**Design and Development of Compiler**

**For C- Language**

**(Phase 3: Design and Implementation of Semantic Analyzer)**

1. **각 단계별 제안서**
2. **각 단계별 결과 보고서**

**(Phase 2: Design and Implementation of Semantic Analyzer)**

\*본 프로젝트는 예시2의 loc 할당 방식을 이용하여 구현되었습니다\*

**과목명: [CSE4120] 기초 컴파일러 구성**

**담당교수: 서강대학교 컴퓨터공학과 정 성 원**

**개발자: 6조 정지원(팀장, 20081638)**

**박경민(20121590)**

**개발기간: 2017. 5. 16 ~ 2017. 6. 02**

**Project3 결과 보 고 서**

**프로젝트 제목: Design and Development of Compiler for C-Language:**

**Phase 3: Design and Implementation of Semantic Analyzer**

**제출일: 2017. 6. 02**

**개발자: 정지원(팀장, 20081638), 박경민(20121590)**

1. **개발 목표**

본 프로젝트의 목표는 Compile 과정의 3번째 과정인 Semantic Analyzer를 구현하는 것입니다. Semantic Analyzer는 다음을 수행합니다. 첫째, hash table을 이용하여 symbol table을 구성합니다. Symbol table을 통해 scope check와 변수 선언 여부 검사, ID의 중복 여부 검사를 수행합니다. 둘째, Semantic Error Check를 수행합니다. 이는 연산에서의 type checking과 함수 호출시의 parameter 일치 여부 등을 검사합니다.

1. **개발 범위 및 내용**
2. **개발 범위**

Semantic Analyzer를 구현하기 위하여, 다음을 개발하여야 한다.

1. [PASS 1] Symbol Table을 구성하고, scope check를 수행
2. [PASS 2] Semantic Error Check를 수행하고, 검출된 Error를 출력
3. **개발 내용**
4. [PASS 1] Symbol Table을 구성하고, scope check를 수행

ID의 정보를 출력하는 symbol table을 구성하고, symbol table을 구성하면서 scope check, duplicate assign check, using not assigned symbol check를 수행한다. 또한, PASS 2에서의 type check를 위해 syntax tree에 typeDecNode를 연결해준다. 이를 통해 다음과 같은 에러 케이스를 검출한다.

* 선언되지 않은 변수나 함수는 사용할 수 없다.
* 변수나 함수, 함수 파라미터 선언 시, 중복된 이름 사용 불가

1. [PASS 2] Semantic Error Check를 수행하고, 검출된 Error를 출력

PASS 1에서 Id에 해당 Id의 선언부 노드의 정보가 추가된 Abstract Syntax Tree를 입력으로 받고, 이를 이용하여 다음의 Semantic Error를 검사한다. 이 때, 오류가 검출되면 적절한 오류메시지를 오류가 발생한 라인 넘버와 함께 출력한다.

* 변수나 parameter 선언 시 void type으로 선언할 수 없다.
* 변수에 값을 assign 하는 경우에는 l-value와 r-value의 type이 같아야 한다.
* 변수가 array인 경우 array index는 integer 값이어야 한다.
* Array가 아닌 변수를 Array처럼 사용하려 하면 안 된다.
* 함수 호출 시 parameter의 개수와 type이 일치하여야 한다.
* 함수 호출 시 호출된 것이 함수가 맞는지 확인하여야 한다.
* 함수의 return 값과 return type이 일치하여야 한다.
* 함수의 return type이 void일 경우에는 return이 없어야 한다.
* Main 함수는 가장 마지막에 선언되어야 한다.
* Main 함수는 반드시 parameter가 존재하지 않는다.
* loop문의 반복 조건을 표시하는 부분에 오는 statement는 integer값 혹은 Boolean 값이어야 한다.

1. **추진 일정 및 개발 방법**
2. **추진 일정**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | 5/16 | 5/17 | 5/18 | 5/19 | 5/20 |
|  |  | 제안서  구상 및 작성 | 제안서  구상 및 작성 | 제안서  구상 및 작성 | 박경민: Semantic Error 정의  정지원:  Base code 작성 및 설계 | 박경민: Semantic Error 정의  정지원:  Base code 작성 및 설계 |
| 5/21 | 5/22 | 5/23 | 5/24 | 5/25 | 5/26 | 5/27 |
| 박경민:  Base Code 작성  정지원:  Symbol table 구현 | 박경민:  Type Check 구현  정지원:  Symbol table 구현 | 박경민:  Type Check 구현  정지원:  Symbol table 구현 | 박경민:  Type Check 구현  정지원:  Symbol table 구현 | 박경민:  Parameter Check 구현  정지원:  Symbol table 출력 형식 수정 | 박경민:  Parameter Check 구현  정지원:  선언되지 않은 변수나 함수 체크 | 박경민:  세부 Error Handle 구현  정지원:  이름 중복 체크 |
| 5/28 | 5/29 | 5/30 | 5/31 | 6/1 | 6/2 |  |
| 박경민:  세부 Error Handle 구현  정지원:  선언되지 않은 변수나 함수 체크 | 코드 통합 및 테스트 | 코드 통합 및 테스트 | 코드 통합 및 테스트 | 결과보고서 작성 | 결과보고서 작성 |  |

