REPORT

Semantic Analysis Implementation

전공 : 소프트웨어전공
학번 : 2013011167
이름 : 진 용 석

1. 파서 구현 방법 및 주요 소스코드 설명

기존 tiny 의 hash table 을 큰 수정 없이 그대로 사용하였으며, Scope 구현을 위해 Table 단위로 구분합니다. struct BucketListRec 는 symbol 하나의 정보를 가지며, 함수의 경우 parameter 를 저장하기 위해 param 변수를 두고, 이외에는 이름이나 line 번호, type 등을 가집니다. struct SymbolTable 은 특정 scope 에서의 심볼 테이블을 의미하며, nested depth, function name, parent, sibling, child 등의 계층 구조 정보, 그리고 출력 및 탐색을 쉽게 하기 위한 각종 변수들을 가집니다.

```
typedef struct BucketListRec
{
    char * name;
    int lineno;
    int is_function;
    Type type;
    struct BucketListRec * next;
    struct BucketListRec * param;
} *BucketList;

/**
 * struct SymbolTable represents a symbol table:
 * an instance of SymbolTable represents a scope of a single block.
 */
struct SymbolTable
{
    /* hashTable has the data of whole declarations. */
    BucketList hashTable[SIZE];

    /* functionName has the name of function. global is set to __GLOBAL__. */
    char functionName[NAME_LENGTH];

    /* depth represents nested level. */
    int depth;

    /* children has the nested block's scope. */
    struct SymbolTable *child;

    /* sibling has the nested block's scope of parent.
       this works like a linked list for parent. */
    struct SymbolTable *sibling;

    /* parent represents the parent scope of this block. */
    struct SymbolTable *parent;

    /* represents that this symbol table is already visited when traversing. */
    int visited;

    /* table traversing order. */
    int order;
};
```

Symbol Table 구현에서 수정한 사항은, scope 에 따라 symbol 을 검색한다는 점입니다. 현재 scope 를 최우선적으로 탐색하며, 없는 경우에는 부모의 scope, 즉 상위 scope 에서 해당 symbol 을 검색합니다. 이렇게 global scope 까지 올라가며 탐색하게 되고, 없는 경우에는 NULL, 찾은 경우에는 BucketList * 레퍼런스 형태를 반환하도록 구현하였습니다.

```
68 /**
69  * st_lookup calls table_lookup recursively finds the variable from bottom to up.
70  */
71 BucketList st_lookup ( struct SymbolTable *table, char * name )
72 {
73     BucketList lookup_result = table_lookup( table->hashTable, name );
74
75     if (lookup_result != NULL)
76     {
77         return lookup_result;
78     }
79     else
80     {
81         if (table->parent == NULL)
82         {
83             /* reached over global scope : return -1. */
84             return NULL;
85         }
86         else
87         {
88             return st_lookup( table->parent, name );
89         }
90     }
91 }
```

Symbol Table 생성 시 호출하는 InsertNode 함수에서는 declaration node 인 경우에는 현재 symbol table 에 해당 symbol 을 추가합니다. parameter 인 경우에는 '현재 함수' node 에 param 으로서 추가합니다.

InsertNode 함수에서는 또한 scope 처리를 위해 특수한 경우에 새 struct SymbolTable instance 를 생성하여, 현재 테이블의 child 로 저장합니다. 이러한 현상이 발생하는 경우는 두 가지인데, 하나는 function declaration, 나머지 하나는 Conpound statement 입니다. function declaration 의 child[1] 은 필연적으로 Compound statement 이므로, 중복하여 Symbol Table 을 생성하지 않도록 처리하였습니다. 또한, Compound Statement 처리가 끝난 이후에는 상위 scope 로 backtrack 하도록 구현하였습니다.

InsertNode 함수에서 다른 node 를 처리하는 경우에는 Identifier 를 체크합니다. st_lookup 함수를 통해 체크하며, scope 에 없는 변수인 경우에는 오류를 표시합니다. 또한, ASSIGN 을 제외한 operator 에 type 을 부여합니다.

InsertNode 함수에서 처리하는 오류는 Void Variable 오류, Undefined Variable/Symbol 오류 등이 있습니다. 코드 상에서 참고 문서와 다른 점은, ASSIGN 연산 시에 자료형 검사를 을 심볼 테이블 생성 시가 아닌 type checking 시에 한다는 점입니다. 현재 C Minus 문법 상으로는 nested ASSIGN 연산 또는 ASSIGN 우변에 각종 expression 이 있을 수 있는데, top-down 방식으로 체크하는 Symbol Table Build 과정에서는 이에 대한 type checking 을 처리하기 어렵습니다. 따라서, type checking 시에 이를 체크하도록 처리하였습니다.

