Assignment2

Parser

제출일	2021.11.15
과목	컴파일러
담당교수	박영준 교수님
전공	컴퓨터소프트웨어과
학번	2015005169
이름	최윤석

1. Intro

■ 목표 : C-Minus Parser의 구현

> Yacc(BISON)을 이용하여 AST를 생성해낸다.

■ BNF Grammar for C-Minus

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*

2. 컴파일 진행 환경, 실행방법

Uvuntu 20.04.3 LTS에서 컴파일을 진행하였으며,
Makefile은 주어진 아래 Makefile을 수정 없이 사용했습니다.

```
# Makefile for C-Minus
#
# ./lex/tiny.l        --> ./cminus.l (from Project 1)
# ./yacc/tiny.y       --> ./cminus.y
# ./yacc/globals.h    --> ./globals.h

CC = gcc

CFLAGS = -W -Wall

OBJS = main.o util.o lex.yy.o y.tab.o

.PHONY: all clean
all: cminus_parser

clean:
  rm -vf cminus_parser *.o lex.yy.c y.tab.c y.tab.h y.output

cminus_parser: $(OBJS)
  $(CC) $(CFLAGS) $(OBJS) -o $@ -lfl

main.o: main.c globals.h util.h scan.h parse.h y.tab.h
  $(CC) $(CFLAGS) -c main.c

util.o: util.c util.h globals.h y.tab.h
  $(CC) $(CFLAGS) -c util.c

scan.o: scan.c scan.h util.h globals.h y.tab.h
  $(CC) $(CFLAGS) -c scan.c
```

```
lex.yy.o: lex.yy.c scan.h util.h globals.h y.tab.h
  $(CC) $(CFLAGS) -c lex.yy.c

lex.yy.c: cminus.l
  flex cminus.l

y.tab.h: y.tab.c

y.tab.o: y.tab.c parse.h
  $(CC) $(CFLAGS) -c y.tab.c

y.tab.c: cminus.y
  yacc -d -v cminus.y
```

실행방법은 위 Makefile을 이용하여 컴파일을 하는 것이며
cminus_parser라는 파일이 생성되면 해당 실행파일을 실행시키면 됩니다.
예를 들어 test1.txt라는 파일을 이용해 AST를 만들고 싶다면
`./cminus_parser ./test1.txt`와 같이 입력하면 됩니다.

3. 구현

- pdf를 참조하여 코드 작성을 시작하였습니다.

<`main.c`>
Syntax Tree만을 출력하기 위해 `NO_PARSE`는 `FALSE;`로
Tracing flags는 아래와 같이 수정하였습니다.

`int echoSource = FALSE;`

`int TraceScan = FALSE;`

`int TraceParse = TRUE;`

`int TraceAnalyze = FALSE;`

`int TraceCode = FALSE;`

`int Error = FALSE;`

<`globals.h`>
yacc/globals.h 파일을 복사하여 수정하였습니다.
Parsing을 위한 syntax tree는 아래와 같이 설정했습니다.
PDF 5~ 7까지의 AST model을 기반으로 설정하였으며 기존에 존재하던 것에 추가로 `DeclarKind`를 추가했습니다.

```
/**************************************************/
/***********   Syntax tree for parsing ************/
/**************************************************/

typedef enum {DeclarK,StmtK,ExpK} NodeKind;
typedef enum {VarK, FunK, ParamK, ArrVarK, ParamVoidK} DeclarKind;
typedef enum {CompK, IfK, IfElseK, WhileK, ReturnK,ReturnOnlyK} StmtKind;
typedef enum {AssignK, OpK, ConstK, CallK, IDK, ArrIDK}ExpKind;

/* ExpType is used for type checking */
typedef enum {Void,Integer, VoidArr, IntegerArr, Boolean} ExpType;
```

AST의 기본 설정을 위와 같이 변경했기 때문에 TreeNode 역시 수정해주었습니다.
DeclarK Nodekind가 추가되었기 때문에 TreeNode kind에 DeclarKind declar를 추가해주었으며
cminus.y파일에서 parsing을 할 때 사용할 char *id를 추가해주었습니다.

```
typedef struct treeNode
   { struct treeNode * child[MAXCHILDREN];
     struct treeNode * sibling;
     int lineno;
     NodeKind nodekind;
     union { StmtKind stmt; ExpKind exp; DeclarKind declar;} kind;
```

```
      union { TokenType op;
              int val;
              char * name;
              char * id; } attr;
      ExpType type; /* for type checking of exps */
    } TreeNode;
```

<`util.c`>
global.h에서 DeclarK Nodekind를 추가하였기 때문에 `newStmtNode`함수를 복사하여 `TreeNode * newDeclarNode(DeclarKind kind)`를 만들어주었습니다.
Variable Declaration과 Function declaration 등의 output format에 type을 작성해야 했기 때문에 아래와 같이 getType 함수를 만들어 타입을 얻어올 수 있도록 만들었습니다.

```
void getType(int typeNum, char * type){
  if(typeNum == 0)
    strcpy(type,"void");
  else if(typeNum==1)
    strcpy(type, "int");
  else if(typeNum==2)
    strcpy(type, "void[]");
  else if(typeNum==3)
    strcpy(type, "int[]");
}
```

