project2. Parser

2015004302 CSE 곽상원

목차

---

1. Compile 환경

- Ubuntu14.04, gcc, flex, yacc
- make command를 입력해 “cminus” 빌드 및 생성

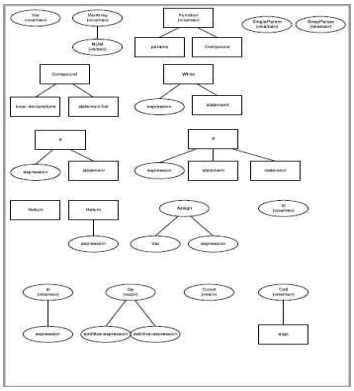
2. Project purpose

- 이전 project에서 만든 scanner를 바탕으로 parser 구현
- CFG의 Nonterminal & terminal은 아래의 명세와 그림을 따른다.

▶CFG

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*

▶Node



3. Code 수정사항

< globals.h >

```
typedef enum {DecK,ParK,StmtK,ExpK} NodeKind;
typedef enum {VarK,VararrK,FuncK} DecKind;
typedef enum {SinpK,ArrpK} ParKind;
typedef enum {IfK,IfelseK,CmstmtK,WhileK,ReturnK} StmtKind;
typedef enum {AssignK,OpK,ConstK,IdK,CallK} ExpKind;

/* ExpType is used for type checking */
typedef enum {Void,Int} ExpType;
```

Scanner에서 정의된 Token(terminal)에 대한 노드가 선언된 곳으로, 크게 DecK, ParK, Stmtk, ExpK로 나누었고 위의 Node 그림에 맞게 노드를 추가했다.
DecKind: 변수선언과 함수 선언을 위한 노드이다.
ParKind: Parameter을 위한 노드이다.
StmtKind: Statement를 위한 노드이다.
ExpKind: expression을 위한 노드이다.

```
union { DecKind dec; ParKind par; StmtKind stmt; ExpKind exp;} kind;
```

추가적으로 TreeNode 구조체의 kind 멤버에 모든 노드종류를 추가했다.

< cminus.y >

```
%union {
    int val;
    char* str;
    struct treeNode* node;
    int Type;
};
```

Scanner에서 얻은 토큰에 대한 정보를 얻기 위해 yylval 전역변수를 이용했다. yylval의 type인 YYSTYPE을 재정의하기 위해 위와 같이 %union구문을 이용했다. 상수를 담는 val, 문자열을 담는 str, 노드를 담는 node, type정보를 구분하기 위한 Type 등 4가지 멤버를 추가했다.

```
%type <Type> type_specifier
%type <node> program declaration_list declaration var_declaration fun_dec
%token <str> IF ELSE INT RETURN VOID WHILE
%token <val> NUM
%token <str> ID PLUS MINUS TIMES OVER LT LE GT GE EQ NE ASSIGN SEMI COMMA
%token <str> ERROR
```

Scanner에서 받아들인 토큰과 CFG에서 쓰이는 nonterminal을 정의한 부분으로, 각자 YYSTYPE의 멤버에 맞게 적절한 자료형을 할당했다.

parse tree를 형성 시 각 노드에서 일어나는 유형은 크게 다음과 같다.

① Sibling이 늘어나는 경우

```
declaration_list    : declaration_list declaration
                        { TreeNode* t = $1;
                            if(t != NULL){
                            while(t->sibling != NULL)
                                t = t ->sibling;
                            t->sibling = $2;
                            $$ = $1; }
                            else $$ = $2;
                        }
                    | declaration { $$ = $1; }
                    ;
```

위의 경우는 sibling이 오른쪽으로 늘어나는 경우로, 가장 왼쪽의 declaration_list에 declaration을 sibling으로 추가해주면서 tree를 확장한다.

② Child가 늘어나는 경우

```
fun_declaration     : type_specifier ID LPAREN params RPAREN compound_stmt
                      { $$ = newDecNode(FuncK);
                          $$->attr.name = $2;
                          $$->type = $1;
                          $$->child[0] = $4;
                          $$->child[1] = $6;
                      }
                    ;
```

Node 그림에 맞게 노드를 구성했다. 위의 경우, FuncK형 노드를 생성해 그 자식으로 params와 compound_stmt를 추가했다.

③ 노드만 생성되는 경우

```
addop               : PLUS { $$ = newExpNode(OpK); $$->attr.op = PLUS; }
                    | MINUS { $$ = newExpNode(OpK); $$->attr.op = MINUS; }
                    ;
```

해당 토큰에 맞는 노드를 생성하고 그 속성을 노드에 저장한다. 위의 경우, PLUS일 때 OpK형 노드를 생성하고 attr.op에 PLUS를 저장해준다.

④ 나머지 경우

```
statement           : expression_stmt { $$ = $1; }
                    | compound_stmt { $$ = $1; }
                    | selection_stmt { $$ = $1; }
                    | iteration_stmt { $$ = $1; }
                    | return_stmt { $$ = $1; }
                    ;
```

'$$ = $1'을 명시적으로 표현한다.

Dangling else problem은 다음과 같이 처리했다.

```
%nonassoc "then"
%nonassoc ELSE
```

```
selection_stmt      : IF LPAREN expression RPAREN statement     %prec "then"
                        { $$ = newStmtNode(IfK);
                            $$->child[0] = $3;
                            $$->child[1] = $5;
                        }
                    | IF LPAREN expression RPAREN statement ELSE statement
                        { $$ = newStmtNode(IfelseK);
                            $$->child[0] = $3;
                            $$->child[1] = $5;
                            $$->child[2] = $7;
                        }
                    ;
```

if if else 구문에서 2번째 if가 else와 결합되기 위해서는 ELSE토큰이 shift되는 경우와 selection_stmt -> if(expression) statement에 의해 reduce되는 경우 중에서 shift가 일어나야한다. 이를 위해 %nonassoc구문과 %prec를 이용해 "then"이 ELSE보다 우선순위가 낮게 정의되어 항상 ELSE토큰을 받아들여 shift/conflict 문제를 해결할 수 있다.

< cminus.l >

```
{number}        { yylval.val = atoi(yytext); return NUM;}
{identifier}    { yylval.str = strdup(yytext); return ID; }
```

NUM, ID는 yylval에 해당 숫자, 문자열을 저장해서 Parser에서 사용할 수 있도록 한다.

< util.c >

각 노드종류별 노드생성 함수를 정의하고, printTree()에 각 노드에 알맞은 출력형태를 지정했다.

4. 실행예제 및 결과

▶test파일(ctest.c)

```
int gcd(int u,int v)
{
    if(v==0) return u;
        if (v==1) return a;
        else return asd(a,b,c);
    /*asdsad*asd/asdasd*/
}

void main(void)
{
    int x;int y;int a[5];
    x=input();y=input();
    output(gcd(x,y));
}
```

▶실행결과

```
ksw@ubuntu:~/2019_ELE4029_2015004302/2_Parser/source$ ./cminus ctest.c

C-MINUS COMPILATION: ctest.c

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)
        Op : ==
          Id : v
          Num : 0
        Return :
          Id : u
      If (condition) (body) (else)
        Op : ==
          Id : v
          Num : 1
        Return :
          Id : a
        Return :
          Call, name : asd, with arguments below
            Id : a
            Id : b
            Id : c
  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
      Array Var declaration, name : a, type : int
        Num : 5
      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
```