Compiler Semantic Analysis Report

학부	컴퓨터소프트웨어학부
학번	2018008977
이름	이종범

Environment

Compiler: clang 14.0.3

Flex: 2.6.4 Apple(flex-34)

OS: Sonoma 14.1

Device: Apple MacBook Pro M1 Max 32GB

How to Run?

- 경로 이동 후 Compile

  $ make all

- 파일 실행

  $ ./cminus_semantic [C-Minus File to Run]

주요 구현 사항

- analyze.c 파일의 스켈레톤 코드를 참고하여, Semantic Analysis에 필요한 checkNode, insertNode 함수를 완성함
- 제공되는 테스트 케이스의 통과 여부를 비교하기 위해, 자동화 테스트 Script를 작성하여 사용

analyze.c 내 checkNode 함수

- Params

```c
// Parameters
case Params:
{
    // Void Parameters: Do Nothing
    if (t->flag == TRUE) break;

    if (t->type == Void || t->type == VoidArray)
    {
        // Semantic Error: Void-Type Parameters
        VoidTypeVariableError(t->name, t->lineno);
        break;
    }

    // Semantic Error: Redefined Variables
    SymbolRec *symbol = lookupSymbolInCurrentScope(currentScope, t->name);
    if (symbol != NULL) RedefinitionError(t->name, t->lineno, symbol);

    // Insert New Variable Symbol to Symbol Table
    insertSymbol(currentScope, t->name, t->type, VariableSym, t->lineno, t);

    break;
}
```

- Void-Type Paramter, Redefined Variables Error을 탐지
- Paramter의 Type을 확인하여 Void 또는 VoidArray인지 확인하고, 맞다면 VoidTypeVariableError를 일으킴
- 현재의 Scope에서 SymbolTable을 검색하고, 동일한 이름의 Symbol이 존재한다면 RedefinedError를 일으킴
- 모든 검사를 통과했다면 Symbol Table에 해당 Symbol을 삽입함

- VarAccessExpr

```
// Variable Access
case VarAccessExpr:
{
    // Semantic Error: Undeclared Variables
    SymbolRec *func = lookupSymbolWithKind(currentScope, t->name, VariableSym);
    if (func == NULL) func = UndeclaredVariableError(currentScope, t);
    // Update Symbol Table Entry
    else
        appendSymbol(currentScope, t->name, t->lineno);

    // Break
    break;
}
```

- Symbol Table에 해당 Variable이 존재하는지 확인하고, 존재한다면 Symbol Table에 Line Number를 추가
- 해당 Variable이 존재하지 않는다면, UndeclaredVariableError를 일으킴

analyze.c 내 checkNode 함수

- IfStmt, WhileStmt

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

    // Break
    break;
}
```

- If 문 또는 While 문 내 조건이 명시되어 있지 않거나, Integer가 아닐 경우 InvalidConditionError를 일으킴

- ReturnStmt

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

- 함수의 Return State가 반환하고자 하는 Expression의 Type과 동일한지 확인하고, 그렇지 않다면 InvalidReturnError를 반환
- 함수의 반환형이 Void일 경우에만 Return Statement의 Expression이 없는 것을 허용

- AssignmentExpr / BinOpExpr

```
// Assignment, Binary Operator Expression
case AssignExpr:
case BinOpExpr:
{
    // Error Check
    ERROR_CHECK(t->child[0] != NULL && t->child[1] != NULL);
    // Semantic Error: Invalid Assignment / Operation
    if (t->child[0] == NULL || t->child[1] == NULL || t->child[0]->type != t->child[1]->type)
        if (t->kind == AssignExpr)
            InvalidAssignmentError(t->lineno);
        else
            InvalidOperationError(t->lineno);
    // Update Node Type
    t->type = t->child[0]->type;
    // Break
    break;
}
```

- Left Child 또는 Right Child가 없거나, 양 Child의 Type이 일치하지 않을 경우에 예외를 일으킴
- AssignExpr일 경우 InvalidAssignmentError를 일으키고, BinOpExpr의 경우 InvalidOperationError를 일으킴

- CallExpr

```
// Call Expression
case CallExpr:
{
    SymbolRec *calleeSymbol = lookupSymbolWithKind(globalScope, t->name, FunctionSym);
    // Error Check
    ERROR_CHECK(calleeSymbol != NULL);
    // Semantic Error: Call Undeclared Function - Already Caused
    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];

if (paramNode->type == Void)
{
    paramNode = NULL;
    if (argNode != NULL && argNode->type == Void)
        argNode = NULL;
}

while (paramNode != NULL && argNode != NULL)
{
    if (paramNode->type != argNode->type)
        InvalidFunctionCallError(t->name, t->lineno);

    paramNode = paramNode->sibling;
    argNode = argNode->sibling;
}

if (paramNode != NULL || argNode != NULL)
    InvalidFunctionCallError(t->name, t->lineno);

// Update Node Type
t->type = calleeSymbol->type;
// Break
break;
}
```

■ Symbol Table 내에 호출하고자 하는 함수가 존재하는지 확인

■ Paramter가 함수가 필요로하는 Parameter와 동일한지 확인

■ paramNode와 argNode의 sibling을 순차적으로 탐색하여 Type이 일치하는지 확인

■ 일치하지 않는다면 InvalidFunctionCallError를 일으킴

■ Paramter의 수가 일치하지 않는다면 InvalidFunctionCallError를 일으킴

- 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
        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;
}
```

■ Symbol Table 내에 해당 Variable이 존재하는지 확인하고, 존재하지 않는다면 Error를 일으킴

- 해당 노드에 Child가 존재하는 경우, Array인 경우로 판단함
- Child가 존재하는 경우, Array인지 확인하고 Array가 아니라면 ArrayIndexingError2를 일으킴
- Array라면, 해당 Child가 Index를 나타내기 위한 Integer Type인지 확인하고, Integer가 아니라면 ArrayIndexingError를 일으킴

자동화 테스트 Script 작성

```
for my_result_file in ./my_result/*
do
    # 파일명 추출
    filename=$(basename "$my_result_file")
    test_result_file="./testresult/$filename"

    # 파일이 존재하는 경우에만 비교 수행
    if [ -e "$test_result_file" ]; then
        echo "CMinus Semantic Test: $filename"
        # 파일 비교
        diff "$my_result_file" "$test_result_file"
        echo "\n"
    else
        echo "파일이 존재하지 않습니다: $test_result_file"
    fi
done
```

- 해당 Project 3에서 testcase_result.sh 파일을 통해 테스트를 자동으로 실행할 수 있는 스크립트를 제공하고 있음
- 하지만, 이는 동작만 자동화된 것으로, 많은 테스트 케이스에 있어 결과를 비교하기에는 오랜 시간이 소요되며, 휴먼 에러가 발생할 수 있을 것으로 판단함
- 이에 자동으로 테스트 결과를 확인할 수 있는 자동화 테스트 Script를 작성함

Test

- $ sh ./compare_test.sh 를 실행하여 테스트 진행
    - 실행 이전 $ sh ./testcase_result.sh 를 실행하여 테스트 결과를 생성해두어야 함
- 테스트 실행 결과 모든 테스트 케이스에 대해서 차이가 없는 것을 확인