CS 442/542

Project Starting Point
yacc4

ExprEval.y

```
%union {
  long val;
  char * string;
  struct ExprRes * ExprRes;
  struct InstrSeq * InstrSeq;
  struct BExprRes * BExprRes;
}

%type <string> Id
%type <ExprRes> Factor
%type <ExprRes> Term
%type <ExprRes> Expr
%type <InstrSeq> StmtSeq
%type <InstrSeq> Stmt
%type <BExprRes> BExpr

%token Ident
%token IntLit
%token Int
%token Write
%token IF
%token EQ
```

ExprEval.y

```
%%

Prog          :   Declarations StmtSeq                                {Finish($2); } ;
Declarations :   Dec Declarations                                    { };
Declarations :                                                       { };
Dec           :   Int Ident {enterName(table, yytext); }';'{};
StmtSeq       :   Stmt StmtSeq                                        {$$ = AppendSeq($1, $2); } ;
StmtSeq       :                                                       {$$ = NULL;} ;
Stmt          :   Write Expr ';'                                      {$$ = doPrint($2); };
Stmt          :   Id '=' Expr ';'                                     {$$ = doAssign($1, $3);} ;
Stmt          :   IF '(' BExpr ')' '{' StmtSeq '}'                    {$$ = doIf($3, $6);};
BExpr         :   Expr EQ Expr                                        {$$ = doBExpr($1, $3);};
Expr          :   Expr '+' Term                                       {$$ = doAdd($1, $3); } ;
Expr          :   Term                                                {$$ = $1; } ;
Term          :   Term '*' Factor                                     { $$ = doMult($1, $3); } ;
Term          :   Factor                                              { $$ = $1; } ;
Factor        :   IntLit                                              { $$ = doIntLit(yytext); };
Factor        :   Ident                                               { $$ = doRval(yytext); };
Id            :   Ident                                               { $$ = strdup(yytext);}

%%
```

lex1.l

```
%%
if                              {return IF;}
int                             {return Int;}
print                           {return Write;}
{letter}({letter}|{digit})*     {return Ident;}
{digit}{digit}*                 {return IntLit;}
\=\=                            {return EQ;}
\=                              {return '=';}
\+                              {return '+';}
\*                              {return '*';}
\;                              {return ';';}
\{                              {return '{';}
\}                              {return '}';}
\(                              {return '(';}
\)                              {return ')';}
```

lex1.l

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

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

%%


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

Semantics.h

```
/* Semantic Records */
struct IdList {
  struct SymEntry * TheEntry;
  struct IdList * Next;
};
struct ExprRes {
  int Reg;
  struct InstrSeq * Instrs;
};
struct ExprResList {
	struct ExprRes *Expr;
	struct ExprResList * Next;
};
struct BExprRes {
  char * Label;
  struct InstrSeq * Instrs;
};
```

Semantics.h

```
/* Semantics Actions */
extern struct ExprRes *  doIntLit(char * digits);
extern struct ExprRes *  doRval(char * name);
extern struct InstrSeq *  doAssign(char * name,  struct ExprRes * Res1);
extern struct ExprRes *  doAdd(struct ExprRes * Res1,  struct ExprRes * Res2);
extern struct ExprRes *  doMult(struct ExprRes * Res1,  struct ExprRes * Res2);
extern struct InstrSeq *  doPrint(struct ExprRes * Expr);
extern struct BExprRes * doBExpr (struct ExprRes * Res1,  struct ExprRes * Res2);
extern struct InstrSeq * doIf(struct BExprRes *bRes, struct InstrSeq * seq);

extern void   Finish(struct InstrSeq *Code);
```

Semantics.c

```
struct ExprRes *  doIntLit(char * digits)  {

   struct ExprRes *res;

  res = (struct ExprRes *) malloc(sizeof(struct ExprRes));
  res->Reg = AvailTmpReg();
  res->Instrs = GenInstr(NULL,"li",TmpRegName(res->Reg),digits,NULL);

  return res;
}
```

Semantics.c

```
struct ExprRes *  doRval(char * name)  {

   struct ExprRes *res;

   if (!FindName(table, name)) {
            WriteIndicator(GetCurrentColumn());
            WriteMessage("Undeclared variable");
   }
  res = (struct ExprRes *) malloc(sizeof(struct ExprRes));
  res->Reg = AvailTmpReg();
  res->Instrs = GenInstr(NULL,"lw",TmpRegName(res->Reg),name,NULL);

  return res;
}
```

