컴파일러설계 프로젝트2 (Parser) 보고서

2019019016 서시언

1. Compilation method and environment

- Ubuntu 20.04.3 LTS

2. Explanation about how to implement and how to operate

- 제공된 Makefile 사용

[main.c]

- NO_ANALYZE, TraceParrsee를 TRUE로 설정한다.

```
11 #define NO_PARSE FALSE
12 /* set NO_ANALYZE to TRUE to get a parser-only compiler */
13 #define NO_ANALYZE TRUE
```

```
39 /* allocate and set tracing flags */
40 int EchoSource = FALSE;
41 int TraceScan = FALSE;
42 int TraceParse = TRUE;
43 int TraceAnalyze = FALSE;
44 int TraceCode = FALSE;
```

[globals.h]

- yacc/glocals.h를 베이스로 한다.
- 주어진 AST format에 맞게 node의 종류를 추가해준다.

```
65 typedef enum {StmtK,ExpK} NodeKind;
66 typedef enum {CompoundK,IfK,IfElseK,ReturnK,NonValueReturnK,WhileK,VarDeclK,VarArrDeclK,FuncK,ParamK,ParamArrK,VoidParamK} StmtKind;
67 typedef enum {OpK,ConstK,VarK,AssignK,CallK} ExpKind;
68
69 /* ExpType is used for type checking */
70 typedef enum {Void,Int} ExpType;
71
72 #define MAXCHILDREN 3
73
74 typedef struct treeNode
75    { struct treeNode * child[MAXCHILDREN];
76      struct treeNode * sibling;
77      int lineno;
78      NodeKind nodekind;
79      union { StmtKind stmt; ExpKind exp;} kind;
80      union { TokenType op;
81              int val;
82              char * name; } attr;
83      ExpType type; /* for type checking of exps */
84    } TreeNode;
85
```

- Variable declaration에서, arr인지 구분을 위해 VarDeclK와 VarArrDeclK를 구분한다. 마찬가지로 ParamK와 ParamArrK도 구분하며, void parameter를 구분하기 위해 VoidParamK도 추가해준다.
- ExpType을 Void, Int로 설정한다.

[util.c]

- newStmtNode()와 newExpNode()는 기존의 코드를 활용한다.

```
60 /* Function newStmtNode creates a new statement
61  * node for syntax tree construction
62  */
63 TreeNode * newStmtNode(StmtKind kind)
64 { TreeNode * t = (TreeNode *) malloc(sizeof(TreeNode));
65   int i;
66   if (t==NULL)
67     fprintf(listing,"Out of memory error at line %d\n",lineno);
68   else {
69     for (i=0;i<MAXCHILDREN;i++) t->child[i] = NULL;
70     t->sibling = NULL;
71     t->nodekind = StmtK;
72     t->kind.stmt = kind;
73     t->lineno = lineno;
74   }
75   return t;
76 }
77
78 /* Function newExpNode creates a new expression
79  * node for syntax tree construction
80  */
81 TreeNode * newExpNode(ExpKind kind)
82 { TreeNode * t = (TreeNode *) malloc(sizeof(TreeNode));
83   int i;
84   if (t==NULL)
85     fprintf(listing,"Out of memory error at line %d\n",lineno);
86   else {
87     for (i=0;i<MAXCHILDREN;i++) t->child[i] = NULL;
88     t->sibling = NULL;
89     t->nodekind = ExpK;
90     t->kind.exp = kind;
91     t->lineno = lineno;
92     t->type = Void;
93   }
94   return t;
95 }
96
```

- enum인 ExpType을 문자열로 변환하기 위해 char* []인 expType을 선언한다.

```
// to convert ExpType to string
static char* expType[] = {"void","int"};
```

- 노드 종류마다 주어진 output format에 맞게 출력해준다.

```
    printSpaces();
    if (tree->nodekind==StmtK)
    { switch (tree->kind.stmt) {
        case CompoundK:
          fprintf(listing,"Compound Statement:\n");
          break;
        case IfK:
          fprintf(listing,"If Statement:\n");
          break;
        case IfElseK:
          fprintf(listing,"If-Else Statement:\n");
          break;
        case ReturnK:
          fprintf(listing,"Return Statement:\n");
          break;
        case NonValueReturnK:
          fprintf(listing,"Non-value Return Statement\n");
          break;
        case WhileK:
          fprintf(listing,"While Statement: \n");
          break;
        case VarDeclK:
          fprintf(listing,"Variable Declaration: name = %s, type = %s\n", tree->attr.name, expType[tree->type]);
          break;
        case VarArrDeclK:
          fprintf(listing,"Variable Declaration: name = %s, type = %s[]\n", tree->attr.name, expType[tree->type]);
          break;
        case FuncK:
          fprintf(listing,"Function Declaration: name = %s, return type = %s\n", tree->attr.name, expType[tree->type]);
          break;
        case ParamK:
          fprintf(listing,"Parameter: name = %s, type = %s\n", tree->attr.name, expType[tree->type]);
          break;
        case ParamArrK:
          fprintf(listing,"Parameter: name = %s, type = %s[]\n", tree->attr.name, expType[tree->type]);
          break;
        case VoidParamK:
          fprintf(listing,"Void Parameter\n");
          break;
        default:
          fprintf(listing,"Unknown StmtNode kind\n");
          break;
    else if (tree->nodekind==ExpK)
    { switch (tree->kind.exp) {
        case OpK:
          fprintf(listing,"Op: ");
          printToken(tree->attr.op,"\0");
          break;
        case ConstK:
          fprintf(listing,"Const: %d\n",tree->attr.val);
          break;
        case VarK:
          fprintf(listing,"Variable: name = %s\n",tree->attr.name);
          break;
        case AssignK:
          fprintf(listing,"Assign:\n");
          break;
        case CallK:
          fprintf(listing,"Call: function name = %s\n",tree->attr.name);
          break;
        default:
          fprintf(listing,"Unknown ExpNode kind\n");
          break;
      }
    }
```

[cminus.y]

- yacc/tiny.y를 베이스로 한다.
- 변수, 함수 등의 이름, 숫자를 저장하기 위해 스택을 사용한다.

