[Compiler Design Report]

한양대학교 컴퓨터소프트웨어학부

2021088304 박현준

1. 과제 설명

: Symbol table과 type checker를 이용하여 모든 Semantic Error를 찾아내는 과제

-> Semantic Analyzer가 input source string을 읽어서 AST를 만들고 AST를 traverse 하면서 semantic error와 line number를 찾아서 출력한다.

2. 개발 환경

: Ubuntu 64-bit –Workstation 17.x virtual machine

3. 코드 설명

[analyze.c]

<Params>

```
177         // Parameters
178         case Params:
179         {
180             // Void Parameters: Do Nothing
181             if (t->flag == TRUE) break;
182             // Semantic Error: Void-Type Parameters
183             if (t->type == Void || t->type == VoidArray){
184                 VoidTypeVariableError(t->name, t->lineno);
185             }
186             // Semantic Error: Redefined Variables
187             SymbolRec *symbol = lookupSymbolInCurrentScope(currentScope, t->name);
188             if (symbol != NULL){
189                 RedefinitionError(t->name, t->lineno, symbol);
190             }
191             // Insert New Variable Symbol to Symbol Tablee
192             insertSymbol(currentScope, t->name, t->type, VariableSym, t->lineno, t);
193             break;
194         }
```

: type이 void나 voidArray인 경우 VoidTypeVariableError로 처리하도록 구현하였다. 또한 현재 범위에서 symbol들을 확인해서 재정의된 parameter가 있는 경우 RedefinitionError 함수를 호출해서 변수명이 동일한 경우의 error를 처리하였다.

마지막으로, error가 발생하지 않은 경우 symbol을 삽입하는 insertSymbol 함수를 호출하였습니다.

<VarAccessExpr>

```
216         case VarAccessExpr:
217         {
218             // Semantic Error: Undeclared Variables
219             SymbolRec *symbol = lookupSymbolWithKind(currentScope, t->name, VariableSym);
220             if (symbol == NULL){
221                 // Undeclared Variable Error
222                 symbol = UndeclaredVariableError(currentScope, t);
223             }else{
224                 appendSymbol(currentScope, t->name, t->lineno);
225             }
226             break;
227         }
```

: 현재 scope에서 name과 VariableSym이 같은 symbol을 찾아서 NULL (선언X)인 경우 UndeclaredVariableError를 호출하고, 선언되어 있는 경우 appendSymbol을 호출하여 현재 scope에서 해당 name과 lineno를 갖는 변수를 append 해줍니다.

<CheckNode – WhileStmt>

```
case WhileStmt:
{
    // Error Check
    ERROR_CHECK(t->child[0] != NULL);
    // Semantic Error: Invalid Condition in If/If-Else, While Statement
    if(t->child[0]->type==Integer){
        break;
    }
    InvalidConditionError(t->lineno);
    // Break
    break;
}
```

: 먼저 ERROR_CHECK를 해준 뒤, 조건문의 type에 따라 ConditionError와 break를 구분해주었습니다.

<CheckNode - ReturnStmt>

```
// Return Statement
case ReturnStmt:
{
    // Error Check
    ERROR_CHECK(currentScope->func != NULL);
    // Semantic Error: Invalid Return
    if(t->child[0]==NULL){
        if (currentScope->func->type == Void){
            break;
        }
    }else{
        // Return expression type must match the function return type
        if (t->child[0]->type == currentScope->func->type){
            break;
        }
    }
    InvalidReturnError(t->lineno);
    // Break
    break;
}
```

: Return도 마찬가지로 ERROR_CHECK를 통해 현재 scope가 함수 내부인지 확인해주고, 함수 내부가 아니라면 error를 return합니다. 함수 type이 void인 경우 return을 하지 않기 때문에 break를 해주고, 함수의 type과 return type이 같은지 확인을 해서 type이 다른 경우 InvalidReturnError 함수를 통해 lineno를 반환해줍니다.

