Project #3 Semantic Analysis Report

2022076062 김유찬

project goal

- C-Minus Semantic Analyzer Implementation

TODO

- Scope Analysis : un/redefined variables and functions
- Built-in Functions
- Type Checking

Semantic Analysis 동작 과정

```
#if !NO_ANALYZE
  if (! Error)
  { if (TraceAnalyze) fprintf(listing,"\nBuilding Symbol Table...\n");
    buildSymtab(syntaxTree);
    if (TraceAnalyze) fprintf(listing,"\nChecking Types...\n");
    typeCheck(syntaxTree);
    if (TraceAnalyze) fprintf(listing,"\nType Checking Finished\n");
  }
```

buildSymtab 함수로 syntaxtree를 pre-traverse하면서 symbol table을 만들고 typeCheck 함수를 통해 syntaxtree를 post-travers하면서 타입 체크(에러 체크)를 한다.

BuildSymtab 함수와 typeCheck traverse 구조와 방법

```
void buildSymtab(TreeNode * syntaxTree)
{
  globalScope = insertScope("global",NULL);
  globalScope = st_insert(globalScope, "input", 0, location++, "int");
  globalScope = st_insert(globalScope, "output", 0, location++, "void");
  currentScope = globalScope;
  currentScope = insertScope("input",globalScope);
  currentScope = exitScope(currentScope);
  currentScope = insertScope("output",globalScope);
  currentScope = st_insert(currentScope, "value", 0, location++, "int");
  currentScope = exitScope(currentScope);
  traverse(syntaxTree,insertNode,leaveScope);
  if (TraceAnalyze)
  { fprintf(listing,"\nSymbol table:\n\n");
    printSymTab(globalScope, listing);
  }
}
```

```
static ScopeList globalScope = NULL;
static ScopeList currentScope = NULL;
```

```
void typeCheck(TreeNode *syntaxTree) {
    // Traverse the tree with scope management
    traverse(syntaxTree, nullProc, checkNode);
}
```

- InsertScope : 새로운 Scope를 만들고 현재 Scope의 자식을 만들어 준다.
- st_insert : 현재 Scope에 대한 symbol table에 넣을 Symbol을 넣는다
- exitScope : 현재 Scope에서 나와 부모 Scope로 이동한다
- insertNode : syntax tree를 순회하며 현재 Node를 symbol table에 적절하게 넣어주는 함수이다.
- currentScope를 전역변수로 두고 symbol을 넣을 때마다 currentScope를 바꿔가며 symbol table에 넣는 방식을 사용한다.
- leaveScope : traverse를 하면서 전역 변수인 currentScope를 적절하게 빠지게 해준다.
- built-in functions를 구현하기 위해 input과 output에 대한 정보를 buildSymtab에서 구현한다.

TreeNode 구조체

ScopeList scope 멤버 변수 추가 → 현재 treeNode가 어떤 scope에 있는 지 확인한다.

ScopeList 구조체

```
typedef struct ScopeListRec
   { char * name;
     BucketList buckets[SIZE];
     struct ScopeListRec * parent;
     struct ScopeListRec * children;
     struct ScopeListRec * sibling;
   } * ScopeList;
```

- 전체적인 구조는 트리 구조임
- 부모 scope가 하나의 자식만 가리키고 그 자식이 형제들을 가리켜 부모가 여러 자식을 갖는 것처럼 구현함
- 기존 BucketList구조체가 갖고 있는 배열의 정보를 Scope로 넘겨옴

BucketList, FuncInfo, Param 구조체

```
typedef struct ParamRec
   { char * typestring;
     char * name;
   } * Param;

typedef struct FuncInfoRec
   { char * typestring;
     int param_num;
     Param params[SIZE];
   } * FuncInfo;

typedef struct BucketListRec
   { char * name;
     LineList lines;
     int memloc ; /* memory locati
     char * type;
     struct FuncInfoRec * func;
     struct BucketListRec * next;
     struct ScopeListRec * scope;
   } * BucketList;
```

