

CS323 Programming Assignments

1. Names [1. **Alex Ho**], (MW [**x**] or R class [])
2. Assignment Number [2]
3. Due Dates **Softcopy** [11/12 (Mon), 11:59 pm], **Hardcopy** [11/14 (Wed) by 5 pm]
4. Turn-In Dates **Softcopy** [11/11 (Sun)], **Hardcopy** [11/14 (Wed)]
5. Executable FileName [CompilerRAT18F.exe]
(A file that can be executed without compilation by the instructor)
6. LabRoom [CS 408]
(Execute your program in a lab in the CS building before submission)
7. Operating System [Window 10 64bit]

To be filled out by the Instructor:

GRADE:

COMMENTS:

CS323 Compiler Project Documentation

I. Problem Statement

1. Rewrite the grammar *Rat18F* to remove any left recursion
(Also, use left factorization if necessary)
2. Use the **lexer()** generated in the assignment 1 to get the tokens
3. **The parser should print to an output file the tokens, lexemes and the production rules used;**

That is, first, write the token and lexeme found

Then, print out all productions rules used for analyzing this token

Note: - a simple way to do it is to have a “print statement” at the beginning of each function that will print

the production rule.

- It would be a good idea to have a “switch” with the “print statement” so that you can turn it on or off.

4. **Error handling:** *if a syntax error occurs, your parser should generate a meaningful error message, such as token, lexeme, line number, and error type etc.*

Then, your program may exit or you may continue for further analysis.

The bottom line is that your program must be able to parse the entire program if it is syntactically correct.

5. **Turn in your assignment according to the specifications given in the project outline**

Example

Assume we have the following statement

```
....more ....  
a = b + c;  
.... more ....
```

One possible output would be as follows:

.... more....

Token: Identifier Lexeme: a

<Statement> -> <Assign>

<Assign> -> <Identifier> = <Expression> ;

Token: Operator Lexeme: =

Token: Identifier Lexeme: b

<Expression> -> <Term> <Expression Prime>

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle \langle \text{Term Prime} \rangle$

$\langle \text{Factor} \rangle \rightarrow \langle \text{Identifier} \rangle$

Token: Operator Lexeme: +

$\langle \text{Term Prime} \rangle \rightarrow ($

$\langle \text{Expression Prime} \rangle \rightarrow + \langle \text{Term} \rangle \langle \text{Expression Prime} \rangle$

Token: Identifier Lexeme: c

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle \langle \text{Term Prime} \rangle$

$\langle \text{Factor} \rangle \rightarrow \langle \text{Identifier} \rangle$

Token: Separator Lexeme: ;











$\langle \text{Term Prime} \rangle \rightarrow ($

$\langle \text{Expression Prime} \rangle \rightarrow ($

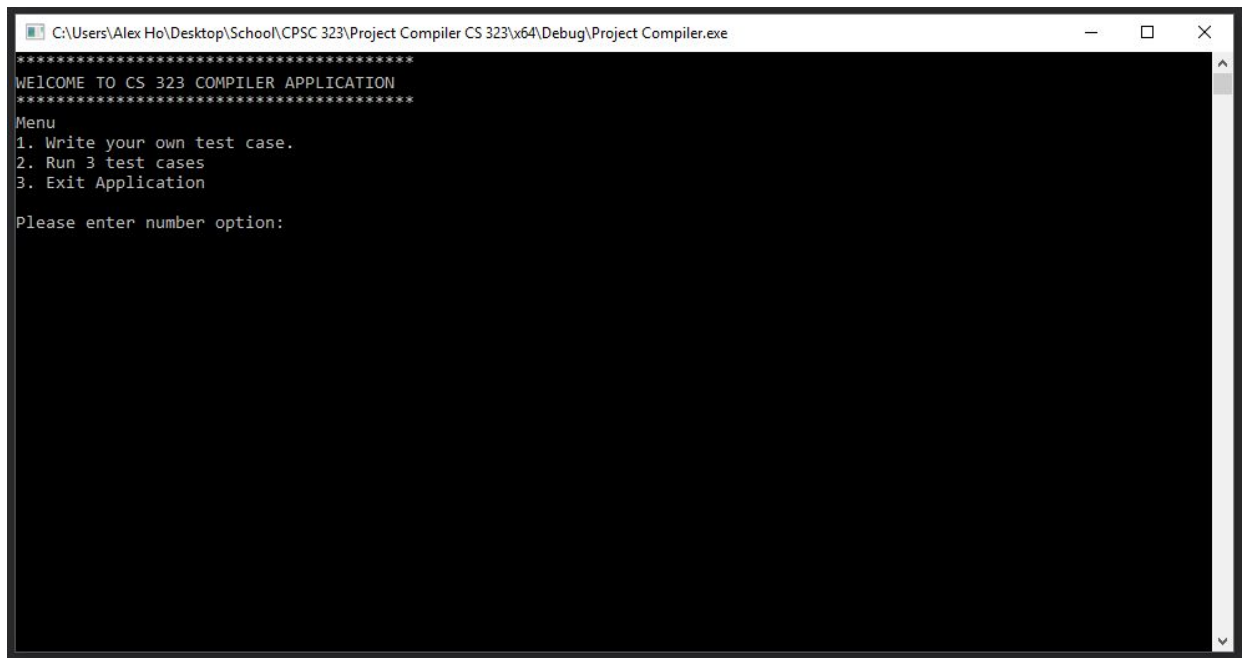
.... more.....

