

Project #2. Parser

컴퓨터소프트웨어학부

2018008722 유수영

1. Compile 환경 및 방법

→ Make를 이용하여 compile(Makefile을 사용하여 compile)

```
yusuyoung@yusuyoung-VirtualBox:~/2020_ELE4029_2018008722/2_Parser$ ls
analyze.c  cminus.l  fakecminus.y  Makefile  sample.tm  syntab.c  tm.c  y.output
analyze.h  cminus.y  globals.h     parse.c   sample.tny  syntab.h  util.c
cgen.c     code.c    lex           parse.h   scan.c      test.1.txt  util.h
cgen.h     code.h    main.c        readme.unx  scan.h      test.2.txt  yacc
yusuyoung@yusuyoung-VirtualBox:~/2020_ELE4029_2018008722/2_Parser$ make
yacc -d -v cminus.y
cminus.y:311 parser name defined to default : "parse"
conflicts: 1 shift/reduce
gcc -c main.c
main.c:48:1: warning: return type defaults to 'int' [-Wimplicit-int]
main( int argc, char * argv[] )
^
gcc -c util.c
util.c:125:8: warning: type defaults to 'int' in declaration of 'indentno' [-Wimplicit-int]
static indentno = 0;
^
flex cminus.l
gcc -c lex.yy.c
gcc -c y.tab.c
/usr/share/bison+/bison.cc: In function 'yyparse':
/usr/share/bison+/bison.cc:198:24: warning: implicit declaration of function 'yyerror'; did you
mean 'yyerrok'? [-Wimplicit-function-declaration]
#define YY_ERROR yyerror
^
/usr/share/bison+/bison.cc:667:4: note: in expansion of macro 'YY_parse_ERROR'
YY_ERROR("parser stack overflow");
^
gcc -c syntab.c
gcc -c analyze.c
gcc -c code.c
gcc -c cgen.c
gcc main.o util.o lex.yy.o y.tab.o syntab.o analyze.o code.o cgen.o -o cminus -lfl
yusuyoung@yusuyoung-VirtualBox:~/2020_ELE4029_2018008722/2_Parser$ ls
analyze.c  fakecminus.y  main.o  sample.tny  test.1.txt  yacc
analyze.h  cminus.l     globals.h  Makefile  scan.c      test.2.txt  y.output
analyze.o  cminus.y    lex       parse.c   scan.h      tm.c        y.tab.c
cgen.c     code.c      lex.yy.c  parse.h   syntab.c    util.c      y.tab.h
cgen.h     code.h      lex.yy.o  readme.unx  syntab.h    util.h      v.tab.o
cgen.o     code.o      main.c    sample.tm  syntab.o    util.o
yusuyoung@yusuyoung-VirtualBox:~/2020_ELE4029_2018008722/2_Parser$
```

→ make명령어를 통해 Makefile(pdf의 Makefile 내용을 그대로 사용) 의 내용들이 실행되어 실행파일 cminus가 생성된 것을 볼 수 있다.

2. Cminus.y 코드 작성 및 파일 별 코드 변경 내용

<globals.h>

- 1) yacc폴더에 있는 globals.h로 상위 폴더의 globals.h를 덮어쓰기
- 2) 새로운 node의 kind들을 추가해주었다. Parse를 할 때, 각 node의 kind에 따라 출력을 달리 해주는데, 올바른 출력을 위해 IfThenK, CompoundK, ReturnK, FunDeK, SingleParaK, ArrParaK, CallK, VarDeK, TypeK를 추가해주었다. 각각 임의로 StmtKind와 ExpKind에 넣어주었다.