- BucketList의 type 멤버 변수는 int, int[], void, void[]를 문자열 형태로 가짐
- FuncInfo는 bucket이 함수일 때 갖는 구조체로 이후 FunDK(함수 선언) case일 때 동적할당함
- param_num으로 현재 parameter가 몇 개 있는지 확인할 수 있고, params 배열을 통해 parameter를 관리할 수 있다.
- Param구조체 역시 함수일 때 가질 수 있는 구조체로 ParamK에서 해당 함수 bucket의 func멤버변수의 params에 들어간다.
- param은 typestring, name에 대한 정보를 갖는다

Symtab.h에 있는 함수들

```
SymbolInfo st_lookup_recursive(ScopeList scope, char *name);
char * return_bucket_type(ScopeList scope, char *name);
char * return_bucket_name(ScopeList scope, char *name);
char * return_bucket_type_recursive(ScopeList scope, char *name);
BucketList st_lookup_bucket(ScopeList scope, char * name);
```

- st_lookup_recursive : symbol을 추가할 때 현재 scope에서 정의되지 않았으면 부모 scope에서 타입을 갖고 오기 위한 함수
- return_bucket_type : 현재 scope에서 해당 이름이 갖고 있는 bucket의 type 정보 return
- return_bucket_name : 현재 scope에서 해당 이름이 갖고 있는 bucket의 이름 정보 return
- return_bucket_type_recursive : 현재 scope에서 해당 이름이 갖고 있는 bucket의 type이 없다면 부모 scope까지 올라가 bucket의 type 정보를 return
- st_lookup_bucket : 현재 scope에서 해당 이름이 갖고 있는 bucket 정보 return

insertNode case별(StmtK,ExpK) 작동 방식

우선 전체적으로 t->scope = currentScope를 할당해준다.

StmtK

- CompK
  - func_flag로 해당 compound statement가 함수로 생긴건지 if, while, 일반{}로 생긴건지 확인
  - 만약 if, while, 일반{}로 생긴거라면 새로운 Scope를 생성

- VarK
  - currentScope에 이미 있으면 lineno만 추가하게 symboltable에 삽입
  - currentScope에 없다면 부모에 정의 됐는지 확인 이후 정의 됐으면 그 타입 정보로 삽입하고 없으면 undeclared 타입으로 삽입
  - 이때 처음 넣는거는 나중에 symbol table에 넣는 역할을 하도록 typestring을 “none”으로 설정
- AssignK, RetK, WhileK, SelectK
  - 그냥 지나감

ExpK

- OpK, ConstK, IdK, TypeK
  - ConstK에서만 typestring을 “int”로 설정해주고 나머지는 그냥 지나감
- FuncDK
  - currentScope에 없을 때 → 사실상 globalScope에 없을 때 새로운 scope 생성
  - func_flag = 1로 설정해 이후 CompK가 나와도 그 때 새로운 Scope를 만들지 않도록 함
  - 함수이므로 FuncInfo 구조체 동적 할당 후 초기화
- VarDK
  - currentScope에 없으면 새로 symbol table에 집어 넣고 있으면 기존 것에 넣음
- ParamK
  - 현재 parameter가 어떤 함수의 parameter인지에 대한 정보를 알기 위해 funcName 선언
  - st_lookup_bucket을 통해 bucket정보를 갖고옴
  - 각 parameter들이 ParamK에 들어갈 때마다 params에 차곡차곡 쌓임
  - 이후 currentScope에 없으면 새로, 있으면 기존 것에 symbol table을 집어 넣음
- CallK
  - 함수 호출에 대한 case로 globalScope에 없으면 undeclared 타입으로 넣음
  - 처음 넣는거는 나중에 symbol table에 넣는 역할을 하도록 typestring을 “none”으로 설정
  - globalScope에 있다면 typestring에 대한 정보를 받고 현재 scope에 없으면 symbol table에 새로 추가하고 현재 scope에 있으면 기존 symbol에 추가한다

checkNode case별(StmtK, ExpK) 작동 방식

ExpK

- OpK
  - child[0]과 child[1]의 타입이 둘다 “int”가 아니면 invalid operation 발생
  - 이때 결과 typestring은 에러면 undeclared, int끼리면 int로 바뀐다.
- ConstK, IdK, FunDK,TypeK,ParamK
  - 그냥 지나감
- VarDK
  - symbol table을 뒤져서 void 타입으로 선언 했을 시 void type 에러 발생
  - currentScope에서 또 선언한게 있는 지 확인하기 위해 bucket정보 가져옴
  - 현재 lineno보다 작은 것은 다 재정의 했다는 뜻(Syntax parser 통해)
  - 현재 lineno보다 작은 것들을 출력하며 redefined 에러 출력