Semantics.c

```
struct ExprRes *  doAdd(struct ExprRes * Res1, struct ExprRes * Res2)  {

   int reg;

  reg = AvailTmpReg();
  AppendSeq(Res1->Instrs,Res2->Instrs);
  AppendSeq(Res1->Instrs,GenInstr(NULL,"add",
                                  TmpRegName(reg),
                                  TmpRegName(Res1->Reg),
                                  TmpRegName(Res2->Reg)));
  ReleaseTmpReg(Res1->Reg);
  ReleaseTmpReg(Res2->Reg);
  Res1->Reg = reg;
  free(Res2);
  return Res1;
}
```

Semantics.c

```
struct ExprRes *  doMult(struct ExprRes * Res1, struct ExprRes * Res2)  {

   int reg;

  reg = AvailTmpReg();
  AppendSeq(Res1->Instrs,Res2->Instrs);
  AppendSeq(Res1->Instrs,GenInstr(NULL,"mul",
                                   TmpRegName(reg),
                                   TmpRegName(Res1->Reg),
                                   TmpRegName(Res2->Reg)));
  ReleaseTmpReg(Res1->Reg);
  ReleaseTmpReg(Res2->Reg);
  Res1->Reg = reg;
  free(Res2);
  return Res1;
}
```

Semantics.c

```
struct InstrSeq * doPrint(struct ExprRes * Expr) {

  struct InstrSeq *code;
  code = Expr->Instrs;

    AppendSeq(code,GenInstr(NULL,"li","$v0","1",NULL));
    AppendSeq(code,GenInstr(NULL,"move","$a0",TmpRegName(Expr->Reg),NULL));
    AppendSeq(code,GenInstr(NULL,"syscall",NULL,NULL,NULL));

    AppendSeq(code,GenInstr(NULL,"li","$v0","4",NULL));
    AppendSeq(code,GenInstr(NULL,"la","$a0","_nl",NULL));
   AppendSeq(code,GenInstr(NULL,"syscall",NULL,NULL,NULL));

    ReleaseTmpReg(Expr->Reg);
    free(Expr);

  return code;
}
```

Semantics.c

```
struct InstrSeq * doAssign(char *name, struct ExprRes * Expr) {

  struct InstrSeq *code;

   if (!FindName(table, name)) {
            WriteIndicator(GetCurrentColumn());
            WriteMessage("Undeclared variable");
   }

  code = Expr->Instrs;

  AppendSeq(code,GenInstr(NULL,"sw",TmpRegName(Expr->Reg), name,NULL));

  ReleaseTmpReg(Expr->Reg);
  free(Expr);

  return code;
}
```

Semantics.c

```
struct BExprRes * doBExpr(struct ExprRes * Res1,  struct ExprRes * Res2) {
      struct BExprRes * bRes;
      AppendSeq(Res1->Instrs, Res2->Instrs);
      bRes = (struct BExprRes *) malloc(sizeof(struct BExprRes));
      bRes->Label = GenLabel();
      AppendSeq(Res1->Instrs,
                  GenInstr(NULL, "bne", TmpRegName(Res1->Reg),
                                                      TmpRegName(Res2->Reg), bRes->Label));
      bRes->Instrs = Res1->Instrs;
      ReleaseTmpReg(Res1->Reg);
      ReleaseTmpReg(Res2->Reg);
      free(Res1);
      free(Res2);
      return bRes;
}
```

Semantics.c

```
struct InstrSeq * doIf(struct BExprRes * bRes, struct InstrSeq * seq) {
    struct InstrSeq * seq2;
    seq2 = AppendSeq(bRes->Instrs, seq);
    AppendSeq(seq2, GenInstr(bRes->Label, NULL, NULL, NULL, NULL));
    free(bRes);
    return seq2;
}
```

Semantics.c

```
void Finish(struct InstrSeq *Code) {
  struct InstrSeq *code;
  struct SymEntry *entry;
  struct Attr * attr;


  code = GenInstr(NULL,".text",NULL,NULL,NULL);
  AppendSeq(code,GenInstr(NULL,".globl","main",NULL,NULL));
  AppendSeq(code, GenInstr("main",NULL,NULL,NULL,NULL));

  AppendSeq(code,Code);

  AppendSeq(code, GenInstr(NULL, "li", "$v0", "10", NULL));
  AppendSeq(code, GenInstr(NULL,"syscall",NULL,NULL,NULL));
  AppendSeq(code,GenInstr(NULL,".data",NULL,NULL,NULL));
  AppendSeq(code,GenInstr(NULL,".align","4",NULL,NULL));
  AppendSeq(code,GenInstr("_nl",".asciiz","\"\\n\"",NULL,NULL));
```