1. **개발 방법**

먼저, TextBook의 Code의 분석을 통하여, 해당 코드의 부족한 부분이 무엇인지 확인한다. 수업 시간에 배운 내용과 Lecture Node, Textbook을 통하여 구현할 Semantic Analyzer의 이론을 확인한다.

1. **연구원 역할 분담**

정지원: PASS 1 구현(create symbol table)

박경민: PASS 2 구현(Semantic Error Check)

1. **연구 결과**
2. 합성 내용

* 전체 소프트웨어 구성도

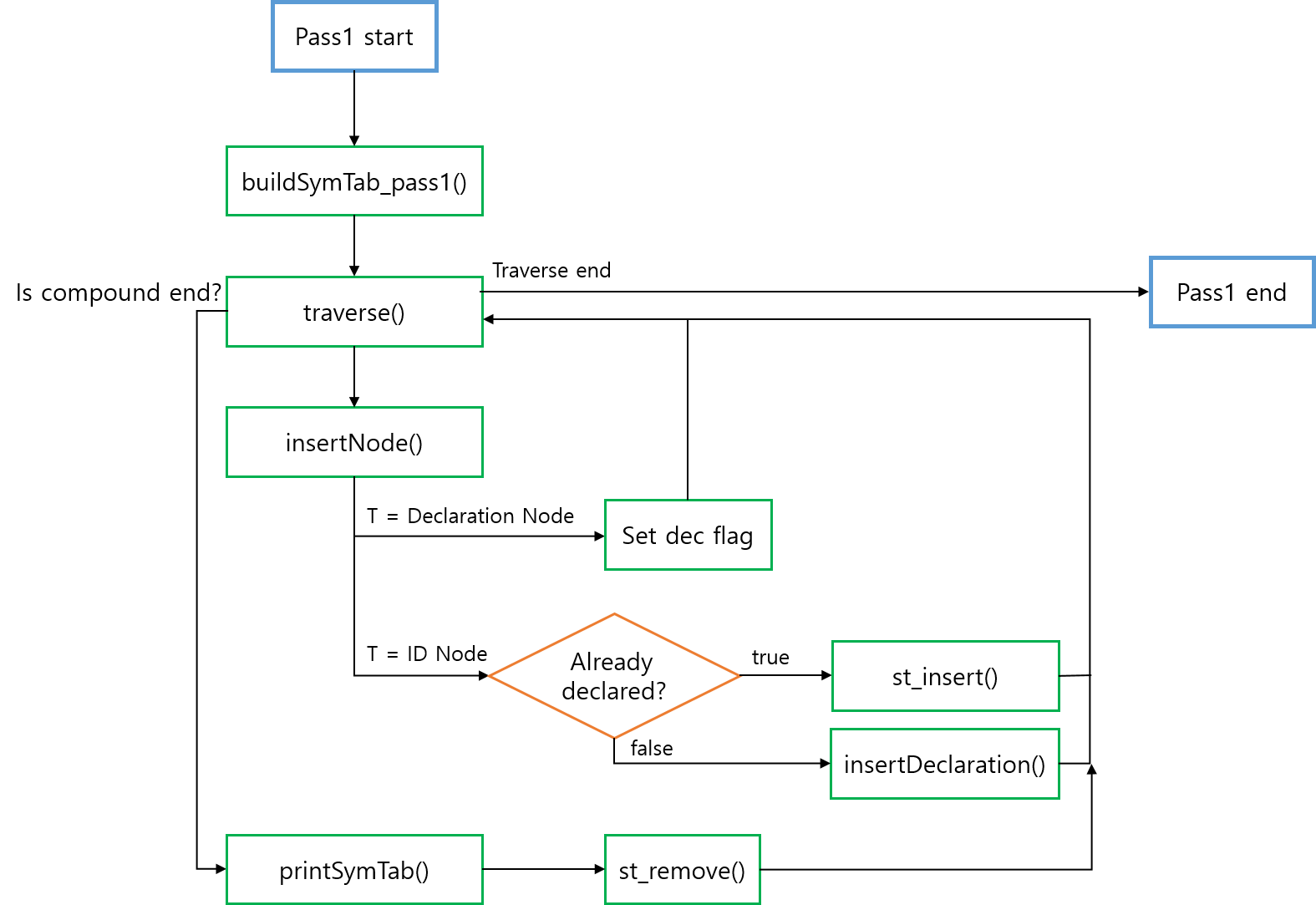


Figure PASS 1 순서도

* 각 부분의 역할 및 구현 방법

1. Main

역할: 사용자가 프로그램의 사용법에 맞게 사용하였는지 확인한다.

