[Project 2 : Parser]

컴퓨터전공 2013011253 김기원

Project 2 : C-Minus Parser

0. Compilation tools & Environment

Basic Environment : VMware(Ubuntu 17.10)

Basic Tools : GCC (version 5.4.0) , Flex (version 2.6.0) , bison(version 3.0.4) Makefile

Process to my code :

$ make (in home directory : cp2) (Create cminus_yacc : Main file)

$ make test (./cminus_yacc test.cm)

$ make clean (rm all created by Makefile)

1. Explanation (step by step)

1-1: Setting up Makefile :

For debugging and basic compilation, Makefile should be the first step.

The main modified of Makefile for parser is like below :

```
cminus_yacc: $(OBJS)
    $(CC) $(CFLAGS) $(OBJS) -o cminus_yacc
y.tab.o: cminus.y globals.h
    bison -d cminus.y --yacc
    $(CC) $(CFLAGS) -c y.tab.c
```

Make the file y.tab.c/y.tab.h with bison through cminus.y

1-2: Modifying code :

In this project, we need to modify compiler's code.

Main.c :

```
#define NO_PARSE FALSE
```

Turn this line to FALSE, because our project is about parser.

```
int EchoSource = TRUE;
int TraceScan = FALSE;
int TraceParse = TRUE;
int TraceAnalyze = FALSE;
int TraceCode = FALSE;
```

EchoSource option is about the scanned source code line, and TraceParse is the option that specify every parser walk through. This option should to be TRUE(1).

Globals.h :

```
#include "y.tab.h"
```

We need to include y.tab.h for the declearation of tokens.

```
typedef enum {StmtK,ExpK,DeclK,ParamK,TypeK} NodeKind;
typedef enum {CompK,IfK,IterK,RetK} StmtKind;
typedef enum {AssignK,OpK,ConstK,IdK,ArrIdK,CallK} ExpKind;
typedef enum {FuncK,VarK,ArrVarK} DeclKind;
typedef enum {ArrParamK,NonArrParamK} ParamKind;
typedef enum {TypeNameK} TypeKind;
```

And we need to define more statements. Use to Print out of parsetree. From util.c.

And for these declaration, we need to use structure to access into it.

```
typedef struct treeNode
   { struct treeNode * child[MAXCHILDREN];
     struct treeNode * sibling;
     int lineno;
     NodeKind nodekind;
     union { StmtKind stmt;
             ExpKind exp;
             DeclKind decl;
             ParamKind param;
             TypeKind type; } kind;
     union { TokenType op;
             TokenType type;
             int val;
             char * name;
             ArrayAttr arr; } attr;
     ExpType type; /* for type checking of exps */
   } TreeNode;
```

The next is about the code we modified in global.h for Util.c.

Util.c :

StmtK = For specify statement(the second line of declaration).

```
if (tree->nodekind==StmtK)
    { switch (tree->kind.stmt) {
       case CompK:
         fprintf(listing,"Compound Statment\n");
         break;
       case IfK:
         fprintf(listing,"If\n");
         break;
       case IterK:
         fprintf(listing,"While\n");
         break;
       case RetK:
         fprintf(listing,"Return\n");
         break;
       default:
         fprintf(listing,"Unknown ExpNode kind\n");
         break;
      }
```

ExpK = For specify the third line of declaration.

```
else if (tree->nodekind==ExpK)
    { switch (tree->kind.exp) {
       case AssignK:
         fprintf(listing,"Assign: ");
         //printToken(tree->attr.op,"\0");
         break;
       case OpK:
         fprintf(listing,"Op: ");
         printToken(tree->attr.op,"\0");
         break;
       case ConstK:
         fprintf(listing,"Const: %d\n",tree->attr.val);
         break;
       case IdK:
         fprintf(listing,"Id: %s\n",tree->attr.name);
         break;
       case ArrIdK:
         fprintf(listing,"ArrId\n");
         break;
       case CallK:
         fprintf(listing,"Call(followings are args) : %s\n", tree->attr.name);
         break;
       default:
         fprintf(listing,"Unknown ExpNode kind\n");
         break;
      }
```

DelK = For Function and variable declaration.

```c
else if (tree->nodekind==DeclK)
    { switch (tree->kind.decl) {
        case FuncK:
          fprintf(listing,"Function Dec: %s\n",tree->attr.name);
          break;
        case VarK:
          fprintf(listing,"Var Dec: %s\n",tree->attr.name);
          break;
        case ArrVarK:
          fprintf(listing,
                  "Var Dec(following const:array length): %s %d\n",
                  tree->attr.arr.name,
                  tree->attr.arr.size);
          break;
        default:
          fprintf(listing,"Unknown DeclNode kind\n");
          break;
      }
```

