ELE4029 2_Parser Project Report

Afif Danial 2019048586

This program implemented C-Minus Parser using C-Minus Scanner and yacc.

Project Environment

Ubuntu 18.0.4.5 LTS

Overview

In order to write the C-Minus Parser, the **cminus.y** file will be modified to define the Syntax Tree and parse the C-Minus code.

BNF Grammar for C-Minus

1. program → declaration-list
2. declaration-list → declaration-list declaration | declaration
3. declaration → var-declaration | fun-declaration
4. var-declaration → type-specifier ID ; | type-specifier ID [NUM] ;
5. type-specifier → int | void
6. fun-declaration → type-specifier ID (params) compound-stmt
7. params → param-list | void
8. param-list → param-list , param | param
9. param → type-specifier ID | type-specifier ID [ ]
10. compound-stmt → { local-declarations statement-list }
11. local-declarations → local-declarations var-declarations | empty
12. statement-list → statement-list statement | empty
13. statement → expression-stmt | compound-stmt | selection-stmt | iteration-stmt | return-stmt
14. expression-stmt → expression ; | ;
15. selection-stmt → if (expression) statement | if (expression) statement else statement
16. iteration-stmt → while (expression) statement
17. return-stmt → return ; | return expression ;
18. expression → var = expression | simple-expression
19. var → ID | ID [expression]
20. simple-expression → additive-expression relop additive-expression | additive-expression
21. relop → <= | < | > | >= | == | !=
22. additive-expression → additive-expression addop term | term
23. addop → + | -
24. term → term mulop factor | factor

25. mulop → * | /
26. factor → (expression) | var | call | NUM
27. call → ID (args)
28. args → arg-list | empty
29. arg-list → arg-list , expression | expression

Implementation

Makefile

```
y.tab.o: cminus.l globals.h util.h scan.h parse.h
	yacc -d cminus.y
	$(CC) $(CFLAGS) -c y.tab.c
```

Since yacc is necessary in order to generate **y.tab.c** which is responsible to parse, so it was added at **Makefile** that has been provided.

main.c

```
#define NO_PARSE FALSE
#define NO_PARSE TRUE

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

In this program, only C-Minus Parser is written, so the flags of **main.c** have been adjusted.

globals.h

```
typedef enum {StmtK,ExpK,DeclK,ParamK,TypeK} NodeKind;
typedef enum {CompK,IfK,IfEK,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;

/* ArrayAttr is used for attributes of array variable */
typedef struct arrayAttr
   { TokenType type;
     char * name;
     int size;
   } ArrayAttr;

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;
```

Basically, the **yacc/globals.h** file was copied and has been modified which is from Parser and it is necessary to classify and add accordingly into each node of the Syntax Tree. Plus, because the arrangement has to be recognized, the **ArrayAttr** structure is created separately. The **treeNod**e structure is modified due to that.

util.c

```
TreeNode * newDeclNode(DeclKind kind)
{ TreeNode * t = (TreeNode *) malloc(sizeof(TreeNode));
  int i;
  if (t==NULL)
    fprintf(listing,"Out of memory error at line %d\n",lineno);
  else {
    for (i=0;i<MAXCHILDREN;i++) t->child[i] = NULL;
    t->sibling = NULL;
    t->nodekind = DeclK;
    t->kind.decl = kind;
    t->lineno = lineno;
  }
  return t;
}
...

void printTree( TreeNode * tree )
{ int i;
  INDENT;
  while (tree != NULL) {
    if (tree->nodekind!=TypeK)
      printSpaces();
    if (tree->nodekind==StmtK)
    { switch (tree->kind.stmt) {
        case CompK:
          fprintf(listing,"Compound statement :\n");
          break;
        case IfK:
          fprintf(listing,"If (condition) (body)\n");
          break;
        case IfEK:
          fprintf(listing,"If (condition) (body) (else)\n");
          break;
        case IterK:
          fprintf(listing,"Repeat : \n");
          break;
```

Since Decl, Param, and Type Node are added into the BNF, so a function that generates them is created. And when these are applied to the Parse Tree, the **printTree** function is modified so that it can generate the output.

cminus.y

Based on BNF, the **cminus.y** file is modified as follows.

```
program     : decl_list
                { savedTree = $1;}
            ;
decl_list   : decl_list decl
                { YYSTYPE t = $1;
                  if (t != NULL)
                  { while (t->sibling != NULL)
                       t = t->sibling;
                    t->sibling = $2;
                    $$ = $1; }
                    else $$ = $2;
                }
            | decl  { $$ = $1; }
            ;
decl        : var_decl  { $$ = $1; }
            | fun_decl { $$ = $1; }
            ;
            ...
```

For most grammar, it has to be modified in order to suit the BNF, however ID and NUM additionally define the grammar as follows.

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

This is to prevent the global variables **savedName** and **savedNumber** from being overwritten in the process of derivation.

Plus, in this case of arrangement, the values of the elements of the **ArrayAttr** structure were substituted as follows.

