**Design and Development of Compiler**

**For C-Language**

**Phase3: Design and Implementation of Semantic Analyzer**

**Result**

과목명: [CSE4120] 기초 컴파일러 구성

담당교수: 서강대학교 컴퓨터공학과 정 성 원

개발자: 12조

팀장 박상욱(20141522)

이예진(20141564)

개발기간: 2017.05.16 – 2017.06.02

**Project3 결과보고서**

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

**Phase3: Design and Implementation of Semantic Analyzer**

제출일 2017. 06. 02

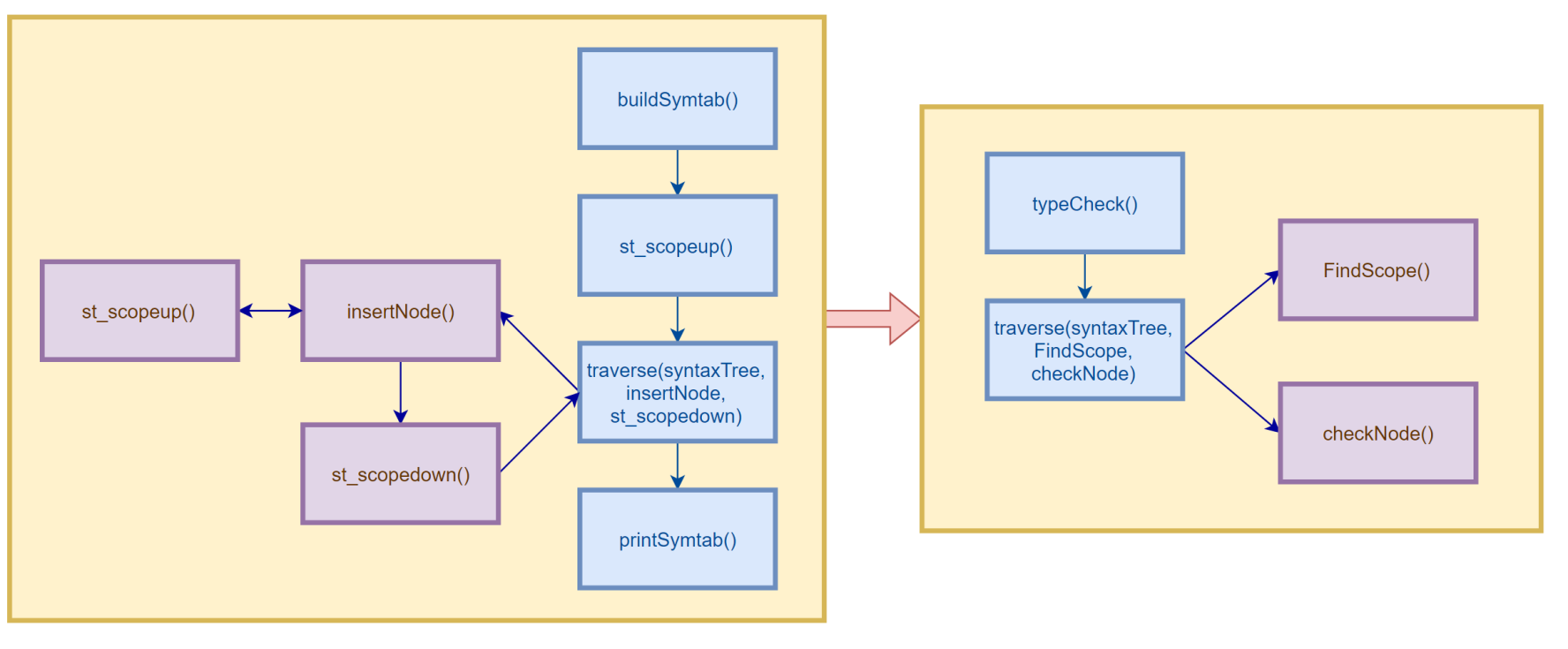
4조 박상욱(20141522), 이예진(20141564)

1. **개발 목표**

단계별 제안서 1. 개발 목표 참조

1. **개발 범위 및 내용**

단계별 제안서 2. 개발 범위 및 내용 참조

1. **추진 일정 및 개발 방법**
2. **연구 결과**
   1. **합성 내용**
      1.  전체 소프트웨어 구성도
      2. 각 부분의 역할 및 사용 기술  
         이번 프로젝트에서는 semantic analyzer를 구현하는데, 크게 전체 코드에 대한 symbol table을 작성과 scoping checking하는 부분, 작성된 symbol table을 탐색하며 type checking을 하는 부분으로 나눌 수 있다.
3. 우선, symbol table을 생성하는 단계는 buildSymtab이라는 함수를 통해서 진행된다. 처음에 buildSymtab이 호출되게 되면 현재 아무런 symbol table이 없는 상태이므로 st\_scopeup이라는 함수를 통해 symbol table을 생성해주고 필요한 초기 작업들을 수행해준다.

그리고 난 뒤, traverse라는 트리 탐색 함수를 호출해주는데 이때 필요한 함수 포인터를 같이 넘겨주어 필요한 연산을 할 수 있도록 해준다. Symbol table을 만드는 단계에서는 traverse하면서 만나는 노드들을 타입에 따라 symbol table에 삽입해주어야 하므로 insertNode함수와 scope에 따른 각각의 symbol table을 만들기 위해 필요한 scope level을 줄여주는 st\_scopedown함수를 parameter로 같이 넘겨준다.

traverse함수에서는 tree를 순회하면서 parameter로 넘어온 함수들을 적절히 사용하여 symbol table을 생성해준다.

1. Type checking 을 진행하는 부분은 symbol table 생성 단계가 모두 완료된 후에 수행된다. TypeCheck라는 함수가 호출되면 다시 한번 AST를 탐색하면서 주어진 Symbol Table을 참고하여 type을 체크한다.

TypeCheck 함수에서는 traverse를 다시 한번 호출해주는 데 이 단계에서는 현재 scope에 연결되어 있는 하위 scope으로 이동시켜주는 함수 FindScope과 type을 체크해주는 checkNode라는 함수를 parameter로 넘겨준다.

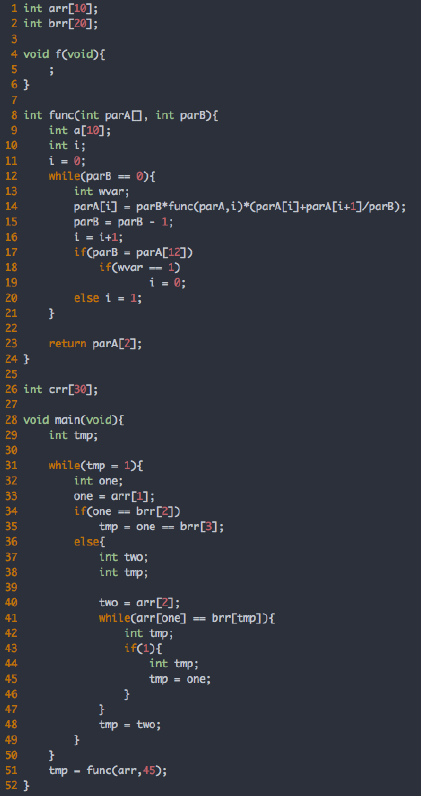
traverse함수에서는 tree를 순회하면서 parameter로 넘어온 함수들을 적적히 사용하여 type을 checking해준다.