Semantics.c

```
 hasMore = startIterator(table);
 while (hasMore) {
   AppendSeq(code,GenInstr((char *)getCurrentName(table),
                                     “.word","0",NULL,NULL));
     hasMore = nextEntry(table);
 }

  WriteSeq(code);

}
```

CodeGen.h

```
#include <stdio.h>

struct InstrSeq {
  char *Label;
  char *OpCode;
  char *Oprnd1;
  char *Oprnd2;
  char *Oprnd3;
  struct InstrSeq *Next;
};
```

CodeGen.h

```
extern struct InstrSeq * GenInstr(char *Label, char *OpCode,
                                  char *Oprnd1, char *Oprnd2, char *Oprnd3);
extern struct InstrSeq * AppendSeq(struct InstrSeq *Seq1,
                                   struct InstrSeq *Seq2);
extern void  WriteSeq(struct InstrSeq *ASeq);

extern char *GenLabel();
```

CodeGen.h

```
extern int  AvailTmpReg();
extern char *TmpRegName(int RegNum);
extern void ReleaseTmpReg(int ANum);
extern void ResetAllTmpReg();
extern struct InstrSeq * SaveSeq();
extern struct InstrSeq * RestoreSeq();

extern char *Imm(int Val);
extern char *RegOff(int Offset, char * Reg);
```

main.c

```c
SymTab *table;
FILE *aFile;

int main(int argc, char * argv[]) {
   table = createSymTab(33);
   //assumes there is a listing file
   openFiles(argv[1], argv[2]);
   if (argc == 4)
      aFile = fopen(argv[3], "w");
   else
      aFile = stdout;

   yyparse();
}
```

Source Program

```
int num1;
int num2;
num1 = 10;
num2 = 20;
if (num1+10+10+10 == 2*num2) {
      print num1;
      print num2;
}
print num1;
print num2;
```

Assembly Language Program

```
        .text
        .globl          main
main:
        li          $t0, 10
        sw          $t0, num1
        li          $t0, 20
        sw          $t0, num2
        lw          $t0, num1
        li          $t1, 10
        add         $t2, $t0, $t1
        li          $t0, 10
        add         $t1, $t2, $t0
        li          $t0, 10
        add         $t2, $t1, $t0
        li          $t0, 2
        lw          $t1, num2
        mul         $t3, $t0, $t1
```

Assembly Language Program

```
    bne     $t2, $t3, L1
    lw      $t0, num1
    li      $v0, 1
    move    $a0, $t0
    syscall
    li      $v0, 4
    la      $a0, _nl
    syscall
    lw      $t0, num2
    li      $v0, 1
    move    $a0, $t0
    syscall
    li      $v0, 4
    la      $a0, _nl
    syscall
L1:
```

Assembly Language Program

```
lw      $t0, num1
li      $v0, 1
move    $a0, $t0
syscall
li      $v0, 4
la      $a0, _nl
syscall
lw      $t0, num2
li      $v0, 1
move    $a0, $t0
syscall
li      $v0, 4
la      $a0, _nl
syscall
```

Assembly Language Program

```
        li      $v0, 10
        syscall
        .data
        .align  4
_nl:    .asciiz "\n"
num1:   .word   0
num2:   .word   0
```

Build the Program

```
> yacc -d ExprEval.y
> lex lex1.l
> cc   -o comp lex.yy.c y.tab.c SymTab.c Semantics.c CodeGen.c IOMngr.c
main.c
```

Execute the Program

```
> ./comp  source listing.lst  asmCode.asm
```

Where to Start

- Download yacc4ForStudents
- Download the Mars or SPIM MIPS simulator
- Build and test the code with your implementation of SymTab and IOMngr
- Try adding one feature at a time. For example add subtraction
- Try adding another feature such as the less than relational operator (see the relational operators slides)
- Keep adding features until you run out of time…
- As you add features keep backup copies of versions that work.
- Have fun!