Semantic Analysis Report

2024062806, 주하진

컴파일 환경 및 방법

주어진 Makefile을 이용해서 컴파일했다. 컴파일 및 빌드시 “cminus_semantic” 실행파일이 생성한다.

세부 구현

구조체 정의

Semantic Analysis를 진행하는데 필요한 대부분의 구조는 `symtab.h`에서 정의한다.

`symtab.h`에서 `LineList`, `BucketList`, `Scope`가 정의된다. 또 전역 상태 변수인 scope_stack과 scope_global이 `symtab.c`에서 정의된다.

Scope는 name, depth, location을 갖고 있으며 BucketList의 배열(HashTable)을 소유한다. 그리고 다른 Scope와 traversal을 도와주는 목적으로 parent, child, last_child, sibling 포인터를 갖고 있다.

BucketList는 struct BucketListEntry *이다. BucketListEntry는 변수를 담는 그릇으로 줄 번호 리스트(LineList), 심볼 타입, 변수 타입, 파라미터 타입 배열, 파라미터 이름 배열, 파라미터 개수, 리턴 타입, memloc, 그리고 다음 엔트리 참조를 갖고 있다.

이 중 파라미터 관련한 변수와 리턴 타입은 심볼 타입이 Func일때만 유효하다.

LineList는 LineNo를 담는 리스트로 연결리스트로 구현돼있다.

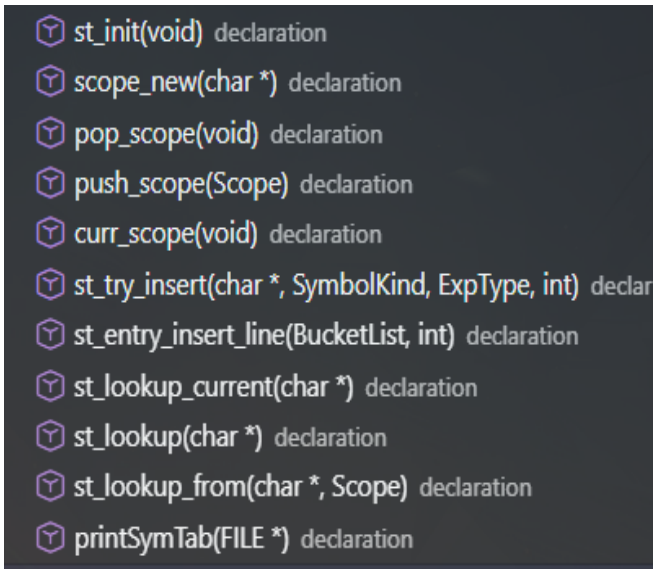
그리고 ParseTree에서 Scope를 쉽게 참조할 수 있도록 TreeNode에 Scope도 추가했다.

구조체의 동작 구현

외부에 노출되어 있는 함수는 다음과 같다.

- 스코프 관련 함수
  - `scope_new`: 새로운 스코프를 만듬. @다른 사이드이펙트는 없음
  - `pop_scope`: 전역 스코프 스택에서 스코프를 뺌. 스코프의 상태는 바꾸지 않음.
  - `push_scope`: 전역 스코프 스택에 스코프를 넣으며, 부모 스코프와 넣는 스코프와 링크를 함.
  - `curr_scope`: 스택 탑

- 심볼테이블 관련 함수
  - `st_try_insert`: 스택 탑 스코프에 심볼을 넣는다. 버킷 엔트리를 반환 @ 만약 존재하면 라인넘버를 넣는다. @NonNull in almost case
  - `st_lookup_current`: 현재 탑에서 심볼을 탐색한다. 버킷 엔트리를 반환 @Nullable
  - `st_entry_insert_line`: 라인넘버를 엔트리에 넣는다.
  - `st_lookup`: 현재 탑 스코프에서부터 루트(글로벌)까지 심볼을 탐색한다. 버킷 엔트리를 반환 @Nullable
  - `st_lookup_from`: 특정 스코프에서부터 루트(글로벌)까지 심볼을 탐색한다. 버킷 엔트리를 반환` @Nullable



- 전역 상태 관련 함수
  - `st_init` 전역 상태 변수 scope_stack, scope_global의 상태를 초기화함

분석의 동작 구현

본격적인 분석은 `buildSymtab`과 `typeCheck`, 두 단계로 이루어진다. buildSymtab과 typeCheck 모두 traverse를 이용하는데, traverse는 SyntaxTree를 루트노트부터 순회를 하는 함수로 순회를 할 때 그 노드에서 자식 노드를 순회하기 전에 실행하는 preProc과 그 자식노드를 모두 순회하고 나서 실행하는 psotProc이 인자로 주어진다.

buildSymtab에서는 parseTree를 traverse하면서 preProc인 insertNode에서 Entry를 넣거나, Scope를 넣거나 등의 동작을하며 최종적으로 scope_global을 루트로 하는 ScopeTree를 구성한다.