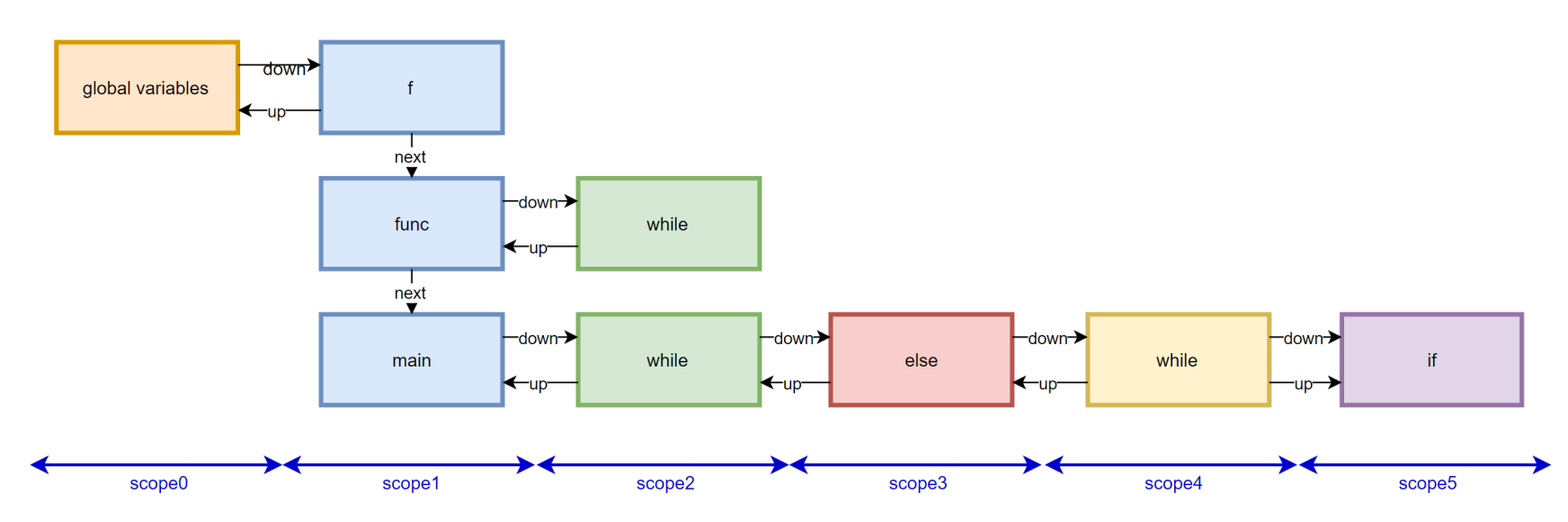
* + 1. 메뉴얼

1. 해당 프로젝트 폴더에 들어가서 make를 입력하여 실행 파일을 생성한다.
2. ./20141522 test1.c 와 같이 ./(실행파일 이름) (실행할 파일)을 입력하여 test를 진행한다.
3. 오류가 없는 파일이라면 symbol table이 출력되고, 오류가 있는 파일일 경우 에러가 발생한 줄 번호와 에러 내용이 출력된다.
4. Test가 끝난 후에는 make clean을 입력하여 실행파일과 오브젝트 파일 등 컴파일 후 생성되는 파일들을 정리 할 수 있다.  
   1. **분석 내용**

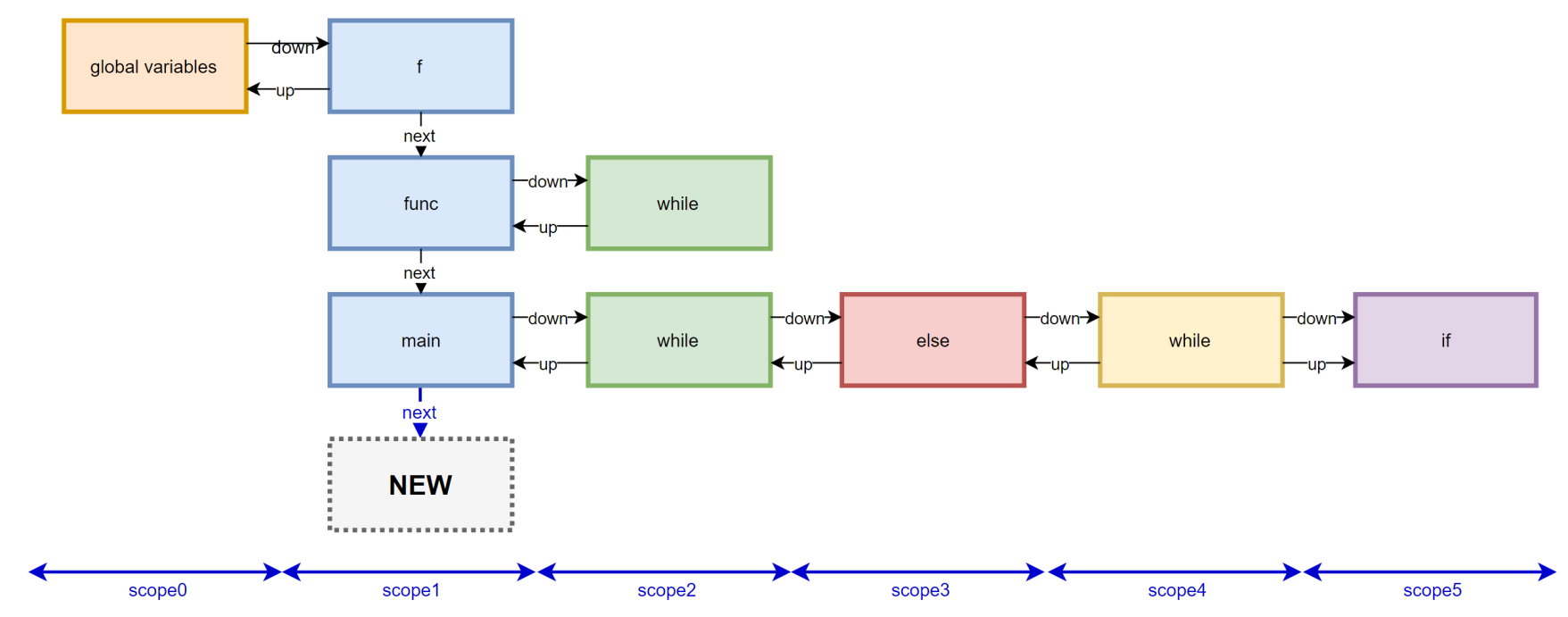
앞서 설명한 바와 같이 이번 프로젝트는 크게 전체 코드에 대한 symbol table을 작성하는 부분과 scoping check을 하는 부분, 작성된 symbol table을 탐색하며 type checking을 하는 부분으로 나눌 수 있다.