ParamK = For parameter

```c
else if (tree->nodekind==ParamK)
    { switch (tree->kind.param) {
        case ArrParamK:
          fprintf(listing,"Array Parameter: %s\n",tree->attr.name);
          break;
        case NonArrParamK:
          fprintf(listing,"Parameter: %s\n",tree->attr.name);
          break;
        default:
          fprintf(listing,"Unknown ParamNode kind\n");
          break;
      }
```

TypeK = For data Type in c-minus which is INT & VOID

```c
else if (tree->nodekind==TypeK)
    { switch (tree->kind.type) {
        case TypeNameK:
          fprintf(listing,"Type: ");
          switch (tree->attr.type) {
            case INT:
              fprintf(listing,"int\n");
              break;
            case VOID:
              fprintf(listing,"void\n");
          }
          break;
        default:
          fprintf(listing,"Unknown TypeNode kind\n");
        break;
      }
```

Util.c is like the past project. It is the output of our process.

Parse.c :

Modify of Parse.c is necessary because of cminus.y for yacc uses for BNF grammar rule. For the construct of parsetree, it will be the main section.

```
static TreeNode * stmt_sequence(void);
static TreeNode * statement(void);
static TreeNode * if_stmt(void);
static TreeNode * while_stmt(void);
static TreeNode * assign_stmt(void);
static TreeNode * read_stmt(void);
static TreeNode * write_stmt(void);
static TreeNode * exp(void);
static TreeNode * simple_exp(void);
static TreeNode * term(void);
static TreeNode * factor(void);
```

to much code to explain, but I will take one or more to explain. Because each TreeNode* is basically same structure. And the basic tiny language in included because I left it. And just need more production rules to define another tokens and symbols.

```
TreeNode * simple_exp(void)
{ TreeNode * t = term();
  while ((token==PLUS)||(token==MINUS))
  { TreeNode * p = newExpNode(OpK);
    if (p!=NULL) {
      p->child[0] = t;
      p->attr.op = token;
      t = p;
      match(token);
      t->child[1] = term();
    }
  }
  return t;
}
```

The default statement(production) is like below, this structure is like util.c because it must be the same to access nodes. So the explanation will be the same too. The simple expression will like PLUS or MINUS.

```
TreeNode * term(void)
{ TreeNode * t = factor();
  while ((token==TIMES)||(token==OVER))
  { TreeNode * p = newExpNode(OpK);
    if (p!=NULL) {
      p->child[0] = t;
      p->attr.op = token;
      t = p;
      match(token);
      p->child[1] = factor();
    }
  }
  return t;
}
```

And we define term to factor of TIMES and OVER , difference with PLUS, MINUS is because of the Precedence.

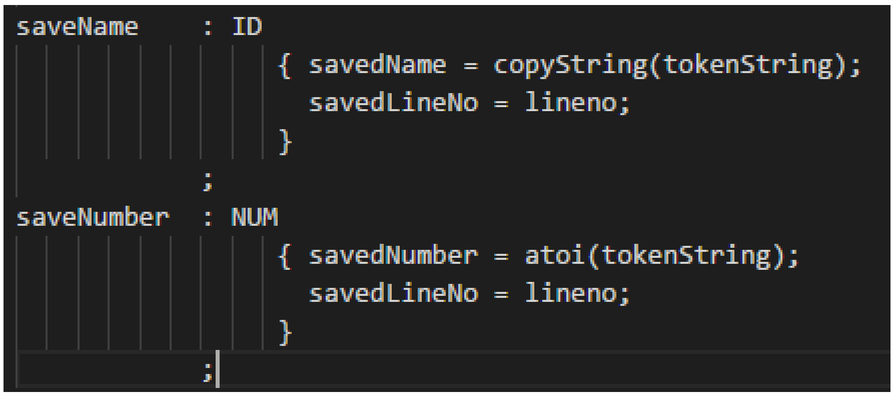
```
TreeNode * factor(void)
{ TreeNode * t = NULL;
  switch (token) {
    case NUM :
      t = newExpNode(ConstK);
      if ((t!=NULL) && (token==NUM))
        t->attr.val = atoi(tokenString);
      match(NUM);
      break;
    case ID :
      t = newExpNode(IdK);
      if ((t!=NULL) && (token==ID))
        t->attr.name = copyString(tokenString);
      match(ID);
      break;
    case LPAREN :
      match(LPAREN);
      t = exp();
      match(RPAREN);
      break;
    default:
      syntaxError("unexpected token -> ");
      printToken(token,tokenString);
      token = getToken();
      break;
    }
  return t;
}
```

At the End of factor will be define it is ID or NUM of "(" or ")" which is Terminal symbols.

