

컴파일러 #3

Usage

make all

./cminus_flex <target_file>

Environment

Ubuntu 18.04.1 LTS, x86_64

Flex 2.6.5 , bison 3.0.4

Implementation

1 Global scope 에서 시작해서, 함수 시작 시 Function scope, { } block 에서 사용되는 scope를 담기 위한 자료구조인 ScopeList를 추가.

```
39 + /* scope List save */
40 + typedef struct ScopeListRec
41 + {
42 +     char *name;
43 +     BucketList bucket[SIZE];
44 +     struct ScopeListRec *parent;
45 + } * ScopeList;
46 +
```

2 symbol table에 추가할 때 위에서 만든 scope정보 안의 bucket에 저장해야 되므로 함수 수정

Symbol table, scope를 위한 함수 정의 (sc_top(), sc_pop() 등)

```
/* Procedure st_insert inserts line numbers and
 * memory locations into the symbol table
 * Loc = memory location is inserted only the
 * first time, otherwise ignored
 */
- void st_insert(char *name, int lineno, int loc);
+ void st_insert(char *scope, char *name, ExpType type, int lineno, int loc);

/* Function st_lookup returns the memory
 * location of a variable or -1 if not found
 */
- int st_lookup(char *name);
+ BucketList st_lookup(char *scope, char *name);
+ BucketList st_lookup_excluding_parent(char *scope, char *name);
```

여기서 부터는 코드가 너무 길어서 설명으로 대체하고 코드는 `analyze.c` `analyze.h` `syntab.c` `syntab.h`에 수정했습니다.

3 global 영역에 있는 변수, 함수를 global scope의 bucket에 추가하고 built in function을 정의

```
+ ScopeList globalScope;
+ static char name[] = "global";
+ static char *scopeName = name;

+ globalScope = sc_create(scopeName);
+ sc_push(globalScope);
+
+ TreeNode *input_func;
+ st_insert(scopeName, input_func->attr.name, input_func->type, input_func);
```

4 symbol table 을 만들기 위해 traverse

```
traverse(syntaxTree, insertNode, nullProc);
traverse(syntaxTree, insertNode, forPop);
```

insertNode : 입력된 tree를 보고 심볼테이블에 넣어주는 역할

Function – 현재 scope를 function name으로 하나 만들고 현재 상태를 function scope로 변경

Compound – 현재 scope에서 nested될 때마다 _n을 concat해서 scope를 중첩시킨다

Variable, Param – 현재 scope의 bucket에 추가한다.

Call, Id, ArrayId – 변수가 사용되는 scope를 찾아서 line no를 추가한다

forPop : insertNode에서는 {}가 끝날 때 scope를 하나 밖으로 빠져나온다

5 symbol table을 만들었으니 symbol error 잡기

- 이미 정의된 함수 정의

```
symbolError(t, "function already declared in scope");
```

- 이미 정의된 변수 정의

```
symbolError(t, "variable already declared in scope");
```

- Void type의 변수 정의

```
symbolError(t, "variable type should not Void");
```

- 정의되지 않은 함수나 변수 사용

```
symbolError(t, "variable or function undeclared");
```

6 TypeCheck를 위해 traverse

```
void typeCheck(TreeNode *syntaxTree)
{
    sc_push(globalScope);
    traverse(syntaxTree, forPush, checkNode);
    sc_pop();
}
```

Check Node : 정의된 **symbol table**에 의해서 내가 사용하는 변수의 타입이나 위치를 찾아 확인

AssignK – left와 right의 type이 같은지 확인 후 type을 둘 중 하나로 지정

OpK – left와 right에 Void가 있다면 에러. + - * / == != > < >= <= 에 Integer만 오도록

연산 후에는 Bool type을 사용하지 않아서 Integer type으로 대체

CallK – 함수 호출시 argument개수와 type 을 확인

ConstK – type을 Integer로 지정

IdK, ArrayIdK – 변수가 정의된 가장 가까운 scope를 찾아서 type을 지정

forPush : scope가 열릴 때 scope를 push한다 (함수의 시작이나 그냥 블록)

```
if (t->nodekind == StmtK) {
    if (t->kind.stmt == CompoundK) sc_push(t->scope);
    else if (t->kind.stmt == FunctionK) scopeName = t->attr.name;
}
```

7 Type Error 잡기

```
typeError(t->child[0], "Assign should be same two var's type");
typeError(t->child[0], "can not be void in op");
typeError(t->child[0], "Op should be same two var's type");
typeError(t->child[0], "+ - * / should be integer");
typeError(t->child[0], "== != > < >= <= should be integer");
typeError(functionDeclaredArgs, "too few args!");
typeError(curArgs, "can not come void");
typeError(curArgs, "should be same type args");
typeError(t->child[0], "ArrayIdK not Integer");
typeError(t->child[0], "can not void in if or while state");
typeError(t, "void function return not void");
typeError(t, "integer function return not integer");
```

1 Assign시 type 확인

2 void는 연산 불가

3 Op는 같은 type 끼리

4 연산은 Integer끼리

5 비교 등 도 Integer끼리

6 함수의 argument가 너무 적다 (많다도있음) `TypeError(curArgs, "too many args!");`

7 argument에 void type못오도록

8 argument의 type이 정의와 다름

9 int 배열과 int 타입 에러

10 if 나 while에 void못옴

11 void 함수에서 int return

12 int 함수에서 void return

8 정리
















함수 안에서 함수를 정의하는 에러를 잡으려 했는데 이미 syntax 에러였다.

symtab에서 symbol table, bucket등의 함수를 정의한다

analyze에서 symtab을 사용하여 symbol table을 실제로 만들고 symbol, type 에러를 체크한다

기존의 Tree 구조에서 자신의 Scope를 찾아갈 수 있게 해서 FunctionK의 tree에서 자신의 scope를 찾아간다.

9 Commit 기록

 Merge branch 'feature/semantic1' into 'master' [M] 정두진 committed about 16 hours ago	
 feat: add more exception [M] clucie committed about 16 hours ago	
 style: unused comment del [M] clucie committed about 17 hours ago	
 feat: 함수 파라미터 타입 맞추기 [M] clucie committed about 17 hours ago	
 fix: 타입이 다른걸 에러로 해야되는데 같은걸 에러로 보던 걸 수정 [M] clucie committed about 17 hours ago	
 feat: TypeError 처리 [M] clucie committed about 18 hours ago	
 feat: 배열 Integer인지 확인 type error exception [M] clucie committed about 20 hours ago	
 feat: symbol error exception [M] clucie committed about 21 hours ago	
 feat: symbol table 완성 [M] clucie committed about 22 hours ago	
14 Dec, 2018 6 commits	
 feat: void int IntegerArray 구분 while 등 지리 안에서 에러상태 [M] clucie committed a day ago	
 feat: 이미 정의된 걸 call 하거나 찾아갈 때 recursive 하게 부르는 것찾기 [M] clucie committed a day ago	
 feat: IDK, ArrayIDK 심볼 테이블에 추가 [M] clucie committed a day ago	
 feat: 함수 내부 변수 정의 심볼테이블에 추가 [M] clucie committed a day ago	
 feat: global 영역에 있는 함수들 추가 [M] clucie committed a day ago	
 feat: 심볼테이블 scope, block 등 정의. 다른 소스에서 에러 만나도록 주석처리 (후에 바꾸기) [M] clucie committed a day ago	