1. **Symbol Table 작성 및 scoping check (analyze.c + symtab.c)**

**본 프로젝트에서 symbol table은 다음 형태와 같이 제작하였다.**

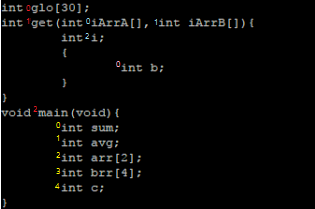
**위의 코드를 예시로 생성되는 symbol table의 형태는 다음과 같다.**

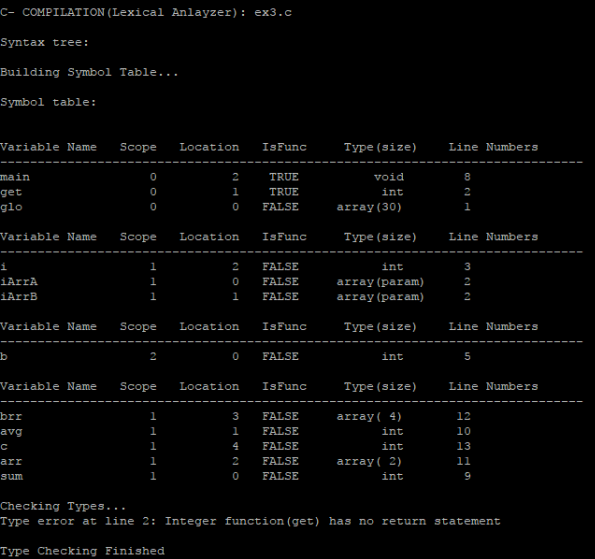
**예를 들어 scope1의 main의 경우 main 함수 안에 while문이 등장하게 된다. 이처럼 현재 scope안에서 또 다른 compound statement가 등장했을 경우 하위 단계의 scope은 down으로 연결되고, 상위 단계의 scope은 up으로 연결된다.**

**또한, 같은 scope에 있는 symbol table들은 next pointer로 연결된다. 이때 같은 scope level을 갖는 scope이 새로 등장했을 경우 다음과 같이 현재 scope에서 next로 연결된 리스트의 마지막에 추가된다.**

**또한 트리를 탐색하면서 노드별로 symbol table에 insert가 진행될 경우 location을 계산해주게 된다.**

**본 프로젝트에서 사용한 location 계산 방식은 다음과 같다. 다음 예시 코드에 대한 location 계산 결과는 다음과 같다.**

****

****

**같은 scope level 에 해당하는 경우, 코드 상에서 등장 순서대로 location값이 0부터 1씩 증가하면서 할당된다.**

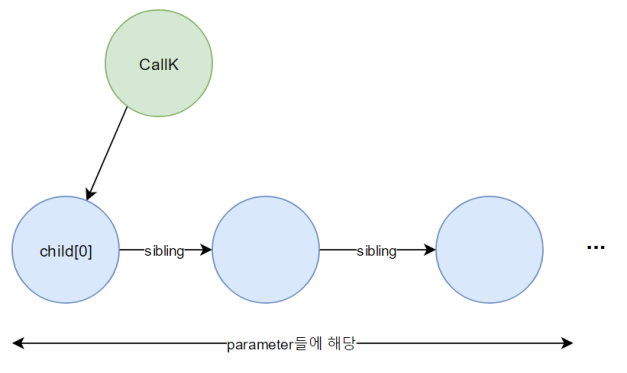
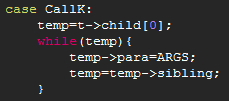
**또한, 함수에 해당하는 scope 에서는 parameter에 우선적으로 location값이 0부터 1씩 증가하면서 앞에서부터 할당되고 난 뒤, local variable들에 계속하여 location 값이 1씩 증가하면서 할당된다.**

1. **Type checking (analyze.c)**
2. **선언되지 않은 변수나 함수는 사용 불가**
3. **변수나 함수, 함수 파라미터 선언 시 중복된 이름 check**
4. **변수나 parameter 선언 시 void type으로 선언할 수 없음.**
5. **변수나 parameter 선언 시 array 변수인지 확인해야 함.**
6. **변수에 값을 assign 하는 경우 type에 대한 check**
7. **변수가 array인 경우 array index가 int가 아닌 경우**
8. **array가 아닌 변수를 array처럼 사용하려 하는 경우**
9. **함수 호출 시 parameter의 개수와 type이 일치하는지 check**
10. **함수 호출 시 호출된 것이 함수가 맞는지 (즉 변수 등이 아니었는지) 확인해야**
11. **함수의 return 값과 return type이 일치하는지 check**
12. **함수의 return type이 void일 경우 return이 없어야 함**
13. **main함수는 가장 마지막에 선언되어야 함**
14. **main함수는 반드시 void로 선언되어야 함**
15. **main함수는 반드시 parameter가 존재하지 않는다.**
16. **loop문의 반복 조건을 표시하는 부분에 오는 statement는 int값을 가질 수 있어야 한다.**

**본 프로젝트에서 제시된 오류 체크 사항을 모두 처리하도록 하며, c- language에서 발생할 수 있는 모든 type error 및 추가적인 사항들을 처리한다.**

* 1. **제작 내용**

앞서 설명한 바와 같이 이번 프로젝트는 크게 전체 코드에 대한 symbol table을 작성하는 부분과 작성된 symbol table을 탐색하며 type checking을 하는 부분으로 나눌 수 있다.

1. **Symbol table 작성 (analyze.c + symtab.c)**
   * 1. **우선 main.c에서 parse까지 에러가 없이 진행되었을 경우 buildSymtab 함수를 호출하여 symbol table의 작성을 시작한다.**
     2. **buildSymtab(analyze.c)의 함수를 진입했을 경우 맨 처음에는 symbol table이 없는 상태이므로 st\_scopeup이라는 함수를 호출하여 새로운 symbol table을 생성해준다.**
     3. **이때, st\_scopeup(symtab.c)이라는 함수는 새로운 symbol table을 생성해주는 함수이다. 이 함수는 새로운 scope이 생겼을 때 호출되기 때문에 저장되어 있던 scope level의 값을 1 증가시켜서 새로운 symbol table에 저장시켜 주고, 새로운 symbol table은 up을 통해 자신이 속해 있는 상위 scope을 가리킨다. 이때, 상위 scope에 속한 하위 scope이 여러 개일 때, 새로 생기는 scope은 맨 마지막에 연결되고, 각각의 scope은 next라는 포인터로 연결된다. 그리고 현재 scope을 의미하는 st가 새로 생긴 symbol table을 가리킨다. (자세한 구조는 나. 분석 내용 참조)**
     4. **buildSymtab에서 st\_scopeup을 진행한 후 tree를 traverse하기 시작한다. 이때 traverse(syntaxTree, insertNode, st\_scopedown)을(analyze.c) 호출하게 되는데, syntaxTree는 abstract syntax tree의 root를 가리키게 된다. Traverse 함수에서 AST 를 대상으로 preorder로 insertNode를 진행하고 (이 과정에서 st\_scopeup이 호출됨), postorder로 compound statement가 끝날 때 마다 st\_scopedown을 진행하게 된다.**
     5. **insertNode(analyze.c) 함수는 tree를 탐색하면서 각 노드의 타입별로 symbol table에 적절한 삽입을 해주거나 scope에 대한 조정을 해주는 함수이다.**
   1. **우선, 함수의 선언이 아닌 compound statement를 나타내는 노드를 만났을 경우 scope가 증가되므로 새로운 symbol table의 생성이 필요하다. 또한, 새로운 scope로 진입했다는 의미이므로 location을 0으로 setting해준다.**
   2. **하지만 함수의 선언에 의한 compound statement를 만났을 경우 scope의 값이 증가되면 안 된다. tree를 순회할 때, 함수 선언에 해당하는 DclrK의 FuncK노드를 먼저 만나고 오기 때문에 이미 scope이 새로 생긴 상태이기 때문에 중복으로 st\_scopeup이 호출되는 것을 방지한다.**
   3. **변수나 배열에 해당하는 노드를 만났을 경우 symbol table을 다 탐색하여 해당 변수나 배열, 함수가 선언되어 있는지 확인하고 그렇지 않다면 에러를 출력해준다. 해당 변수나 배열, 함수가 선언되어 있음을 확인했다면 symbol table에 insert를 위해 st\_insert를 호출해준다.**
   4. **함수의 호출에 해당하는 변수를 만났을 경우에는 함수 호출과 함께 넘겨주는 parameter 노드를 명시해주는 처리를 해주어야 한다. 현재 AST에서는 다음과 같이 CallK의 노드의 child[0]에 sibling으로 하지만 실제 AST에서는 이 parameter에 해당하는 노드들에 따로 함수 호출에 의해 넘겨지는 parameter라는 명시가 없기 때문에 추후 traverse를 진행했을 때 해당 노드가 함수 호출에 쓰여진 parameter인지 아닌지에 대한 정보가 없다. 따라서 추후에 type checking 처리에 유용하도록, 현재 노드가 CallK일 경우 다음과 같이 자식 노드들 중 함수 호출에 넘어가는 parameter에 해당하는 노드의 para값에 -10에 해당하는 ARGS 값을 대입해주어 명시해준다.  
      **
   5. **배열이나 일반 변수가 선언되는 노드를 만났을 경우. 해당 symbol 이름을 가지는 변수가 이미 선언되어 있는지 st\_lookup함수를 이용해 체크 후, 동일한 이름을 갖는 변수가 존재한다면 에러 메시지를 출력해준다. 그렇지 않다면 해당 변수를 st\_insert를 통해 symbol table에 삽입해준다. 이때, location을 계산해서 parameter로 넘겨주는데 선언 시에는 scope의 변화가 없으므로 기존 location값에 1을 더해준다.**
   6. **함수가 선언되는 노드를 만났을 경우   
      해당 함수의 이름이 symbol table에 이미 있을 경우에 대해 에러 메시지를 출력해준다. 그게 아니라면 st\_insert를 이용해symbol table에 삽입해주고 이때 location을 계산해서 parameter로 넘겨준다. 함수의 이름의 경우 현재 scope의 symbol table에 그대로 삽입되는 것이므로 기존 location값에 1을 더해준다. 그리고함수의 선언에 의해 새로운 scope이 생기게 되므로 st\_scopeup을 호출해주어 새로운 symbol table을 생성해준다. 추후 함수 선언에 의한 compound statement가 나올 경우 중복으로 st\_scopeup이 되는 것을 방지하기 위해 funcscope flag변수를 선언하여 true로 설정해 주었다.**
      1. **St\_insert(symtab.c) 함수는 실제로 symbol table에 정보 삽입에 대한 구현이 이루어진 부분이다. Parameter로 insert될 AST의 node pointer와 계산된 location값, vpf(variable, parameter, function)에 대한 정보가 넘어온다. 이때 mode라는 flag변수가 있다. 선언에 해당하는 노드에서 st\_insert 함수가 호출되었다면 현재 scope의 symbol table에 정보를 저장하게 된다. 하지만 expression부분에서 변수가 참조되어 st\_insert가 호출되었다면 그 변수가 현재 scope이 아닌 다른 scope에 있는 경우도 존재하기 때문에 두 경우에 대해 처리해야 하는 일이 다르다. 따라서 mode값이 0이면 전자, 1이면 후자를 가리키도록 하여 코드를 분리해주었다.**