Type Checking 수행 시에는 parse tree 를 다시 탐색하며, 각각의 자료형을 체크합니다. 이 때 처리하는 오류로는 type inconsistance error, return type error, invalid function error 등이 있습니다. type inconsistance error 에 개인적으로 추가한 부분은 연산 수행 시 좌변과 우변의 type 이 Integer 가 아닌 경우 오류를 출력하도록 하였습니다. C Minus 에서는 Integer, Void, IntegerArray

밖에 없으므로, equal 과 not equal 을 제외한 모든 연산에 있어서는 Integer 만 처리 가능합니다. 따라서, Integer 가 아닌 경우에는 Type Consistancy 를 위반한 것으로 간주합니다.

```
case ExpK:
    /* check the validity of assignment */
    if (t->kind.exp == OpExp)
    {
        if (t->attr.op == ASSIGN)
        {
            if (t->child[0]->type != t->child[1]->type)
            {
                typeError( t );
            }
            else
            {
               t->type = t->child[0]->type;
            }
        }
        else if (t->attr.op != EQ && t->attr.op != NE)
        {
            if (t->child[0]->type != Integer
                    || t->child[1]->type != Integer)
            {
                typeError( t );
            }
        }
    }
```

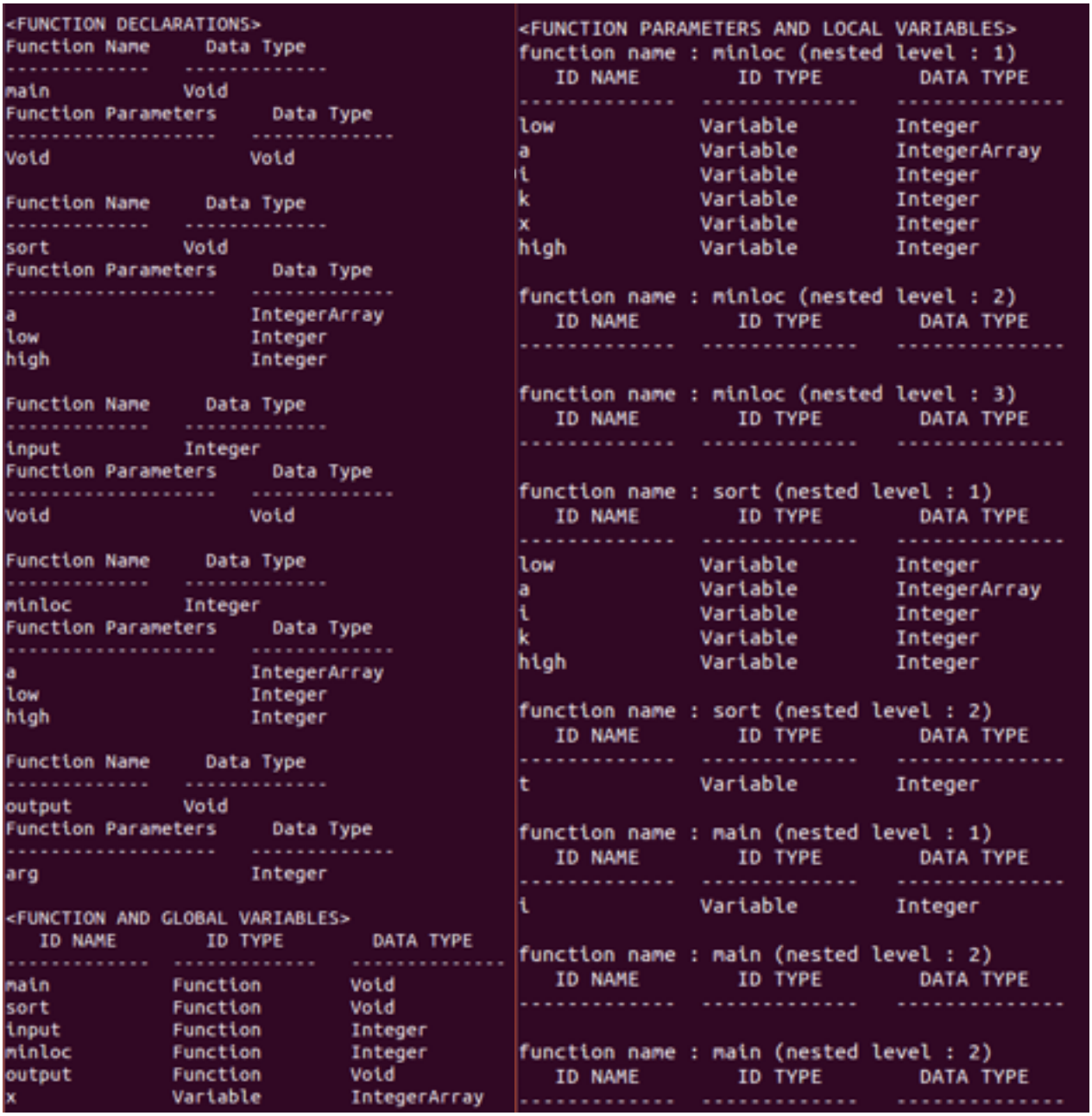
2. 컴파일 방법 및 환경

이 프로젝트는 Ubuntu 14.04, 커널 버전 3.13.0-34-generic 에서 개발 및 테스트하였습니다.