II. How to use your program

Open the folder “CompilerRAT18F_CS323”, then run the “CompilerRAT18F.exe”

Name	Date modified	Type	Size
 CompilerRAT18F	10/7/2018 12:59 AM	Application	217 KB
 Lexical.cpp	10/7/2018 12:45 AM	C++ Source	9 KB
 Lexical.h	10/7/2018 12:45 AM	C/C++ Header	3 KB
 Main.cpp	10/7/2018 12:10 AM	C++ Source	3 KB
 Output	10/7/2018 1:06 AM	Text Document	5 KB
 test1	10/1/2018 1:39 AM	Text Document	1 KB
 test2	9/30/2018 11:54 PM	Text Document	1 KB
 test3	9/30/2018 11:54 PM	Text Document	1 KB
 Token.cpp	10/6/2018 11:49 PM	C++ Source	1 KB
 Token.h	10/6/2018 11:49 PM	C/C++ Header	1 KB

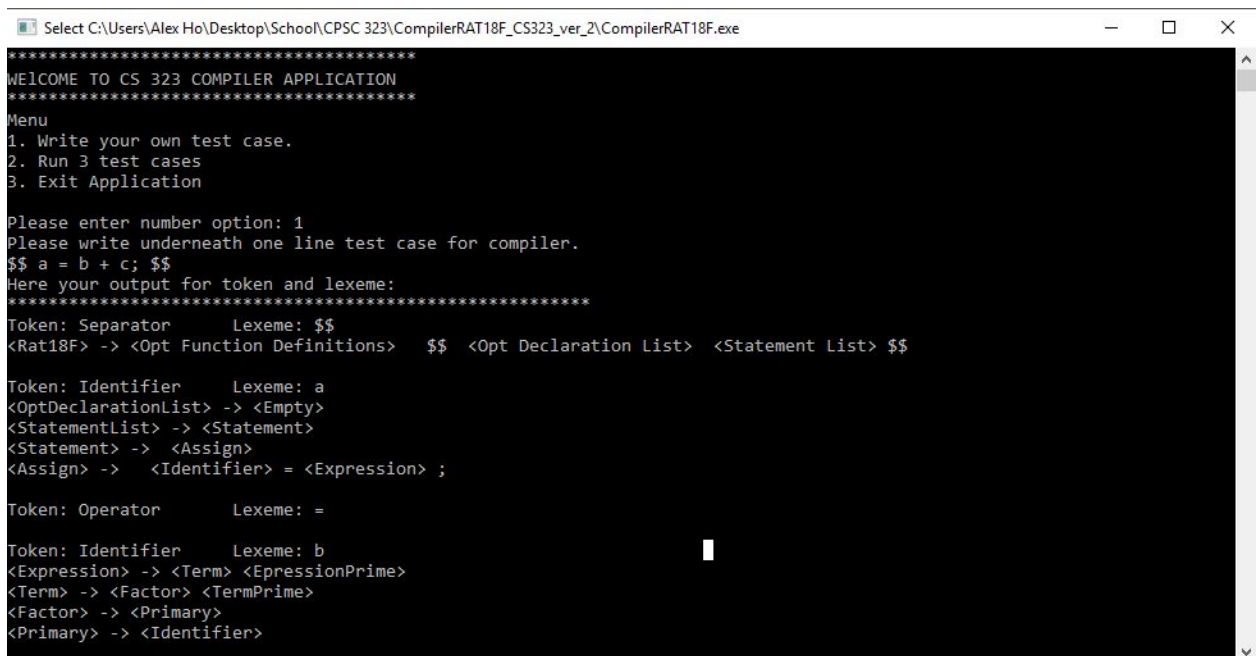
The Window Program will show Menu three Options as the picture



```
C:\Users\Alex Ho\Desktop\School\CPSC 323\Project Compiler CS 323\x64\Debug\Project Compiler.exe
*****
WELCOME TO CS 323 COMPILER APPLICATION
*****
Menu
1. Write your own test case.
2. Run 3 test cases
3. Exit Application
Please enter number option:
```