<CheckNode - BinOpExpr>

```
case BinOpExpr:
{
    // Error Check
    ERROR_CHECK(t->child[0] != NULL && t->child[1] != NULL);
    // Semantic Error: Invalid Assignment / Operation
    TreeNode* t1=t->child[0];
    TreeNode* t2=t->child[1];

    if((t1->type!=t2->type)||(t1->type==Void || t2->type==Void)){
        if(t->opcode==-1)
            InvalidAssignmentError(t->lineno);
        else
            InvalidOperationError(t->lineno);
        break;
    }
    // Update Node Type
    t->type = t->child[0]->type;
    // Break
    break;
}
```

: 먼저 ERROR_CHECK를 통해 left, right child가 있는지 확인을 해주고 left child를 t1, right child를 t2라고 선언해준 뒤, type를 비교해서 opcode가 -1이면 Assign에 문제가 있다고 판단해서 Assign error를 선언하고, 그렇지 않은 경우 Operation에 error를 선언해줍니다. 이 두 경우에 해당하지 않는 경우 child node에 type을 assign하고 switch문을 종료합니다.

<CheckNode - CallExpr>

```
// Call Expression
case CallExpr:
{
    SymbolRec *calleeSymbol = lookupSymbolWithKind(globalScope, t->name, FunctionSym);

    // Error Check
    ERROR_CHECK(calleeSymbol != NULL);

    // Semantic Error: Call to Undeclared Function
    if (calleeSymbol->state == STATE_UNDECLARED)
    {
        t->type = calleeSymbol->type;
        break;
    }

    // Semantic Error: Invalid Arguments
    TreeNode *paramNode = calleeSymbol->node->child[0];
    TreeNode *argNode = t->child[0];

    int argumentsMatch = 1;

    if (argNode && argNode->type == Void && paramNode->type == Void)
        argumentsMatch = 0;
    else if (argNode == NULL && paramNode->type == Void);
    else if ((argNode == NULL && paramNode->type != Void) || (argNode != NULL && paramNode->type == Void)){
        argumentsMatch = 0;
    }else{
        while (argNode && paramNode){
            if (argNode->type != paramNode->type){
                argumentsMatch = 0;
                break;
            }
            argNode = argNode->sibling;
            paramNode = paramNode->sibling;
            if ((argNode && !paramNode) || (!argNode && paramNode)){
                argumentsMatch = 0;
                break;
            }
        }
    }
    if (argumentsMatch == 0){
        InvalidFunctionCallError(t->name, t->lineno);
        break;
    }
    // Update Node Type
    t->type = calleeSymbol->type;
    // Break
    break;
}
```

