

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, TraceParse를 TRUE로 설정한다.

```
10 /* Set NO_PARSE to TRUE to get a scanner-only compiler */
11 #define NO_PARSE FALSE
12 /* set NO_ANALYZE to TRUE to get a parser-only compiler */
13 #define NO_ANALYZE TRUE
14
39 /* allocate and set tracing flags */
40 int EchoSource = FALSE;
41 int TraceScan = FALSE;
42 int TraceParse = TRUE; [Red Box]
43 int TraceAnalyze = FALSE;
44 int TraceCode = FALSE;
```

[globals.h]

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

```
61 typedef enum {StmtK,ExpK} NodeKind;
62 typedef enum {CompoundK,IfK,IfElseK,ReturnK,NonValueReturnK,WhileK,VarDeclK,VarArrDeclK,FuncK,ParamK,ParamArrK,VoidParamK} StmtKind;
63 typedef enum {OpK,ConstK,VarK,AssignK,CallK} ExpKind;
64
65 /* ExpType is used for type checking */
66 typedef enum {Void,Int} ExpType;
67
72 #define MAXCHILDREN 3
73
74 typedef struct treeNode
75 {
76     struct treeNode * child[MAXCHILDREN];
77     struct treeNode * sibling;
78     int lineno;
79     NodeKind nodekind;
80     union { StmtKind stmt; ExpKind exp;} kind;
81     union { TokenType op;
82             int val;
83             char * name; } attr;
84     ExpType type; /* for type checking of exps */
85 } TreeNode;
```

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

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

```
128 // to convert ExpType to string
129 static char* expType[] = {"void", "int"};
130
- 노드 종류마다 주어진 output format에 맞게 출력해준다.
131     printf("%s\n", tree->nodekind);
132     if (tree->nodekind==StmtK)
133     { switch (tree->kind.stmt) {
134         case CompoundK:
135             fprintf(listing,"Compound Statement:\n");
136             break;
137         case Ifk:
138             fprintf(listing,"If Statement:\n");
139             break;
140         case IfElseK:
141             fprintf(listing,"If-Else Statement:\n");
142             break;
143         case ReturnK:
144             fprintf(listing,"Return Statement:\n");
145             break;
146         case NonValueReturnK:
147             fprintf(listing,"Non-value Return Statement\n");
148             break;
149         case WhileK:
150             fprintf(listing,"While Statement: \n");
151             break;
152         case VarDeclK:
153             fprintf(listing,"Variable Declaration: name = %s, type = %s\n", tree->attr.name, expType[tree->type]);
154             break;
155         case VarArrDeclK:
156             fprintf(listing,"Variable Declaration: name = %s[], type = %s[]\n", tree->attr.name, expType[tree->type]);
157             break;
158         case FuncK:
159             fprintf(listing,"Function Declaration: name = %s, return type = %s\n", tree->attr.name, expType[tree->type]);
160             break;
161         case ParamK:
162             fprintf(listing,"Parameter: name = %s, type = %s\n", tree->attr.name, expType[tree->type]);
163             break;
164         case ParamArrK:
165             fprintf(listing,"Parameter: name = %s[], type = %s[]\n", tree->attr.name, expType[tree->type]);
166             break;
167         case VoidParamK:
168             fprintf(listing,"Void Parameter\n");
169             break;
170         default:
171             fprintf(listing,"Unknown StmtNode kind\n");
172             break;
173     }
174
175     else if (tree->nodekind==ExpK)
176     { switch (tree->kind.exp) {
177         case Opk:
178             fprintf(listing,"%Op: ");
179             printToken(tree->attr.op,"\\0");
180             break;
181         case ConstK:
182             fprintf(listing,"Const: %d\n",tree->attr.val);
183             break;
184         case Vark:
185             fprintf(listing,"Variable: name = %s\n",tree->attr.name);
186             break;
187         case AssignK:
188             fprintf(listing,"Assign:\n");
189             break;
190         case CallK:
191             fprintf(listing,"Call: function name = %s\n",tree->attr.name);
192             break;
193         default:
194             fprintf(listing,"Unknown ExpNode kind\n");
195             break;
196     }
197 }
```

[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)

```
22 char* nameStack[MAX_STACK];
23 int numStack[MAX_STACK];
24 int linenoStack[MAX_STACK];
25 ExpType typeStack[MAX_STACK];
26
27 int nameIndex = 0;
28 int numIndex = 0;
29 int linenoIndex = 0;
30 int typeIndex = 0;
31
32 void pushName(char* name) { nameStack[nameIndex++] = name; }
33 void pushNum(int num) { numStack[numIndex++] = num; }
34 void pushLinenos(int lineno) { linenoStack[linenoIndex++] = lineno; }
35 void pushType(ExpType type) { typeStack[typeIndex++] = type; }
36
37 char* popName() { return nameStack[--nameIndex]; }
38 int popNum() { return numStack[-numIndex]; }
39 int popLinenos() { return linenoStack[--linenoIndex]; }
40 ExpType popType() { return typeStack[--typeIndex]; }
```