Option 1:

You can write down your own one-line test case.

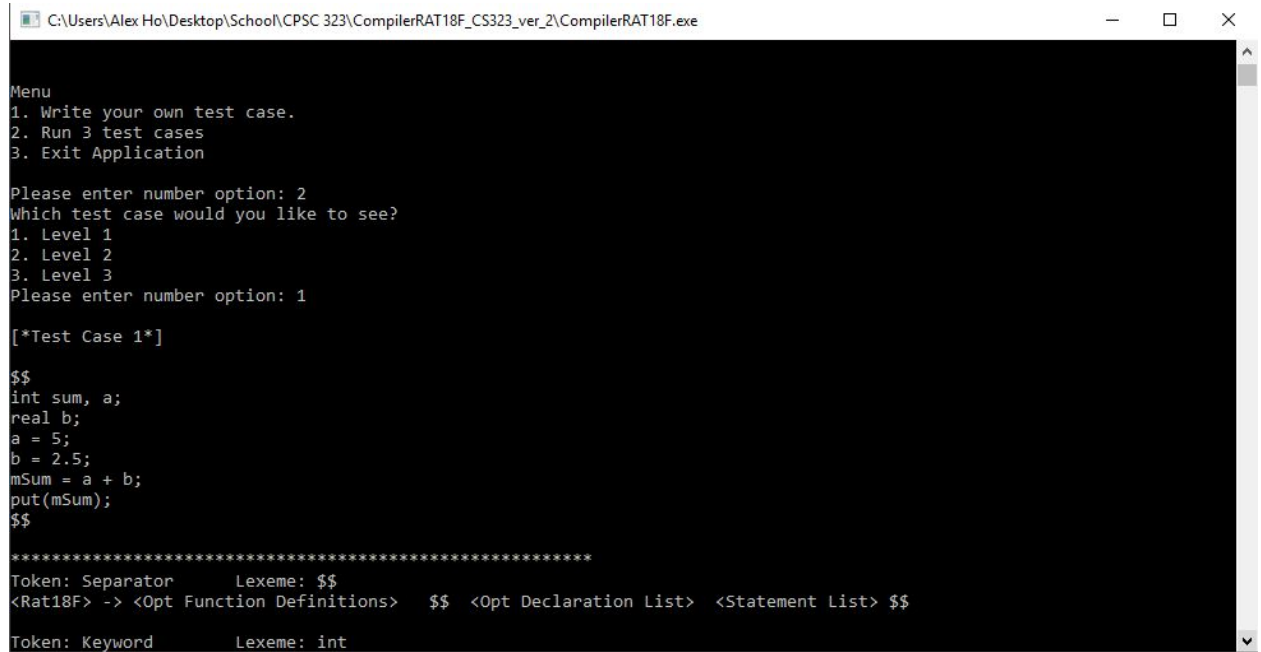


```
Select C:\Users\Alex Ho\Desktop\School\CPSC 323\CompilerRAT18F_CS323_ver_2\CompilerRAT18F.exe
*****
WELCOME TO CS 323 COMPILER APPLICATION
*****
Menu
1. Write your own test case.
2. Run 3 test cases
3. Exit Application
Please enter number option: 1
Please write underneath one line test case for compiler.
$$ a = b + c; $$
Here your output for token and lexeme:
*****
Token: Separator      Lexeme: $$
<Rat18F> -> <Opt Function Definitions>  $$ <Opt Declaration List> <Statement List> $$
Token: Identifier     Lexeme: a
<OptDeclarationList> -> <Empty>
<StatementList> -> <Statement>
<Statement> -> <Assign>
<Assign> -> <Identifier> = <Expression> ;
Token: Operator       Lexeme: =
Token: Identifier     Lexeme: b
<Expression> -> <Term> <EpressionPrime>
<Term> -> <Factor> <TermPrime>
<Factor> -> <Primary>
<Primary> -> <Identifier>
```