구현: 프로그램의 사용법인 “{프로그램 명} {C-Language 소스 코드}”에 맞게 프로그램을 실행하였는지, 실행 시 명시한 C-Language 소스 코드가 존재하는지 확인한 후, 파일 스트림을 열고 parse() 함수를 호출한다. 이를 통해, 소스 코드에 대한 abstract syntax tree를 생성한다. 그 후, buildSymtab\_pass1()을 이용하여 scope check를 수행하고 typeCheck()에서 사용할 정보(id와 해당 선언부 연결)를 생성한다. 그 후, typeCheck()를 이용하여 semantic error를 check한다.

1. PASS 1 **(Symtab.c Conceptual Implementation)**
2. buildSymtab\_pass1  
   역할 : syntax tree를 순회하며 symbol table을 생성하기 위한 함수들을 호출한다.  
   구현 : 프로젝트2에서 구현한 parser를 통해 구성한 syntax tree의 root를 받아 syntax tree 순회하는 traverse 함수를 호출한다.
3. Traverse  
   역할 : syntax tree를 순회하며 symbol table을 구성하기 위한 함수들을 호출한다.  
   구현 : 재귀함수로 구성되어 있으며, syntax tree를 pre order로 순회하도록 구현되어 있다. 하나의 scope가 끝나면 printSymtab을 호출하여 symbol table을 출력한다.
4. insertNode  
   역할 : syntax tree의 현재 node에 대해 symbol handling이 필요한 node인지 판단하고 필요하다면 각각의 함수를 호출하여 symbol table을 구성하도록 한다.  
   구현 : 현재 node의 statement type, expression type을 체크하여 어떤 동작을 해야 할지 결정한다. 선언인 경우, PASS 2에서 Id의 type check를 수행하기 위해 node를 저장한다. Return 문의 경우, Symbol table에서 함수 선언을 찾아 typeDecNode에 연결한다. Id의 경우, symbol table에 해당 id의 정보를 삽입하고 동일한 이름을 가진 node를 찾아 typeDecNode에 연결한다.
5. nullProc  
   역할 : syntax tree 탐색 시, post process에서 아무 동작도 하지 않도록 한다.  
   구현 : 어떠한 동작도 하지 않고 파라미터에 대한 워닝이 발생하지 않도록 구현하였다.
6. st\_create  
   역할 : scope 별 hash table을 생성한다.  
   구현 : ExpandHash structure를 통해 Hash table의 list를 가지고 있을 수 있도록 구현하였다. 각각의 hash table에 해당 symbol의 정보를 저장할 수 있도록 hash table을 생성하여 list에 연결한다.
7. removeHash  
   역할 : scope를 벗어난 hash table을 제거한다.  
   구현 : ExpandHash node를 제거하기 전에 hash table 먼저 free 해주도록 구현하였다.
8. st\_remove  
   역할 : scope를 벗어난 hash table을 제거한다.  
   구현 : ExpandHash node를 제거하기 전에 removeHash 함수를 호출한 후, ExpandHash를 free하는 형태로 구현하였다.
9. st\_insert  
   역할 : symbol table에 해당 symbol의 정보를 저장한다.  
   구현 : ExpandHash list를 가장 가까운 scope의 hash table부터 가장 먼 scope의 hash table까지 탐색하며 동일한 이름을 가진 symbol을 찾는다.
10. St\_lookup  
    역할 : symbol table을 탐색하여 location을 반환한다.  
    구현 : st\_insert와 마찬가지 방식으로 symbol table을 탐색하여 해당 symbol의 location을 반환하도록 구현하였다. 해당하는 symbol이 존재하지 않으면, -1을 반환한다.
11. findDeclaration  
    역할 : DeclarationList에서 해당하는 이름의 선언을 찾는다. 찾은 node는 중복 체크와 typeDecNode 연결에 사용한다.  
    구현 : DeclarationList를 탐색하여 해당 이름을 가진 node가 있으면 TreeNode \*를, 없으면 NULL을 반환한다.
12. insertDeclarationList  
    역할 : 변수 및 함수 선언 시, DeclarationList에 저장한다. 이때, 중복체크도 함께 진행한다.  
    구현 : VarDecK, FunDecK, ParaDecK에 해당하는 node를 받아서 DeclarationList에 저장한다. 이 과정에서 DeclarationList에 같은 이름의 node가 이미 존재한다면, 에러를 출력한다.
13. printSymtab  
    역할 : 생성된 각 scope의 symbol table을 출력한다.  
    구현 : 생성된 symbol table을 탐색하여 symbol들의 정보를 출력한다. 가장 최근 scope에 해당하는 첫 번째 symbol table만 출력하도록 구현하였다.
14. PASS 2 **(analyze.c conceptual implementation)**
    1. typeCheck

역할: Semantic Error를 Check하는 checknode를 syntax tree를 traverse하며 수행하도록 traverse()를 호출한다.

