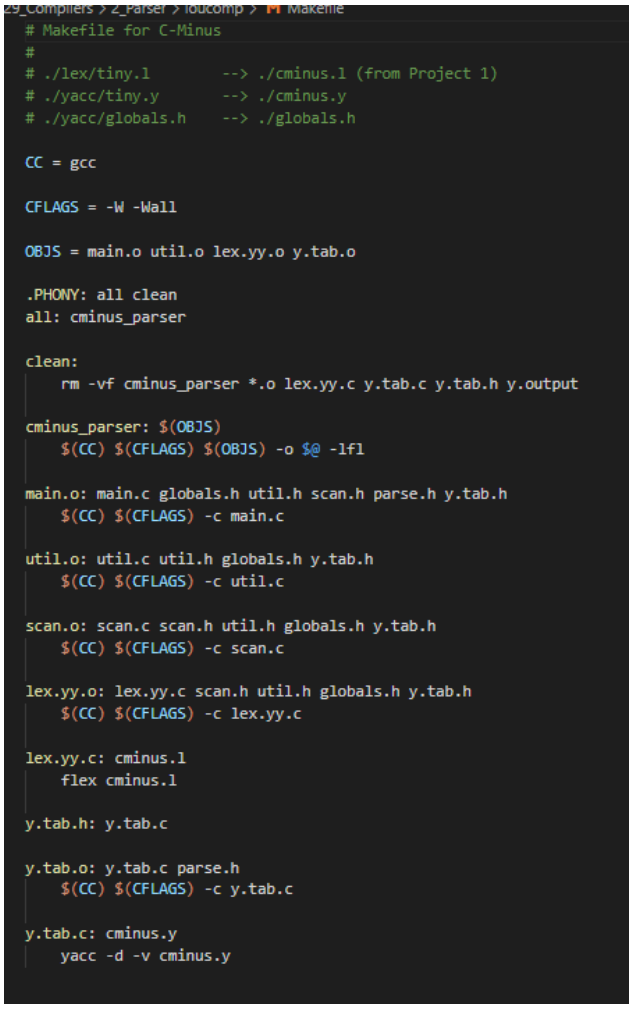
컴파일러 설계 프로젝트 2 Parser

2019062833 김유진

1. 컴파일 방법과 개발 환경

1) 컴파일 – make 명령어로 cminus_parser 프로그램 생성 후 ./cminus_plus [file_name]으로 파싱



```
# 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
```

2) Ubuntu 20.04

2. 구현과 작동 방법

1)main.c

-syntax 트리만 출력하기 위해서 NO PARSE를 False로, NO ANALYZE를 True로 변경해주고, traceParse flag만 True로 설정해준다.

2)globals.h

```
typedef enum {StmtK,ExpK,DeclareK} NodeKind;
typedef enum {CompK,SelectK,IterK,RetK} StmtKind;
typedef enum {AssignK,VarK,BinK,ConstK,IdK,CallK,TypeK} ExpKind;
typedef enum {VarDK,FuncDK,ParamDK} DeclareKind;
typedef enum {ArrayK,NArrayK,IfElseK,IfK,NonValRetK,ValRetK} TypeKind;
```

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

#define MAXCHILDREN 3

typedef struct treeNode
   { struct treeNode * child[MAXCHILDREN];
     struct treeNode * sibling;
     int lineno;
     NodeKind nodekind;
     union { StmtKind stmt; ExpKind exp; DeclareKind declare;} kind;
     union { TokenType op;
             int val;
             char * name; } attr;
     TypeKind typeK;
     ExpType type; /* for type checking of exps */
   } TreeNode;
```

- Parse Tree의 노드를 정의한다. 노드는 크게 statement, expression, declare 타입으로 나뉜다. stmt 타입은 복합문,if-else문, 반복문, 반환문으로 나뉘며, exp 타입은 할당, 변수, 바이너리(연산자), 상수, Id, 함수 Call, 매개변수의 타입을 저장하는 타입을 나뉜다. 마지막으로 declare 타입은 변수 선언문, 함수 선언문, 매개변수 선언문 타입으로 나뉜다.

- 노드는 자식노드를 가리키는 포인터와 형제를 가리키는 포인터를 가지고 있으며, 파싱을 위해 추가적인 정보를 들고있다.

- 줄의 넘버를 저장하는 lineno와, 노드의 문장 종류, 문장 내부에서의 타입, 값, 이름, 연산자 등을 저장하여서 파싱할 때 사용하게 된다.

- TypeKind에 들어가는 정보를 통해 배열인지 아닌지, if-else문인지 if 문인지, 반환값이 있는지 없는 지를 체크할 수 있게 된다.

3)util.c

```
TreeNode * newStmtNode(StmtKind kind)
{ TreeNode * t = (TreeNode *) malloc(sizeof(TreeNode));
  int i;
  if (t==NULL)
    fprintf(listing,"Out of memory error at line %d\n",lineno);
  else {
    for (i=0;i<MAXCHILDREN;i++) t->child[i] = NULL;
    t->sibling = NULL;
    t->nodekind = StmtK;
    t->kind.stmt = kind;
    t->lineno = lineno;
  }
  return t;
}

