[컴파일러설계]

Parser

컴퓨터전공

2013011822

김병조

Modification code (globals.h, main.c, util.c, cminus.y)

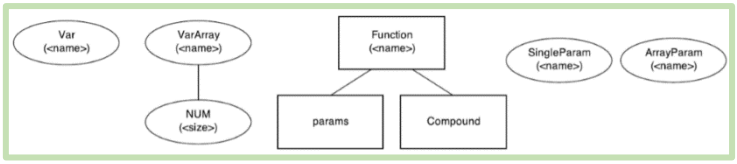
globals.h -> (./yacc/globals.h 를 복사)

```
typedef enum {StmtK, ExpK, DecK} NodeKind;
typedef enum {VarK, FunK, ParamK} DecKind;

typedef enum {CompK, IfK, WhileK, RetK} StmtKind;
typedef enum {OpK, ConstK, IdK, ArrIdK, CallK} ExpKind;
/* ExpType is used for type checking */
typedef enum {Void, Integer, Array} Type;
```

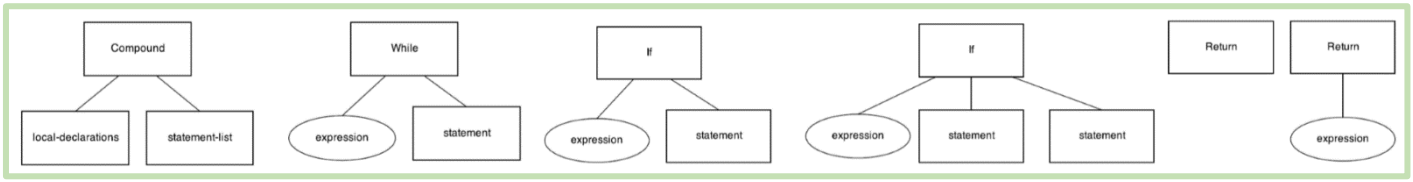
노드의 종류로 StmtK, ExpK, DecK를 설정 해놓는다. 각 종류에는 또 여러가지 종류가 있다.

DecKind 에는 Declaration 되어지는 것들이 담겨진다.



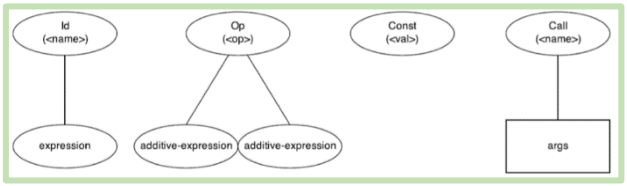
그림과 같이 Var, Function, Parameter 종류가 있다. 이때 Var과 Parameter은 배열의 형식으로도 파싱이 가능해야 하기 때문에 Type에 Array를 하나 추가 해 놓아서 언제든지 체크가 가능 하도록 하였다.

StmtKind 에는 statement로 이용할 수 있는 것들이 담겨진다.



그림과 같이 Compound, While문, If문 (Else) Return 의 종류를 추가하였다. 이때 Else를 따로 추가하지 않은 이유는 If를 처리할 때 Child-node를 하나 더 추가해서 처리하기 때문이다.

ExpKind 에는 나머지 것들이 담겨진다.



그림과 같이 Id, Operation, Constant, Call 그리고 id가 배열일때를 생각하여 ArrayId로 구성되어 있다.

이렇게 노드의 종류들을 구성해 놓고, 노드가 가질 수 있는 변수들을 설정 하였다.

```
typedef struct treeNode
   { struct treeNode * child[MAXCHILDREN];
     struct treeNode * sibling;
     int lineno;

     NodeKind nodekind;
     union {
        StmtKind stmt;
        ExpKind exp;
        DecKind dec;
    } kind;

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

TreeNode에는 최대 3개의 자식 노드(child)와 형제 노드(sibling)를 가질 수 있다. 그리고 모든 노드에는 노드종류(nodekind)라는 성질을 가지고 있어야한다. 위에서 언급한 StmtK, ExpK, DecK가 그것들 이다. 그리고 또한 세부

하게 어떤 노드인지에 대한 성질을 가지고 있어야한다. 마찬가지로 위에서 언급한 DecKind, StmtKind, ExpKind가 그것들 이다. 변수명 혹은 값 혹은 operation을 저장하기 위해 attr union을 만들었고, 배열인 경우 size를 저장할 수 있도록 size라는 성질도 만들었다. 그리고 마지막으로 int형인지 void형인지 또는 배열 형태인지를 알 수 있게 type 성질을 만들었다. 이제 모든 준비는 끝났다.

util.c

```
TreeNode * newDecNode(DecKind 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 = DecK;
    t->kind.dec = kind;
    t->lineno = lineno;
  }
  return t;
}
```

새로 추가한 DecK 종류의 노드를 설정 할 수 있는 newDecNode 함수이다. malloc으로 동적할당이나 다른부분은 다른 종류와 동일하지만 nodekind부분과 kind의 dec 부분에 저런식으로 추가하여 이것이 DecK 종류의 노드임을 알려야한다.

