# CS 442/542 Homework 4

Due Monday November 2

# 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 yyleng;
extern int yyerror(char *);

extern SymTab *table;
extern SymEntry *entry;

%}
```

# 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();
}
```

# Build Executable

```
toms-air:a3 gendreau$ yacc -d boolExpr.y

toms-air:a3 gendreau$ lex boolLex.l

toms-air:a3 gendreau$ cc -o h4 y.tab.c lex.yy.c semantics.c SymTab.c
IOMngr.c main.c
```

# Homework 4 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 -> INT
Factor -> FLOAT
Id -> Ident
```

## Homework 4

- Build an interpreter for the arithmetic grammar shown on the previous slide.
- As in the boolean expression grammar use the symbol table to remember the values of variables.
- In addition to the values of variables the symbol table also should remember the type of the variable. The type will either be int or float depending on the type of value assigned to the variable. See examples in the next slide.
- When the program is finished, print the values of the variables to standard output

## Homework 4

```
x = 10;
y = 12.2;
z = x * 2;
x = 15.5;
w = z + y;
a = 2;
n = -(w + a)*3+x;
```

## Homework 4

- As you parse the source file (move through the grammar in the yacc file) you will need to keep track of the values and types of expressions.
- Do this by associating value and type information with non-terminals in the grammar.

# Homework 4 Type Information

| | |
|---|---|
| Prog -> StmtSeq | |
| StmtSeq -> Stmt StmtSeq | |
| StmtSeq -> ε | |
| Stmt -> Id = Expr ; | Id has the same type as Expr |
| Expr -> Expr + Term | If both Expr and Term are ints then Expr is an int otherwise Expr is a float |
| Expr -> Term | Expr has the same type as Term |
| Term -> Term * Factor | If both Term and Factor are ints then Term is an int otherwise Term is a float |
| Term -> Factor | Term has the same type as Factor |
| Factor -> - Factor | Factor has the same type as Factor |
| Factor -> ( Expr ) | Factor has the same type a Expr |
| Factor -> Id | Factor has the same type as Id |
| Factor -> INT | Factor is an int |
| Factor -> FLOAT | Factor is a float |
| Id -> Ident | |

# Homework 4 Submission

- Upload one zip file named H4yourName.zip (i.e. I would send a file named H4TomGendreau.zip).
- H4yourName must contain two folders. One folder must be named h4 and must contain some test programs on which your h4 program works and the following files: arithLex.l, arithYacc.y, SymTab.h, SymTab.c, IOMngr.h, IOMngr.c, semantics.h, semantics.c, main.c. The other folder must be named h3. It should include some test programs on which your h3 program works and the files you produced for homework 3. This does mean that the files SymTab.h, SymTab.c, IOMngr.h, IOMngr.c will be in both h3 and h4.
- Please use the exact file names shown above for h4.

## Homework 4 Submission

- You will demo homework 4 to me sometime during the week of November 2.
- The homework (including SymTab and IOMngr, and homework 3) is worth 50 points.
-