**mode값이 0인 경우, symbol로 hash값을 구해 해당하는 BucketList를 찾아준다. 그리고 BucketList의 끝을 찾아서 새로운 BucketList를 생성해준다. 그리고 parameter로 넘어온 노드에 담겨 있는 정보를 BucketList에 저장해준다. 이 정보들에는 이름과 선언된 줄 번호, vpf(variable인지 paramter인지 function인지)에 대한 정보가 있다. 함수라면 parameter들에 대한 정보도 같이 저장하고, 배열이라면 배열의 크기를 저장해준다. 또한 변수의 타입과 location에 대한 정보가 담기게 된다.**

**mode값이 1인 경우, 현재scope부터 시작해 참조된 변수를 찾을 때까지 scope pointer를 계속 타고 올라가면서 반복해주어야 한다. 참조된 변수에 대한 정보가 담긴 BucketList를 찾으면 참조된 변수가 있는 줄 번호와 type을 BucketList 안에 저장해준다.**

* + 1. **St\_scopedown(symtab.c) 함수는 postorder로 tree를 traverse하면서 compound statement에 해당하는 노드를 만났을 경우, 현재 scope에 해당하는 symbol table을 출력해주고 symbol table pointer를 현재 연결되어 있는 상위 레벨의 symbol table을 가리키도록 하여 scope을 빠져 나온 것을 처리하였다.**
    2. **다시 buildSymtab함수로 돌아왔을 때 마지막으로 현재 해당하는 scope에 대한 symbol table을 출력해준다.**

1. **Type checking(analyze.c)**

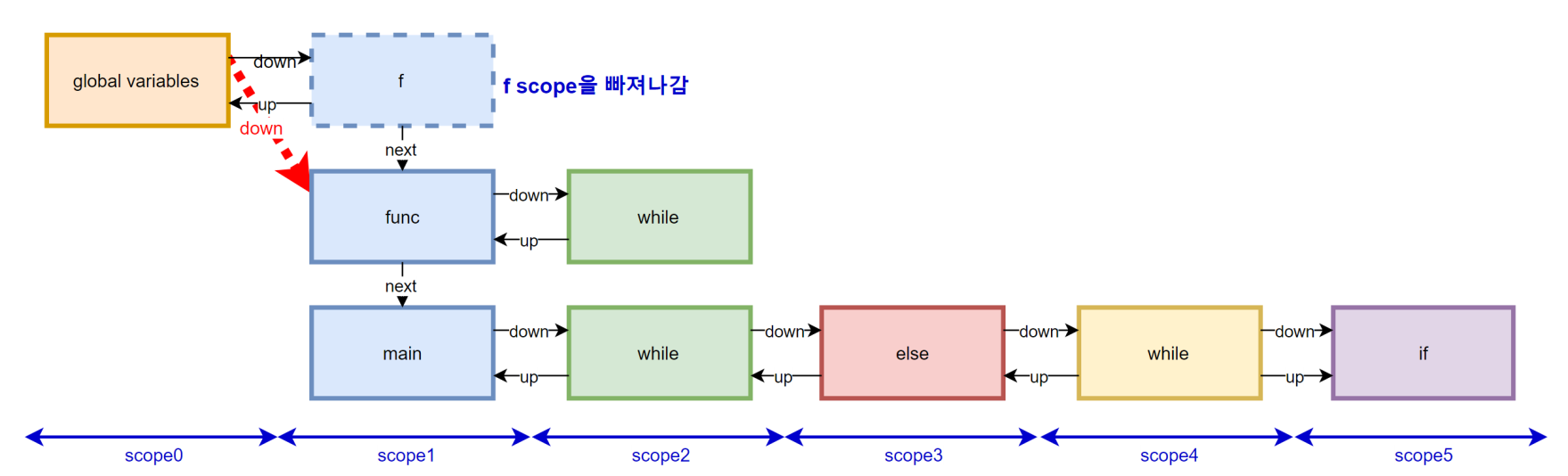
**Main.c 에서 buildSymtab함수를 호출하며 symbol table을 생성한 뒤 typeCheck함수를 통해 생성된 symbol table을 참고하여 type checking이 이루어진다.**

**typeCheck (analyze.c)함수에서는 traverse(syntaxTree, FindScope, checkNode)를 호출하게 된다.**

**이 후, traverse함수에서는 preorder로 AST를 탐색하면서 FindScope함수를 통해 대응되는 symbol table로 pointer를 옮겨주고 preorder로 AST를 탐색하면서 checkNode함수를 통해 type checking을 해준다.**