구현: typeCheck는 Abstract Syntax Tree의 root 노드를 입력으로 받는다. 이를 각 노드에서 semantic error를 체크하는 checkNode()함수, 아무것도 수행하지 않는 nullProc()함수와 함께 parameter로 하여 traverse()함수를 호출한다. 이를 통해 abstract syntax tree의 모든 노드에서 semantic error를 검사한다.

* 1. traverse

역할: Abstract Syntax Tree를 traverse하며 post order의 순서로 checkNode를 수행한다.

구현: traverse는 세 개의 parameter를 입력으로 받는다. 1) 트리의 루트노드 t. 2) pre-order의 순서로 수행할 함수 preProc. 3) post-order의 순서로 수행할 함수 postProc. PASS 2에서는 preProc에는 nullProc(아무 수행이 없는 함수)가 들어가고, postProc에는 checkNode가 들어간다.

수행 순서는 먼저 preProc을 수행하고, 모든 자식 노드에 대해서 traverse를 재귀적으로 호출한다. 그 후, postProc을 수행하고, 자신의 sibling 노드에 대해서 traverse를 재귀적으로 호출한다.

* 1. typeError

역할: typeError에 대해서, 오류 메시지를 출력한다.

구현: typeError가 발생한 노드의 lineno(라인 넘버)와 오류메시지를 fprintf를 이용하여 출력한다.

* 1. checkNode

역할: 입력받은 노드에 대해서 Semantic Error가 있는지 검사한다.

구현: 노드의 nodekind를 분석하여 해당 노드가 어떤 타입의 노드인지를 구분한다. 노드의 종류에는 다음이 있다. 1) Parameter Declaration, 2) Parameter Array Declaration, 3) Variable Declaration, 4) Array Declaration, 5) Function Declaration, 6) Compound Statement, 7) Selection Statement, 8) Iteration Statement, 9) Return Statement, 10) Operator, 11) Constant, 12) Identifier, 13) Type, 14) Assignment, 15) Function Call, 16) Array Access. 노드에는 위와 같은 총 16가지의 종류가 있다.

위의 종류의 노드에 따라 Semantic Rule에 의한 Error Check를 수행한다. 각 종류에 따라 다음과 같은 검사를 수행한다.

1. Parameter Declaration, Parameter Array Declaration, Variable Declaration, Array Declaration
   1. Void 타입으로 변수와 배열을 선언할 수 없다.
2. Function Declaration
   1. Main 함수는 가장 마지막에 선언되어야 한다.
   2. Main 함수의 return type은 void이어야 한다.
   3. Main 함수는 parameter를 가질 수 없다.
3. Compound Statement
   1. 검사할 사항이 없다.
4. Selection Statement, Iteration Statement
   1. 반복 조건으로 오는 statement는 integer나 Boolean 형이어야 한다.
5. Return Statement
   1. 반환형이 void인 함수는 return문이 없어야 한다.
   2. 반환하는 statement의 타입과 선언부의 반환타입은 일치하여야 한다.
6. Operator
   1. Operand는 Integer형이어야 한다.
7. Constant, Identifier, Type
   1. 검사할 사항 없음
8. Assignment
   1. L-value는 변수, 배열, parameter 중 하나이어야 한다.
   2. L-value와 R-value의 타입은 같아야 한다. 이 때 같은 타입은 타입의 이름이 같음을 의미한다. (name-equivalence)
9. Function Call
   1. 함수 이름이 아닌 identifier를 함수 호출에 사용할 수 없다.
   2. Formal Parameter와 Actual Parameter는 같은 개수와 같은 타입을 가져야 한다.
10. Array Accessing
    1. Array가 아닌 Identifier를 이용하여 배열 연산을 하면 안 된다.
    2. Array의 Index는 Integer형이어야 한다.

- 사용된 툴(Tool)

Lexical Analyzer를 생성하는데 사용되는 공개소프트웨어 LEX와, Parser를 생성하는데 사용되는 공개소프트웨어 Bison을 사용하였다. 작업환경은 학교 cspro9서버이고, 컴파일러는 gcc를 사용하였다.

* 사용자 매뉴얼

Usage: ./program\_name <file\_name>

Ex) ./20121590 test1.tny

프로그램의 사용법은 C-Language로 작성된 소스 코드의 파일명을 <file\_name>자리에 입력하고, 프로그램의 이름을 program\_name자리에 입력하고, Enter를 누릅니다.