  (기존 방식처럼 스택이 아닌 변수로 할 경우, overwrite되는 문제를 해결하기 위해)

```
char* nameStack[MAX_STACK];
int numStack[MAX_STACK];
int linenoStack[MAX_STACK];
ExpType typeStack[MAX_STACK];

int nameIndex = 0;
int numIndex = 0;
int linenoIndex = 0;
int typeIndex = 0;

void pushName(char* name) { nameStack[nameIndex++] = name; }
void pushNum(int num) { numStack[numIndex++] = num; }
void pushLineno(int lineno) { linenoStack[linenoIndex++] = lineno; }
void pushType(ExpType type) { typeStack[typeIndex++] = type; }

char* popName() { return nameStack[--nameIndex]; }
int popNum() { return numStack[--numIndex]; }
int popLineno() { return linenoStack[--linenoIndex]; }
ExpType popType() { return typeStack[--typeIndex]; }
```

- C-MINUS에 맞는 토큰들을 추가해준다.

```
%token IF ELSE WHILE RETURN INT VOID
%token ID NUM
%token ASSIGN EQ NE LT LE GT GE PLUS MINUS TIMES OVER LPAREN RPAREN LBRACE RBRACE LCURLY RCURLY SEMI COMMA
%token ERROR
```

- If / If-Else에서 발생하는 shift/reduce 문제를 해결하기 위해 %nonassoc을 이용해 우선순위를 정한다.

```
/* to solve shift/reduce conflict */
%nonassoc RPAREN
%nonassoc ELSE
```

- 기본적으로 주어진 BNF를 그대로 사용하되,
  - non-terminal id, num을 추가해주고, ID는 id, NUM은 num으로 대체한다. (overwrite문제 해결을 위해)

```
id              : ID
                                {
                                  pushName(copyString(tokenString));
                                  pushLineno(lineno);
                                }
                    ;
num             : NUM
                                {
                                  pushNum(atoi(tokenString));
                                  pushLineno(lineno);
                                }
                    ;
```

  - 다음과 같이 스택에 저장된 type, name, num을 사용한다. 또한 array declaration의 경우, child로 size를 추가해준다.

```
var_declaration         : type_specifier id SEMI
                                { $$ = newStmtNode(VarDeclK);
                                  $$->type = popType();
                                  $$->attr.name = popName();
                                  $$->lineno = popLineno();
                                }
                        | type_specifier id LBRACE num RBRACE SEMI
                                { $$ = newStmtNode(VarArrDeclK);
                                  $$->type = popType();
                                  $$->attr.name = popName();
                                  $$->lineno = popLineno();

                                  // add array size to child
                                  $$->child[0] = newExpNode(ConstK);
                                  $$->child[0]->attr.val = popNum();
                                  $$->child[0]->lineno = popLineno();
                                }
                        ;
type_specifier          : INT { pushType(Int); }
                        | VOID { pushType(Void); }
                        ;
```

  - relop, addop, mulop의 경우, 다음과 같이 대체해준다.

```
simple_expression       : additive_expression LE additive_expression
                                { $$ = newExpNode(OpK);
                                  $$->child[0] = $1;
                                  $$->child[1] = $3;
                                  $$->attr.op = LE;
                                }
                        | additive_expression LT additive_expression
                                { $$ = newExpNode(OpK);
                                  $$->child[0] = $1;
                                  $$->child[1] = $3;
                                  $$->attr.op = LT;
                                }
                        | additive_expression GT additive_expression
                                { $$ = newExpNode(OpK);
                                  $$->child[0] = $1;
                                  $$->child[1] = $3;
                                  $$->attr.op = GT;
                                }
                        | additive_expression GE additive_expression
                                { $$ = newExpNode(OpK);
                                  $$->child[0] = $1;
                                  $$->child[1] = $3;
                                  $$->attr.op = GE;
                                }
                        | additive_expression EQ additive_expression
                                { $$ = newExpNode(OpK);
                                  $$->child[0] = $1;
                                  $$->child[1] = $3;
                                  $$->attr.op = EQ;
                                }
                        | additive_expression NE additive_expression
                                { $$ = newExpNode(OpK);
                                  $$->child[0] = $1;
                                  $$->child[1] = $3;
                                  $$->attr.op = NE;
                                }
                        | additive_expression { $$ = $1; }
                        ;
additive_expression     : additive_expression PLUS term
                                { $$ = newExpNode(OpK);
                                  $$->child[0] = $1;
                                  $$->child[1] = $3;
```

[실행]

```
make
./cminus_parser [input file]
```

3. Example and Result Screenshot

- test.1.txt

```
/* A program to perform Euclid's
   Algorithm to computer gcd */

int gcd (int u, int v)
{
        if (v == 0) return u;
        else return gcd(v,u-u/v*v);
        /* u-u/v*v == u mod v */
}

void main(void)
{
        int x; int y;
        x = input(); y = input();
        output(gcd(x,y));
}
```

make

./cminus_parser test.1.txt