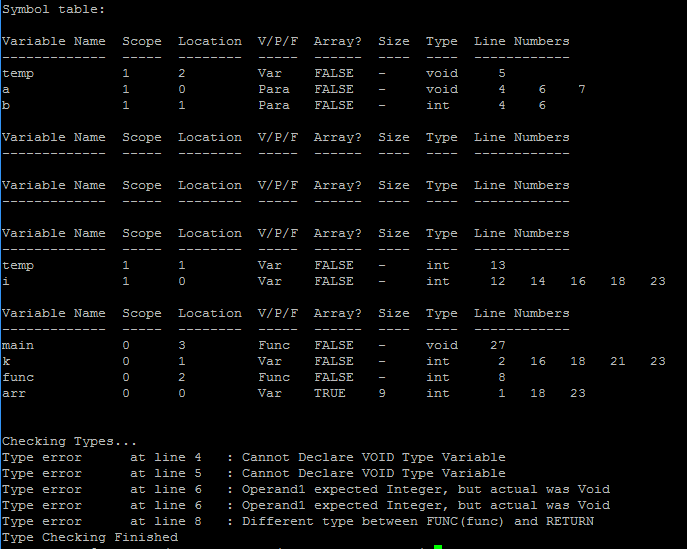
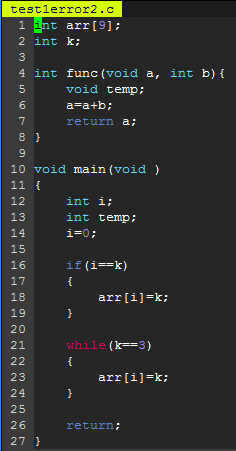
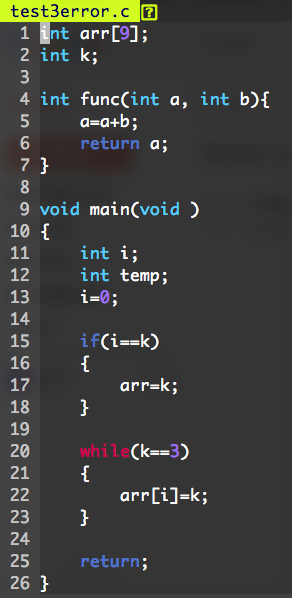
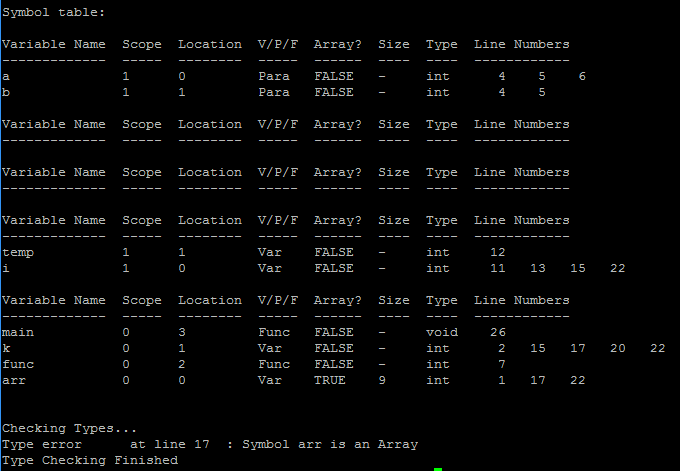
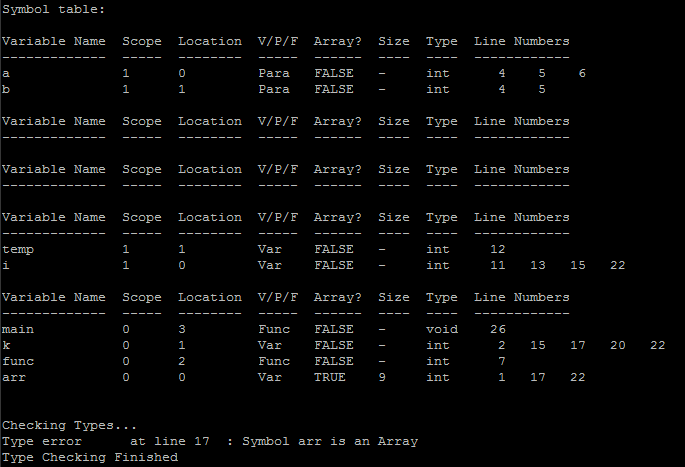
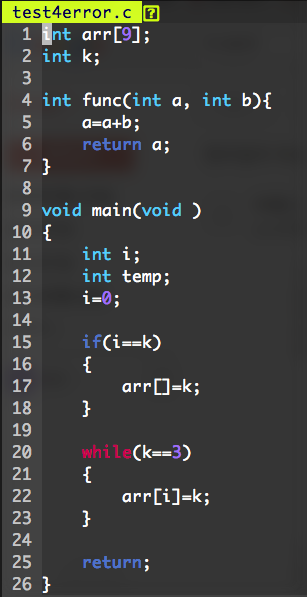
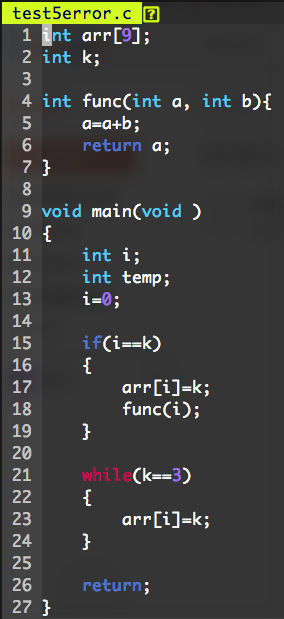
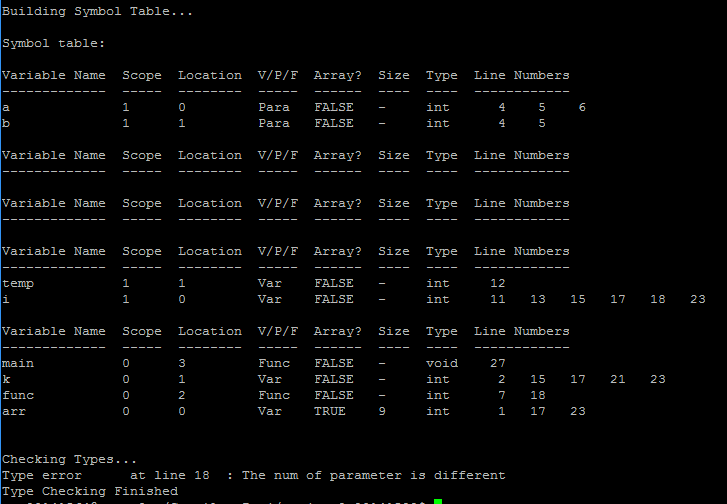
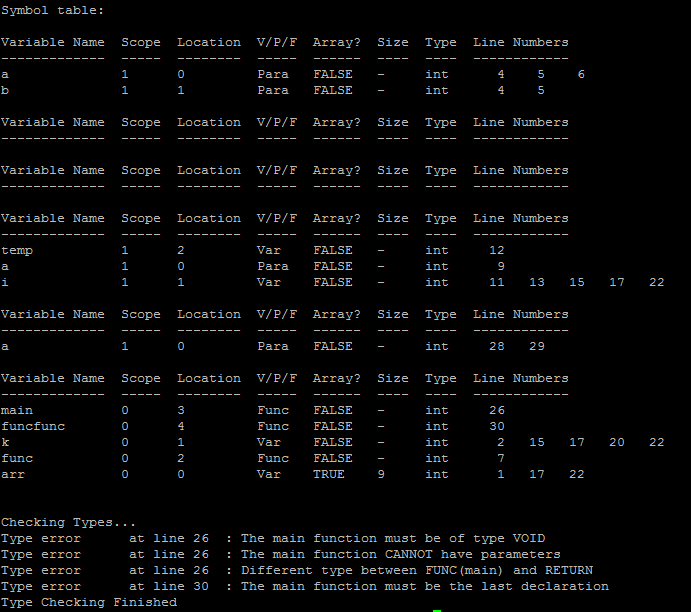