이때 특이할 점은 Scope를 넣는 동작(push-scope)이 Compound에서만 동작한다는 점이다. Function Decl에서는 먼저 임시 변수인 func_entry, func_scope, func_params 등에 넣어놓고. 이후 Compound를 처리할 때 func_entry NULL 체크로 판단한다.

Scope Naming Convention은 `{func_name}(.{parent scope child count at push time})*` (만약 function이 이미 decl됐을때는 랜덤 이름의 더미 스코프를 만들고 진행)

그리고 이후 postProc인 afterNode에서 pop_scope를 한다.

TypeCheck는 다시 TreeNode를 순회하며 타입체킹을 한다. 이때 postProc에서 대부분의 일을 하며 bottom-up방식으로 expType을 갱신해간다.

이때 특이할점은 Return 처리인데 return 처리를 하기 위해 자기가 어떤 function 스코프에 와있는지 알아야 한다. 따라서 preProc인 beforeCheckNode에서 function Decl이면 func_entry에다가 현재 function의 버킷 엔트리를 추가한다. 그리고 다시 postProc에서 해지 해준다.

Implicit Declaration을 구현하기 위해 ExpType에 undetermined를 추가하기도 하였다.

이외에도 세부적인 동작은 코드에 구현 되어있다.

예시 및 결과

예시1

```c
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));
}
```

```
C-MINUS COMPILATION: ./test_1.cm

Building Symbol Table...


< Symbol Table >
 Symbol Name   Symbol Kind   Symbol Type    Scope Name   Location  Line Numbers
-------------  -----------  -------------  ------------  --------  ------------
main           Function     void           global        3          8
input          Function     int            global        0          0   11   11
output         Function     void           global        1          0   12
gcd            Function     int            global        2          1    4   12
value          Variable     int            output        0          0
u              Variable     int            gcd           0          1    3    4    4
v              Variable     int            gcd           1          1    3    4    4    4
x              Variable     int            main          0         10   11   12
y              Variable     int            main          1         10   11   12


< Functions >
Function Name   Return Type   Parameter Name  Parameter Type
-------------  -------------  --------------  --------------
main           void                           void
input          int                            void
output         void
-              -              value           int
gcd            int
-              -              u               int
-              -              v               int


< Global Symbols >
 Symbol Name   Symbol Kind   Symbol Type
-------------  -----------  -------------
main           Function     void
input          Function     int
output         Function     void
gcd            Function     int


< Scopes >
 Scope Name   Nested Level   Symbol Name   Symbol Type
------------  ------------  -------------  -----------
output        1             value          int

gcd           1             u              int
gcd           1             v              int

main          1             x              int
main          1             y              int


Checking Types...

Type Checking Finished
```

예시2

```c
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.cm

Building Symbol Table...


< Symbol Table >
 Symbol Name   Symbol Kind   Symbol Type    Scope Name   Location  Line Numbers
-------------  -----------  -------------  ------------  --------  ------------
main           Function     void           global        2         1
input          Function     int            global        0         0    8
output         Function     void           global        1         0   18
value          Variable     int            output        0         0
i              Variable     int            main          0         3    5    6    8   10   10   13   14   16
18
x              Variable     int[]          main          1         3    8   16   18


< Functions >
Function Name   Return Type   Parameter Name  Parameter Type
-------------  -------------  --------------  --------------
main           void                          void
input          int                           void
output         void
-              -              value           int


< Global Symbols >
 Symbol Name   Symbol Kind   Symbol Type
-------------  -----------  -------------
main           Function     void
input          Function     int
output         Function     void


< Scopes >
 Scope Name   Nested Level   Symbol Name   Symbol Type
------------  ------------  -------------  -----------
output        1             value          int

main          1             i              int
main          1             x              int[]


Checking Types...

Type Checking Finished
```

예시3

```c
int main(void)
{
    int x[5];
    x[output(5)] = 3 + 5;

    return 0;
}
```

```
C-MINUS COMPILATION: ./test_4.cm

Building Symbol Table...


< Symbol Table >
 Symbol Name   Symbol Kind   Symbol Type    Scope Name   Location  Line Numbers
-------------  -----------  -------------  ------------  --------  ------------
main           Function     int            global        2          1
input          Function     int            global        0          0
output         Function     void           global        1          0    4
value          Variable     int            output        0          0
x              Variable     int[]          main          0          3    4


< Functions >
Function Name   Return Type   Parameter Name  Parameter Type
-------------  -------------  --------------  --------------
main           int                            void
input          int                            void
output         void
-              -              value           int


< Global Symbols >
 Symbol Name   Symbol Kind   Symbol Type
-------------  -----------  -------------
main           Function     int
input          Function     int
output         Function     void


< Scopes >
 Scope Name   Nested Level   Symbol Name   Symbol Type
------------  ------------  -------------  -----------
output        1             value          int

main          1             x              int[]


Checking Types...
Error: Invalid array indexing at line 4 (name : "x"). indicies should be integer

Type Checking Finished
```