In Parse.c is simple to see the all structure when we just think of the CFGs and Production Rules.

Cminus.y :

Actually it is the main part of the project. Uses for yacc in parsetree is just the matching of Parse.c , Not the main of it. We need to modify cminus.y for our project because we use YACC for this project 2. The basic of BNF Grammar is shown in Appendix A.2. All modified by it but little difference.

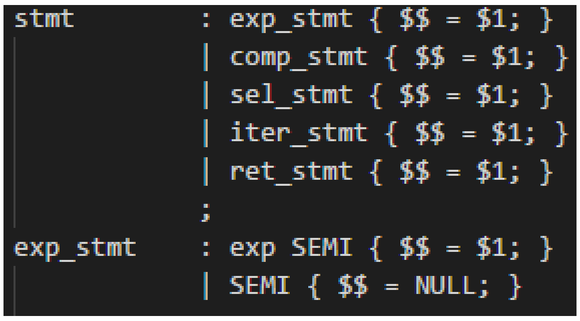


```
saveName     : ID
               { savedName = copyString(tokenString);
                 savedLineNo = lineno;
               }
             ;
saveNumber   : NUM
               { savedNumber = atoi(tokenString);
                 savedLineNo = lineno;
               }
             ;
```

Difference for ID and NUM, we need another process (which is showing). For ID and NUM. It will be the last token(non-Terminal in Parsetree), so the token came out after them will be non-identified. Use of saveName for saving ID, and it can be access like global variables.

For the other explain. Will choose the main section to explain. Let's trace about "==".

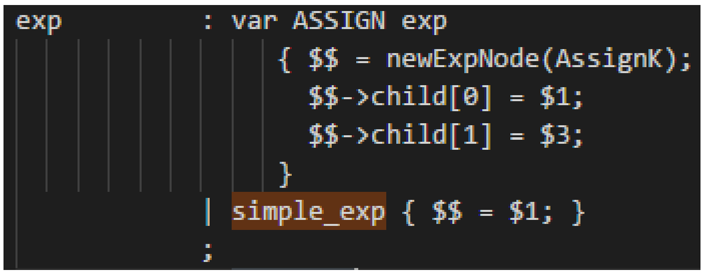
Start from Stmt Statement.



```
stmt          : exp_stmt { $$ = $1; }
              | comp_stmt { $$ = $1; }
              | sel_stmt { $$ = $1; }
              | iter_stmt { $$ = $1; }
              | ret_stmt { $$ = $1; }
              ;
exp_stmt      : exp SEMI { $$ = $1; }
              | SEMI { $$ = NULL; }
```

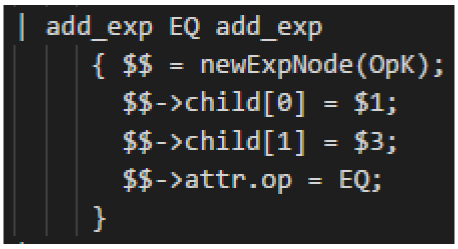
"=" match of S->exp_stmt / exp_stmt-> exp SEMI | SEMI which SEMI is null.

$$ will be the LHS of production, RHS for $S1 is first and continued split with blank.



```
exp          : var ASSIGN exp
               { $$ = newExpNode(AssignK);
                 $$->child[0] = $1;
                 $$->child[1] = $3;
               }
             | simple_exp { $$ = $1; }
             ;
```

