Project #3 Semantic Analysis

2018007938 김민관

Goal

- Symbol Table 과 type Checker 를 사용해서 모든 Semantic Error를 찾아낸다.
    - 생성된 AST 를 읽는다.
    - semantic error와 해당 error 의 line을 출력한다.

Implement

1. main.c

과제 설명서에 나온대로 관련 부분을 수정한다.

```
/* set NO_PARSE to TRUE to get a scanner-only compiler */
#define NO_PARSE FALSE
/* set NO_ANALYZE to TRUE to get a parser-only compiler */
#define NO_ANALYZE FALSE

/* allocate and set tracing flags */
int EchoSource = FALSE;
int TraceScan = FALSE;
int TraceParse = FALSE;
int TraceAnalyze = TRUE;
int TraceCode = FALSE;

int Error = FALSE;
```

2. symtab.h & symtab.c

Syntax tree 를 확인하면서, node를 저장할 bucketlist 구조체와 이를 담을 ScopeList 구조체를 선언한다.

```
typedef struct LineListRec
   { int lineno;
     struct LineListRec * next;
   } * LineList;

/* The record in the bucket lists for
 * each variable, including name,
 * assigned memory location, and
 * the list of line numbers in which
 * it appears in the source code
 */
typedef struct BucketListRec
   { char * name;
     TreeNode * treeNode;  /* tree node that having variable */
     LineList lines;
     int memloc ; /* memory location for variable */
     struct BucketListRec * next;
   } * BucketList;

/* The record for each scope
 * including name, its bucket,
 * and parent scope.
 */
 typedef struct ScopeListRec
   { char * funcName;
     BucketList hashTable[SIZE]; /* the hash table */
     struct ScopeListRec * parent;
```

```
    int nestedLevel;
  } * ScopeList;
```

static scope또한 구현하기 위해, scope를 stack으로 관리할 수 있는 함수를 추가한다. 그리고 symbol table을 출력하기 위한 함수 또한 구현한다.

```
void st_insert( char * name, int lineno, int loc, TreeNode * treeNode )
{ int h = hash(name);
  ScopeList nowScope = sc_top();
  BucketList l =  nowScope->hashTable[h];
  while ((l != NULL) && (strcmp(name,l->name) != 0))
    l = l->next;
  if (l == NULL) /* variable not yet in table */
  {
    //printf("variable not in table %d\n",loc);
    l = (BucketList) malloc(sizeof(struct BucketListRec));
    l->name = name;
    l->treeNode = treeNode;
    l->lines = (LineList) malloc(sizeof(struct LineListRec));
    l->lines->lineno = lineno;
    l->memloc = loc;
    l->lines->next = NULL;
    l->next = nowScope->hashTable[h];
    nowScope->hashTable[h] = l;
  }
  else /* found in table, so just add line number */
  {
    // LineList t = l->lines;
    // while (t->next != NULL) t = t->next;
    // t->next = (LineList) malloc(sizeof(struct LineListRec));
    // t->next->lineno = lineno;
    // t->next->next = NULL;
  }
} /* st_insert */

//table info
void printSymTab(FILE * listing);
void print_SymTab(FILE * listing);
void print_FuncTab(FILE * listing);
void print_Func_globVar(FILE * listing);
void print_FuncP_N_LoclVar(FILE * listing);
```

3. analyze.c

Compound State를 추가할 때 마다 새로운 Scope를 생성하여 Stack에 Push한다. 그리고, afterInsertNode 함수를 통해 Compound State를 빠져나갈 때 Stack을 Pop한다.

