[Project 3 : Semantic]

컴퓨터전공 2013011253 김기원

Project 3 : C-Minus Semantic Analysis

0. 컴파일 환경 및 도구

기본 Environment : VMware(Ubuntu 17.10)

기본 Tools : GCC (version 5.4.0) , Flex (version 2.6.0) , bison(version 3.0.4) Makefile

실행을 위한 명령어 :

$ make (in home directory : cp3) (Create cminus_sm : Main file)

$ make gcd (./cminus_sm gcd.cm)

$ make sort (./cminus_sm sort.cm)

$ make clean (rm all created by Makefile)

1. 코드 설명 (간략)

1-1: Main :

```
/* allocate and set tracing flags */
int EchoSource = FALSE;
int TraceScan = FALSE;
int TraceParse = FALSE;
int TraceAnalyze = TRUE;
int TraceCode = FALSE;
```

Analyze 에 대한 출력만 허용하게 한다.

만약 테스트 하는 코드의 출력을 원하는경우, EchoSource = TRUE 로 수정하여도 된다.

1-2: Global.h , symtab.h :

```
struct ScopeListRec;

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;
     ExpType type; /* for type checking of exps */
     struct ScopeListRec *scope;

   } TreeNode;
```

해당 TreeNode가 어느 Scope에 속해있는지 쉽게 판단하기 위해 scope를 저장할 수 있도록 TreeNode의 구조체를 위와 같이 수정하였습니다.

```
typedef struct BucketListRec
   { char * name;
     ExpType type;
     LineList lines;
     int memloc ; /* memory location for variable */
     struct BucketListRec * next;
     TreeNode *treenode;
   } * BucketList;

typedef struct ScopeListRec{
  char * name;
  BucketList bucket[SIZE];
  struct ScopeListRec * parent;
} * ScopeList;
```

BucketListRec에 해당 bucket이 어느 TreeNode에서 파생된 것인지 쉽게 확인해 주기 위해 TreeNode변수인 node를 멤버로 추가합니다.

심볼 테이블을 만들 때 static scope rule을 이용하기 위해 위와 같이 수정된 BucketList를 저장해주는 bucket과 scope의 이름, 중첩깊이, 조상 scope를 저장할 수 있도록 ScopeListRec를 위와 같이 구현합니다.

1-3: symtab.c :

```
static ScopeList totalScope[1000];
static int ntotalScope = 0;
static ScopeList scopeStack[1000];
static int nScopeStack = 0;
```

scope 배열은 모든 scope들을 저장하기 위한 배열이며 ntotalScope 변수는 총 몇 개의 scope가 저장되어 있는지를 나타내는 변수입니다.

scopeStack은 static scope rule을 적용하기 위한 스택 자료구조이며 nScopeStack은 스택의 top 변수 역할을 해주는 변수입니다. scopestack의 가장 밑은 항상 global scope입니다.

```
if (l == NULL) /* variable not yet in table */
{

  l = (BucketList) malloc(sizeof(struct BucketListRec));
  l->name = name;

  l->lines = (LineList) malloc(sizeof(struct LineListRec));
  l->lines->lineno = lineno;
  l->lines->next = NULL;

  l->type = type;
  l->memloc = loc;
  l->next = sc->bucket[h];
  l->treenode = t;
  sc->bucket[h] = l;
}
```

```
else /* found in table, so just add line number */
{
  LineList t = l->lines;

  while (t->next != NULL) t = t->next;
  t->next = (LineList) malloc(sizeof(struct LineListRec));
  t->next->lineno = lineno;
  t->next->next = NULL;
}
```

st_insert / 심볼 테이블에 정보 삽입

static scope rule에 따르면 현재 변수는 스택의 가장 위에 있는 scope에 저장해야 하므로 scopeStack의 맨 위에 있는 scope에 변수가 저장되도록 합니다.

