Introduction to Compiler
Fall 2017
Programming Assignment 1
(A recursive descent parser)

Due: before class, Oct. 31, 2017

You are to complete the following two tasks :

1. Implement a **recursive descent parser** for the following grammar. The parser will translate infix Boolean expressions to postfix Virtual Stack Machine intermediate code and dumps (print out) all identifiers in the symbol table.
2. (Bonus 30%) Implement an **interpreter** for the Virtual Stack Machine and Print out the final values of each variable in the variable table.

You may write your programs in C, C++, or Java.

Task 1 Recursive Descent Parser (required)

The following attributed translation grammar translates infix Boolean expressions into postfix stack machine code. You are to write a recursive descent parser for it. User defined identifiers should be inserted into the symbol table when the first time they appear. The action routine *dumpIDs()* refers to printing out all the identifiers at the symbol table. Error handling is highly recommended, but panic mode for errors is acceptable. You may assume the language is case-sensitive. (大小寫有差)

1. start → stmtseq eof {dumpIDs()}
2. stmtseq→ assignment stmtseqlist
3. stmtseqlist → ; assignment stmtseqlist | ∈
4. assignment → id {emit_code (‘push lvalue’ id.lexeme) } = expr {emit_code(“=”)}
5. expr → term moreterms
6. moreterms → 'or' term { emit_code (“or”)} moreterms
7. moreterms → ∈
8. term → factor morefactors
9. morefactors → 'and' factor {emit_code (“and”)} morefactors
10. morefactors →∈
11. factor ->(expr)
12. factor -> true{ emit_code (“ push true") }
13. factor -> false { emit_code (“ push false") }
14. 12 factor--> id { emit_code (“ push rvalue ” id.lexeme) }

For example, the following input will generate the stack code listed below:

Input:

```
  X =true;
  Y = (false or X) and X
```

Parser Output:

```
push lvaue    X
push true
=
push lvalue Y
push false
push rvalue X
or
push rvalue X
and
  =
%% IDs:    X      Y
```

Task 2 (Extra Credit) : Interpreter for the stack machine code

You are to write an interpreter for the stack machine code generated above. You need ***a stack*** to keep the pushed items. Besides, you also need ***a variable table*** to keep identifier's name and value. At the end of the interpretation, the variable table's contents will be print out. For example, the output of the above sample will be:

*Interpreter results (output the **variable table**)*

```
X: true
Y: true
```

Interpreter actions

For *"push"* operation, the interpreter pushes the item **<** ***type, value*****>** into its stack.

For operations '*and*' and *'or*', the interpreter will act as follows:

{ pop(operand2); pop(operand1); result = operand1 operation operand2; push result)}.

For operation "=", the interpreter will perform:

*{ pop(rhs); pop(lhs);   assign the value rhs to lhs's slot in the **variable table** }}.*

Test Data for your parser

(1)

```
X = true and (false or false);
Y = false and true or true;
Cat = (false or true) and X or (false or true );
```

X = X or Y or Cat

(2)
sea = false;
fish = true;
Dog= true or fish of sea

<u>Test Data for the interpreter (Bonus)</u>

The output from your parser

<u>Reminder 提醒</u>

Good programming style and documentation are required. (須註解—含文法規則，須用有意義變數名) Hand in a hard copy of your parser and interpreter, as well as your test run with results for both programs. Moreover, upload your source program and executable code to Portal. (繳交程式碼與執行畫面並上傳程式碼及執行檔至portal)

<u>Hint:</u>

Please refer to the program listing of the recursive descent parser discussed in class. (From Aho’s book version 1)