새로운 선언이 있을 경우, 현재의 Scope의 HashTable를 검사하여 중복이 있는지 확인한 하고, 변수를 사용할 때는 현재 Scope Stack의 Top부터 탐색하여 해당 변수가 있는지 확인한다.

```
static void typeError(TreeNode * t, char * name)
// { fprintf(listing,"Error: Type error at line %d: %s\n",t->lineno,message);
{
  fprintf(listing, "Error: Invalid function call at line %d (name : \"%s\")\n", t->lineno, name);
  Error = TRUE;
}

static void undeclaredError(TreeNode * t)
{ if (t->kind.exp == CallK)
  fprintf(listing, "Error: undeclared function \"%s\" is called at line %d\n", t->attr.name,t->lineno);
  else if (t->kind.exp == IdK || t->kind.exp == ArrIdK)
    fprintf(listing, "Error: undeclared variable \"%s\" is used at line %d\n", t->attr.name,t->lineno);
  Error = TRUE;
}

static void redefinedError(TreeNode * t)
{ if (t->kind.decl == FunctionK)
    fprintf(listing, "Error: Invalid function call at line %d (name : \"%s\")\n", t->attr.name,t->lineno);
  else if (t->kind.decl == VariableK)
    fprintf(listing, "Error: invalid assignment at line %d\n", t->lineno);
  else if (t->kind.decl == ArrayVariableK)
    fprintf(listing, "Error: Invalid array indexing at line %d (name : \"%s\"). indicies should be integer\n", t->attr.arr.name,t->lin