1. 분석 내용
2. Main

Main함수의 input인 argc의 값이 2가 아닐 경우, argv[1]의 파일명의 파일이 존재하지 않을 경우를 각각 비교연산자와 fopen함수의 반환값(파일이 존재하지 않을 경우 NULL을 반환)으로 확인한다. 그리고 parse 함수를 호출하여, abstract syntax tree를 생성한다. 이를 매개변수로 하여 buildSymtab\_pass1을 호출하여 pass1을 수행하고, typeCheck를 호출하여 pass2를 수행한다.

1. PASS 1
2. buildSymtab\_pass1  
   프로젝트2에서 구현한 parser를 통해 구성한 syntax tree의 root를 받아 syntax tree 순회하는 traverse 함수를 호출한다. Traverse 함수가 종료되면 declarationList를 remove한다.
3. Traverse  
   재귀 호출을 통해 syntax tree를 pre order로 순회하며 symbol table을 구성하기 위한 함수들을 호출한다. 하나의 scope가 끝나면 printSymtab을 호출하여 symbol table을 출력한다. Parameter가 함수의 scope의 바깥에 존재하는 것을 막기 위해 parameter의 경우 따로 저장해두었다가 compound statement node를 만나면 탐색을 진행하는 처리가 되어있다.
4. insertNode  
   syntax tree의 현재 node에 대해 symbol handling이 필요한 node인지 판단하고 필요하다면 현재 node의 statement type, expression type을 체크하여 다음과 같은 함수를 호출하는 것으로 symbol table을 구성하도록 한다.
   1. VarDecK, ArrDecK, ParaDecK, ParaArrDecK, FunDecK의 경우, PASS 2에서 Id의 type check를 수행하기 위한 node를 저장할 수 있도록 flag를 세팅한다.
   2. ReturnK의 경우, global variable로 지정된 함수 선언 node를 typeDecNode에 연결한다.
   3. IdK의 경우, symbol table에 해당 id의 정보를 삽입하고 symbol table로부터 동일한 이름을 가진 node를 찾아 typeDecNode에 연결한다.
5. nullProc  
   어떠한 동작도 하지 않고 파라미터에 대한 warning이 발생하지 않도록 한다.
6. st\_create  
   ExpandHash structure를 통해 Hash table의 list를 가지고 있을 수 있도록 각각의 hash table에 해당 scope 별 symbol의 정보를 저장할 수 있도록 hash table을 생성하여 list에 연결한다.
7. removeHash  
   scope를 벗어난 hash table을 제거한다. ExpandHash node와 hash table 모두 free해주도록 한다.
8. st\_insert  
   ExpandHash list를 가장 가까운 scope의 hash table부터 가장 먼 scope의 hash table까지 탐색하며 동일한 이름을 가진 symbol을 찾는다. 동일한 이름의 symbol이 존재하지 않으면 새 BucketList를 생성하여 hash table에 넣어준다. 동일한 이름의 symbol이 존재하는 경우, 해당 symbol에 line number를 추가한다.
9. St\_lookup  
   st\_insert와 같은 방식으로 symbol table을 탐색하여 해당 symbol의 location을 반환한다. 해당하는 symbol이 존재하지 않으면, -1을 반환한다.
10. printSymtab  
    생성된 symbol table을 탐색하여 symbol들의 정보를 출력한다. 가장 최근 scope에 해당하는 첫 번째 symbol table만 출력해야하므로, ExpandHashList의 첫번째 hash table만 출력한다.
11. PASS 2
    1. typeCheck

typeCheck는 Semantic Error Check를 수행할 Abstract Syntax Tree를 입력으로 받아, traverse함수에 nullProc과 checkNode와 함께 매개변수로 전달한다.

* 1. Traverse

Traverse는 재귀적으로 호출되는 함수이다. 먼저, 종료조건인 입력 받은 노드 t가 null인지 확인한다. 그 후, null이면 함수를 종료한다. 아니라면, 먼저 t의 모든 자식에 대해서 traverse함수를 호출하고, t에 대해서 checkNode를 호출한다. 그 후, t->sibling에 대해서 traverse함수를 호출한다.

* 1. typeError

typeError는 자신을 호출한 노드 t와 오류 메시지 message를 입력으로 받는다. T->lineno를 통해서 해당 error가 발생한 코드의 lineno를 출력하고, message를 이용하여 오류 메시지를 출력한다. 이는 fprintf를 호출함으로써 수행된다.

* 1. checkNode

checkNode는 Semantic Error를 검사할 노드 t를 입력으로 받는다. T->nodekind를 통해서 해당 노드가 Statement인지 Expression인지 확인한다. 그 후, Statement라면 t->kind.stmt를 이용하고, Expression이라면 t->kind.exp를 이용하여 해당 노드의 종류를 확인한다. 그 후, 각 노드의 종류에 따라 다음의 과정을 수행한다.