/* Function newExpNode creates a new expression
 * node for syntax tree construction
 */
TreeNode * newExpNode(ExpKind kind)
{ TreeNode * t = (TreeNode *) malloc(sizeof(TreeNode));
  int i;
  if (t==NULL)
    fprintf(listing,"Out of memory error at line %d\n",lineno);
  else {
    for (i=0;i<MAXCHILDREN;i++) t->child[i] = NULL;
    t->sibling = NULL;
    t->nodekind = ExpK;
    t->kind.exp = kind;
    t->lineno = lineno;
```

- 우선 newStmtNode와 newExpNode 함수는 기존의 함수를 그대로 사용하였으며 추가적으로 내가 정의내린 DeclareNode에 대한 함수만 새로 생성하였다

.

```
TreeNode * newDeclareNode(DeclareKind kind)
{ TreeNode * t = (TreeNode *) malloc(sizeof(TreeNode));
  int i;
  if (t==NULL)
    fprintf(listing,"Out of memory error at line %d\n",lineno);
  else {
    for (i=0;i<MAXCHILDREN;i++) t->child[i] = NULL;
    t->sibling = NULL;
    t->nodekind = DeclareK;
    t->kind.declare = kind;
    t->lineno = lineno;
  }
  return t;
}
```

- newDeclareNode에서는 받아온 kind로 선언 노드의 종류를 저장하고 해당 노드가 선언 타입임을 알려주는 DeclareK를 저장한다.

```
void printTree( TreeNode * tree )
{ int i;
  INDENT;
  while (tree != NULL) {
    printSpaces();
    if (tree->nodekind==StmtK)
    { switch (tree->kind.stmt) {
        case CompK:
          fprintf(listing,"Compound Statement:\n");
          break;
        case SelectK:
          if(tree->typeK == IfElseK)
            fprintf(listing,"If-Else Statement:\n");
          else
            fprintf(listing,"If Statement:\n");

          break;
        case IterK:
          fprintf(listing,"While Statement:\n");
          break;
        case RetK:
          if(tree->typeK == ValRetK)
            fprintf(listing,"Return Statement:\n");
          else
            fprintf(listing,"Non-Value Return Statement\n");

          break;
        default:
          fprintf(listing,"Unknown ExpNode kind\n");
          break;
      }
    }
```

-받아온 트리 노드가 statement 타입일 때, 해당 stmt가 어떤 종류인지에 따라 알맞은 출력문을 프린트한다.

-이때, If-Else는 Else문의 유무에 따라 구별 가능하게 출력한다.

-Return 문 또한 반환값의 유무에 따라 구분해서 출력해준다.

```
else if (tree->nodekind==ExpK)
{ switch (tree->kind.exp) {
    case BinK:
      fprintf(listing,"Op: ");
      printToken(tree->attr.op,"\0\n");
      break;
    case ConstK:
      fprintf(listing,"Const: %d\n",tree->attr.val);
      break;
    case IdK:
      fprintf(listing,"Id: %s\n",tree->attr.name);
      break;
    case AssignK:
      fprintf(listing,"Assign: \n");
      break;
    case VarK:
      fprintf(listing,"Variable: name = %s\n",tree->attr.name);
      break;
    case CallK:
      fprintf(listing,"Call: function name = %s\n",tree->attr.name);
      break;
    case TypeK:
      fprintf(listing,"Void Parameter\n");
      break;

    default:
      fprintf(listing,"Unknown ExpNode kind\n");
      break;
  }
}
```

-Expression 타입의 노드의 경우, BinK, ConstK, IdK, AssignK, VarK, CallK, TypeK의 타입으로 나뉘게 되는데 각각의 타입에 맞게 출력을 해준다.

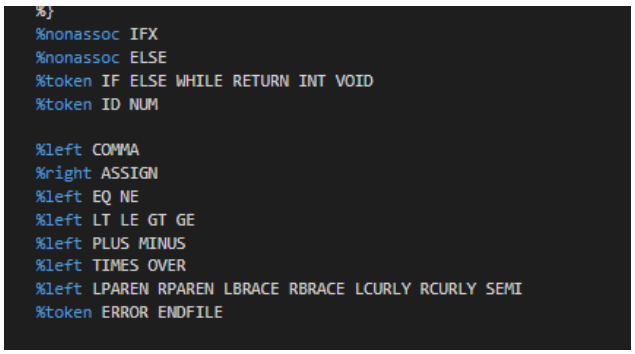
```
else if (tree->nodekind==DeclareK)
{ switch (tree->kind.declare) {
    case VarDK:
      fprintf(listing,"Variable Declaration: name = %s, type = %s\n",
      tree->attr.name, tree->type?
      (tree->typeK==ArrayK?"int[]":"int"):
      (tree->typeK==ArrayK?"void[]":"void"));
      break;
    case FuncDK:

      fprintf(listing,"Function Declaration: name = %s, return type = %s\n",
      tree->attr.name,tree->type?"int":"void");
      break;
    case ParamDK:
      fprintf(listing,"Parameter: name = %s, type = %s\n",
      tree->attr.name,tree->type?
      (tree->typeK==ArrayK?"int[]":"int"):
      (tree->typeK==ArrayK?"void[]":"void"));
      break;
    default:
      fprintf(listing,"Unknown ExpNode kind\n");
      break;
  }
}
else fprintf(listing,"Unknown node kind\n");
for (i=0;i<MAXCHILDREN;i++)
     printTree(tree->child[i]);
tree = tree->sibling;
}
```

-Declare 문인 경우 VarDk, FuncDk, ParamDk로 세가지 타입이 존재하며 각 타입에 맞게 출력을 해준다.

- 선언문의 타입의 경우 받아온 노드 내부의 정보를 통해서 해당 타입이 배열인지 아닌지를 구분한다.

4) cminus.y



