Compiler Fall 2017
Programming Assignment 2

LR Parser (Flex and Bison)
Due: before class, Dec. 19, 2017

You are to use Flex and Bison to redo the work similar to that of program 1 using the following left recursive grammar.

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

User defined identifiers should be inserted into the symbol table. Dump all the identifiers at the end of your parser output, which is performed by action routine dumpIDs()}

Test your program using the above sample as well as the following two test data sets:

(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

Write your program with good coding standards and documentation. Turn in a hard copy (紙本) of your program along with test runs (執行畫面). Moreover, you also need to upload an electronic copy of your source program along with executable code to the portal homework section.

The following flex and bison specification fragments are given for your reference.

```
/*****************    sample file: xx.l
****************************************/
/* 定義區 */
%{
#include <stdio.h>
#include <stdlib.h>
#include "xx.tab.h" /* 會自動生成的頭檔 */
void yyerror(char *); /* 偵錯函數 */
%}
/* Regular 定義區 */
…..
letter [A-Za-z]
id {letter}({letter}|{digit})*
number {digit}+
……..
/* 規則區 */
%%
">=" return GE;
"<=" return LE;
……..
"true" return TRUE;
……..
{id} { yylval.string = strdup(yytext);
       return VARIABLE;}
{number} { yylval.string = strdup(yytext);
       return INTEGER; }
%%
/* 偵錯函數 */
yywrap(void)
{
return 1;
}


/*****************************************************************/
/******************** file : xx.y ***********************************/
* 初始定義區 */
%{
#include <ctype.h>
#include <stdio.h> /* C內建函數 */
#include <string.h> /* 用來幫數比較字串函數 */
void yyerror(char *); /* 偵錯函數 */
int myfunction (char *); /* 其他需要自己寫的副程式 */
int count;         /*及global 變數*/
%}
/* 保留字定義區 */
%token WRITE TRUE FALSE BLEFT BRIGHT AND OR NOT S
…
```

```
/* yylval type */
%union
{
char *string;
int value;
}
/* 指定token value的 Type */
%token <string> INTEGER
%token <string> VARIABLE
%token<value>
/* 規則定義區 */
%%
program:
    block;
block:
    BLEFT decls stmts BRIGHT
    |;
……..……...
bool:
    bool OR join            { printf("Or\n"); }
    | join;
……
%%
/* 函數定義區 */
/* 偵錯函數 */
void yyerror(char *s)
{
fprintf(stderr, "%s\n", s);
}
/* 主程式函數 */
int main(void)
{
……..; /* 放入初始值 */
yyparse();
return 0;
}

int myfunction(char *temp)
{    …… /*自己寫的副程式*/
……..

}
```