# yacc1

Look at this before you start homework 3

# Simple Boolean Expression Grammar

```
Prog -> StmtSeq
StmtSeq -> Stmt StmtSeq
StmtSeq ->  ε
Stmt -> Id = Expr ;
Expr -> Expr || Term
Expr -> Term
Term -> Term && Factor
Term -> Factor
Factor -> ! Factor
Factor -> ( Expr )
Factor -> Id
Factor -> True
Factor -> False
Id -> Ident
```

# Example Program

```
x = true;
y = false;
w = !x || y;
z = (x && !y) || w;
```

## Implementation of an Interpreter for the Grammar

- An interpreter executes the source program as the program is parsed.
- The interpreter uses the symbol table to remember the values of variables.
- When the source program finishes the interpreter prints the values of the variables to standard output
- This is different from the main project you will do but it will give you experience with lex and yacc

## Implementation of an Interpreter for the Grammar

- lex
- yacc
- semantics
- main
- Building the compiler executable

# boolLex.l

```
%{
    #include "IOMngr.h"
    #include "y.tab.h"

    #define YY_INPUT(buf,result,max_size) \
    { int c = getNextSourceChar(); \
    result = (c == EOF) ? YY_NULL : (buf[0] = c, 1); \
    }

%}


letter [A-Za-z]
digit [0-9]
```

# boolLex.l

```
%%

true                            {return TRUE;}
false                           {return FALSE;}
{letter}({letter}|{digit})*      {return Ident;}
\|\|                            {return OR;}
\&\&                            {return AND;}
\=                              {return '=';}
\!                              {return '!';}
\;                              {return ';';}
\(                              {return '(';}
\)                              {return ')';}
```

# boolLex.l

```
[ ]                  {}
\t                   {}
\r                   {}
\n                   {}

.                   {
                       WriteIndicator(getCurrentColumnNum());
                       WriteMessage("Illegal Character in lex");
                    }

%%

int yywrap () {
    return 1;
}
```

# boolExpr.y

```
%{
#include "semantics.h"
#include "IOMngr.h"
#include <string.h>

extern int yylex();       /* The next token function. */
extern char *yytext;    /* The matched token text.  */
extern int yyerror(char *s);


%}
```

# boolExpr.y

```
%union {
    bool boolean;
    char * string;
}

%type <string> Id
%type <boolean> Expr
%type <boolean> Term
%type <boolean> Factor

%token Ident
%token TRUE
%token FALSE
%token OR
%token AND
```

# boolExpr.y

```
%%

Prog          :   StmtSeq                 {printSymTab();};
StmtSeq       :   Stmt StmtSeq            { };
StmtSeq       :                           { };
Stmt          :   Id '=' Expr ';'         {storeVar($1, $3);};
Expr          :   Expr OR Term            {$$ = doOR($1, $3);};
Expr          :   Term                    {$$ = $1;};
Term          :   Term AND Factor         {$$ = doAND($1, $3);};
Term          :   Factor                  {$$ = $1;};
Factor        :   '!' Factor              {$$ = doNOT($2);};
Factor        :   '(' Expr ')'            {$$ = $2;};
Factor        :   Id                      {$$ = getVal($1);};
Factor        :   TRUE                    {$$ = true;};
Factor        :   FALSE                   {$$ = false;};
Id            :   Ident                   {$$ = strdup(yytext);};

%%
```

# boolExpr.y

```
int yyerror(char *s)  {
    WriteIndicator(getCurrentColumnNum());
    WriteMessage("Illegal Character in YACC");
    return 1;
}
```

## semantics.h

```
#include <stdbool.h>

extern void printSymTab();
extern void storeVar(char * name, bool v);
extern bool doOR(bool v1, bool v2);
extern bool doAND(bool v1, bool v2);
extern bool doNOT(bool v1);
extern bool getVal(char * name);
```

# semantics.c

```
#include <stdio.h>
#include <stdlib.h>
#include “SymTab.h"
#include “semantics.h”


extern struct SymTab *table;

void printSymTab() {
    Int hasMore = startIterator(table);
    printf("%20s\t%10s\n", "Variable", "Value");
    while (hasMore) {
        printf("%20s\t%10s\n",getCurrentName(table),
                                    getCurrentAttr(table) ? "true" : "false");
        hasMore = nextEntry(table);
    }
}
```

# semantics.c

```
void storeVar(char * name, bool v) {
    enterName(table, name);
    setCurrentAttr(table, (void *) v);

}

bool getVal(char * name) {
    If (enterName(table, name)) {
                WriteIndicator(getCurrentColumnNum());
                WriteMessage("Initialize variable to false");
                setCurrentAttr(table, (void *) false);
    }
    return (bool) getCurrentAttr(table);

}
```

# semantics.c

```
bool doOR(bool v1, bool v2) {
    return v1 || v2;
}

bool doAND(bool v1, bool v2) {
    return v1 && v2;
}

bool doNOT(bool v1) {
    return !v1;
}
```

## main.c

```
#include <stdio.h>
#include "SymTab.h"
#include "IOMngr.h"

extern int yyparse();

SymTab *table;

int main(int argc, char * argv[]) {
    table = CreateSymTab(17);
    if (!openFiles(argv[1], "listing")) {
        printf("open failed\n");
        exit(0);
    }
    yyparse();
    destroySymTab(table);
}
```

# Build Executable

```
> yacc -d boolExpr.y

> lex boolLex.l

>  cc -o h3 y.tab.c lex.yy.c semantics.c SymTab.c IOMngr.c main.c
```