Option 2:

The program will input file test case for compiler lexer and output the result on screen as well as the output file text

You can choose any level from 1 to 3.



```
C:\Users\Alex Ho\Desktop\School\CPSC 323\CompilerRAT18F_CS323_ver_2\CompilerRAT18F.exe

Menu
1. Write your own test case.
2. Run 3 test cases
3. Exit Application

Please enter number option: 2
Which test case would you like to see?
1. Level 1
2. Level 2
3. Level 3
Please enter number option: 1

[*Test Case 1*]

$$
int sum, a;
real b;
a = 5;
b = 2.5;
mSum = a + b;
put(mSum);
$$

*****
Token: Separator      Lexeme: $$
<Rat18F> -> <Opt Function Definitions>  $$ <Opt Declaration List> <Statement List> $$
Token: Keyword        Lexeme: int
```

Option 3:

Exit program and “thank you” message.

To open output file text “Output.txt”, go to Compiler folder. It must be there after you run compiler program

Name	Date modified	Type	Size
CompilerRAT18F	10/7/2018 12:59 AM	Application	217 KB
Lexical.cpp	10/7/2018 12:45 AM	C++ Source	9 KB
Lexical.h	10/7/2018 12:45 AM	C/C++ Header	3 KB
Main.cpp	10/7/2018 12:10 AM	C++ Source	3 KB
Output	10/7/2018 1:12 AM	Text Document	1 KB
test1	10/1/2018 1:39 AM	Text Document	1 KB
test2	9/30/2018 11:54 PM	Text Document	1 KB
test3	9/30/2018 11:54 PM	Text Document	1 KB
Token.cpp	10/6/2018 11:49 PM	C++ Source	1 KB
Token.h	10/6/2018 11:49 PM	C/C++ Header	1 KB

Output file should show something similar to the picture below.

```

Output - Notepad
File Edit Format View Help
Token: Separator      Lexeme: $$
<Rat18F> -> <Opt Function Definitions>  $$  <Opt Declaration List>  <Statement List> $$

Token: Identifier     Lexeme: a
<OptDeclarationList> -> <Empty>
<StatementList> -> <Statement>
<Statement> -> <Assign>
<Assign> ->  <Identifier> = <Expression> ;

Token: Operator       Lexeme: =

Token: Identifier     Lexeme: b
<Expression> -> <Term> <EpressionPrime>
<Term> -> <Factor> <TermPrime>
<Factor> -> <Primary>
<Primary> -> <Identifier>

Token: Operator       Lexeme: +
<TermPrime> -> <Empty>
<ExpressionPrime> ->  + <Term> <ExpressionPrime>

Token: Identifier     Lexeme: c
<Term> -> <Factor> <TermPrime>
<Factor> -> <Primary>
<Primary> -> <Identifier>

```

Note: To make a new test case by a different text file, you have to add a new text file into the directory of the project by replacing old file or modify any test case file.

III. Design of your program

The project has been modified to fulfill the second assignment, the following points are modified or added to the project:

- ❖ Main
 - ❖ Create a new SyntaxAnalyzer object with an argument that is a list from the lexical analyzer
 - ❖ Using print function from SyntaxAnalyzer
- ❖ Token Class
 - ❖ Add one more overload constructor with rule string.
 - ❖ Add overload operator "cout" for token object
- ❖ Lexical Class
 - ❖ Reduce keywords array to 14 in total
 - ❖ Add getListToken function (return the list of lexical analyzer)
 - ❖ Add destructor to delete the list.
- ❖ SysntaxAnalyzer Class (New)
 - ❖ Variable:
 - mProductionRule: a string contains current production rule
 - mCurrentTokenOject: a token object is in the mListToken list
 - mCurrentLexeme: a string contains a lexeme of current token object
 - mCurrentToken: a string contains a token of current token object
 - mListToken: a queue list is used to receive the list from lexical
 - mListTokenSyntax: a queue list of token after syntax analyzer
 - ❖ Functions:
 - SyntaxAnalyzer(): default constructor; initial mProductionRule, mCurrentToken, and mCurrentLexeme to empty string.
 - SyntaxAnalyzer(queue <Token>list): overload constructor; receive parameter is a list from lexical analyzer then transfer tokens to mListToken as well as get and pop the first token in the list.
 - errorMessage(string typeSyntax, string rules):
 - Parameter: a string of type syntax error, a string of name rule
 - Print the error line number, token, lexeme, what missing and rule name
 - return nothing but back to menu.
 - nextToken()
 - Set production rule for the mCurrentTokenOject
 - Insert mCurrentTokenOject to the mListTokenSyntax
 - Assign mProductionRules to empty string
 - mCurrentTokenOject to the next token in the mListToken and pop mListToken.
 - Change mCurrentToken, mCurrentLexeme.
 - Handle error if list empty then print the error message
 - printListSyntax()

