Compiler Parser Report

학부	컴퓨터소프트웨어학부
학번	2018008977
이름	이종범

Environment

Compiler: clang 14.0.3

Flex: 2.6.4 Apple(flex-34)

OS: Sonoma 14.1

Device: Apple MacBook Pro M1 Max 32GB

How to Run?

- 경로 이동 후 Compile

  $ make all

- 파일 실행

  $ ./cminus_parser [C-Minus File to Run]

주요 구현 사항

- cminus.y 파일을 수정하여 C-Minus Parser가 정상적으로 동작하도록 함
- 추가적인 테스트 코드를 작성하고, 자동화 스크립트를 통해 테스트 함

cminus.y 구현 내용

- fun_declaration

```
type_specifier identifier LPAREN params RPAREN compound_stmt
  {
        $$ = newTreeNode(FunctionDecl);
        $$->lineno = lineno;
        $$->type = $1->type;
        $$->name = $2->name;
        $$->child[0] = $4;
        $$->child[1] = $6;
        free($1);
        free($2);
  }
```

  - 해당 노드의 Child로 params와 compound_stmt 지정
  - type과 name은 $1과 $2의 값을 사용
  - $1과 $2는 AST의 노드로 포함되지 않으므로 동적할당 해제

- params

```
param_list { $$ = $1; }
VOID
  {
        $$ = newTreeNode(Params);
        $$->lineno = lineno;
        $$->type = Void;
        $$->flag = TRUE;
  }
```

  - param_list의 경우 해당 노드를 params로 사용
  - VOID의 경우 Params 노드를 생성하고, VoidParameter로 출력될 수 있도록 flag를 True로 설정

- param_list

```
: param_list COMMA param
    {
            YYSTYPE t = $1;
            if (t != NULL)
            {
                while (t->sibling != NULL) t = t->sibling;
                t->sibling = $3;
                $$ = $1;
            }
            else $$ = $3;
    }
| param { $$ = $1; }
```

- param_list의 가장 끝 sibling을 param으로 지정
- declaration_list 로직을 참고하여 적용
- param의 경우 노드 그대로 param_list로 할당

- param

```
: type_specifier identifier
    {
            $$ = newTreeNode(Params);
            $$->lineno = $2->lineno;
            $$->type = $1->type;
            $$->name = $2->name;
            free($1);
            free($2);
    }
| type_specifier identifier LBRACE RBRACE
    {
            $$ = newTreeNode(Params);
            $$->lineno = $2->lineno;
            if ($1->type == Integer) $$->type = IntegerArray;
            else if ($1->type == Void) $$->type = VoidArray;
            else $$->type = None;
            $$->name = $2->name;
            free($1);
            free($2);
    }
;
```

- Params 노드를 생성하고, type과 name을 각각 $1과 $2에서 가져옴
- $1과 $2는 값을 AST에 포함되지 않으므로 동적 할당 해제
- LBRACE와 RBRACE가 포함된 경우에는 type_specifier를 보고 대응되는 Type의 Array를 Type으로 지정

- compound_stmt

```
: LCURLY local_declarations statement_list RCURLY
    {
            $$ = newTreeNode(CompoundStmt);
            $$->lineno = lineno;
            $$->child[0] = $2;
            $$->child[1] = $3;
    }
```

- CompoundStmt 노드를 생성하고, child로 local_declarations과 statement_list를 지정

- local_declarations

```
: local_declarations var_declaration
    {
            YYSTYPE t = $1;
            if (t != NULL)
            {
                while (t->sibling != NULL) t = t->sibling;
                t->sibling = $2;
                $$ = $1;
            }
            else $$ = $2;
    }
| empty { $$ = $1; }
```

- local_declaration의 가장 끝 sibling을 var_declaration으로 지정
- Declaration_list 로직을 참교하여 적용
- Empty의 경우 노드 그대로 statement_list로 할당

- statement

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

- 노드 그대로 statement로 할당

- selection_stmt

```
: IF LPAREN expression RPAREN statement ELSE statement
    {
            $$ = newTreeNode(IfStmt);
            $$->lineno = lineno;
            $$->flag = TRUE;
            $$->child[0] = $3;
            $$->child[1] = $5;
            $$->child[2] = $7;
    }
    | IF LPAREN expression RPAREN statement
    {
            $$ = newTreeNode(IfStmt);
            $$->lineno = lineno;
            $$->child[0] = $3;
            $$->child[1] = $5;
    }
    ;
```

- IF ELSE Statement의 경우 expression, statement, statement를 child로 지정
- Flag를 TRUE로 설정하여 IF Statement로 출력될 수 있도록 함
- 일반 IF Statement의 경우 expression과 statement를 child로 지정

