<컴파일러 Semantic Analysis>

2012003686
임준수

1. Data structure and function

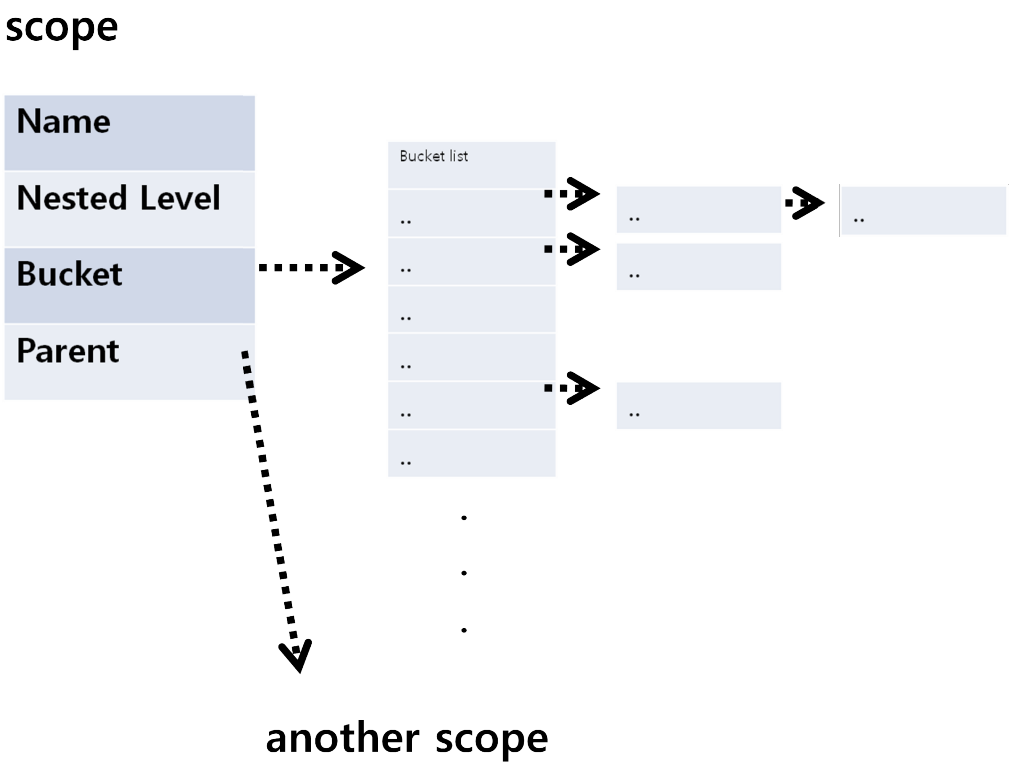
1) Data structure

BucketListRec

: name – 해당 symbol의 name 을 저장한다.
Type – globals.h 에 정의 되어있는 ExpType을 저장한다.
Param_opt – 해당 symbol이 param 이라면 몇번째 param 인지 저장한다.
Next – linked list의 next 를 가리킨다.
I_type – 해당 symbol 이 normal var, function ,param var 인지를 정의해준다.

ScopeListRec

: name – scope 의 이름을 정의한다.
Nested_level – 해당 scope 의 nested_level 알려줌
Bucket – 해당 scope 이 가지고 있는 bucket
Parent – scope이 parent로 가지고 있는 scope을 가리킨다.



2) Function

Symbol related function

st_insert(Scope scope, char* name, ExpType type, int lineno, IdType i_type, int param_opt)

해당 scope 에 symbol의 정보를 insert 해준다.
scope – symbol을 insert 할 scope
name – insert symbol 의 name
type – symbol type
lineno – symbol exist line num
i_type – id type이 어떻게 되는가 (normal var, function, paramval)
param_opt – param의 경우 몇번째 param인지

st_lookup(Scope scope, char *name)

해당 scope 부터 그 scope 의 root parent 까지 name 에 대한 symbol search를 진행한다. 존재한다면 가장 가까운 symbol 의 bucket을 반환한다.
scope – lookup 할 scope를 찾는다. Lookup의 범위는 해당 scope 부터 scope의 root parent 까지이다.
Name – lookup 할 symbol의 이름이다.
Return value – 해당 symbol bucket을 return 한다.

Scope related function

sc_init()

scope init 을 수행한다. 자세히 말하자면 global scope 생성, built-in function input() output(arg) function 을 정의 한다.

sc_push(char *scope, int nested_level)

해당 name 과 nested_level을 가진 scope을 추가한다. 이는 현재 참조하고 있던 scope을 parent로 가지게 되며 현재 참조하는 scope의 push 한 scope으로 바뀌게 된다.

sc_pop()

참조 할 scope를 현재 scope의 parent로 바꾼다.

sc_top()

현재 참조하고 있는 scope를 반환해준다.

set_cur_scope(Scope scope)

param으로 들어간 scope를 현재 참조하는 scope로 바꿔준다.

