----->   Read is matched

 Number(5)----->   Identifier is matched

 Number(6)----->   ; is matched

 Number(7)----->   If is matched

 Number(8)----->   ( is matched

 Number(9)----->   New is matched

 Number(10)----->   Identifier is matched

 Number(11)----->   ( is matched

 Number(12)----->   ) is matched

 Number(13)----->   ) is matched

 Number(14)----->   { is matched

 Number(15)----->   Read is matched

 Number(16)----->   Identifier is matched

 Number(17)----->   } is matched

 Number(18)----->   ; is matched

 Number(19)----->   Identifier is matched
```

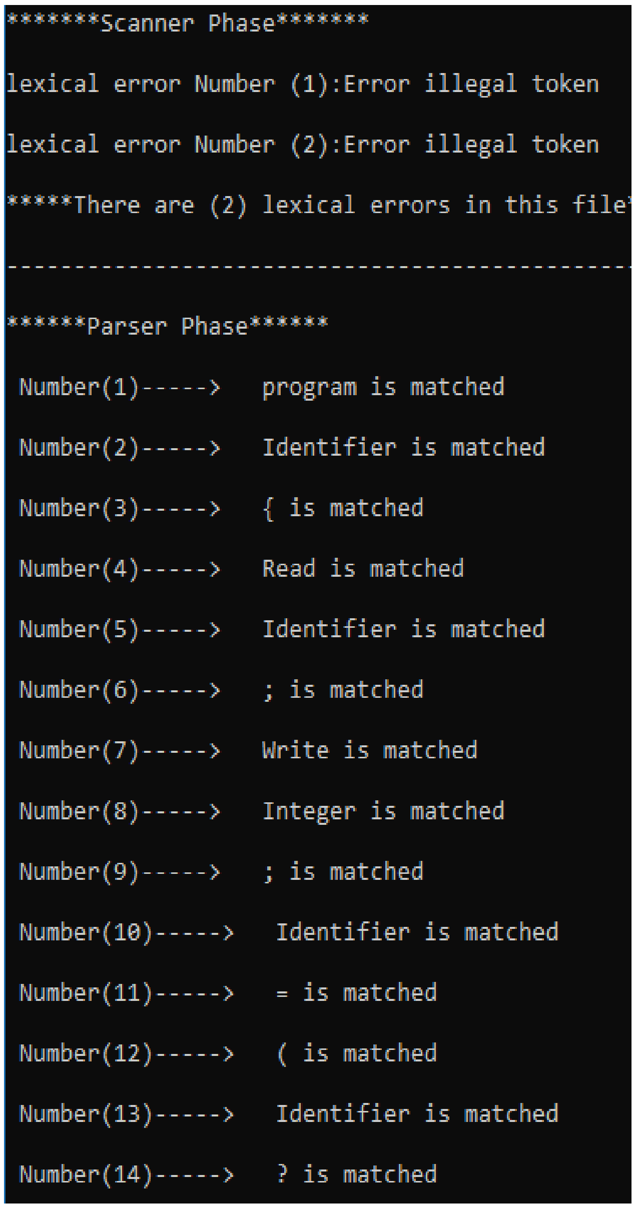