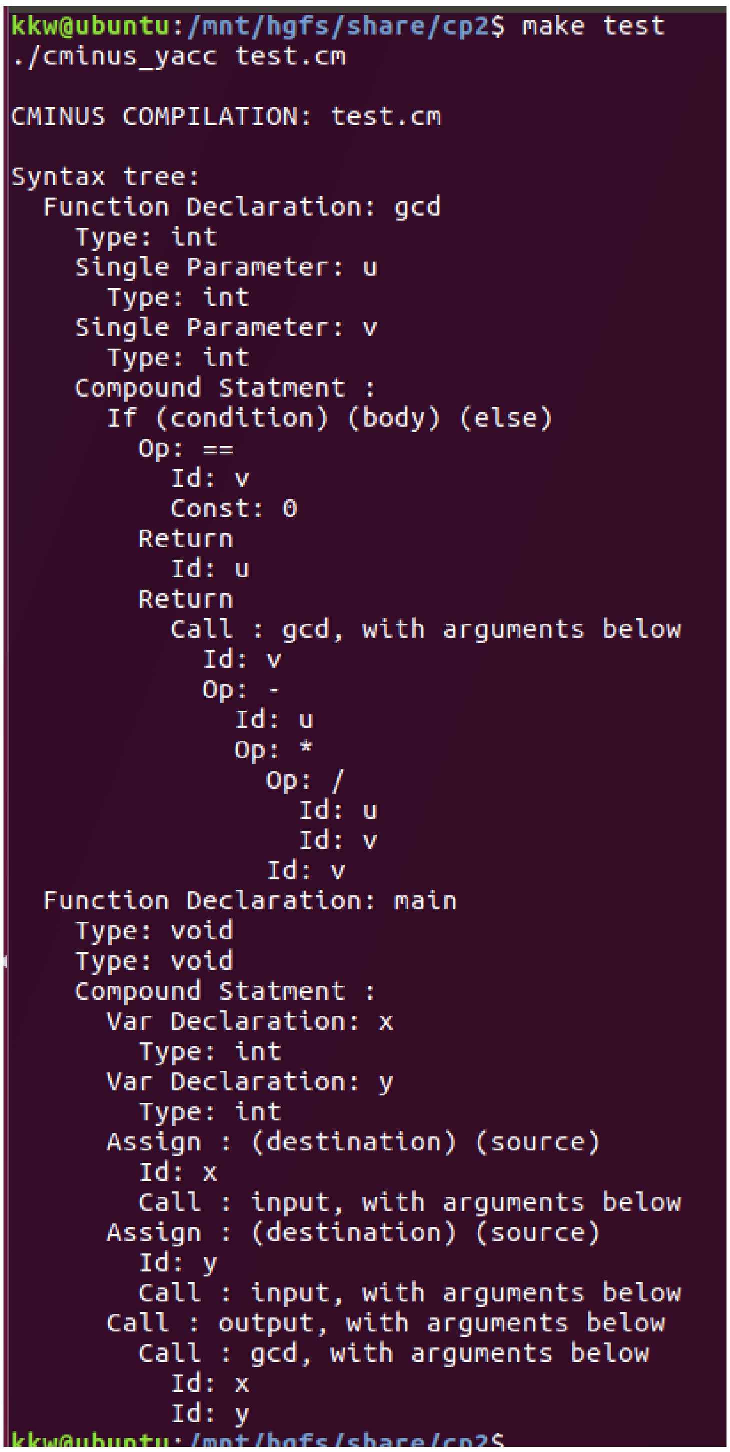
exp -> car ASSIGN exp | simple_exp get exp -> simple_exp



```
| add_exp EQ add_exp
      { $$ = newExpNode(OpK);
        $$->child[0] = $1;
        $$->child[1] = $3;
        $$->attr.op = EQ;
      }
```

which is simple_exp. Change current production rule. For LHS is the newExpNode , and two matches for RHS is add_exp first or add_exp secoond. To setting of child[]. And switch the attr.op . In parse.c. uses.

2. The Result of TESTS.



```
kkw@ubuntu:/mnt/hgfs/share/cp2$ make test
./cminus_yacc test.cm

CMINUS COMPILATION: test.cm

Syntax tree:
  Function Declaration: gcd
    Type: int
    Single Parameter: u
      Type: int
    Single Parameter: v
      Type: int
    Compound Statment :
      If (condition) (body) (else)
        Op: ==
          Id: v
          Const: 0
        Return
          Id: u
        Return
          Call : gcd, with arguments below
            Id: v
            Op: -
              Id: u
              Op: *
                Op: /
                  Id: u
                  Id: v
                Id: v
  Function Declaration: main
    Type: void
    Type: void
    Compound Statment :
      Var Declaration: x
        Type: int
      Var Declaration: y
        Type: int
      Assign : (destination) (source)
        Id: x
        Call : input, with arguments below
      Assign : (destination) (source)
        Id: y
        Call : input, with arguments below
      Call : output, with arguments below
        Call : gcd, with arguments below
          Id: x
          Id: y
kkw@ubuntu:/mnt/hgfs/share/cp2$
```

The output of syntax tree is not perfectly same. But it is not important. Just the combine of printf in util.c. all these terminals can be recognize and parsing is going well.