```
var_decl    : type_spec saveName SEMI
                { $$ = newDeclNode(VarK);
                  $$->child[0] = $1;
                  $$->lineno = lineno;
                  $$->attr.name = savedName;
                }
            | type_spec saveName LBRACE saveNumber RBRACE SEMI
                { $$ = newDeclNode(ArrVarK);
                  $$->child[0] = $1;
                  $$->lineno = lineno;
                  $$->attr.arr.name = savedName;
                  $$->attr.arr.size = savedNumber;
                }
            ;
var         : saveName
                { $$ = newExpNode(IdK);
                  $$->attr.name = savedName;
                }
            | saveName
                { $$ = newExpNode(ArrIdK);
                  $$->attr.name = savedName;
                }
              LBRACE exp RBRACE
                { $$ = $2;
                  $$->child[0] = $4;
                }
            ;
```

Operation

```
$ make cminus
$ ./cminus test.cm
```

Result

test1.cm

```
Syntax tree:
  Function declaration, name : main, return type : void
    Single parameter, name : (null), type : void
    Compound statement :
      Var declaration, name : i, type : int
      Arr Var declaration, name : x, size : 5, type : int
      Assign : (destination) (source)
        Id : i
        Const : 0
      Repeat :
        Op : <
          Id : i
          Const : 5
        Compound statement :
          Assign : (destination) (source)
            ArrId : x
              Id : i
            Call, name : input, with arguments below
          Assign : (destination) (source)
            Id : i
            Op : +
              Id : i
              Const : 1
      Assign : (destination) (source)
        Id : i
        Const : 0
      Repeat :
        Op : <=
          Id : i
          Const : 4
        Compound statement :
          If (condition) (body)
            Op : !=
              ArrId : x
                Id : i
              Const : 0
            Compound statement :
              Call, name : output, with arguments below
                ArrId : x
                  Id : i
```

test2.cm

```
Syntax tree:
  Function declaration, name : gcd, return type : int
    Single parameter, name : u, type : int
    Single parameter, name : v, type : int
    Compound statement :
      If (condition) (body) (else)
        Op : ==
          Id : v
          Const : 0
        Return :
          Id : u
        Return :
          Call, name : gcd, with arguments below
            Id : v
            Op : -
              Id : u
              Op : *
                Op : /
                  Id : u
                  Id : v
                Id : v
  Function declaration, name : main, return type : void
    Single parameter, name : (null), type : void
    Compound statement :
      Var declaration, name : x, type : int
      Var declaration, name : y, type : int
      Assign : (destination) (source)
        Id : x
        Call, name : input, with arguments below
      Assign : (destination) (source)
        Id : y
        Call, name : input, with arguments below
      Call, name : output, with arguments below
        Call, name : gcd, with arguments below
          Id : x
          Id : y
```

test3.cm

```
Syntax tree:
  Arr Var declaration, name : aaa, size : 1234, type : int
  Function declaration, name : function, return type : int
    Single parameter, name : a, type : int
    Single parameter, name : b, type : int
    Array parameter, name : c, type : int
    Single parameter, name : d, type : int
    Compound statement :
      Assign : (destination) (source)
        ArrId : aaa
          ArrId : a
            Id : i
        Const : 1
```

test4.cm

```
Syntax tree:
  Var declaration, name : x, type : int
  Var declaration, name : y, type : int
  Var declaration, name : k, type : int
  Function declaration, name : abc, return type : int
    Single parameter, name : qwe, type : int
    Single parameter, name : lol, type : int
    Compound statement :
      Var declaration, name : aa, type : int
      Var declaration, name : bb, type : int
      Var declaration, name : cc, type : int
      Var declaration, name : dd, type : int
      Arr Var declaration, name : zzz, size : 5324, type : int
      Arr Var declaration, name : ee, size : 123, type : int
      Var declaration, name : qre, type : int
      Assign : (destination) (source)
        Id : cc
        Const : 2
      Assign : (destination) (source)
        Id : qre
        Const : 123
      If (condition) (body) (else)
        Op : ==
          Id : aa
          Id : bb
        Compound statement :
          Repeat :
            Op : <=
              Id : aa
              Id : cc
            Assign : (destination) (source)
              Id : aa
              Const : 5
        Return :
          Const : 1
      Assign : (destination) (source)
        ArrId : ee
          Const : 1
        Op : +
          Id : aa
```

```
        Id : aa
    Assign : (destination) (source)
      ArrId : ee
        Const : 2
      Op : -
        Id : bb
        Id : bb
    Assign : (destination) (source)
      ArrId : ee
        Const : 3
      Op : *
        Id : cc
        Id : cc
    Assign : (destination) (source)
      ArrId : ee
        Const : 4
      Op : /
        Id : dd
        Id : dd
    Assign : (destination) (source)
      ArrId : ee
        Const : 5
      Op : <
        Id : aa
        Id : bb
    Assign : (destination) (source)
      ArrId : ee
        Const : 6
      Op : >
        Id : bb
        Id : cc
    Assign : (destination) (source)
      ArrId : ee
        Const : 7
      Op : <=
        Id : cc
        Id : dd
    Assign : (destination) (source)
      ArrId : ee
        Const : 8
```

```
      Op : >=
        Id : dd
        Id : cc
    Return :
      Id : aa
Function declaration, name : main, return type : int
  Single parameter, name : (null), type : void
  Compound statement :
    Return :
      Const : 1
```

For **test1.cm** and **test2.cm**, these are test cases provided by Project 1, while **test3.cm** and **test4.cm** are additional test cases created in arrangement.