1. Parameter Declaration, Parameter Array Declaration, Variable Declaration, Array Declaration
   1. Void 타입으로 변수와 배열을 선언할 수 없다.

t->child[0]는 해당 선언의 type을 가리키고 있다. t->child[0]->attr.op를 확인하여 해당 type이 void인지 확인하고, 만약 그렇다면 “Parameter, Variable or Array Declare in type Void”라는 오류 메시지를 출력한다.

1. Function Declaration
   1. Main 함수는 가장 마지막에 선언되어야 한다.

전역변수로 선언되어 있는 findMainFlag는 처음에 False값을 가지고 있다. 이는 “main”이라는 이름을 가진 Function Declaration을 발견 시 True로 바뀐다. 이때 findMainFlag가 true이면서 다른 함수의 선언에 의해서 checkNode에 Function Declaration노드가 입력으로 들어온다면, 이는 main함수가 가장 마지막에 선언되는 것이 아니므로, “main function must be declared at the end”라는 오류 메시지를 출력한다.

* 1. Main 함수의 return type은 void이어야 한다.

입력 노드 t의 child[0]->attr.name은 선언하는 함수의 이름이 저장되어 있다. 이름이 main인지 확인한 후 맞다면, findMainFalg에 True를 대입한다. 그 후, 함수의 반환 타입을 나타내는 t->child[0]->attr.op를 확인하여 Void가 아니라면 “function main’s return type must be void”라는 오류 메시지를 출력한다.

* 1. Main 함수는 parameter를 가질 수 없다.

입력 노드 t의 child[2]는 parameter를 가리킨다. 이 때, t->child[2]가 NULL 혹은 t->child[2]->attr.op가 void가 아니라면, 이는 main 함수가 parameter를 가지고 있다는 것이다. 이 때는 “function main can’t have any parameter”라는 오류 메시지를 출력한다.

1. Compound Statement
   1. 검사할 사항이 없다.
2. Selection Statement, Iteration Statement
   1. 반복 조건으로 오는 statement는 integer나 Boolean 형이어야 한다.

입력 노드 t의 child[0]는 반복 조건으로 오는 statement를 가리킨다. 이 때, t->child[0]->type은 해당 statement의 결과의 type을 나타내는데, 이것이 Integer 혹은 Boolean이 아니라면, “Condition is not both Integer and Boolean”이라는 오류 메시지를 출력한다.

1. Return Statement
   1. 반환형이 void인 함수는 return문이 없어야 한다.

입력 노드 t의 typeDecNode는 해당 Return Statement가 있는 함수의 선언문 노드를 가리킨다. 해당 함수의 반환형을 나타내는 t->typeDecNode->child[0]->attr.op를 확인하여 Void일 경우에는 “Function which of return type is void can’t have return statement”라는 오류 메시지를 출력한다.

* 1. 반환하는 statement의 타입과 선언부의 반환타입은 일치하여야 한다.

입력 노드 t의 child[0]는 return하고자 하는 statement를 나타내는데, t->child[0]가 NULL이 아니면서 Integer형이 아니라면 “Return Value must be integer type”이라는 오류메시지를 출력한다. 이는 C—에서는 Return을 가질 수 있는 함수의 반환형은 INT밖에 없기 때문이다.

1. Operator
   1. Operand는 Integer형이어야 한다.

입력 노드 t의 child[0]와 child[1]은 각각 operand를 가리킨다. t->child[0]->type과 t->child[1]->type을 검사하여, 이 때 둘 중 하나라도 Integer형이 아니라면 “Op applied to non-integer”라는 오류 메시지를 출력한다.

Operator가 사칙연산이라면, t->type에 Integer를 넣고, 비교연산자라면 t->type에 Boolean을 넣는다.

1. Constant, Identifier, Type

Identifier의 경우, 해당 Identifier의 선언문의 종류를 살펴보고, 해당 선언이 Variable, Parameter이면 t->type에 Integer를 대입하고, Array, Parameter Array, Function이면 Void를 대입한다.

1. Assignment
   1. L-value는 변수, 배열, parameter 중 하나이어야 한다.

입력 노드 t의 child[0]는 L-value를 가리킨다. t->child[0]->kind.stmt를 이용하여 l-value가 Identifier인지 Array Accessing인지 확인한다. Identifier라면 t->child[0] ->typeDecNode를 이용하고, Array Accessing이라면 t->child[0]->child[0]->typeDecNode를 이용하여 l-value의 선언문을 접근한다. 그 후, 해당 선언문의 kind.stmt를 이용하여, 어떤 선언문인지 확인하고, 만약 Parameter Array Declaration, Array Declaration, Parameter Declaration, Variable Declaration 중 하나에 속하지 않는다면, “Assign applied to non-variable”이라는 오류 메시지를 출력한다.