0, 1, 2, 3은 `globals.h`의
`typedef enum {Void,Integer, VoidArr, IntegerArr, Boolean} ExpType;`
에 대응됩니다.
Void와 VoidArr, Integer와 IntegerArr을 2칸씩 떨어뜨려 ciminus.y파일에서 파싱을 진행할 때 type을 구분할 수 있도록 만들었습니다.
`void printTree( TreeNode * tree )` 함수는 길기 때문에 요약하여 기술하겠습니다.
- char type[16]; 이라는 local variable을 선언하여 getType함수로 타입을 받아올 수 있게 하였습니다.
- PDF 5~7쪽에 기술된 Output Format, 제공된 input, result file, globals.h의 Syntax Tree node종류를 참고하여 각 노드에 맞는 출력이 발생할 수 있도록 코드를 작성했습니다.
num value의 경우 `tree->attr.val,` name의 경우 `tree->attr.id`에 접근하여 값을 얻어옵니다.

<`ciminus.y`>
주어진 명세서의 BNF Grammar와 동일하게 구현하였습니다.
문법에 없는 사항으로

```
static char * savedID;
static int savedNum;
static int savedType;
```

를 선언하여 identifier, num, type에 해당하는 값을 저장할 때 사용하였습니다.
이는 TokenString이 ID, num token의 정확한 값을 포함하지 못하기 때문에 별도로 저장해 두었다가 노드에 저장하기 위함입니다.

lineno를 별도로 활용하는 부분은 없지만 Some comments부분에 project 3을 위해 line no 도 유지해두라고 나와있어 새로운 노드를 만들 때 마다 lineno값도 저장했습니다.

처음 컴파일을 하였을 때 ciminus.y부분에서 shift/redce conflict가 발생했습니다.
따라서 %nonassoc과 %prec을 활용하여 해결하였습니다.
ciminus.y를 보면 if statement 부분을 다음과 같이 구현했습니다.

```
%nonassoc NO_ELSE
%nonassoc ELSE
...
```

```
%%
...
select_stmt : IF LPAREN exp RPAREN stmt %prec NO_ELSE
                { $$ = newStmtNode(IfK);
                  $$->child[0] = $3;
                  $$->child[1] = $5;
                  $$->lineno = lineno;
                }
            | IF LPAREN exp RPAREN stmt ELSE stmt
                { $$ = newStmtNode(IfElseK);
                  $$->child[0] = $3;
                  $$->child[1] = $5;
                  $$->child[2] = $7;
                  $$->lineno = lineno;
                }
            ;
...
```

Else가 있는 If statement와 Else가 없는 Statement의 우선 순위를 구분하기 위해 ELSE보다 낮은 우선순위의 NO_ELSE를 %nonassoc으로 정의하고 ELSE의 우선순위를 높이기 위해 뒤이어 %nonassoc을 정의하였습니다.

4. 실행결과

- Dangling Else Problem 코드 화면

  yacc -d cminus.y를 통해 conflict가 없음을 확인하였으며, PDF에 나왔던 코드역시 문제 없이 AST가 형성됨을 확인할 수 있었습니다.

- test.1.txt 실행결과화면

```
          Const: 3
        Assign:
          Variable: name = a
          Const: 3
        Assign:
          Variable: name = a
          Const: 4
choi@choi ~/2021Compiler/tmp_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
choi@choi ~/2021Compiler/tmp_parser
```