```
%}
%nonassoc IFX
%nonassoc ELSE
%token IF ELSE WHILE RETURN INT VOID
%token ID NUM

%left COMMA
%right ASSIGN
%left EQ NE
%left LT LE GT GE
%left PLUS MINUS
%left TIMES OVER
%left LPAREN RPAREN LBRACE RBRACE LCURLY RCURLY SEMI
%token ERROR ENDFILE
```

- 앞의 두 문장은 if-else 문에서의 reduce/shift 충돌을 해결하기 위해 추가해 주었다.

- 나머지 토큰들은 C언어에서의 연산자 우선순위를 기준으로 Associativity를 결정하였다.

-program: declare_list를 루트로 가지게 되며 이를 savedTree로 설정해서 루트로 인식하게 해준다

-declare_list: declare가 더 이상 존재하지 않을 때까지 반복문을 돌면서 형제 노드에 추가해준다

-declare: var 선언문 또는 fun 선언문으로 나뉜다.

- var_declare: VarDk 타입인 노드를 생성해주고 해당 노드에 타입과 id에서 저장해둔 savedName, savedLineNo를 저장해준다. 배열인지 아닌지에 따라 $$->typeK에 정보를 다르게 적어준다.

-id: 토큰 스트링을 읽어와서 savedName에 저장해두고 줄 넘버또한 저장해준다

-num: ConstK 타입의 노드를 생성한 후 현재 라인 넘버와 토큰 스트링을 저장해준다.

-type_spec: TypeK 타입의 노드를 생성한 후 각 타입과 라인 넘버를 저장한다

-func_declare: Funck 타입의 노드를 생성한 후 id에서 저장해둔 이름과 라인 넘버를 추가하고 타입도 기록한다. 매개변수 노드를 자신의 0번째 자식노드로 두고 복합문 노드를 자신의 1번째 노드로 둔다. id 이후 inner 방식을 사용한 이유는 params, compound_stmt 를 파싱하는 동안 savedName이 덮어씌워질 수 있기 때문이다

-param_list: param이 더 이상 나오지 않을 때까지 반복문을 돌면서 형제 노드에 추가해준다.

param: ParamDK 타입의 선언문 노드를 새로 생성하고 id에서 저장해둔 이름과 라인 넘버, 타입을 저장해준다. 매개변수가 배열인 경우 typeK에 ArrayK, 배열이 아닌 경우에는 NArrayK를 넣어준다

-compound_stmt: Compk타입의 노드를 생성한 후 지역 선언문을 자신의 0번째 자식 노드로 갖고 stmt list를 자신의 1번째 자식 노드를 가진다.

-local_declare과 stmt list 모두 더 이상 추가 문장이 나오지 않을 때까지 반복문을 돌면서 형제 노드에 추가해 준다.

-stmt는 5 종류로 나뉘게 되는데 각각의 종류에 맞는 타입의 stmtNode를 생성해서 자식 노드를 연결 시켜준다.

-exp: var=exp 형식 또는 single_exp 형식으로 나뉘게 되며, 첫번째의 경우 AssignK 타입의 노드를 생성해서 var와 exp를 자식 노드로 연결해준다.

-relop: greater than, greater equal, less than, less equal, equal, not equal 에 대해 노드를 생성하게 되며 이때 노드의 타입은 Bink이다. attr->op에 연산자를 저장하게 된다.

-addop, mulop도 위와 유사하게 구현

3. 예시 및 결과 화면

1) 예시 1

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

```
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
```

2) 예시2

```
void main(void)
{
    int i; int x[5];

    i = 0;
    while( i < 5 )
    {
        x[i] = input();

        i = i + 1;
    }

    i = 0;
    while( i <= 4 )
    {
        if( x[i] != 0 )
        {
            output(x[i]);
        }
    }
}
```

```
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
```

3) 예시 3(pdf) – if-else Rule

```
void main(void)
{
    if(a < 0) if (a > 3) a= 3; else a =4;
}
```

```
C-MINUS COMPILATION: test.3.txt

Syntax tree:
  Function Declaration: name = main, return type = void
    Void Parameter
    Compound Statement:
      If Statement:
        Op: <
          Variable: name = a
          Const: 0
        If-Else Statement:
          Op: >
            Variable: name = a
            Const: 3
          Assign:
            Variable: name = a
            Const: 3
          Assign:
            Variable: name = a
            Const: 4
```