```
 Number(64)----->   Integer is matched

 Number(65)----->   &&= is matched

 Number(66)----->   Identifier is matched

 Number(67)----->   ) is matched

 Number(68)----->   { is matched

 Number(69)----->   Identifier is matched

 Number(70)----->   = is matched

 Number(71)----->   ( is matched

 Number(72)----->   Identifier is matched

 Number(73)----->   += is matched

 Number(74)----->   Identifier is matched

 Number(75)----->   ? is matched

 Number(76)----->   Identifier is matched

 Number(77)----->   : is matched

 Number(78)----->   Identifier is matched

 Number(79)----->   ) is matched

 Number(80)----->   ; is matched

 Number(81)----->   Write is matched

 Number(82)----->   Integer is matched

 Number(83)----->   } is matched

 Number(84)----->   } is matched

*****There are (0) Synatx errors in this file*

        press 1 to try again =
```

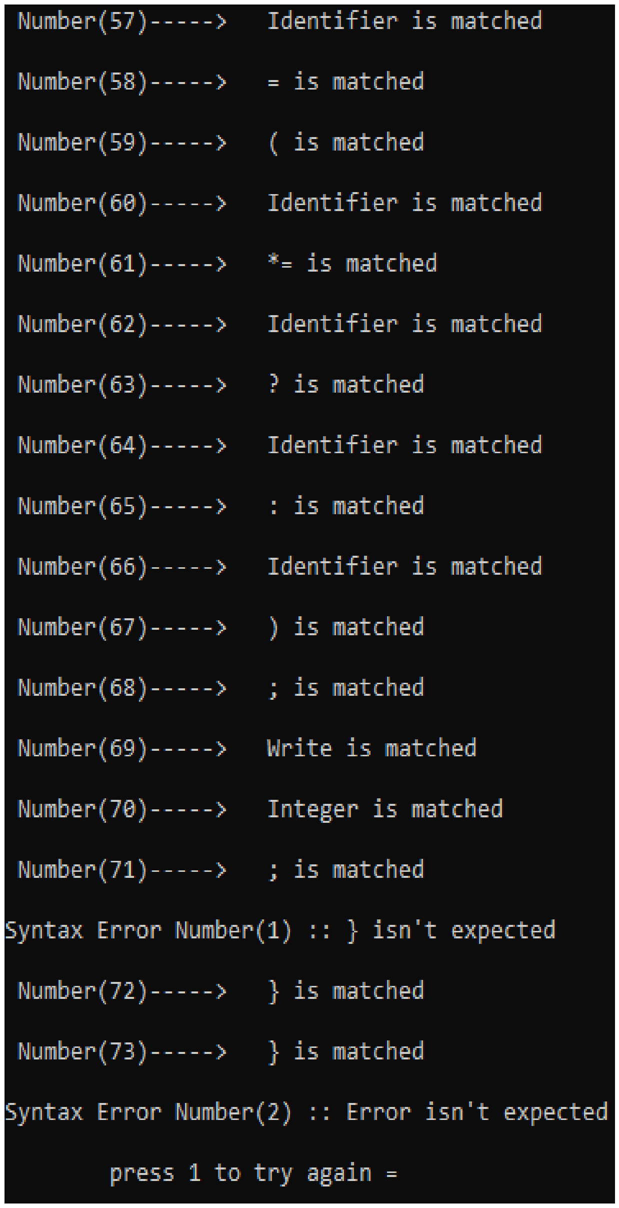
Test (2)



```
program fffff
{
read s3 ;
write 8989 ;
sss=( gggg ? rrrrr : qqqqqq ) ;
ddddd &&= ! 9999 ;
if (! ccccc += 9090 ) { read gtgt ; bbbb &&= ! 1111 };
write new oooo () ;
if ( 7777 += 4747 &&= xxxxx ) { ddodd = (9009 *= loaa ? uuuuu : nnnnn); write 8484;
}##
$
```



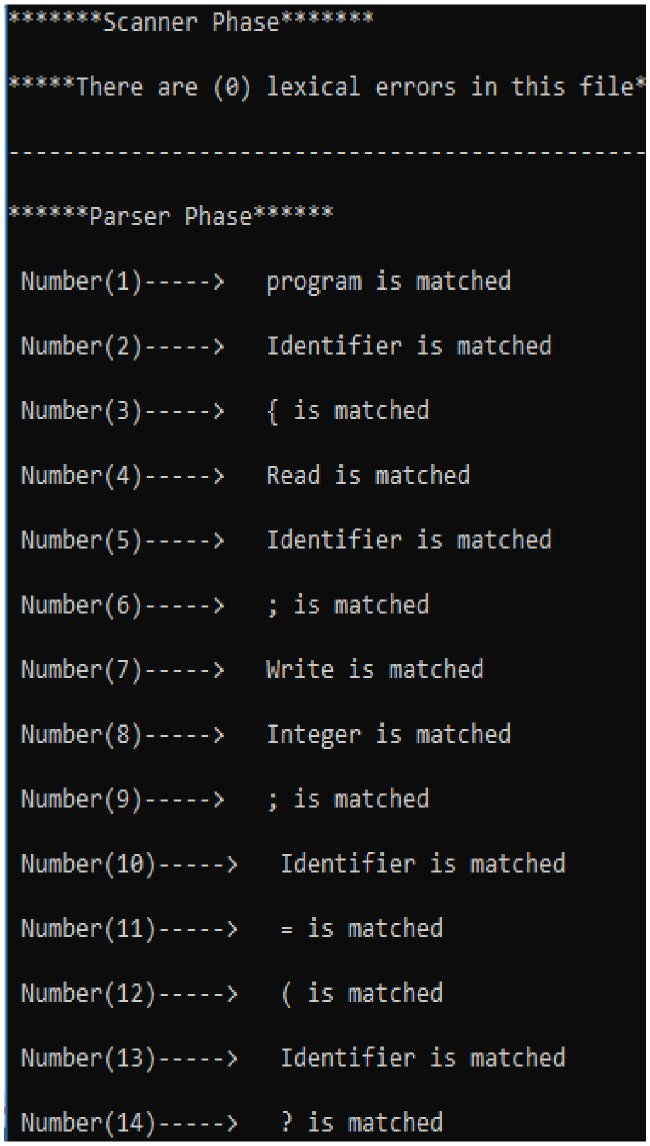
```
*******Scanner Phase*******