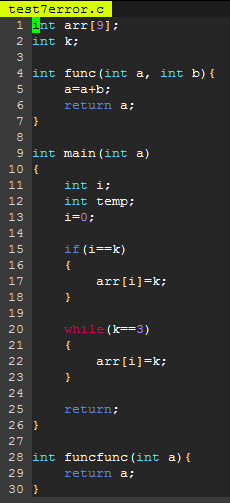
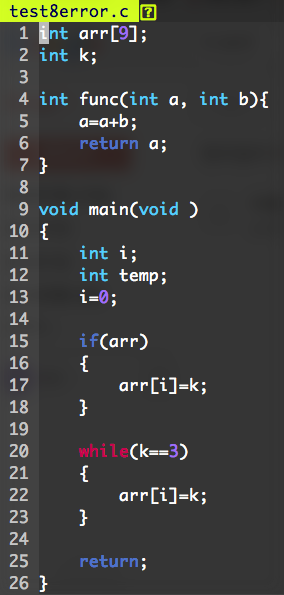
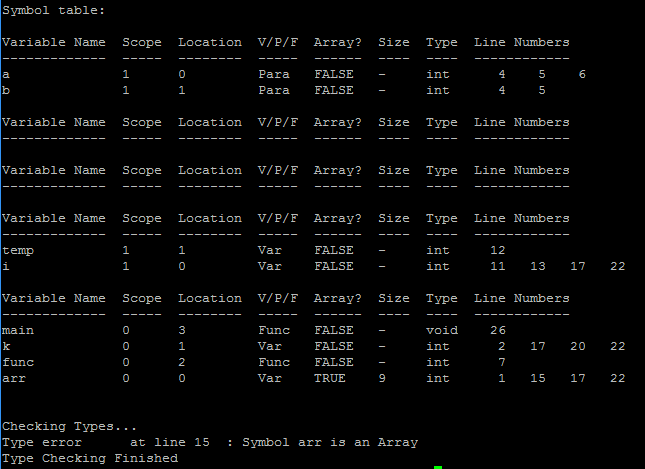
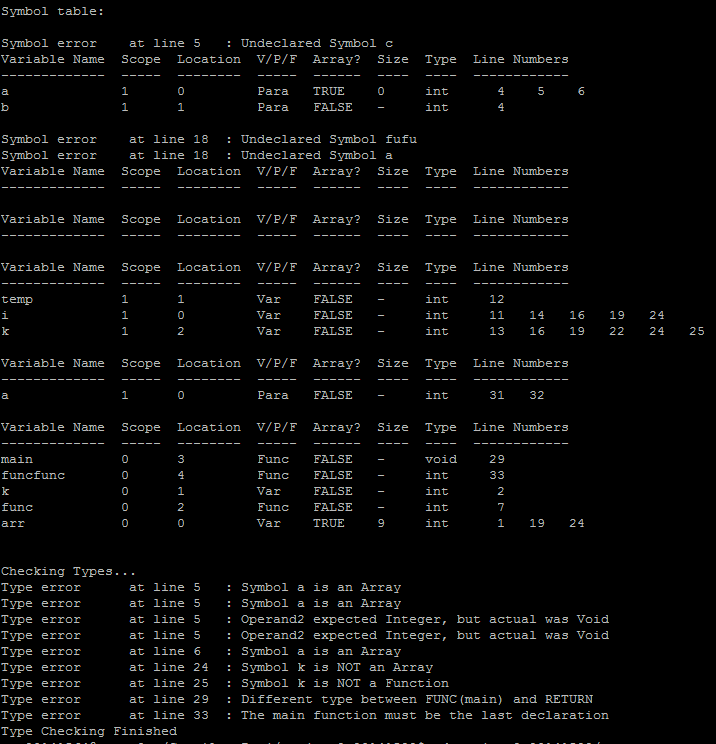
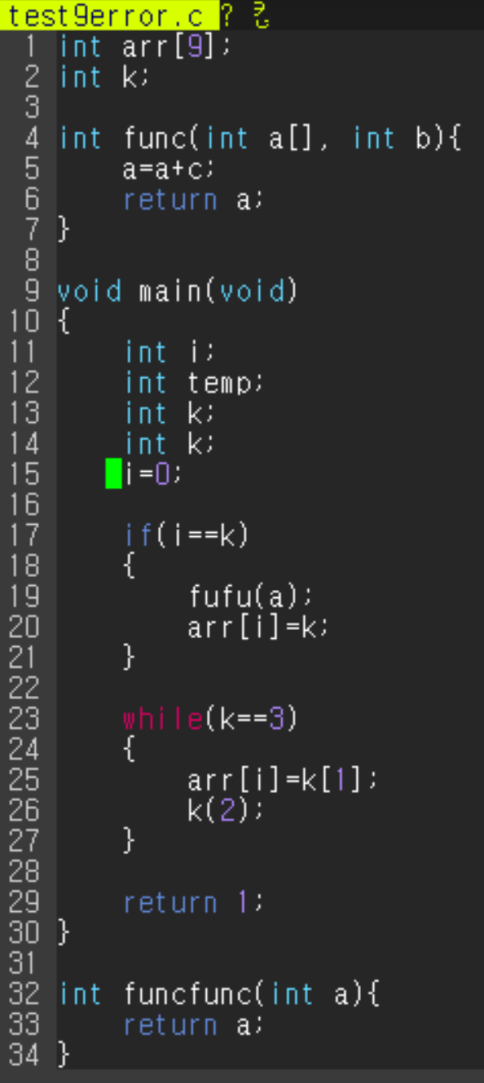
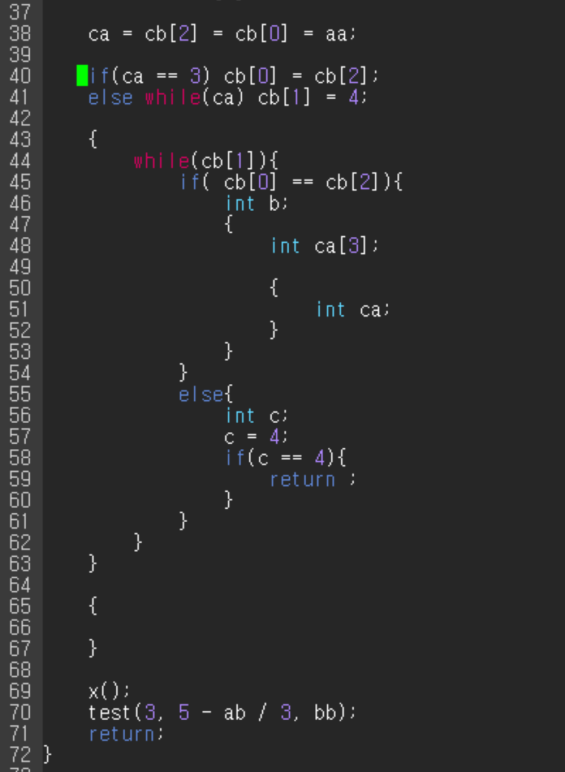
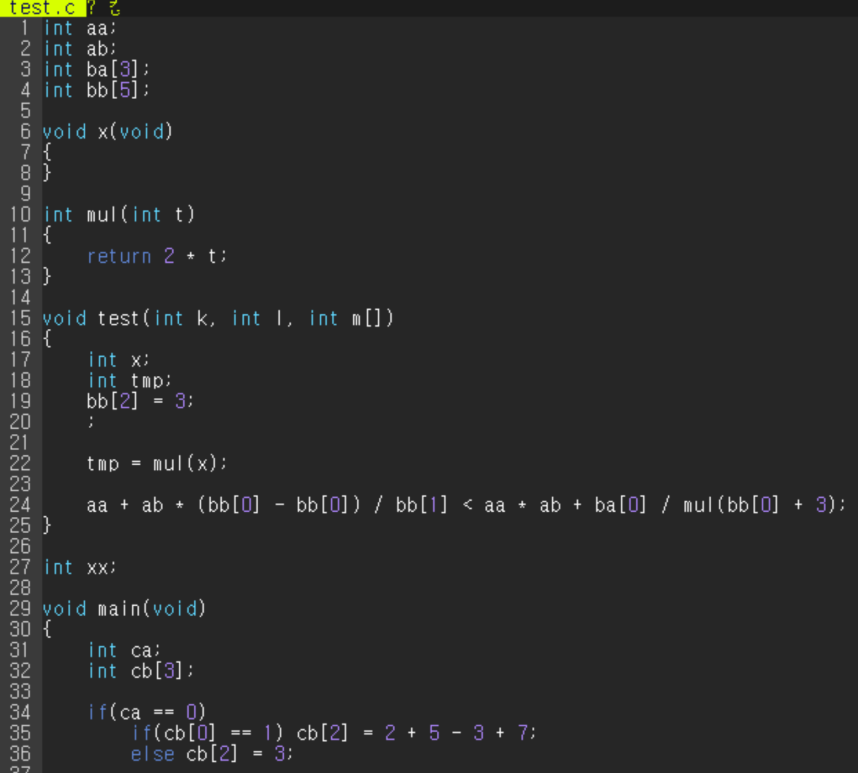
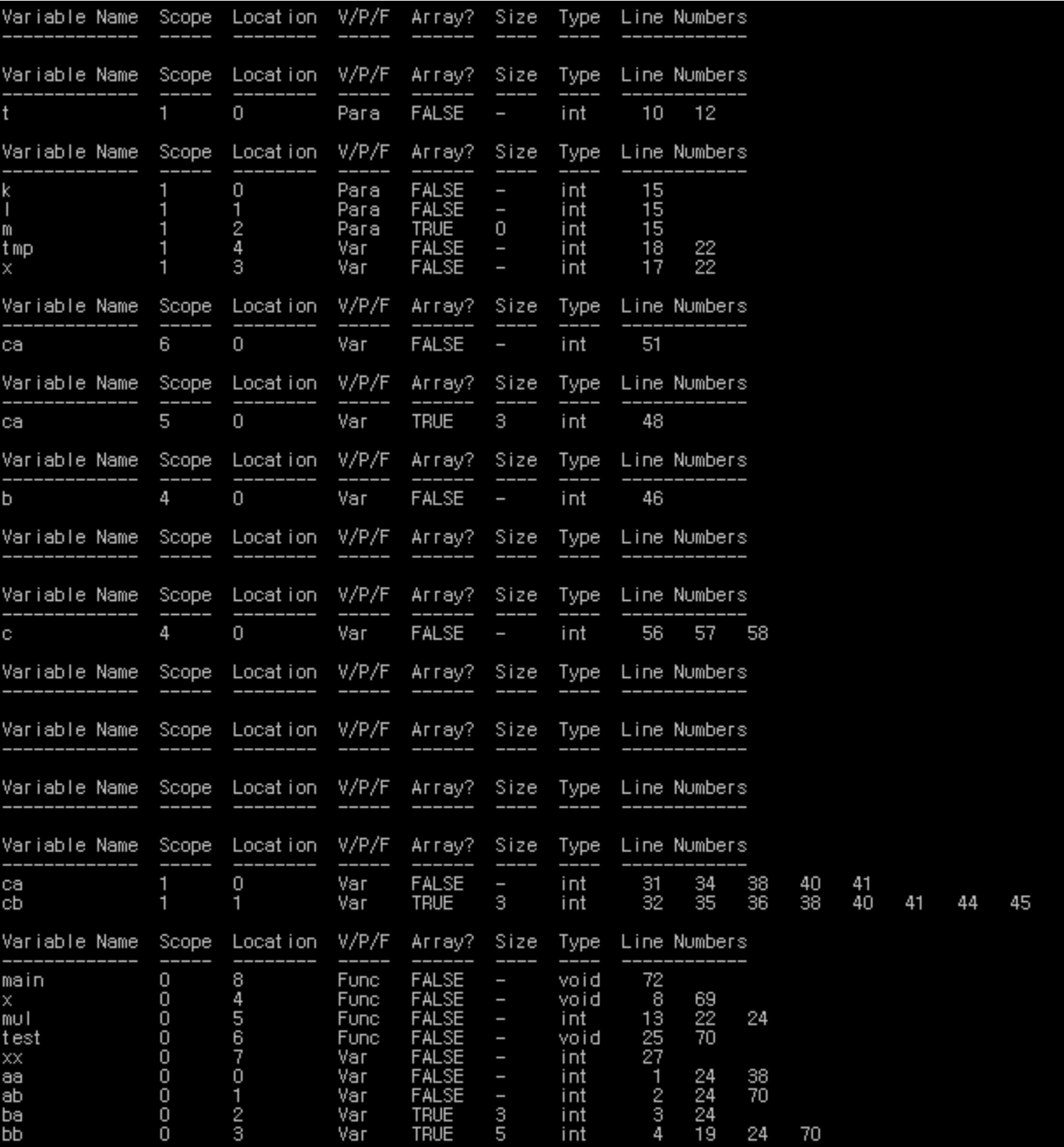
**FindScope(analyze.c)함수에서는 호출될 때마다 해당 노드가 함수의 선언에 해당한다면 설정해주고 symbol table pointer를 하위 scope을 가리키도록 옮겨준다. Tree를 preorder로 순회하면서 FindScope함수가 호출되는데 이 경우 함수가 선언되고 parameter부터 새로운 scope으로 들어가게 된다. 따라서 함수의 scope의 여부를 판단하는 flag은 funcscope을 true로 설정해준다.  
만약 해당 노드가 statement타입의 compound statement에 해당한다면 symbol table pointer를 하위 scope을 가리키도록 옮겨준다. Compound statement에 해당하는 노드를 만났을 경우에 funcscope이 true라면 함수 scope이 종료되었다는 의미이므로 funcscope을 false로 바꿔준다.**