  Error = TRUE;
}

static void funcDeclNotGlobal(TreeNode * t)
{ fprintf(listing, "Error: Invalid function call at line %d (name : \"%s\")\n", t->lineno,t->attr.name);
  Error = TRUE;
}

static void voidVarError(TreeNode * t, char * name)
{ fprintf(listing, "Error: The void-type variable is declared at line %d (name : \"%s\")\n", t->lineno,name);
  Error = TRUE;
}
```

명세에서 주어진 print 문을 확인하여 출력 내용을 맞춘다.

3. globals.h

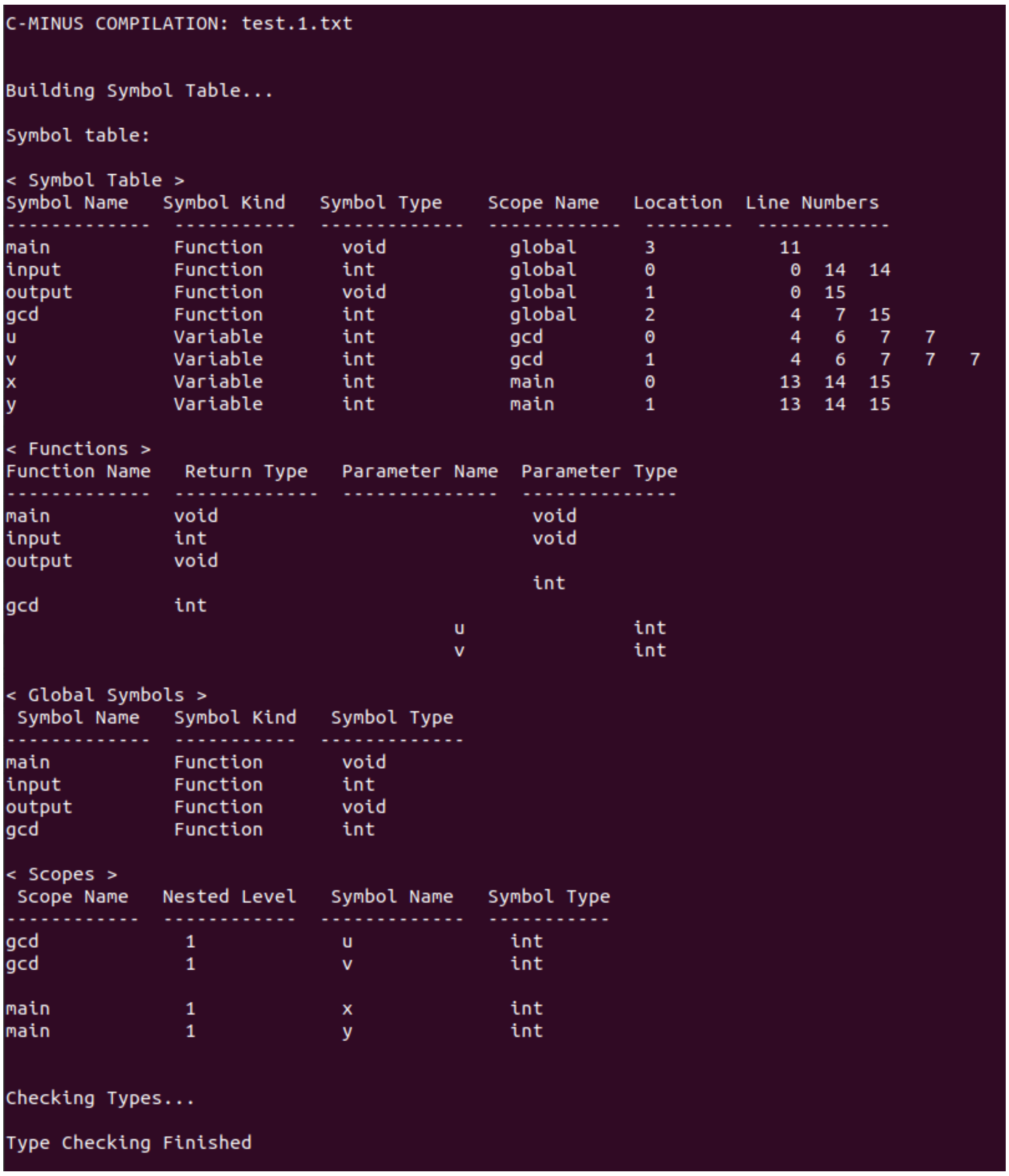
Tree 확인할 때, node를 통해 다른 Scope로 접근하는 경우가 발생할 수 있기 때문에 attr union에 Scope 구조체를 추가해준다.

```
typedef struct treeNode
   { struct treeNode * child[MAXCHILDREN];
     struct treeNode * sibling;
     int lineno;
     NodeKind nodekind;
     union { StmtKind stmt;
             ExpKind exp;
             DeclareKind decl;
             ParameterKind param;
             TypeKind type; } kind;
     union { TokenType op;
             TokenType type;
             int val;
             char * name;
             ArrayAttr arr;
            // 추가한 부분ㄴ
             struct Scope * scope} attr;
     ExpType type; /* for type checking of exps */
 } TreeNode;
```