- Create or open Output file to write out
- Print the list syntax token object include token, lexeme, and production rules in the console and write out the file
- Close the file
- 31 functions of syntax rules, these functions neither have any parameter nor return.

How to create syntax rule functions?

Top Down Parser method

This project will apply the *Predictive Recursive Descent Parser (PRDP) method*


1. Removing left recursive and backtracking in the syntax rule list
 - ❖ None backtracking
 - ❖ Two left recursive will be replaced by new rules:
 - $\langle \text{Expression} \rangle ::= \langle \text{Expression} \rangle + \langle \text{Term} \rangle \mid \langle \text{Expression} \rangle - \langle \text{Term} \rangle \mid \langle \text{Term} \rangle$
 - $\langle \text{Expression} \rangle ::= \langle \text{Term} \rangle \langle \text{ExpressionPrime} \rangle$
 - $\langle \text{ExpressionPrime} \rangle ::= + \langle \text{Term} \rangle \langle \text{ExpressionPrime} \rangle \mid - \langle \text{Term} \rangle \langle \text{ExpressionPrime} \rangle \mid \langle \text{Empty} \rangle$
 - $\langle \text{Term} \rangle ::= \langle \text{Term} \rangle * \langle \text{Factor} \rangle \mid \langle \text{Term} \rangle / \langle \text{Factor} \rangle \mid \langle \text{Factor} \rangle$
 - $\langle \text{Term} \rangle ::= \langle \text{Factor} \rangle \langle \text{TermPrime} \rangle$
 - $\langle \text{TermPrime} \rangle ::= * \langle \text{Factor} \rangle \langle \text{TermPrime} \rangle \mid / \langle \text{Factor} \rangle \langle \text{TermPrime} \rangle \mid \langle \text{Empty} \rangle$
2. Find the first set for all rules and follow set for any rule with empty on the left-hand side:
 - ❖ First Sets:
 - $\text{First}(\text{Primary}) = \{\text{id, Integer, Real, (, true, false}\}$
 - $\text{First}(\text{Factor}) = \{\text{id, Integer, Real, (, true, false, -}\}$
 - $\text{First}(\text{TermPrime}) = \{*, /, \text{empty}\}$
 - $\text{First}(\text{Term}) = \{\text{id, Integer, Real, (, true, false, -}\}$
 - $\text{First}(\text{ExpressionPrime}) = \{+, -, \text{empty}\}$
 - $\text{First}(\text{Expression}) = \{\text{id, Integer, Real, (, true, false, -}\}$
 - $\text{First}(\text{Relop}) = \{=, ^, >, <\}$
 - $\text{First}(\text{Condition}) = \{\text{id, Integer, Real, (, true, false, -}\}$
 - $\text{First}(\text{While}) = \{\text{while}\}$
 - $\text{First}(\text{Scan}) = \{\text{get}\}$
 - $\text{First}(\text{Print}) = \{\text{put}\}$
 - $\text{First}(\text{Return}) = \{\text{return}\}$
 - $\text{First}(\text{If}) = \{\text{if}\}$
 - $\text{First}(\text{Assign}) = \{\text{id}\}$
 - $\text{First}(\text{Compound}) = \{ \{ \}$
 - $\text{First}(\text{Statement}) = \{ \text{while, get, put, return, if, \{, id} \}$