이미 선언된 변수가 사용될 경우 심볼 테이블에 삽입할 필요가 없고 어느 라인 번호에서 나타났는지만 추가하면 될 때 사용되는 부분입니다.

st_lookup / global scope까지 name이라는 이름의 변수, 함수가 존재하는지 검색

```
BucketList st_lookup (char * name){
  int h = hash(name);
  ScopeList sc = scope_top();

  while(sc){
      BucketList l = sc->bucket[h];
      while(l!=NULL){
        if(strcmp(l->name, name) == 0) return l;
        l = l->next;
      }

    sc = sc->parent;
  }

  return NULL;
}
```

함수의 이름은 소스코드 전체에서 유일하게 구별되어야 하므로 전체 scope에 대해 같은 이름이 있는지 검색할 필요가 있다. 이 때 st_lookup 함수를 사용한다.

```
BucketList st_lookup_excluding_parent ( char *
  int h = hash(name);
  ScopeList sc = scope_top();

  if(strcmp(sc->name, scope)){
    BucketList l = sc->bucket[h];
    while(l!=NULL){
      if(strcmp(l->name, name) == 0) return l;
      l = l->next;
    }
  }
```

st_lookup_top / 현재 scope에서만 name이라는 이름의 변수, 함수가 존재하는지 검색

변수의 이름은 해당 scope내에서만 유일하게 구별되면 되므로 해당 scope에 같은 이름을 갖는 변수가 있는지 검색할 때 사용된다.

1-3: analysis.c analysis.h :

analysis.h 의 수정사항은 analysis.c에 새로 추가된 함수의 선언부를 추가한 것이 전부이므로 설명을 생략합니다. analysis.c의 경우 여러 경우에 대한 예외처리가 많아 소스코드 전체를 설명하는 것은 힘들기 때문에 전체 소스코드에 대한 설명은 생략하는 대신 어떠한 알고리즘으로 심볼테이블을 만들었는지, 각 타입검사를 어떻게 실시하였는지만 설명하도록 하겠습니다.

먼저 심볼테이블은 input, output 함수에 대한 정보를 제일 먼저 삽입한 후 파스트리의 각 TreeNode들을 insertNode 와 Pop 순서로 함수를 실행하며 만들어집니다. 이 때 중요한 것은 TreeNode들은 부모노드보다 자식노드가 먼저, 오른쪽 노드보다 왼쪽 노드가 먼저 탐색되는데 이는 속성 계산이 아래에서 위로, 왼쪽에서 오른쪽으로 이루어짐을 의미합니다. insertNode 함수는 각 Node의 정보들을 심볼 테이블에 넣을지 말지를 판단하며 이 과정에서 오류가 발생할 경우 오류까지 출력해주는 역할을 수행합니다.

2. 결과 :

$ make gcd :

```
kkw@ubuntu:/mnt/hgfs/share/semantic$ make gcd
./cminus_sm gcd.cm

CMINUS COMPILATION: gcd.cm

Building Symbol Table...

Symbol table:

Variable Name   Type    Location        Scope           Line Numbers
-------------   -----   ---------       ------          ------------
main            void            3       Global          11,
input           int             0       Global          0,15,16,
output          void            1       Global          0,17,
gcd             int             2       Global          4,7,17,
u               int             0       gcd             4,6,7,7,
v               int             1       gcd             4,6,7,7,7,
x               int             0       main            13,15,17,
y               int             1       main            14,16,17,

Checking Types...

Type Checking Finished
```

$ make sort :

```
kkw@ubuntu:/mnt/hgfs/share/semantic$ make sort
./cminus_sm sort.cm

CMINUS COMPILATION: sort.cm

Building Symbol Table...

Symbol table:

Variable Name   Type    Location        Scope           Line Numbers
-------------   -----   ---------       ------          -------------
main            void            5       Global          32,
sort            void            4       Global          19,39,
input           int             0       Global          0,36,
minloc          int             3       Global          3,25,
output          void            1       Global          0,42,
x               int[]           2       Global          2,36,39,42,
low             int             1       minloc          3,7,8,9,
a               int[]           0       minloc          3,8,11,12,
i               int             3       minloc          4,9,10,11,12,13,15,15,
k               int             5       minloc          6,7,13,17,
x               int             4       minloc          5,8,11,12,
high            int             2       minloc          3,10,
low             int             1       sort            19,22,
a               int[]           0       sort            19,25,26,27,27,28,
i               int             3       sort            20,22,23,25,27,28,29,29,
k               int             4       sort            21,25,26,27,
high            int             2       sort            19,23,25,
t               int             0       sort            24,26,28,
i               int             0       main            33,34,35,36,37,37,40,41,42,43,43,

Checking Types...

Type Checking Finished
```