**Report - Project3**

**2016025196**

**김동규**

**1. Build environment**

컴파일은 windows의 ubuntu 20.04 LTS를 사용한 리눅스 환경에서 진행하였다

Makefile를 이용하여 make명령어를 통해 자동으로 컴파일이 되며, project1과 project2의 파일에 symtab, analyze파일을 통해 semantic analysis를 하는 프로그램이 생성된다.

**2. Implementation**

Semantic analysis 프로그램의 구현을 위해 analyze.c와 symtab.c파일이 사용된다.

Symtab파일에서는 크게 symbol table생성, insert table, lookup table, print table을 하는 함수가 구현되어있으며, analyze파일에서는 project2에서 구현된 기능에 따라 만들어진 syntax tree를 따라가며 1차로 symbol table 생성, 2차로 type checking을 진행한다.

**3. Algorithm**

Symbol table을 생성하기 위한 구조체는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

BucketList와 ScopeList는 전체 함수와 변수를 저장하기 위한 구조체이고, VarList와 FuncList는 Function table을 저장하기 위한 구조체이다.



Table을 직접적으로 구성하는 ScopeList와 FuncList는 hash table에 담겨 저장되며, hash함수를 통해 name에 따른 위치에 저장된다.

ScopeList는 hash table의 각 영역에 linked list의 형태로 구현된 BucketList를 갖고 있으며, FuncList는 hash table의 영역에 FuncList로 이루어진 Linked list로 되어있다.

insert함수는 function table을 위한 st\_insert\_func함수와, symbol table을 위한 st\_insert함수로 나누어 구현되었다. 각 함수는 hash()함수를 통해 해당 위치에 데이터가 있는지 검사한 뒤 존재할 시 hash값의 충돌여부를 검사하기 위해 linked list를 목표name까지 순회하며 탐색한다.

1. 이후 만약 일치하는 것이 존재하지 않으면, 새로운 scope를 생성하고 데이터를 저장한다.
2. 만약 일치하는 것이 존재하면 해당 scope에 추가 데이터를 저장한다.

Lookup 함수 또한 st\_lookup함수와 st\_lookup\_func함수에 추가적으로 st\_lookup\_including\_parent함수를 구현하였다.

st\_lookup함수와 st\_lookup\_func함수는 해당 스코프에서만 데이터를 찾는 함수이고 st\_lookup\_including\_parent함수는 1차로 해당 스코프에서 데이터를 찾은 뒤 global영역에서 데이터를 찾는 함수이다. 즉, 전역변수까지 탐색한다.

이렇게 구현한 이유는 scope특성상 function declaration영역 내부에서 scope가 해당 function name으로 바뀌기 때문에(compound statement의 경우 nested level로 구현했다) scope영역에서 데이터가 존재하지 않을 시, global영역만 찾아보면 되기 때문이다.

analyze에서는 symtab생성을 위한 insertNode함수와 type checking을 위한 checkNode함수, 이 둘을 실행하기 위한 traverse함수, 에러를 출력하기 위한 typeError, semantic\_error함수로 이루어져있다.

Traverse는 syntax tree와 함수포인터를 인자로 받아 동작하는데, preProc에 함수를 줄 경우 위에서 아래로 depth first탐색을 하고, postProc에 함수를 줄 경우 아래부터 위로 breadth first탐색을 하게 된다.

이를 통해 insertNode함수의 경우 preProc에 인자로 줘서 syntax tree를 scope별로 table에 insert하고 checkNode의 경우 postProc에 인자로 줘서 하위 노드를 type checking한 뒤, 상위 노드의 type이 정해지는 방식으로 오류 없이 동작할 수 있게 된다.

checkNode의 경우 노드의 expType이 OP나 Assign라면 하위 두 노드의 타입이 일치하는지, int형인지 확인하고, Func이나 VarDe의 경우 type specifier가 legal인지 확인한다. VarArr의 경우 []사이의 값이 존재할 경우 해당 식이 룰을 위반하는지 확인하며 Call의 경우 argument값들이 Definition과 일치하는지 확인한다. 문제가 없을 경우 노드의 type을 결정해준다.

insertNode의 경우 semantic check를 진행하며 symbol table에 insert하며 중복 선언이 되었는지 undefined 된 변수나 함수를 사용하는지 확인한다. Forward reference문제를 해결하기 위해 building table은 두 번 시도하여 undefined function을 확인한다.