1- IfThenK : if-then-else문을 가지는 token이 들어온 경우의 node

- 2- CompoundK : function declaration과 statement에서 파생될 수 있는 compound statement의 token을 가지고 있을 때의 node
 - 3- ReturnK : return을 하는 token이 들어온 경우의 node로, return_stmt일 때 사용
 - 4- FunDeK : function declaration의 rule에 맞는 node일 경우로, fun_declaration일 때 사용
 - 5- SingleParaK : single parameter의 rule에 맞는 token이 들어온 경우로, param->type_specifier ID인 경우에 사용
 - 6- ArrParaK : array parameter의 rule에 맞는 token이 들어온 경우로, param->type_specifier ID [] 인 경우에 사용
 - 7- CallK : 함수 호출의 형태를 가지는 token이 들어온 경우로, call-> ID(args) 일 때 사용
 - 8- VarDeK : 변수 선언인 var_declaration의 경우에 맞는 token이 들어온 경우 사용
 - 9- TypeK : 각 변수와 함수, parameter의 type을 저장할 수 있도록 해주는 node로, type_specifier에서 사용
- 3) treeNode struct의 attr에 type이라는 char * 변수를 추가해주었다. Type의 값은 함수, 변수, parameter들의 첫번째 자식 노드에서 설정해주며, "int" 혹은 "void"의 값을 가지며 각 함수, 변수, parameter들이 어느 type인지를 저장하고 있는 변수이다.

<scan.h & cminus.l>

- 1) parser에서 용이하게 ID의 값들을 저장할 수 있도록 char previousToken[MAXTOKENLEN+1]을 추가 해주었다. previousToken은 tokenString 직전에 읽었던 token을 저장하고 있는 값이며, 아래와 같이 사용된다.

```
var_declaration : type_specifier ID{ 1
    savedName = copyString previousToken; 2
    savedLineNo = lineno;
    $$ = newExprNode(VarDeK);
    3 $$->attr.name = savedName;
    $$->lineno = savedLineNo;
}
```

- ➔ previousToken은 1번에 있는 ID 위치에 오는 token의 실제 값을 저장하고 있다. 2번과 같이 previousToken의 값을 copyString을 통해 savedName에 저장한 뒤, 3번과 같이 node의 name 변수에 저장해준다. 이렇게 ID의 값을 저장할 수 있다.

<main.c>

- 1) syntax tree만을 출력할 수 있도록 NO_PARSE의 값을 FALSE로, NO_ANALZE의 값은 TRUE로 설정하였다.

- 2) Facing flags에 관한 변수들 역시, TraceParse만 TRUE로 설정하고, 나머지 EchoSource, TraceScan, TraceAnalyze, TraceCode의 값은 FALSE로 설정하였다.

<cminus.y>

- 1) Yacc의 timy.y를 그대로 복사하여 이름을 바꾼 다음, 내용을 cminus에 맞도록 수정해주었다.
- 2) Definition 수정 : token들을 수정해주었다. 저번 과제의 scanner 구현에 사용했던 globals.h에서 정의했던 token들을 그대로 작성해주었다.
- 3) Rule 수정 : C-minus의 BNF grammar를 그대로 구현하되, relop -> <= | < | > | >= | == != 과 addop -> + | - , mulop -> * | / 는 해당 rule들이 사용되는 rule에 substitute하여 구현해주었다. (EX) term -> term mulop factor를 term->term * factor와 term->term / factor로 변경) 이렇게 바꾸어 구현한 이유는, 각 operator들이 operand들의 parent가 되어야 하기 때문이며, 각 연산자들은 node의 op 부분에 저장될 수 있도록 한다.
 - 1- Program -> declaration_list : 코드의 시작 부분이 오기 때문에 savedTree에 program을 넣어준다.
 - 2- Declaration_list -> declaration_list declaration : declaration_list의 마지막 sibling 다음에 declaration을 연결해준 뒤, \$\$에 넣어준다.
 - 3- Var_declaration -> type_specifier ID SEMI : 변수 하나를 선언하는 rule로, 변수의 이름인 ID를 node의 name에 저장하고 변수의 type인 type_specifier를 node의 child[0]에 저장
 - 4- Var_declaration -> type_specifier ID LBRACE NUM RBRACE SEMI : 배열을 선언하는 rule로, 변수의 이름인 ID를 node의 name에 저장하고, 변수의 type인 type_specifier를 node의 child[0]에 저장한 뒤, child[1]은 상수를 저장하는 node로 만든 다음, NUM에 해당하는 배열의 크기를 child[1]->attr.val에 저장 (배열 variable의 경우 배열의 크기를 child로 가져야 하기 때문에 child에 그 값을 넣어주었다.)
 - 5- Type_sepcifier -> INT / type_specifier -> VOID : type이 int인지 void인지를 설정해주며, TypeK의 kind를 가지는 node를 만들어 attr.type에 'int' 또는 'void'를 저장해준다.
 - 6- Fun_declaration -> type_specifier ID LPAREN params RPAREN compound_stmt : 함수 선언하는 부분으로, 함수의 이름인 ID는 node에 name으로 저장해주고, params는 child[1]에, compound_stmt는 child[2]에 저장해준다. 함수의 type은 child[0]에 저장해준다. (함수의 경우 tree의 모양이 함수의 이름을 가지는 parent와 왼쪽 child에 params, 오른쪽 child에 compound가 들어가야 하기 때문에 이처럼 코드 작성)
 - 7- Params -> VOID : parameter에 void가 있는 경우, SingleParaK kind의 node를 만들어 주고, child에 TypeK의 kind를 가지는 node를 만들어 type을 void로 작성해준다. (나중에 tree를 출력할 때 name은 null이 나오고, type은 void가 나올 수 있게 하기 위함)
 - 8- Param_list -> param_list param : param_list(\$1)의 마지막 sibling에 param을 연결

