[Compiler] 3. Semantic

2019067101 CSE 황인용

Overview

C-Minus Semantic Analyzer Implementation

[Goal]

1. Hierarchical Symbol Table 구현
2. Semantic Error 검출

[Compilation method and environment]

VirtualBox Linux: Ubuntu 18.0.4

Implementation

1. Symbol Table

Hierarchical Symbol Table을 구현하기 위해 다음과 같은 사항을 고려했다.

- Original Compiler의 Symbol Table
- Symbol Table이 생성되는 조건
- Hierarchical Symbol Table의 특성과 구현

(1) Original Compiler의 Symbol Table

- 기존의 Symbol Table은 소스 코드 하나에 하나의 Symbol Table을 가정하고 구현되어 있다.
- 하나의 Symbol은 하나의 BucketList에 할당되며, hashTable이라는 전역 변수를 두고 할당된다.
- 따라서 Hierarchical Symbol Table을 구현하기 위해 BucketList의 상위 계층인 ScopeList를 만들었다.

```
// symtab.c
typedef struct ScopeListRec {
  char * name; // Scope의 이름
  BucketList hashTable[SIZE]; // ScopeList의 하위 계층
  struct ScopeListRec * parent; // Scope의 parent scope
  int depth; // global depth = 0 기준 child scope + 1
} * ScopeList
```

(2) Symbol Table이 생성되는 조건

- Symbol Table이 만들어지는 조건은 크게 두가지이다.
  1. **Function Declaration**
  2. **Variable Declaration**
- 이전 2_Parser에서 구현한대로, Function Declaration은 TreeNode가 FunK일때, Variable Declaration은 CompoundK일때 만들어진다.
- 그러므로 analysis.c의 insertNode(TreeNode * t)에서 FunK일때, CompoundK일때 새로운 Scope를 생성하도록 하였다.
- FunK 일때, child[0]이 arguments, child[1]이 compound_stmt이다. 위의 사항대로 단순히 Scope를 생성하면 하나의 Function에 두 개의 Scope가 연속적으로 생길 수 있으므로 flag를 두어 FunK일때 set하고, CompoundK 일때 flag가 올라가 있으면 flag를 내리게 하였다.

(3) Hierarchical Symbol Table의 특징과 구현

- AST를 traverse하며 Hierarchical Symbol Table을 생성하는 과정에서 , Symbol Table이 완성되기 전에 새로운 Scope를 형성하기도 한다.
- 이는 **Stack** 구조를 통해 구현할 수 있다고 판단하였고, 전체 결과를 담을 전역 변수도 두었다.
- 또한 Scope 생성 뿐만이 아니라, 만들어진 ScopeList에 쌓이는 location을 추가하기 위해 location에 해당하는 stack도 만들어 주었다.

```
// symtab.c
static ScopeList scopes[SIZE]; // Hierarchical Symbol Table 결과
static int num = -1;
static ScopeList stack[SIZE]; // Building Table에 사용되는 Stack 1
static int top = -1;
```

```
static int loc_stack[SIZE]; // Building Table에 사용되는 Stack 2
static loc_top = -1;

// symtab.h
void st_init(void); //int input(void), void output(int data)와 같은 것들
void st_newScope(char * name); // FunK, CompoundK 시, 실행
void st_push(ScopeList s); // stack에 필요한 함수들
void st_pop(void);
ScopeList st_get_top(void);
int addLoc(void); // 새로운 Symbol 등록시 실행되는 함수
void popLoc(void); // st_pop()시, loc_stack에서도 pop이 필요
```

2. Checking Semantic Error

(0) AST Traversal

- Hierarchical Symbol Table을 저장하고 있는 전역변수인 scopes는, pre-order의 순서로 저장되어 있다.
- type checking은 input, output 다음 scope부터 진행하면 된다.
- 이런 점을 종합했을 때, checkNode(TreeNode * t)에서 t→kind.stmt가 CompoundK 일때, analyze.c의 전역변수로 지정한 static int idx = 3 의 값을 하나씩 증가시켜주면, 해당 node가 생성된 곳부터 lookup 함수를 호출할 수 있다.

(1) Undefined / Redefined Variables / Functions

- Undefined/Redefined Variables/Functions Error를 검출
- insertNode에서,
  - Function Declaration, Variable Declaration의 경우, st_lookup의 반환값이 -1이 아닌 경우 **Redefined Error**
  - Call Expression, Variable Expression의 경우, st_lookup의 반환값이 -1인 경우 **Undefined Error**
- typeCheck에서,
  - VariableK 일때, CallK 일때 check

(2) Array Index Check

- typeCheck에서,
  - VariableK 일때, t→child[0]이 NULL이 아니고 type이 Integer가 아닐 경우 Error

(3) Function Call Check

- int callCheck(TreeNode * arg, ScopeList f)에서 구현
  - cnt는 table 내에서 Arg인 Bucket의 개수, cnt_는 function call에 들어간 argument의 개수
  - BucketList에서 kind가 Arg이고, memloc이 N이라면, arg = arg→silbing을 N번 반복하면 위치가 동기회된다고 볼수 있다.
  - type이 일치하지 않거나 마지막 cnt와 cnt_가 같지 않을 경우 -1을 반환

(4) Type Check

- Original Compiler의 Traversal을 C-Minus Specification에 맞추어 이용

3. Output Format

- 출력 양식은 자유라고 알고 있어, 그냥 모든 Scope를 출력하게 하였습니다.

Test

```
user@user-VirtualBox:~/assignment$ ./cminus_semantic test.txt

C-MINUS COMPILATION: ./test.txt

Building Symbol Table...

Symbol table:

[Scope : global (depth = 0)]
Symbol Name  Symbol Type  Symbol Kind  Location  Line Numbers
-----------  -----------  -----------  --------  ------------
main         void         function      3        11
input        int          function      0         0 14 14
output       void         function      1         0 15
gcd          int          function      2         4  7 15

[Scope : input (depth = 1)]
Symbol Name  Symbol Type  Symbol Kind  Location  Line Numbers
-----------  -----------  -----------  --------  ------------

[Scope : output (depth = 1)]
Symbol Name  Symbol Type  Symbol Kind  Location  Line Numbers
-----------  -----------  -----------  --------  ------------
value        int          argument      0         0
```

```
[Scope : gcd (depth = 1)]
Symbol Name  Symbol Type  Symbol Kind  Location  Line Numbers
-----------  -----------  -----------  --------  ------------
u            int          argument      0         4  6  7  7
v            int          argument      1         4  6  7  7  7

[Scope : main (depth = 1)]
Symbol Name  Symbol Type  Symbol Kind  Location  Line Numbers
-----------  -----------  -----------  --------  ------------
x            int          variable      0        13 14 15
y            int          variable      1        13 14 15


Checking Types...

Type Checking Finished
user@user-VirtualBox:~/assignment$
```