RESULT

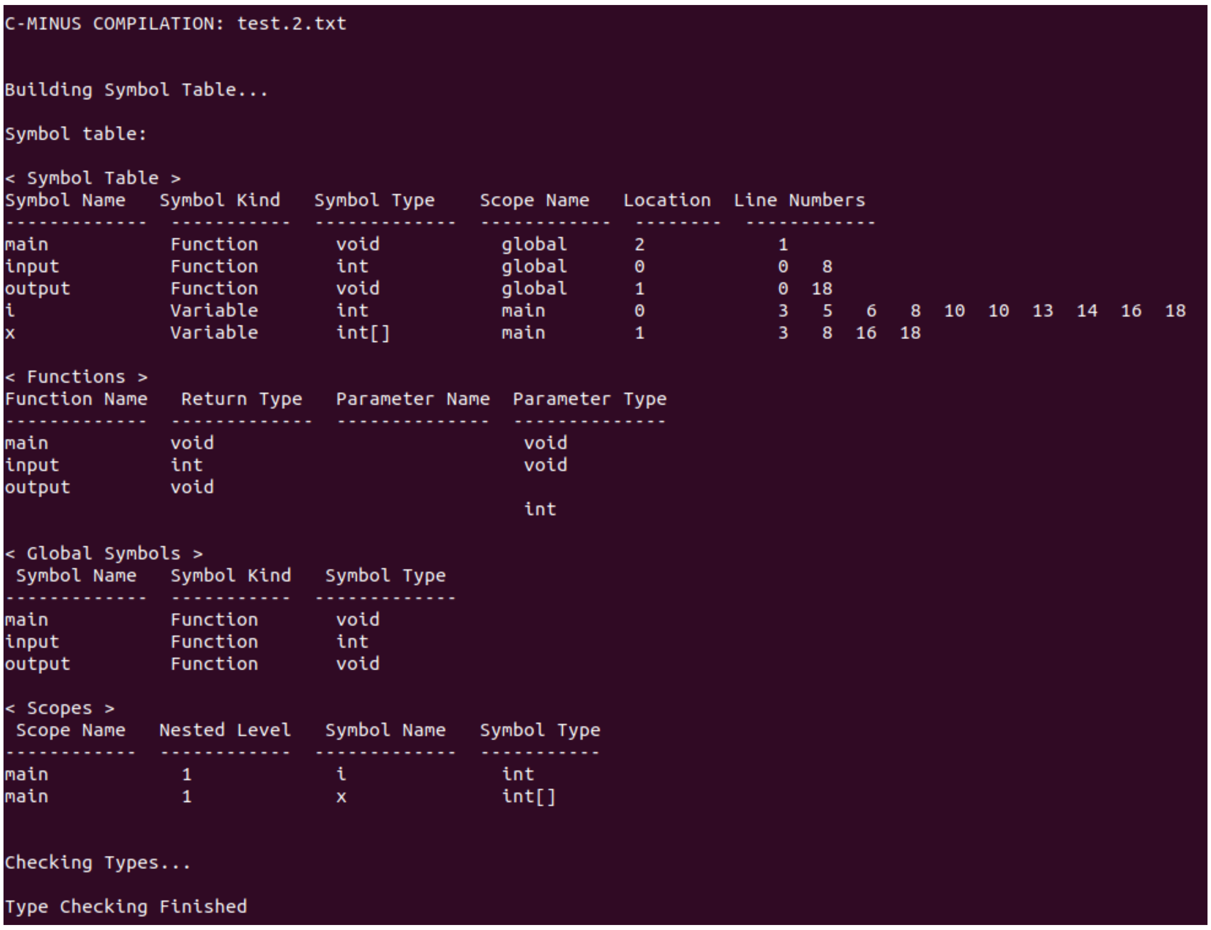
```
make cminus_semantic
./cminus_semantic test.1.txt
```