컴파일 방법은 이전의 Parser 와 동일합니다. 컴파일은 'make cminus_yacc' 명령어로 수행할 수 있으며, 각종 object file 들과 실행 파일인 cminus_yacc 파일이 생성됩니다.

make clean 명령어를 통해 중간 생성물과 최종 생성된 바이너리를 삭제할 수 있습니다.

3. 예시 및 결과 화면



```
<FUNCTION DECLARATIONS>
Function Name     Data Type
------------   ------------
main           Void
Function Parameters     Data Type
------------------   ------------
Void                 Void

Function Name     Data Type
------------   ------------
sort           Void
Function Parameters     Data Type
------------------   ------------
a                    IntegerArray
low                  Integer
high                 Integer

Function Name     Data Type
------------   ------------
input          Integer
Function Parameters     Data Type
------------------   ------------
Void                 Void

Function Name     Data Type
------------   ------------
minloc         Integer
Function Parameters     Data Type
------------------   ------------
a                    IntegerArray
low                  Integer
high                 Integer

Function Name     Data Type
------------   ------------
output         Void
Function Parameters     Data Type
------------------   ------------
arg                  Integer

<FUNCTION AND GLOBAL VARIABLES>
   ID NAME        ID TYPE        DATA TYPE
------------  ------------   -------------
main          Function       Void
sort          Function       Void
input         Function       Integer
minloc        Function       Integer
output        Function       Void
x             Variable       IntegerArray
```

```
<FUNCTION PARAMETERS AND LOCAL VARIABLES>
function name : minloc (nested level : 1)
   ID NAME        ID TYPE        DATA TYPE
------------  ------------   -------------
low           Variable       Integer
a             Variable       IntegerArray
i             Variable       Integer
k             Variable       Integer
x             Variable       Integer
high          Variable       Integer

function name : minloc (nested level : 2)
   ID NAME        ID TYPE        DATA TYPE
------------  ------------   -------------

function name : minloc (nested level : 3)
   ID NAME        ID TYPE        DATA TYPE
------------  ------------   -------------

function name : sort (nested level : 1)
   ID NAME        ID TYPE        DATA TYPE
------------  ------------   -------------
low           Variable       Integer
a             Variable       IntegerArray
i             Variable       Integer
k             Variable       Integer
high          Variable       Integer

function name : sort (nested level : 2)
   ID NAME        ID TYPE        DATA TYPE
------------  ------------   -------------
t             Variable       Integer

function name : main (nested level : 1)
   ID NAME        ID TYPE        DATA TYPE
------------  ------------   -------------
i             Variable       Integer

function name : main (nested level : 2)
   ID NAME        ID TYPE        DATA TYPE
------------  ------------   -------------

function name : main (nested level : 2)
   ID NAME        ID TYPE        DATA TYPE
------------  ------------   -------------
```

참고 문서에 있는 예제들을 그대로 실행한 결과입니다. 예제 문서와 동일한 결과를 출력하며, 특별한 문제점을 보이고 있지 않습니다.

```
parallels@ubuntu:~/Desktop/COMPILER$ ./cminus_yacc input/input.cm

TINY COMPILATION: input/input.cm

Building Symbol Table...
error : undeclared variable v at line 10

parallels@ubuntu:~/Desktop/COMPILER$ vi input/input.cm
parallels@ubuntu:~/Desktop/COMPILER$ ./cminus_yacc input/input.cm

TINY COMPILATION: input/input.cm

Building Symbol Table...
error : Variable type cannot be Void at line 7

Symbol table:
```

Symbol Table 생성 시 출력하는 오류에는 undeclared variable/function error, void variable error 가 있습니다. 이는 symbol table 생성 전에 출력하며, 오류가 발생하더라도 테이블은 최대한 그대로 출력합니다.

```
-------------  -------------   -------------


Checking Types...
type error at line 21 : return type inconsistance

Type Checking Finished

Checking Types...
type error at line 20 : invalid function call

Type Checking Finished

Checking Types...
error : type inconsistance at line 20

Type Checking Finished
```

Type Checking 시 출력하는 오류에는 return type error, invalid function call(parameter) error, type inconsistancy error 가 있습니다. symbol table 생성 후 이를 확인하며 출력합니다.