- C-MINUS에 맞는 토큰들을 추가해준다.

```
44 %token IF ELSE WHILE RETURN INT VOID
45 %token ID NUM
46 %token ASSIGN EQ NE LT LE GT GE PLUS MINUS TIMES OVER LPAREN RPAREN LBRACE RBRACE LCURLY RCURLY SEMI COMMA
47 %token ERROR
```

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

```
49 /* to solve shift/reduce conflict */
50 %nonassoc RPAREN
51 %nonassoc ELSE
```

- 기본적으로 주어진 BNF를 그대로 사용하되,

- non-terminal id, num을 추가해주고, ID는 id, NUM은 num으로 대체한다. (overwrite문제 해결을 위해)

```
58 id      : ID
59           {
60             pushName(copyString(tokenString));
61             pushLineno(lineno);
62           }
63 ;
64 num     : NUM
65           {
66             pushNum(atoi(tokenString));
67             pushLineno(lineno);
68           }
69 ;
```

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

```
85 var_declaration : typeSpecifier id SEMI
86   {
87     $$ = newStmtNode(VarDeclK);
88     $$->type = popType();
89     $$->attr.name = popName();
90     $$->lineno = popLineno();
91   }
92   | typeSpecifier id LBRACE num RBRACE SEMI
93   {
94     $$ = newStmtNode(VarArrDeclK);
95     $$->type = popType();
96     $$->attr.name = popName();
97     $$->lineno = popLineno();
98     // add array size to child
99     $$->child[0] = newExpNode(ConstK);
100    $$->child[0]->attr.val = popNum();
101    $$->child[0]->lineno = popLineno();
102  }
103 typeSpecifier : INT { pushType(Int); }
104   | VOID { pushType(Void); }
105 ;
```

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

```
231 simple_expression : additive_expression [LE] additive_expression
232   {
233     $$ = newExpNode(OpK);
234     $$->child[0] = $1;
235     $$->child[1] = $3;
236     $$->attr.op = LE;
237   }
238   | additive_expression [LT] additive_expression
239   {
240     $$ = newExpNode(OpK);
241     $$->child[0] = $1;
242     $$->child[1] = $3;
243     $$->attr.op = LT;
244   }
245   | additive_expression [GT] additive_expression
246   {
247     $$ = newExpNode(OpK);
248     $$->child[0] = $1;
249     $$->child[1] = $3;
250     $$->attr.op = GT;
251   }
252   | additive_expression [GE] additive_expression
253   {
254     $$ = newExpNode(OpK);
255     $$->child[0] = $1;
256     $$->child[1] = $3;
257     $$->attr.op = GE;
258   }
259   | additive_expression [EQ] additive_expression
260   {
261     $$ = newExpNode(OpK);
262     $$->child[0] = $1;
263     $$->child[1] = $3;
264     $$->attr.op = EQ;
265   }
266   | additive_expression [NE] additive_expression
267   {
268     $$ = newExpNode(OpK);
269     $$->child[0] = $1;
270     $$->child[1] = $3;
271     $$->attr.op = NE;
272   }
273 additive_expression : additive_expression [PLUS] term
274   {
275     $$ = newExpNode(OpK);
276     $$->child[0] = $1;
277     $$->child[1] = $3;
278   }
```

[실행]

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

3. Example and Result Screenshot

- test.1.txt

```
1 /* A program to perform Euclid's
2  * Algorithm to computer gcd */
3
4 int gcd (int u, int v)
5 {
6     if (v == 0) return u;
7     else return gcd(v,u-u/v*v);
8     /* u-u/v*v == u mod v */
9 }
10
11 void main(void)
12 {
13     int x; int y;
14     x = input(); y = input();
15     output(gcd(x,y));
16 }
```

```
make
```

```
./cminus_parser test.1.txt
```

```
stun@stun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/2_Parser$ ./cminus_parser test.1.txt
C-MINUS COMPILATION: test.1.txt

Syntax tree:
  Function Declaration: name = gcd, return type = int
    Parameter: name = u, type = int
    Parameter: name = v, type = int
  Compound Statement:
    If-Else Statement:
      Op: ==
        Variable: name = v
        Const: 0
      Return Statement:
        Variable: name = u
      Return Statement:
        Call: function name = gcd
        Variable: name = v
      Op: -
        Variable: name = u
      Op: *
        Op: /
          Variable: name = u
          Variable: name = v
        Variable: name = v
  Function Declaration: name = main, return type = void
  Void Parameter
  Compound Statement:
    Variable Declaration: name = x, type = int
    Variable Declaration: name = y, type = int
  Assign:
    Variable: name = x
    Call: function name = input
  Assign:
    Variable: name = y
    Call: function name = output
    Call: function name = gcd
      Variable: name = x
      Variable: name = y
```