- expression_stmt

```
: expression SEMI { $$ = $1; }
| SEMI { $$ = NULL; }
```

- expression SEMI의 경우 expression 노드를 그대로 할당
- SEMI의 경우 노드에 NULL을 할당

- iteration_stmt

```
: WHILE LPAREN expression RPAREN statement
    {
            $$ = newTreeNode(WhileStmt);
            $$->lineno = lineno;
            $$->child[0] = $3;
            $$->child[1] = $5;
    }
;
```

- WhileStmt 노드를 만들고 expression과 statement를 child로 지정

- return_stmt

```
: RETURN SEMI
    {
            $$ = newTreeNode(ReturnStmt);
            $$->lineno = lineno;
            $$->flag = TRUE;
    }
| RETURN expression SEMI
    {
            $$ = newTreeNode(ReturnStmt);
            $$->lineno = lineno;
            $$->child[0] = $2;
    }
;
```

- ReturnStmt 노드 생성
- return 하고자 하는 expression이 있는 경우 expression을 child 노드로 이를 지정
- non-value return의 경우 flag를 TRUE로 설정

- expression

```
: var ASSIGN expression
    {
            $$ = newTreeNode(AssignExpr);
            $$->lineno = lineno;
            $$->child[0] = $1;
            $$->child[1] = $3;
    }
| simple_expression { $$ = $1; }
;
```

- Assignment Statement의 경우 var과 expression을 child로 지정
- simple_expression인 경우 노드를 그대로 사용

- var

```
: identifier
    {
            $$ = newTreeNode(VarAccessExpr);
            $$->lineno = $1->lineno;
            $$->name = $1->name;
            free($1);
    }
| identifier LBRACE expression RBRACE
    {
            $$ = newTreeNode(VarAccessExpr);
            $$->lineno = $1->lineno;
            $$->name = $1->name;
            $$->child[0] = $3;
            free($1);
    }
;
```

-VarAccessExpr 노드를 생성
-배열 변수의 Access일 경우, expression도을 child로 지정

- simple_expression

```
: additive_expression relop additive_expression
    {
            $$ = newTreeNode(BinOpExpr);
            $$->lineno = lineno;
            $$->opcode = $2->opcode;
            $$->child[0] = $1;
            $$->child[1] = $3;
            free($2);

    }
| additive_expression { $$ = $1; }
```

- relop의 opcode를 opcode로 그대로 활용
- 양변을 child로 사용
- additive_expression만 존재하는 경우 그대로 노드로 할당

- relop
  - 각 연산자에 대응하는 opcode를 지정
- additive_expression

```
: additive_expression addop term
    {
            $$ = newTreeNode(BinOpExpr);
            $$->lineno = lineno;
            $$->opcode = $2->opcode;
            $$->child[0] = $1;
            $$->child[1] = $3;
            free($2);
    }
| term { $$ = $1; }
```

- $2의 opcode 그대로 사용하고 양변을 child로 지정
- 단일 terrm의 경우 Node로 그대로 사용

- addop
  - +, -에 각각 대응되는 opcode를 노드에 지정
- term
  - additive_expression과 동일
- mulop
  - *, /에 각각 대응되는 opcode를 노드에 지정
- factor

    - 각 단일 노드를 그대로 사용
- call
    - identifier에서 호출하고자 하는 함수 이름을 가져오고, args를 child로 사용
- args
    - 각 단일 노드를 그대로 사용
- args_list
    - arg_list의 가장 끝 sibling을 expression으로 지정
    - declaration_list 로직을 참고하여 적용
    - expression의 경우 노드 그대로 arg_list로 할당

Test

- 제공된 Example 코드와 추가적인 C-Minus 코드를 작성하여 테스트
- 해당 명령어를 이용하여 실행
    - $ sh ./test.sh
- 결과 AST와 생성된 AST를 비교하여, diff 명령어를 활용하여 차이를 보여줌
- 제공된 TestSet 외에 Dangling Else와 단일 return;을 테스트하는 케이스를 추가함

Result

```
leejongbeom@JBMacBookPro 2_Parser % sh test.sh
CMinus Parser Test 1
2c2
< C-MINUS COMPILATION: ./example/test.1.txt
---
> C-MINUS COMPILATION: ./test.1.txt
CMinus Parser Test 2
2c2
< C-MINUS COMPILATION: ./example/test.2.txt
---
> C-MINUS COMPILATION: ./test.2.txt
CMinus Parser Test 3
CMinus Parser Test 4
```

- 파일명 외에 차이가 없는 것을 확인할 수 있음
- Parser가 정상적으로 동작
- Dangling Else Problem과 return statement도 정상적으로 구현되었음을 확인