lexical error Number (1):Error illegal token

lexical error Number (2):Error illegal token

*****There are (2) lexical errors in this file

-----------------------------------------------

******Parser Phase******

 Number(1)----->   program is matched

 Number(2)----->   Identifier is matched

 Number(3)----->   { is matched

 Number(4)----->   Read is matched

 Number(5)----->   Identifier is matched

 Number(6)----->   ; is matched

 Number(7)----->   Write is matched

 Number(8)----->   Integer is matched

 Number(9)----->   ; is matched

 Number(10)----->   Identifier is matched

 Number(11)----->   = is matched

 Number(12)----->   ( is matched

 Number(13)----->   Identifier is matched

 Number(14)----->   ? is matched
```



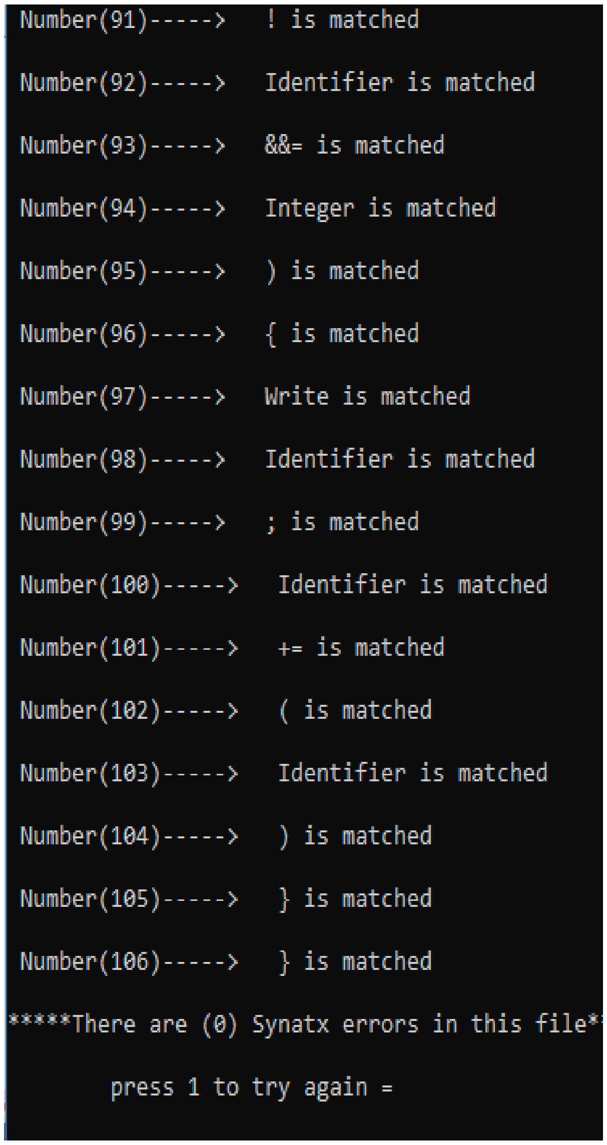
```
 Number(57)----->   Identifier is matched

 Number(58)----->   = is matched

 Number(59)----->   ( is matched

 Number(60)----->   Identifier is matched

 Number(61)----->   *= is matched

 Number(62)----->   Identifier is matched

 Number(63)----->   ? is matched

 Number(64)----->   Identifier is matched

 Number(65)----->   : is matched

 Number(66)----->   Identifier is matched

 Number(67)----->   ) is matched

 Number(68)----->   ; is matched

 Number(69)----->   Write is matched

 Number(70)----->   Integer is matched

 Number(71)----->   ; is matched

Syntax Error Number(1) :: } isn't expected

 Number(72)----->   } is matched

 Number(73)----->   } is matched

Syntax Error Number(2) :: Error isn't expected

        press 1 to try again =
```

Test (3)

```
program fffff
{
read s3 ; write 8989 ; sss=( gggg ? rrrrr : qqqqqq ) ;
ddddd &&= ! 9999 ; if (! ccccc += 9090 ) { read gtgt ; bbbb &&= ! 1111 };
write new oooo () ; if ( 7777 += 4747 &&= xxxxx ) { ddodd *= 9009 &&= loaa };
read plplp ; plpl = ( new oko [ 1000 ] ? jjjjj &&= moi : 34567);
samy *= ( 10000 ) ; if (! kpokp &&= 8888 ) { write zzzz ; yyyy += ( nnnn ) }
}
$
```



```
*******Scanner Phase*******

*****There are (0) lexical errors in this file*

----------------------------------------------

******Parser Phase******

 Number(1)----->   program is matched

 Number(2)----->   Identifier is matched

 Number(3)----->   { is matched

 Number(4)----->   Read is matched

 Number(5)----->   Identifier is matched

 Number(6)----->   ; is matched

 Number(7)----->   Write is matched

 Number(8)----->   Integer is matched

 Number(9)----->   ; is matched

 Number(10)----->   Identifier is matched

 Number(11)----->   = is matched

 Number(12)----->   ( is matched

 Number(13)----->   Identifier is matched

 Number(14)----->   ? is matched
```



```
Number(91)----->   ! is matched

Number(92)----->   Identifier is matched

Number(93)----->   &&= is matched

Number(94)----->   Integer is matched

Number(95)----->   ) is matched

Number(96)----->   { is matched

Number(97)----->   Write is matched

Number(98)----->   Identifier is matched

Number(99)----->   ; is matched

Number(100)----->   Identifier is matched

Number(101)----->   += is matched

Number(102)----->   ( is matched

Number(103)----->   Identifier is matched

Number(104)----->   ) is matched

Number(105)----->   } is matched

Number(106)----->   } is matched

*****There are (0) Synatx errors in this file*

        press 1 to try again =
```