* 1. L-value와 R-value의 타입은 같아야 한다. 이 때 같은 타입은 타입의 이름이 같음을 의미한다. (name-equivalence)

t->child[0]->type이 Integer가 아니라면, l-value는 Variable이 아니라는 뜻이므로, “L-value is not variable”을 출력하고, 만약 t->child[0]->type이 Integer이고 t->child[1]->type이 Integer가 아니라면 이는 l-value와 r-value의 타입이 다르다는 뜻이므로, “L-value’s type is not same as R-value’s type”이라는 오류 메시지를 출력한다.

1. Function Call
   1. 함수 이름이 아닌 identifier를 함수 호출에 사용할 수 없다.

입력 노드 t의 child[0]->typeDecNode를 확인하여 해당 Identifier의 선언문을 확인한다. 선언문의 kind.stmt를 확인하여 Function Declaration이 아니라면 “Function Call by non-function”이라는 오류 메시지를 출력한다.

* 1. Formal Parameter와 Actual Parameter는 같은 개수와 같은 타입을 가져야 한다.

이는 ParameterChecking이라는 static 함수를 호출함으로써 확인한다. 이 함수에는 입력 노드 t와 함수 선언문의 child[1](Formal Parameter), t->child[1](Actual Parameter)를 parameter로 전달한다.

ParameterChecking함수에서는 formal parameter와 actual parameter 둘 중 하나가 NULL이 아닐 때까지 sibling을 대입하며(다음 parameter로 넘어가며) 두 parameter의 타입을 비교한다. 오류메시지가 출력되는 상황은 다음이 있다.

formal parameter가 variable parameter인데, actual parameter의 타입이 integer가 아닐 경우

“Formal Parameter is integer variable, but Actual Parameter is not integer”

formal parameter가 array parameter인데, actual parameter가 array의 이름이 아닌 경우

“Formal Parameter is Array, but Actual Parameter is not Array’s Name”

formal parameter와 actual parameter 둘 중 하나가 NULL이라서 반복문이 종료되었는데, 둘 중 하나가 NULL이 아닐 때. 이는 두 parameter의 개수가 다르다는 뜻.

“Formal Parameter’s number is not same as Actual Parameter’s number”

1. Array Accessing
   1. Array가 아닌 Identifier를 이용하여 배열 연산을 하면 안 된다.

입력 노드 t의 child[0]->typeDecNode를 확인하여, 해당 array접근에 사용되는 identifier가 선언된 선언문을 확인한다. 이 때 선언문의 kind.stmt가 Parameter Array Declaration이나 Array Declaration이 아니라면 “Array expression is applied to non-array name”이라는 오류메시지를 출력한다.

* 1. Array의 Index는 Integer형이어야 한다.

입력 노드 t의 child[1]->type을 확인하여, 해당 array접근에 사용되는 index를 확인한다. 이 때 index의 type이 integer가 아니라면, “Array index is not Integer”라는 오류 메시지를 출력한다.

1. 제작 내용



Figure 수행 command

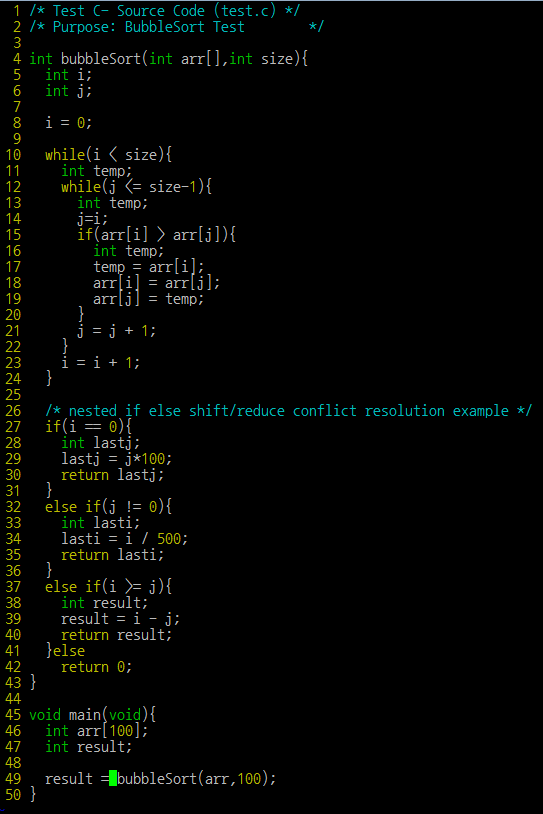


Figure C—language test code(test.c)

