

Compiler Construction

Project #2: Parser

제출일: 2021.11.28

컴퓨터소프트웨어학부

2019000982 이지원

Assignment

C-Minus Parser를 Yacc(bison)을 이용하여 구현한다.

Environment

기본 환경: Mac OS

Yacc(bison): Homebrew 3.2.17을 이용해 설치한 bison(GNU Bison) 2.3

Implementation

1. term

과제 명세서에 표기된 용어들을 과제에서 다음과 같은 용어로 표기하였다.

	과제 명세서의 용어	과제의 용어
1	program	program
2	declaration-list	decl_list
3	declaration	decl
4	var-declaration	var_decl
5	fun-declaration	func_decl
6	type-specifier	type_spec
7	params	params
8	compound-stmt	comp_stmt
9	param-list	paramlist
10	param	param
11	local-declarations	local_decl
12	statement-list	stmt_list
13	statement	stmt
14	expression-stmt	exp_stmt
15	selection-stmt	selec_stmt
16	iteration-stmt	iter_stmt
17	return-stmt	return_stmt
18	expression	expression
19	var	var
20	simple-expression	simple_exp
21	relop	relop
22	additive-expression	addi_exp
23	addop	addop
24	term	term
25	mulop	mulop
26	factor	factor
27	call	call
28	args	args
29	arg-list	arg_list

2. BNF & AST

과제 명세서의 p.3 BNF Grammar for C-Minus와 p.5-7 AST and Output Format을 기반으로 구현

하였다. id와 num을 이름(attr.name)과 값(attr.val)을 저장하기 위해 새로 구현하였다. id에서는 임의로 정의해둔 IdK라는 ExpNode를 만들고 그 안에 attr.name과 lineno를 strcpy(tokenString), lineno를 이용해 저장하였다. num도 마찬가지로 NumK라는 ExpNode를 만들고 attr.val를 atoi(tokenString)을 이용해 저장해 주었다. 다른 node에서 id, num이 나오면 이 노드들의 attr.name, attr.val, lineno를 이용해서 값을 저장하게 된다.

3. Print

void printType(ExpType type) : type을 출력하기 위해 만든 함수이다. switch-case문을 이용하여 각각의 type마다 출력문을 만들어주었다. printTree() 함수에서 type을 출력할 때 이 함수를 부르면 type 부분을 출력해 준다.

Void printTree(TreeNode * tree) : stmtK, ExpK 일 때 각각에서 switch-case문을 이용하여 과제 명세서의 형태에 맞게 fprintf를 이용하여 출력문을 적어주었다.

Operation

cminus.y에 정의한 grammar를 바탕으로 Yacc(bison)을 이용하도록 make하면 y.output에 grammar, state 등의 정보가 출력되고 cminus_parser라는 실행파일이 만들어진다. cminus_parser를 이용해 test file들을 실행시키면 만들어진 grammar에 따라 state들을 이동하며 Abstract Syntax Tree가 만들어진다. 만들어진 Abstract Syntax Tree를 출력한다.

Modified Code

1. globals.h

StmtKind와 ExpKind에 새로운 노드들을 추가로 정의해 주었다.

StmtKind : IfK, IfElseK, WhileK, ReturnK, VarDeclK, FuncDeclK, ParamK, VoidParamK, CompK

ExpKind : OpK, ConstK, AssignK, VarK, CallK, IdK, NumK

ExpType에도 array type들을 추가해 주었다.

ExpType : Void, Integer, VoidArr, IntegerArr

2. util.c

void printType(ExpType type) 함수 추가 : type을 출력하기 위해 switch-case문을 이용하여 각각의 type 별로 void, int, void[], int[]를 출력하도록 했다. 이외의 type이 들어온다면 “Unknown Exp type”이라는 문구가 나오도록 했다.

void printTree(TreeNode * tree) 함수 수정 : 기존에 있던 노드들에 대한 출력 이외의 과제를 구현하며 필요한 노드들에 대한 출력을 추가했다. type을 출력할 때는 printType() 함수를 tree->type을 parameter로 하여 호출하였다.

3. cminus.y

IdK와 NumK를 이용해 savedName, savedLineNo를 쓰지 않았기 때문에 주석처리 하였다.

Definition : c-minus prae를 위해 필요한 token들을 추가했다. shift/reduce conflict를 없애기 위해 ELSE와 RPAREN을 %nonassoc으로 지정해주었다.

Rules : 구현에서 사용된 grammar 정의들을 추가해 주었다. 기존에 있던 몇개를 제외하고 새로 작성하였다.

Subroutines : 수정하지 않았다.

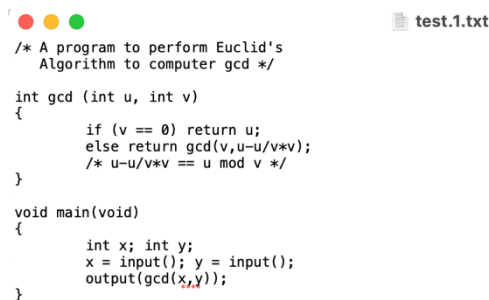
Execution Method

1. 2_Scanner 폴더 안의 파일들을 같은 위치에 저장한다.
2. cd 명령어를 이용해 저장한 위치로 이동한다.
3. 터미널에서 make 명령어를 통해 make clean 후 make를 진행한다.
4. cminus_parser 실행 파일을 원하는 test file과 함께 실행시킨다.
 - 출력 결과를 새로운 파일에 작성하여 확인하고 싶으면 > [file name]을 붙인다.
 - Ex. ./cminus_parser test.1.txt
 - Ex. ./cminus_parser test.1.txt > result1.txt

Result

test.1.txt와 이를 실행한 결과의 사진을 각각 첨부하였다.

- test.1.txt



The image shows a code editor window titled 'test.1.txt'. The code is a C program for calculating the Greatest Common Divisor (GCD) using Euclid's algorithm. It includes a main function that takes two integers as input and outputs their GCD. The code is as follows:

```
/* 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));
}
```

- ./cminus_parser test.1.txt

```
leejiwon@MacBook-Pro 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 = input
  Call: function name = output
  Call: function name = gcd
  Variable: name = x
  Variable: name = y
leejiwon@MacBook-Pro 2_Parser %
```