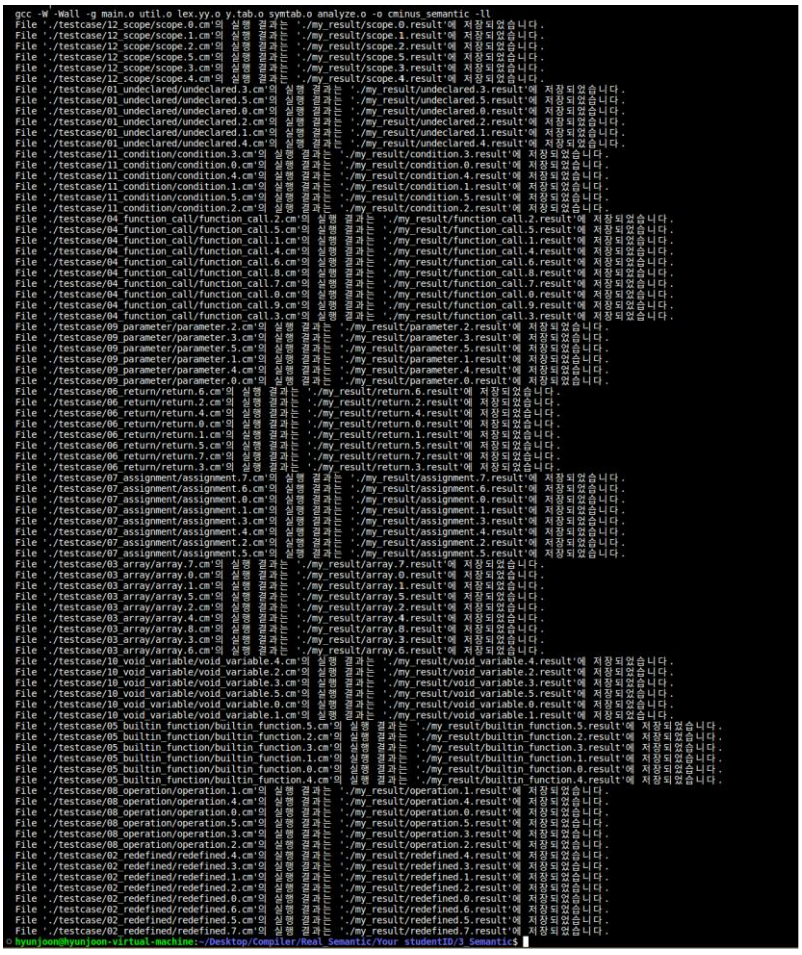
: global scope에서 이름과 functionSym을 가진 symbol을 찾아서 calleeSymbol로 선언해준 뒤, calleeSymbol이 없는 경우를 ERROR 처리해줍니다. 그 다음 calleeSymbol의 state가 선언되지 않은 상태라면, type을 t type으로 선언해줍니다. paramNode와 argNode, argumentsMatch 를 선언해서 if문을 들어가는데, parameter의 수가 매개변수의 수와 일치하지 않거나 Void type인 경우 일치 하지 않는다는 의미로 argumentMatch를 0으로 설정합니다.
while문에서는 paramNode와 argNode가 둘다 NULL이 아닌 경우 루프를 진행하는데, type이 다른 경우 불일치 선언을 하고, 둘 다 sibling node로 이동한 뒤 개수가 같은지 확인해줍니다.
If문과 while문을 돌면서 argumentMatch가 return되었을 때 0이면 error를, 0이 아니면 calleeSymbol의 type을 t type으로 update해주고 case를 종료합니다.

<CheckNode - VarAccessExpr>

```
// Variable Access
case VarAccessExpr:
{
    SymbolRec *symbol = lookupSymbolWithKind(currentScope, t->name, VariableSym);
    // Error Check
    ERROR_CHECK(symbol != NULL);
    // Semantic Error: Access Undeclared Variable - Already Caused
    if (symbol->state == STATE_UNDECLARED)
    {
        t->type = symbol->type;
        break;
    }
    // Array Access or Not
    if (t->child[0] != NULL)
    {
        // Semantic Error: Index to Not Array
        if(symbol->type!=IntegerArray){
            ArrayIndexingError2(t->name, t->lineno);
        }
        // Semantic Error: Index is not Integer in Array Indexing
        else if(t->child[0]->type!=Integer){
            ArrayIndexingError(t->name, t->lineno);
        }
        // Update Node Type
        t->type = Integer;
    }
    // Update Node Type
    else
        t->type = symbol->type;
    // Break
    break;
}
```

: 현재 scope에서 name과 variableSym을 가진 symbol을 찾아서 NULL인지 check를 해주고 만약 선언되지 않은 상태라면, symbol의 type를 t의 type로 update 해줍니다. 그 다음으로, child[0]이 NULL이 아닐 때, symbol의 type이 IntegerArray가 아니라면 IndexingError2, Integer가 아니라면 IndexingError 처리를 합니다. 그 뒤, Node의 type를 Interger로 update해주고, NULL인 경우 type를 update하고 case문을 종료합니다.

4. 결과



testcase_result.sh 파일을 $bash testcase_result.sh로 실행했을 경우 제대로 실행이 되고, my_result 폴더에 결과 파일이 .result 파일로 저장되는 것을 알 수 있다.