**checkNode(analyze.c) 함수에서는 type checking을 실시하게 된다.**

* + - * 1. **OpK에 해당하는 노드가 나왔을 경우 operator의 두 operand들이 void type이거나, 두 operand의 type이 다를 경우 에러를 처리해주었다.**
        2. **Idk일 경우 해당 Idk에 대한 정보가 담겨있는 struct를 찾아온다. 이 경우 함수인데 변수처럼 쓰인 경우와 배열인데 일반 변수처럼 사용된 경우에 대한 에러 처리가 이루어 졌다.**
        3. **배열 변수가 나왔을 경우 st\_lookup함수를 이용하여 생성되어 있는 symbol table에서 해당 배열 변수에 대한 정보가 담겨있는 struct를 찾는다. 이때 배열의 크기가 -1로 지정돼있는 경우 올바른 배열이 아니므로 에러 메시지를 출력해준다. 또한, 배열의 index가 없거나 integer 타입이 아닌 경우 에러 메시지를 출력해준다.**
        4. **함수 호출이 나왔을 경우 st\_lookup함수를 이용하여 생성되어 있는 symbol table 에서 해당 함수에 대한 정보가 담겨있는 struct를 찾는다. 이때, struct에 Func라고 분류되어 있지 않으면 해당하는 symbol이 함수가 아니므로 에러 메시지를 출력해준다. 또한, struct 에 저장되어 있는 함수의 parameter정보(즉, 함수 선언 시 parameter들)와 함수 호출 시 사용된 paramter들을 비교하여 parameter의 개수, type 일치 여부를 확인하여 에러 처리를 해준다.**
        5. **Compound statement에 해당하는 노드를 만났을 경우에 현재 scope을 빠져나간다는 것을 의미하기 때문에 현재 scope이 연결돼있는 상위 scope의 down을 다음과 같이 변경해준다.**
        6. **Return 에 해당하는 노드를 만났을 경우에 이제 함수가 끝나고 return했다는 것을 추후에 알려주기 위하여 return\_flag를 사용하여 1로 지정해주었다. 그리고 추후 return type비교를 위해 현재 노드를 returnNode라는 임시 변수에 저장해놓는다. 그리고 실제 return 된 값과 type을 저장해놓는다.**
        7. **배열이나 변수의 선언 노드가 나왔는데 void 타입일 경우, C- language에서 void타입으로 변수가 선언될 수 없기 때문에 에러 메시지를 출력해준다.**
        8. **함수의 선언에 해당하는 노드가 나온 경우에는 우선 main함수인지 체크해준다. main함수가 나왔을 때 final\_flag를 1로 설정해두고, 다시 typeCheck가 호출되었을 때 만약 선언에 해당하는 노드라면 main 함수의 선언 뒤에 다른 선언이 왔다는 의미이고, 이때 final\_flag는 1일 것이다. 따라서 선언에 해당하는 노드일 경우 final\_flag를 검사하여 에러 메시지를 출력해준다. 또한, main 함수일 경우 parameter가 void인지 검사하여 에러 메시지를 출력해준다.  
           또한, 해당 함수가 return된 상태일 경우에는 실제로 return된 값과 함수 선언 시 지정했던 return type이 일치하는지 확인하여 에러 메시지를 출력해준다. 또한 함수 선언 시 integer return 타입을 지정했었는데 실제로 return 된 값이 없을 때에도 에러 메시지를 처리해주었다. 모든 처리가 끝난 후 return\_flag를 다시 0으로 초기화해주어 다른 함수에 대해서도 작동할 수 있도록 처리해준다.**
  1. **시험 내용**

1. 선언되지 않은 변수나 함수는 사용 불가
2. 변수나 함수, 함수 파라미터 선언 시 중복된 이름 check
3. 변수나 parameter 선언 시 void type으로 선언할 수 없음.
4. 변수나 parameter 선언 시 array 변수인지 확인해야 함.
5. 변수에 값을 assign 하는 경우 type에 대한 check
6. 변수가 array인 경우 array index가 int가 아닌 경우
7. array가 아닌 변수를 array처럼 사용하려 하는 경우
8. 함수 호출 시 parameter의 개수와 type이 일치하는지 check
9. 함수 호출 시 호출된 것이 함수가 맞는지 (즉 변수 등이 아니었는지) 확인해야
10. 함수의 return 값과 return type이 일치하는지 check
11. 함수의 return type이 void일 경우 return이 없어야 함
12. main함수는 가장 마지막에 선언되어야 함
13. main함수는 반드시 void로 선언되어야 함
14. main함수는 반드시 parameter가 존재하지 않는다.
15. loop문의 반복 조건을 표시하는 부분에 오는 statement는 int값을 가질 수 있어야 한다.

* 본 프로젝트는 위의 15가지 사항에 대해 잘 처리할 수 있어야 하므로 여러 test를 통해서 모두 잘 처리되었는지 확인한다.

1. test1 ③,⑤ 조건 처리
2. ⑤ 조건 처리
3. ⑥ 조건 처리
4. ⑧ 조건 처리
5. ⑩⑫⑬⑭ 조건 처리
6. ⑮ 조건 처리
7. ①②④⑦⑨⑪ 조건 처리
8. Test.c  
   1. **평가 내용**
      1. 신뢰성  
          여러 테스트 케이스를 실험해본 결과 모두 올바른 결과를 출력하는 것을 확인하였다. 제시된 모든 오류에 대하여 테스트 파일을 작성하여 올바르게 동작하는 것을 확인하였다.
      2. 안정성  
          긴 테스트 케이스에 대하여 반복적으로 테스트 하였을 때 정상적으로 돌아가는 것을 확인하였다. 발생할 수 있는 모든 오류사항들에 대한 처리가 이루어졌다.
      3. 가독성  
          가독성 있는 프로그램의 작성을 위하여, 각 함수와 변수가 기능에 맞는 이름을 갖도록 하였으며, 한 함수가 이름에 맞지 않게 과도한 기능을 할 경우 두 개의 함수로 분리하여 추상화 수준을 높였다.
      4. 효율성  
          hash 함수 구현 시 BUCKET\_SIZE나 가중치에 소수를 사용하여 좀 더 uniform distribution에 가까운 분포를 갖도록 하였다. 따라서 symbol이 bucket에 비교적 균등하게 분배되어 저장되므로 hash table 탐색 시간이 줄어든다.
9. **기타**
   1. **연구 조원 기여도**

|  |  |  |
| --- | --- | --- |
| 이름 | 박상욱 | 이예진 |
| 비율 | 50 % | 50% |
|  | Symbol Table  테이블에 Symbol 추가  테이블에 Symbol 검색  Type Check  Type check 트리 순회  Declaration type check  Array subscription 시 array인지 check  Function call시 function인지 check  Type Check  Return Check  Assign 시 left value인지 check(variable, array) | Symbol Table  해시 함수 제작  해시테이블의 내용 출력  Type Check  Data Type Check  Array index  If, while condition  Operator left right  Call Check 시 argument 비교  Symbol Table  Symbol Table에 해시 테이블 추가  Symbol Table에 해시 테이블 제거 |

* 1. **자체 평가**C- language에 대해 완벽하게 이해하고 제대로 된 symbol table이 작성될 수 있도록 노력하였고, 완벽하게 생성된 symbol table을 통해 정확한 type checking을 진행하였다. 그리고 발생할 수 있는 모든 오류 체크 사항을 면밀히 분석하여 최대한 완성도가 높은 완성물을 제작하도록 노력하였다. 여러 테스트를 거치고 검증 단계를 거친 결과 본 프로젝트는 매우 우수하게 semantic analyze를 진행하는 것으로 평가된다.
  2. **느낀 점** 팀 작업을 한 적이 많지 않은데, 이번 프로젝트를 통해 팀 별 프로젝트를 해 볼 수 있어서 좋았으며, 특히 의사소통의 중요성을 느낄 수 있는 계기가 되었다. 다음 프로젝트부터는 코딩 스타일 등을 통일 시키는 등 convention을 정하여 프로젝트를 수행해 볼 계획이다.  
      프로젝트 진행 도중 단기적인 목표를 세우고, 그에 맞는 데드라인을 정하여 수행했던 것이 도움이 되었던 것 같다.  
      큰 프로젝트를 접해본 경험이 많지 않았는데, 새로운 경험이었던 것 같다. 인터넷에서 여러 가지 프로젝트 방법론을 본 적이 있었는데, 기회가 된다면 이러한 것들을 시험해 보고 싶다.