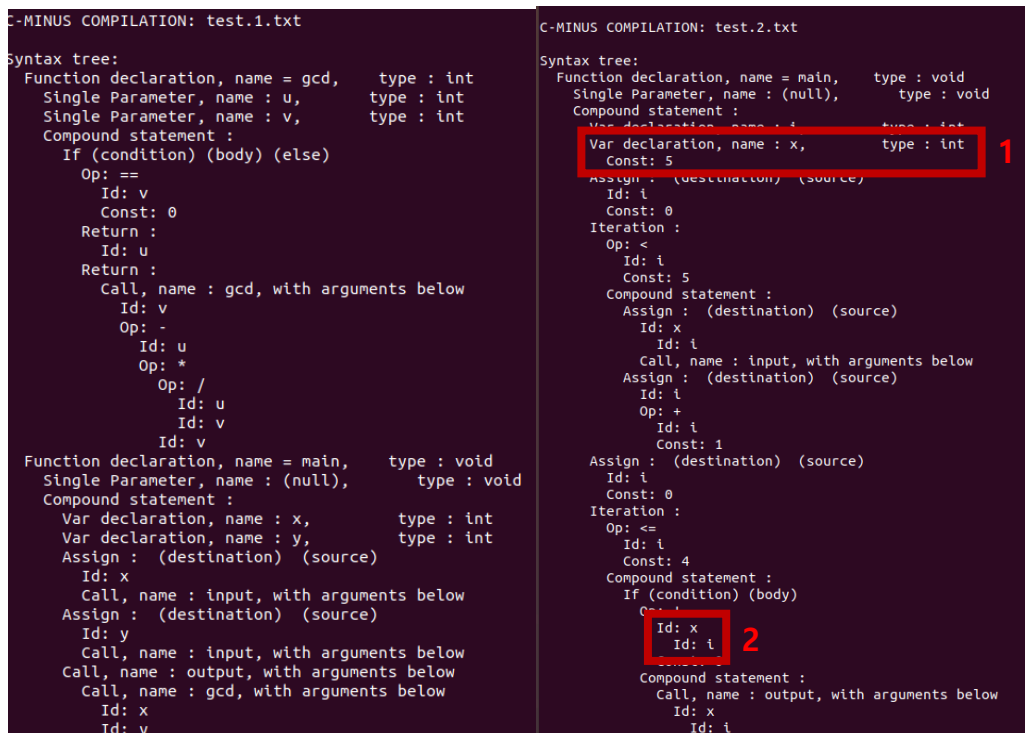
- 9- Param -> type_specifier ID : single parameter의 형태이기 때문에, SingleParaK의 node를 형성하여 name에 parameter의 이름인 ID를 저장하고, type_specifier는 child[0]에 저장해준다.
- 10- Param -> type_specifier ID LBRACE RBRACE : 배열 parameter인 경우로, ArrParaK의 node를 만들어 9번과 동일하게 이름과 type을 저장해준다.
- 11- Compound_stmt -> LCURLY local_declarations statement_list RCURLY : 대괄호가 있는 token들로, compound가 parent에 들어가고, 각각이 child로 들어가기 때문에 CompoundK의 parent node를 만들어, local_declarations은 child[0]에, statement_list는 child[1]에 저장해준다.
- 12- Statement_list -> statement_list statement : statement_list(\$1)의 마지막 sibling에 statement를 연결
- 13- Selection_stmt -> IF LPAREN expression RPAREN statement : if문을 나타내는 rule로, IfK의 node를 만들어, expression은 child[0]에, statement는 child[1]에 저장(if는 tree에서 if가 parent가 되고, 각각이 child로 들어가기 때문)
- 14- Selection_stmt -> IF LPAREN expression RPAREN statement ELSE statement : 13번과 동일하나, else가 추가된 rule로, IfThenK의 node를 만들고, else뒤의 statement도 child로 들어가기 때문에 child[2]에 statement를 저장해준다.
- 15- Iteration_stmt -> WHILE LPAREN expression RPAREN statement : while을 사용한 반복문으로, tree에서 while이 parent에, 나머지 expression과 statement는 child로 들어가기 하므로 IterK node를 형성하고 expression을 child[0]에, statement는 child[1]에 넣어준다.
- 16- Return_stmt -> RETURN expression SEMI : expression을 return하는 구문으로, tree에서 return이 parent로, expression이 child가 되어야 하므로, ReturnK node를 만들어 expression을 child[0]로 넣어준다.
- 17- Expression -> var ASSIGN expression : 변수에 expression을 assign하는 rule로, AssignK node를 만들고, var와 expression을 각각 child에 넣어주어 parent에 Assign이, child에 var와 expression이 오도록 해준다. 그리고, attr.op를 ASSIGN으로 설정해준다.
- 18- Var -> ID : 단순한 변수이므로 IdK node를 만들고, 변수의 이름인 ID 자리의 값을 attr.name에 저장해준다.
- 19- Var -> ID LBRACE expression RBRACE : 배열 형태의 변수로, 18번과 같이 변수의 이름을 저장하고, expression을 child에 저장한다.
- 20- Simple_expression -> additive_expression LE / LT / GT / GE / NE / EQ additive_expression, additive_expression -> additive_expression PLUS / MINUS term, term -> term TIMES / OVER factor : 모두 OpK node를 만들어주고, 양쪽의 additive_expression 혹은 term, factor들을 child로 만들어준 다음, 맞는 attr.op를 설정해준다.