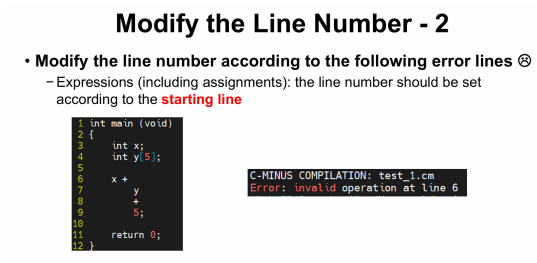
- CallK
  - typestring이 “none”이면 undeclared 됐다는 뜻이고 이는 param에 대한 에러도 자동적으로 발생하므로 두 에러를 동시에 출력
  - typestring이 “undeclared”이면 앞서 “none”으로 symbol table에 추가했다는 작동과 똑같아 지므로 param에 대한 에러만 출력
  - output은 따로 처리를 해줘야하는데 output의 param은 value하나가 있으므로 t→child[0]의 타입이 int가 아니거나 child의 sibling이 존재(param이 2개 이상)하면 invalid function call 에러 출력
  - child가 따로 없어도 에러 출력(output의 param이 없음)
  - t→child가 있으면 param이 있다는 뜻이므로 param에 대한 검사를 해야함
  - param의 child의 sibling들과 func구조체 안에 param배열안에 param 정보들의 타입들을 하나하나 비교해가며 다르면 에러 출력, 마지막에 개수가 달라도 에러 출력

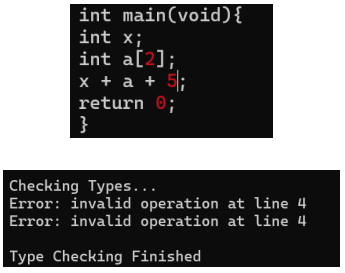
StmtK

- ComK
  - 그냥 지나감
- VarK
  - typestring이 “none”이면 undeclared 됐다는 뜻이고 undeclared에러 출력 이후 symbol table에 넣는 것과 같은 역할
  - “int[]”타입인데 child([]안에 값)가 있으면 child는 int 타입이어야 함
  - int type이 아니라면 indices에러 출력하고 int type이라면 int취급해야하므로 typestring을 int로 바꿈
  - 배열이 아닌 변수가 배열처럼 사용할 때 indexing 에러 출력
- AssignK
  - child[0]과 child[1]의 typestring이 다르면 assign 에러 출력
  - 타입이 같다면 LHS의 typestring도 같은 타입으로 설정
- RetK
  - child가 있을 때 그 typestring이 함수의 typestring과 같아야 하므로 globalScope에서 함수 정보를 찾고 함수의 typestring과 다르면 return type 에러 출력
- WhileK, SelectK
  - child는 조건에 관한 정보이므로 typestring이 “int”가 아니면 condition 에러 출력
  - line정보는 statement를 기준으로 한다

project pdf와 다른 점

- OpK





pdf에서는 해당 연산에 대해 operation error를 한번만 출력한다. 하지만 OpK연산을 할 때 결과에 영향을 주도록 만들었으므로 x + a가 먼저 연산을 하여 operation error출력과 함께 그 결과가 undeclared가 된다. 이후 undeclared와 5(int)가 또 연산을 하여 operation에러를 출력한다.

```
int main(void){
int x;
int a[2];
int z;
z = x + a + 5;
return 0;
}
```

```
Checking Types...
Error: invalid operation at line 5
Error: invalid operation at line 5
Error: invalid assignment at line 5
```

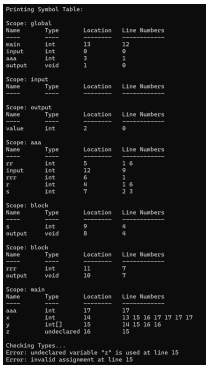
이후 다음과 같은 상황에선 x+a+5 역시 undeclared type이 돼있으므로 z에 assign을 할 때 assign에러가 발생한다.