- First(StatementList) = { while, get, put, return, if, {, id }
 - First(IDs) = {id}
 - First(Declaration) = {“int”, “boolean”, “real” }
 - First(DeclarationList) = {“int”, “boolean”, “real” }
 - First(OPTDeclarationList) = {“int”, “boolean”, “real”, empty }
 - First(Body) = { { }
 - First(Qualifier) = { “int”, “boolean”, “real” }
 - First(Parameter) = {id}
 - First(ParameterList) = {id}
 - First(OPTParameterList) = {id, empty}
 - First(Funfunction) = {function}
 - First(FunfunctionDefinition) = {function}
 - First(OPTFunfunctionDefinition) = {function, empty}
 - First(Rat18F) = { First(OPTFunfunctionDefinition), \$ }
 - ❖ Follow Sets:
 - Follow(OptFunctionDefinition) = { \$ }
 - Follow(OptParameterList) = { } }
 - Follow(OptDeclarationList) = {while, get, put, return, if, {, id}
 - Follow(ExpressionPrime) = { =, ^, >, <,), ; }
 - Follow(TermPrime) = { +, -, =, ^, >, <,), ; }
3. Create Predictive Recursive Descent Parser of syntax rule functions with First and Follow Sets.
- ❖ mProductionRules will be added one more string after going down one rule.
 - ❖ If the nextToken function is called, mProductionRules will be reset to empty.
 - ❖ If there is an error, the function rule will call the errorMessage function to display the message and stop the process.

The new Syntax rules have 31 in total

Syntax rules : The following BNF describes the Rat18F.

- R1. <Rat18F> ::= <Opt Function Definitions> \$\$ <Opt Declaration List> <Statement List> \$\$
- R2. <Opt Function Definitions> ::= <Function Definitions> | <Empty>
- R3. <Function Definitions> ::= <Function> | <Function> <Function Definitions>
- R4. <Function> ::= function <Identifier> (<Opt Parameter List>) <Opt Declaration List> <Body>
- R5. <Opt Parameter List> ::= <Parameter List> | <Empty>
- R6. <Parameter List> ::= <Parameter> | <Parameter> , <Parameter List>
- R7. <Parameter> ::= <IDs> : <Qualifier>
- R8. <Qualifier> ::= int | boolean | real
- R9. <Body> ::= { <Statement List> }
- R10. <Opt Declaration List> ::= <Declaration List> | <Empty>
- R11. <Declaration List> ::= <Declaration> ; | <Declaration> ; <Declaration List>
- R12. <Declaration> ::= <Qualifier> <IDs>

R13. <IDs> ::= <Identifier> | <Identifier>, <IDs>
 R14. <Statement List> ::= <Statement> | <Statement> <Statement List>
 R15. <Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Print> | <Scan> | <While>
 R16. <Compound> ::= { <Statement List> }
 R17. <Assign> ::= <Identifier> = <Expression>;
 R18. <If> ::= if (<Condition>) <Statement> ifend |
 if (<Condition>) <Statement> else <Statement> ifend
 R19. <Return> ::= return ; | return <Expression>;
 R20. <Print> ::= put (<Expression>);
 R21. <Scan> ::= get (<IDs>);
 R22. <While> ::= while (<Condition>) <Statement> whileend
 R23. <Condition> ::= <Expression> <Relop> <Expression>
 R24. <Relop> ::= == | ^= | > | < | => | ==<
 R25. <Expression> ::= <Term> <ExpressionPrime>
 R26. <ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> |
 <Empty>
 R27. <Term> ::= <Factor> <TermPrime>
 R28. <TermPrime> ::= * <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>
 R29. <Factor> ::= - <Primary> | <Primary>
 R30. <Primary> ::= <Identifier> | <Integer> | <Identifier> (<IDs>) | (<Expression>) |
 <Real> | true | false
 R31. <Empty> ::= 

The Table has been reduced some keywords

Table 1.1 Keywords, operators, and separators

	Keywords	Operators	Separators
1	function	=	\$\$
2	int	+	,
3	real	-	:
4	boolean	*	;
5	if	/	
6	else	=>	(
7	ifend	=<)
8	while	^=	{
9	return	==	}
10	get	>	

11	put	<	
12	whileend		
13	false		
14	true		

Note: Keywords are case sensitive

IV. Any Limitation

- ❖ Keyword, Separators, Operators must be matched in Table 1.1, otherwise, it will be defined as unknown
- ❖ The letter and digit or real number must be matched with Regular Expression, otherwise, it will be defined as unknown.
- ❖ Example: input “23.” instead of “23.0”, “23.” will be defined as unknown
- ❖ The source codes have to match the production rules otherwise it will get the error.

V. Any shortcomings

None