2. Symbol Table 만들기

1) Implementation overview

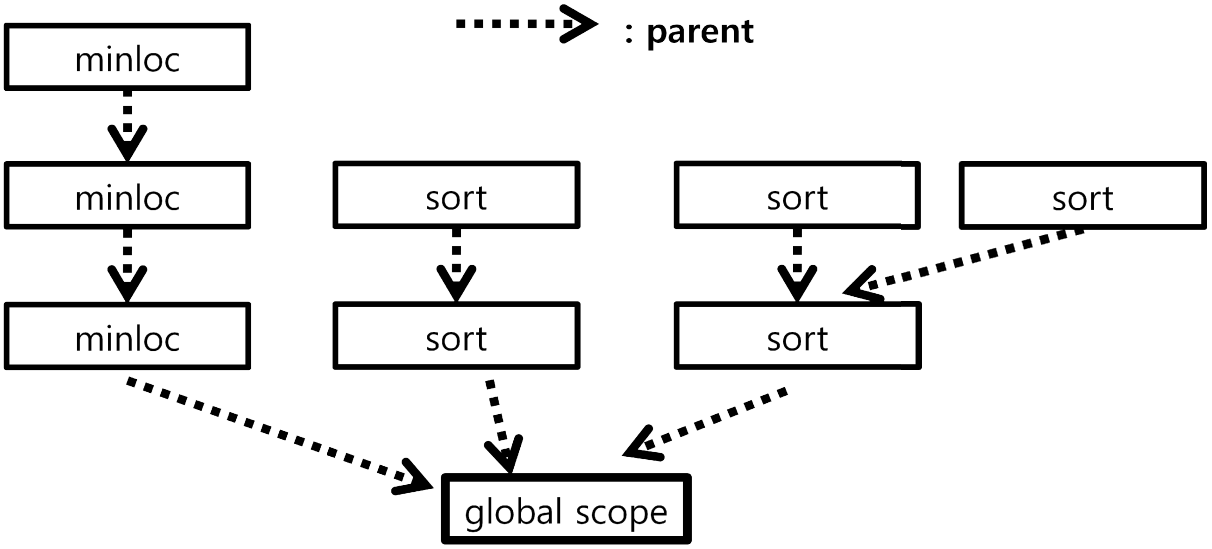
Symbol table build 는 InsertNode 와 afterInsertNode 부분으로 나뉜다. Scope 는 compound statement를 기준으로 분할이 된다.

<scope control>
InsertNode 에서는...
Compound statement를 생성하는 Node는 selection, Iteration 그리고 function 이 세가지가 있다. 해당 node 에 대해선 function의 이름을 가진 scope를 push 해주고 해당 treeNode에 scope pointer를 넣어준다. 이 scope pointer는 나중에 typecheck에서 쓰인다..

AfterInsertNode 에서는...
Scope를 빠져 나가는 것은 compound statement 를 만났을 때 일어나면 되는 일이다. 따라서 해당 상황에서는 sc_pop() 으로 현재 node의 parent를 참조하게 해준다.

<symbol insert and error check>
Expression 에서의 Id, Function call, Declaration, Parameter declaration 부분에서 symbol 의 insert 가 필요하다.
Symbol table build 에서는 Undefined symbol, void variable, assignment expression 에 대한 error check 가 일어난다.

2) explanation



Build symbol table 과정이 끝나고 나면 scope들은 다음과 같은 상관관계를 가지게 된다. 또한 각 scope 들은 scope이 push 되었던 treeNode에 저장이 된다. Type check 를 위한 traverse시 이를 가지고 scope의 정보를 가져온다.

- Undefined symbol
  Function 이나 var를 호출하는 경우 st_lookup을 통해 현재 scope 에서 참조 할 수 있는 id 인지 확인한다. var이나 function은 decl 이 먼저 나와야 사용 할 수 있다. Decl이 되어 있는 symbol 이라면 symbol table에 존재 할 것이므로 st_lookup의 결과가 NULL 이면 Error를 출력하면 된다.

- Void variable
  Variable declaration 에서 해당 var의 type이 void 인지 확인한다. Void 라면 scope에 insert하지 않고 error를 출력한다.

- Assignement check
  Assignment 시 맞는 assign인지 확인한다. cminus에서는 variable 이 가질 수 있는 type이 Integer와 IntegerArray 만 존재하기 때문에 해당 case에 대한 처리만 하면 된다.

3. Type checker

1) Implementation overview

Type checker 는 beforeCheckNode와 checkNode로 나뉜다.

<scope control>
Scope은 compound statement로 나뉘며, beforeCheckNode에서는 treeNode에 저장되있는 scope를 현재 scope로 참조한다.

checkNode에서는 beforeCheckNode에서 참조한 scope부터 탐색을 실행해 해당 symbol type 이 valid 한지를 check한다.
Type checker 에서 하는 error check 에는 return type check 와 function call check 가 있다.

2) Explanation

- Return type check
  해당 scope function 의 return type이 void 인 경우는 return 이후 expression의 유무로 type check 가 가능하다.
  하지만 return type이 void가 아닌 경우는 여러 case로 분류해서 비교해야 한다.
  Expression의 경우에는 항상 Integer의 value를 가진다. 그게 아닌 Id나 function call 의 경우에는 symbol table lookup을 통한 type check로 error 여부를 판별한다.

- Function call parameter check
  1) Param type check
     이전에 st_insert시 param_opt 란 value를 통해 param이 몇번째에 있는지 명시해 두었다. I_type 을 통해 해당 variable이 param 인지 알 수 있으며, tree traverse 시 param들은 순서대로 traverse 된다. 이는 param은 sibling 관계를 가지기 때문이다.
  2) Param 개수 check
     Traverse시 sibling 하나를 지나갈때마다 ++ 되는 변수를 두었다. 그리고 이를 scope 내의 param 개수와 비교한다. Scope내의 param 개수는 i_type 을 통해 구할 수 있다.