위 명령어를 입력하여 test.1.txt 의 symbol table과 에러가 있는지 확인한다.



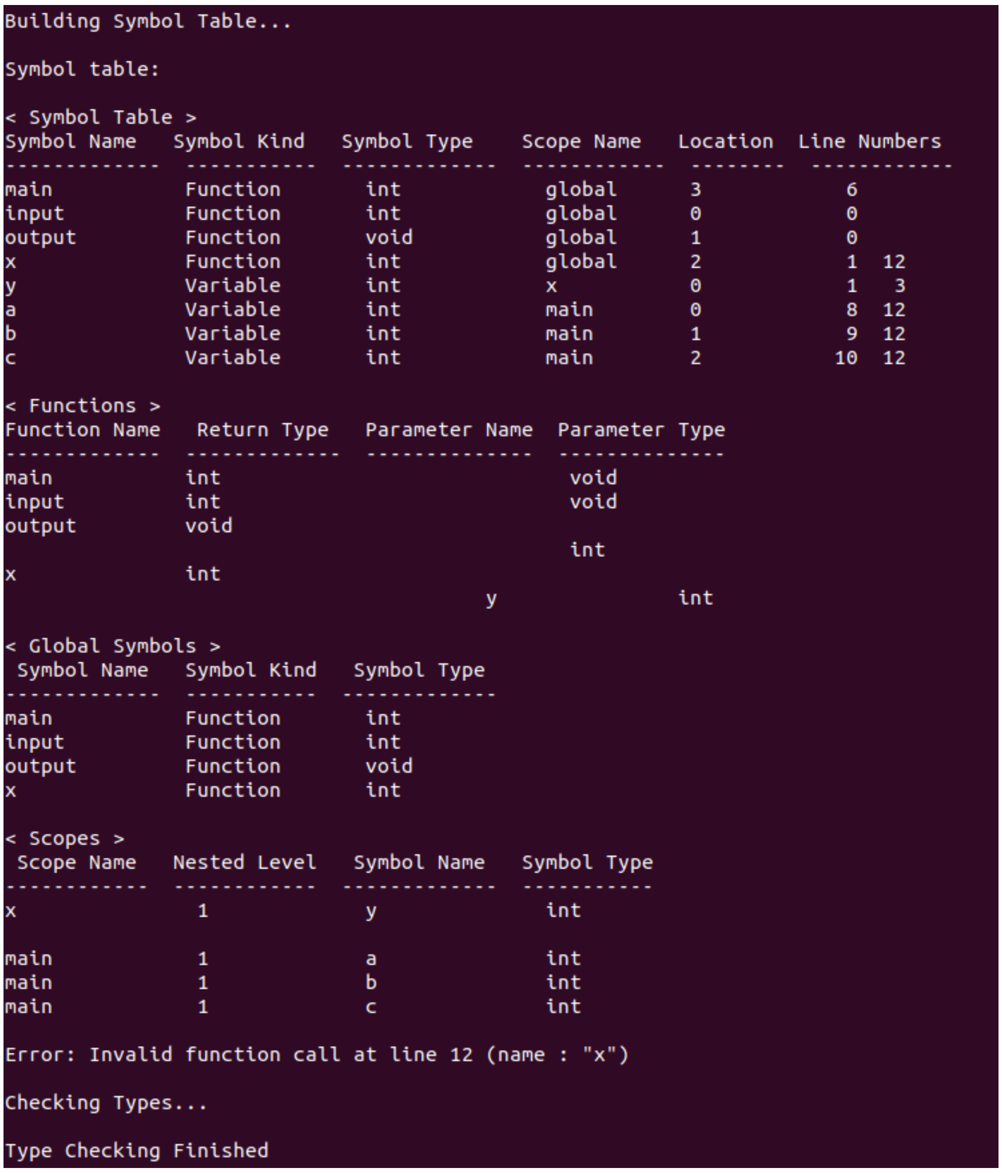
```
C-MINUS COMPILATION: test.1.txt


Building Symbol Table...

Symbol table:

< Symbol Table >
Symbol Name   Symbol Kind   Symbol Type    Scope Name   Location  Line Numbers
-------------  -----------  -------------  ------------  --------  ------------
main           Function       void           global      3           11
input          Function       int            global      0            0   14   14
output         Function       void           global      1            0   15
gcd            Function       int            global      2            4    7   15
u              Variable       int            gcd         0            4    6    7    7
v              Variable       int            gcd         1            4    6    7    7    7
x              Variable       int            main        0           13   14   15
y              Variable       int            main        1           13   14   15

< Functions >
Function Name   Return Type   Parameter Name  Parameter Type
-------------  -------------  --------------  --------------
main           void                            void
input          int                             void
output         void
                                               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
------------  ------------  -------------  -----------
gcd             1             u              int
gcd             1             v              int

main            1             x              int
main            1             y              int


Checking Types...

Type Checking Finished
```

```
./cminus_semantic test.2.txt
```



```
C-MINUS COMPILATION: test.2.txt


Building Symbol Table...

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
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

                                              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
-----------  -----------  ------------  ----------
main            1             i              int
main            1             x              int[]


Checking Types...

Type Checking Finished
```

```
./cminus_semantic test_3.txt
```



```
Building Symbol Table...

Symbol table:

< Symbol Table >
Symbol Name   Symbol Kind   Symbol Type    Scope Name   Location  Line Numbers
-------------  -----------  -------------  ------------  --------  ------------
main           Function       int            global       3            6
input          Function       int            global       0            0
output         Function       void           global       1            0
x              Function       int            global       2            1   12
y              Variable       int            x            0            1    3
a              Variable       int            main         0            8   12
b              Variable       int            main         1            9   12
c              Variable       int            main         2           10   12

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

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

< Scopes >
 Scope Name   Nested Level   Symbol Name   Symbol Type
------------  ------------  -------------  -----------
x               1             y              int

main            1             a              int
main            1             b              int
main            1             c              int

Error: Invalid function call at line 12 (name : "x")

Checking Types...

Type Checking Finished
```

3의 경우 12번째 줄에 x 라는 변수에서 return x(a, b, c) 부분에서 에러가 발생하는 것을 알 수 있다.