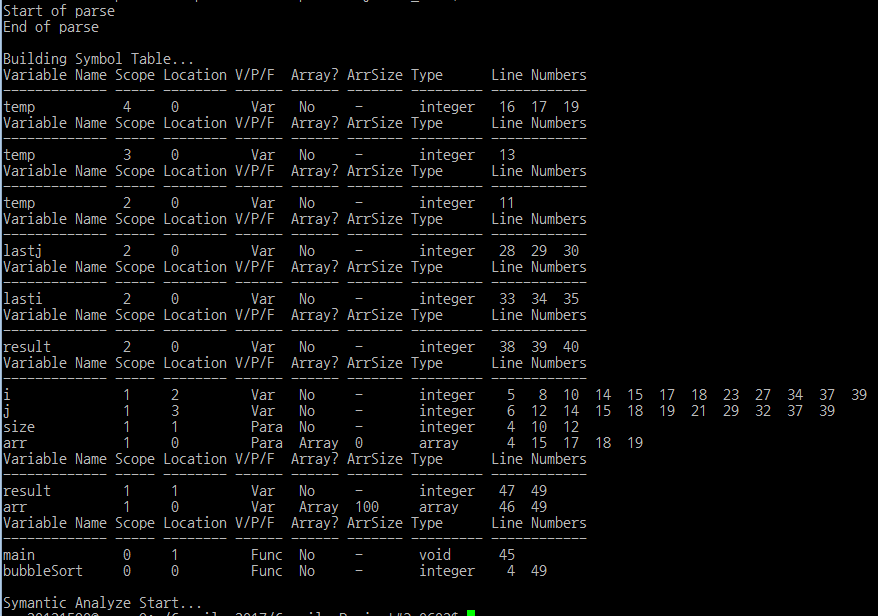


Figure 수행 결과

* 결과 분석

소스코드에 의해 생성된 Symbol Table을 보면, 각 Scope에 대해서 적절한 처리가 되어 있음을 알 수 있다. 또한, nested된 scope에서 temp를 계속 선언하였음에도, 에러 없이 잘 처리 되었고, 또한 가장 안쪽 scope의 temp로 접근하는 것 또한 잘 처리되었음을 scope 4의 temp에 line number가 증가하는 것을 보고 알 수 있다.

1. 시험 내용

- 평가 기준 #1 – 반복된 수행 시 동일한 결과 출력

위의 테스트 파일에 대한 프로그램의 수행을 10번 반복한 결과 매번 동일한 결과를 출력하였다.

- 평가 기준 #2 – PASS1에서 오류 검출 시 PASS2 미 수행

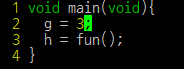


Figure Semantic Error가 있는 C—Code

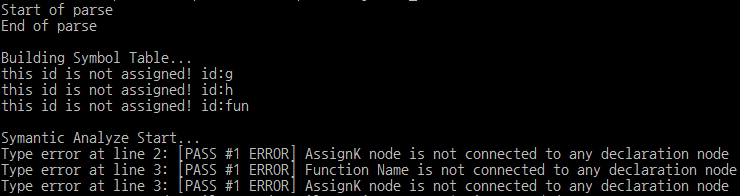


Figure 변경 전 수행 결과

PASS1에서 Scope Error나 Semantic Error가 발견되었을 경우 PASS2가 수행되면, PASS2의 결과가 PASS1의 에러 때문인지, 혹은 다른 이유의 Error때문인지 혼란을 가질 수 있다. 이를 방지하기 위해서 PASS1에서 Error가 발생할 경우 main의 Error Flag를 TRUE로 설정하고, 이를 PASS2를 수행하기 위한 조건으로 설정하였다.

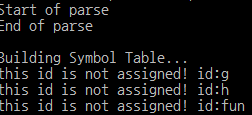


Figure 변경 후 수행 결과

1. 평가 내용

- 안정성

프로그램 실행 시 사용자의 적절한 입력인지를 검사하여 잘못된 입력일 경우 사용법(Usage)를 출력하고 프로그램을 종료한다. 또한, semantic error가 발생할 경우 적절한 오류 메시지를 출력하여, 사용자가 자신의 소스 코드를 고칠 수 있도록 하여, 안정성을 향상시킨다.

- 신뢰성

동일한 입력 파일에 대한 반복수행에도 동일한 결과를 출력한다. 이를 통해 프로그램의 신뢰성이 높음을 알 수 있다.

V. 기타

가. 자체 평가

프로젝트를 진행함에 있어, 제안서에서 기술하였던 프로젝트 추진 일정에 맞게 프로젝트를 단계적으로 진행하였습니다. 또한, textbook에서 제공하는 frame code를 분석하여, 이번 프로젝트에 필요한 부분을 제외한 코드를 제외하여, 간결한 프로젝트 코드를 완성하였습니다.

나. 느낀 점

수업 시간에 배운 Semantic Analyzer를 직접 구현해 봄으로써 컴파일러의 수행 원리를 좀 더 잘 이해할 수 있게 되었습니다.