- 21- Factor -> NUM : 상수를 가지는 ConstK node를 만들어주고, NUM의 값을 attr.val에 저장해준다.
- 22- Call -> ID LPAREN args RPAREN : 함수 호출하는 rule로, 함수의 이름인 ID는 name에 저장해주고, args를 child에 넣어준다.
- 23- Arg_list -> arg_list COMMA expression : arg_list의 마지막 sibling에 expression을 연결
- ➔ 위의 설명은 rule들 중 코드가 \$\$ = \$1 인 것은 제외하고 작성한 것이며, terminal들은 token의 이름으로 표현

<util.c>

- 1) printTree에서 알맞게 tree를 출력할 수 있도록 새로 만든 kind들의 case를 모두 추가해주었다.

3. 실행 내용(리눅스, ./cminus test.1.txt와 cminus test.2.txt로 실행)



```

C-MINUS COMPILATION: test.1.txt
Syntax tree:
Function declaration, name = gcd, 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, 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

C-MINUS COMPILATION: test.2.txt
Syntax tree:
Function declaration, name = main, type : void
Single Parameter, name : (null), type : void
Compound statement :
Var declaration, name : i, type : int
Assign : (destination) (source)
Id: i
Const: 0
Iteration :
Op: <
Id: i
Const: 5
Compound statement :
Assign : (destination) (source)
Id: 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
Iteration :
Op: <=
Id: i
Const: 4
Compound statement :
If (condition) (body)
Id: x
Id: i
Compound statement :
Call, name : output, with arguments below
Id: x
Id: i
  
```

- 1) Project1의 예제였던 test.1.txt와 test.2.txt를 실행해본 결과이다. 두 예제 모두 잘 실행됨을 볼 수 있다.
- 2) 출력의 나머지 부분들은 pdf에서의 예시와 동일하다. Pdf에 나와있지 않은 1번과 2번과 같은 array 변수에 대한 출력은 각 배열의 크기가 변수의 이름의 밑에 출력되도록 하였다(크기를 child로 가지고 있기 때문) 1번은 int x[5]; 이기 때문에 변수 선언 부분에 const로 배열의 크기가 출력된 것을 볼 수 있으며, 2번은 x[i] 이기 때문에 변수 x 밑으로 배열의 크기이자, id인 i가 출력됨을 볼 수 있다.