```
void printTree( TreeNode * tree )
{ int i;
  INDENT;
  while (tree != NULL) {
    printSpaces();
    if (tree->nodekind==StmtK)
    { switch (tree->kind.stmt) {...
      }
    }
    else if (tree->nodekind==ExpK)
    { switch (tree->kind.exp) {...
      }
    }
    else if (tree->nodekind==DecK)
    { switch (tree->kind.dec) {...
      }
    }
    else fprintf(listing,"Unknown node kind\n");
    for (i=0;i<MAXCHILDREN;i++)
         printTree(tree->child[i]);
    tree = tree->sibling;
  }
  UNINDENT;
}
```

Tree를 그려주는 prtinTree 함수에는 각 노드의 종류별로 분기가 되어지고 그안에서 그것들의 종류별로 또 분기가 이루어진다. 소스가 매우 긴관계로 깃에 올려놓은 소스를 확인하면 된다.

```
else if (tree->nodekind==DecK)
{ switch (tree->kind.dec) {
    case VarK:
      if( tree-> type == Array)
        fprintf(listing,"Var declaration, name: %s, type: %s, size: %d\n",tree->attr.name, typeStrings[tree->type], tree->size );
      else
        fprintf(listing,"Var declaration, name: %s, type: %s\n",tree->attr.name, typeStrings[tree->type]);
      break;
    case FunK:
      fprintf(listing,"Function declaration, name: %s, return type: %s\n",tree->attr.name, typeStrings[tree->type]);
      break;
    case ParamK:
```

Nodekind가 Deck 일경우를 예를 들어보겠다. tree->kind.dec에 세부 종류를 집어넣었기 때문에 그것들로 분기가 이루어진다. 이때 Var인 경우 type이 Array인지 아닌지를 판단하여 다르게 트리를 출력한다. 소스가 매우 긴관계로 다른 부분은 깃에 올려놓은 소스를 확인하면 된다.

```
char *typeStrings[] = {"void", "int", "int[]"};
```

트리를 출력 할 때 enum으로 되어있는 타입을 스트링으로 바꿔주기 위해서 배열을 만들어 놓는다. cminus 문법에는 배열이 int array (int[]) 밖에 없으므로 array부분을 int[]로 바꾸어 놓았다.

cminus.y

```
%{
#define YYPARSER /* distinguishes Yacc output from other code files */

#include "globals.h"
#include "util.h"
#include "scan.h"

#define YYSTYPE TreeNode *
static char * savedName; /* for use in assignments */
static int savedNum;
static int savedLineNo;  /* ditto */
static TreeNode * savedTree; /* stores syntax tree for later return */
static int yylex(void); // added 11/2/11 to ensure no conflict with lex

static TreeNode *l = NULL; /* local dec list */
static TreeNode *s = NULL; /* stmt list */
static TreeNode *d = NULL; /* dec list */
static TreeNode *p = NULL; /* params list */
static TreeNode *a = NULL; /* argument list */

%}
```

cminus.y의 Definitions 부분 이다. savedNum을 추가하였고, 각 상황별 리스트에 담기 위해 treenode들을 미리 선언 하였다. 이것들은 리스트에 담길 때 sibling들을 이어주는 용도로 쓰인다. 예를들어 local declaration list를 만들때 TreeNode d부터 연결하여 표현 되도록 한다.

과제 pdf (2_Parser_2017.pdf) 6page에 나와있는 BNF Grammer for C-minus를 그대로 cminus.y에 옮겼다.

type-specifier → **int** | **void**

```
type_spec   : INT
              {
                $$ = newDecNode(VarK);
                $$->type = Integer;
              }
            | VOID
              {
                $$ = newDecNode(VarK);
                $$->type = Void;
              }
            ;
```

예를 들어 위에 간단한 type-specifier가 저런식으로 표현이 된다. 이렇게 총 29개의 CFG를 코드로 바꿔주면 된다. 이때 종류와 타입에 맞춰서 노드를 추가하고 그에 따라 타입과 종류를 넣어주면 된다. 자세한 소스는 깃을 참고하면 알 수 있다.