실행결과

- symbol table은 함수, compound statement에 대한 Scope단위로 출력
- 선언되지 않았으면 undeclared type으로 설정
- pdf 실행 결과는 위에서 언급한 operation error를 제외한 나머지는 동일

```
int aaa(int r, int rr, int rrr){
  int s;
  if(s > 3){
    output(s);
  }
  while(r < rr){
    output(rrr);
  }
  return input();
}

int main(void){
  int x;
  int y[5];
  z = x + y[1];
  x = y[1] + y[2];
  x = aaa(x, x+1, x+2);
  return 0;
}
```



```
Printing Symbol Table:

Scope: global
Name        Type        Location   Line Numbers
----        ----        --------   ------------
main        int         13         12
input       int         0          0
aaa         int         3          1
output      void        1          0

Scope: input
Name        Type        Location   Line Numbers
----        ----        --------   ------------

Scope: output
Name        Type        Location   Line Numbers
----        ----        --------   ------------
value       int         2          0

Scope: aaa
Name        Type        Location   Line Numbers
----        ----        --------   ------------
rr          int         5          1 6
input       int         12         9
rrr         int         6          1
r           int         4          1 4
s           int         7          3 3

Scope: block
Name        Type        Location   Line Numbers
----        ----        --------   ------------
s           int         9          4
output      void        8          4

Scope: block
Name        Type        Location   Line Numbers
----        ----        --------   ------------
rrr         int         11         7
output      void        10         7

Scope: main
Name        Type        Location   Line Numbers
----        ----        --------   ------------
aaa         int         17         17
x           int         14         13 15 16 17 17 17 17
y           int[]       15         14 15 16 16
z           undeclared 16          15

Checking Types...
Error: undeclared variable "z" is used at line 18
Error: invalid assignment at line 15
```

```
int main(void)
{
int x;
int xx;
int t[3];
void k;
x[3] = xx[2];
x = xx;
t[2] = x;
k = k + 3;
z = z + z;
return 0;
}
```

```
Symbol table:

Printing Symbol Table:

Scope: global
Name        Type        Location   Line Numbers
----        ----        --------   ------------
main        int         3          1
input       int         0          0
output      void        1          0

Scope: input
Name        Type        Location   Line Numbers
----        ----        --------   ------------

Scope: output
Name        Type        Location   Line Numbers
----        ----        --------   ------------
value       int         2          0

Scope: main
Name        Type        Location   Line Numbers
----        ----        --------   ------------
k           void        7          6 10 10
t           int[]       6          5 9
x           int         4          3 7 8 9
z           undeclared 8           11 11 11
xx          int         5          4 7 8

Checking Types...
Error: The void-type variable is declared at line 6 (name : "k")
Error: Invalid array indexing at line 7 (name : "x"). indexing can only allowed for int[] variables
Error: invalid array indexing at line 7 (name : "xx"). indexing can only allowed for int[] variables
Error: invalid operation at line 10
Error: invalid assignment at line 10
Error: undeclared variable "z" is used at line 11
Error: invalid operation at line 11
Error: invalid assignment at line 11
```

난관

typestring에 대한 처리를 완벽하게 하지 않아 segmentation fault 자주 발생

ldK가 호출되지 않아 다른 방법으로 ParamK구현

프로젝트 후기

처음에 설계를 잘못해서 segmentation fault가 아주 많이 발생했다. 그럴 때마다 새로운 함수를 만들고 구조를 새로 바꿨다. 그러다보니 거의 다 완성할 때 쯤 에러를 하나 고치면 잘 되던 것도 다시 에러 뜨고, 또 고치면 잘 되던게 에러 뜨고가 무한 반복이라 아예 새로 다 지우고 다시 시작했다. 그래도 기존 것을 최대한 이용하여 빠르게 다시 완성할 수 있었지만, 여전히 제출본에 쓸 데 없는 함수들과 변수들이 남아있어 아쉬움이 있다. 그래도 프로젝트 1,2를 나름 탄탄하게 구현해서 프로젝트 3에만 집중할 수 있었고, 대부분의 testcase 결과가 잘 작동하는 것 같아 잘 마무리 된 것 같다.