- test.2.txt 실행결과 화면

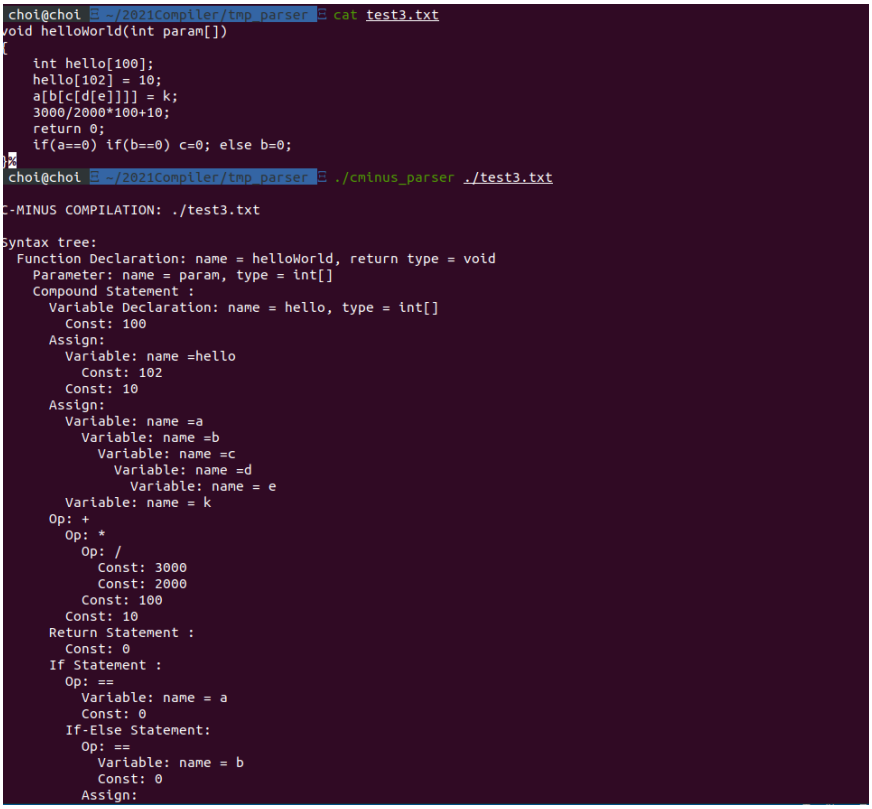
```
          Variable: name = x
          Variable: name = y
choi@choi ~/2021Compiler/tmp_parser ./cminus_parser ./test.2.txt

C-MINUS COMPILATION: ./test.2.txt

Syntax tree:
  Function Declaration: name = main, return type = void
    Void Parameter
    Compound Statement :
      Variable Declaration: name = i, type = int
      Variable Declaration: name = x, type = int[]
        Const: 5
      Assign:
        Variable: name = i
        Const: 0
      While Statement :
        Op: <
          Variable: name = i
          Const: 5
        Compound Statement :
          Assign:
            Variable: name =x
              Variable: name = i
            Call: function name = input
          Assign:
            Variable: name = i
            Op: +
              Variable: name = i
              Const: 1
      Assign:
        Variable: name = i
        Const: 0
      While Statement :
        Op: <=
          Variable: name = i
          Const: 4
        Compound Statement :
          If Statement :
            Op: !=
              Variable: name =x
                Variable: name = i
              Const: 0
            Compound Statement :
              Call: function name = output
                Variable: name =x
                  Variable: name = i
choi@choi ~/2021Compiler/tmp_parser
```

test.1.txt와 test.2.txt모두 주어진 result와 동일하게 출력됨을 확인할 수 있었습니다.

- test3.txt 실행결과 화면



연산이 여러개 연속으로 나올때는 문제가 없는지 혹은 배열을 연속적으로 사용할때는 에러가 발생하지는 않는지 Semantic이 맞지 않아도 AST가 만들어지는지를 추가로 확인하기 위하여 test3.txt를 만들어 확인해보았습니다. 문제없이 동작함을 확인할 수 있었습니다.

피드백

컴파일 후 cminus_parser를 사용하여 !=를 포함한 코드를 파싱했을 때 자꾸 에러가 났었는데, 결과적으로 `cminus.l` 파일에 `"!=" {return NE;}`가 빠져있었기 때문이었습니다.
전 프로젝트에도 빠져있음을 확인하였고, 이번 경험을 통해 테스트케이스 선정의 중요함(당시 테스트는 모두 정상적으로 출력이 되었기 때문에)을 알게 되었습니다.
또한 Dangling Else Problem을 해결할 때 처음에 `%nonassoc ELSE`을 선언해주지 않아 Else문이 뒤쪽 else문과 문제없이 엮임에도 불구하고 yacc -d로 확인했을 때 계속하여 shift/reduce confilct가 발생하였는데, 이 부분은 잘 이해가 되지 않아 해결에 시간이 오래소모되었습니다.
마지막으로 cminus.y파일을 작성할 때

```
fun_declar  : type_spec identifier
                { $$ = newDeclarNode(FunK);
                  $$->attr.id = savedID;
                  $$->type = savedType;
                  $$->lineno = lineno;
                }
                LPAREN params RPAREN comp_stmt
                {
                  $$ = $3;
                  $$->child[0] = $5;
                  $$->child[1] = $7;
                }
            ;
```

이 부분을 처음에는

```
fun_declar  : type_spec identifier LPAREN params RPAREN comp_stmt
                  { $$ = newDeclarNode(FunK);
                    $$->attr.id = savedID;
                    $$->type = savedType;
                    $$->lineno = lineno;
                    $$->child[0] = $4;
                    $$->child[1] = $6;
                  }
            ;
```

과 같이 작성했었는데 아래 방법으로 작성했을 때 function의 이름이 identifier에 저장된 이름이 아닌 params에 저장된 값이 저장되는 문제가 발생했습니다.
구글링을 통해서 위와 같이 Yacc를 작성할 수 있고 중간에 위치한 {}부분의 $$를 $3으로 지정하여 작성할 수 있음을 알게되었고 수정할 수 있었습니다.