```
var_dec     : type_spec id SEMI
              {
                $$ = $1;
                $$->attr.name = savedName;
              }
            | type_spec id
              {
                $$ = $1;
                $$->attr.name = savedName;
                $$->type = Array;
              }
              LBRACE NUM
              {
                $$ = $3;
                $$->size = atoi(tokenString);
              }
              RBRACE SEMI
              {
                $$ = $6;
              }
            ;
```

이때 ID와 NUM과 같은 경우 copyString(tokenString)으로 토큰 문자열을 넣어줘야하지만 토큰 스트링은 각문법의 가장 마지막 토큰스트링만 남기 때문에 위와 같은 방법으로 각각 끊어서 설정 해줘야 한다. 이때 yacc의 문법상 $x의 x가 한칸 밀러나게 된다. 설정하는 부분({ ~ })도 하나의 symbol로 생각되어지는 것 같다. 이와 같이 총 29가지의 문법을 처리 하였다.

Makefile

```
OBJS = y.tab.o lex.yy.o main.o util.o symtab.o analyze.o code.o cgen.o

cminus: $(OBJS)
	$(CC) $(CFLAGS) $(OBJS) -o cminus
```

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

parse.o와 scan.o를 모두 제외 시켰다. 뒤에 특히 cgen.o를 만들 때 많은 에러가 발생하는데 이는 모두 주석 처리하여 에러가 나지 않게 하였다. 이로써 lex와 yacc로 만들어진 parser가 완성 되었다.

Compilation environment

Os : Ubuntu 16.04.3 LTS

gcc : gcc (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609

Example and Result

Example : ./example에 첨부

test.cm

```
/* A program to perform Euclid's
  Algorithm to computer gcd */
int gcd(int u, int v)
{
    if (v==0) return u;

    if (v<0) return u;
    else return gcd(v,u-u/v*v);
}

void main(int a [])
{
    int x;
    int y;
    int z;
    int sample[35];
    x = input();
    y = input();

    if(x<0) x = 0;

    z = gcd(x,y);
    output(z);
}
```

result_test ->첨부

```
CMINUS COMPILATION: test.cm

Syntax tree:
  Function declaration, name: gcd, return type: int
    single Parameter, name: u, type: int
    single Parameter, name: v, type: int
    Compound statement :
      If (condition) (body)
        Op: ==
          Id: v
          Const: 0
        Return
          Id: u
      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, return type: void
    array Param, name: a, type: int[] , size: 0
    Compound statement :
      Var declaration, name: x, type: int
      Var declaration, name: y, type: int
      Var declaration, name: z, type: int
      Var declaration, name: sample, type: int[], size: 35
      Assign :(destination) (source)
        Id: x
        Call, name: input, with arguments below
      Assign :(destination) (source)
        Id: y
        Call, name: input, with arguments below
      If (condition) (body)
        Op: <
          Id: x
          Const: 0
        Assign :(destination) (source)
          Id: x
          Const: 0
      Assign :(destination) (source)
        Id: z
        Call, name: gcd, with arguments below
          Id: x
          Id: y
      Call, name: output, with arguments below
        Id: z
```

기본 예제에 배열선언, 배열 argument 등을 추가하여 만든 test.cm 이다.

test.cm에 있는 syntax들이 파싱이 되어 트리 형식으로 나타내어지는 것을 확인 할 수 있다. 이 뿐만 아니라 더 복잡한 함수 또한 꽤 완벽히 파싱 되어지는 것을 확인 할 수 있다.

test2.cm의 sort 함수

```
void sort(int a[], int low, int high)
{ int i; int k;
  i = low;
  while (i < high-1)
  { int t;
    k = minloc(a,i,high);
    t = a[k];
    a[k] = a[i];
    a[i] = t;
    i = i + 1;
  }
}
```

result_test2 의 sort 함수 부분

```
Function declaration, name: sort, return type: void
  array Param, name: a, type: int[] , size: 0
  single Parameter, name: low, type: int
  single Parameter, name: high, type: int
  Compound statement :
    Var declaration, name: i, type: int
    Var declaration, name: k, type: int
    Assign :(destination) (source)
      Id: i
      Id: low
    While
      Op: <
        Id: i
        Op: -
          Id: high
          Const: 1
      Compound statement :
        Var declaration, name: t, type: int
        Assign :(destination) (source)
          Id: k
          Call, name: minloc, with arguments below
            Id: a
            Id: i
            Id: high
        Assign :(destination) (source)
          Id: t
          Id: a
            Array : [k]
        Assign :(destination) (source)
          Id: a
            Array : [k]
          Id: a
            Array : [i]
        Assign :(destination) (source)
          Id: a
            Array : [i]
          Id: t
        Assign :(destination) (source)
          Id: i
          Op: +
            Id: i
            Const: 1
```

또한 CFG에 맞지 않은 소스가 들어오면 틀린 부분의 줄을 출력 하도록 하였다.

아래는 틀린 소스의 파싱 결과이다.

```
 void main(){
        int;
        int a;
}
```

```
CMINUS COMPILATION: test3.cm
Syntax error at line 1: syntax error
Current token: )
```