Design and Development of Compiler  
for C- Language  
(설계 프로젝트 수행 결과)

과목명: [CSE4120] 기초 컴파일러 구성

담당교수: 서강대학교 컴퓨터공학과 정 성 원

개발자: 14조 20131547 김한길, 20131612 최대운

개발기간: 2019. 5. 09. ~ 2019. 5. 28.

각 단계별 결과 보 고 서

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

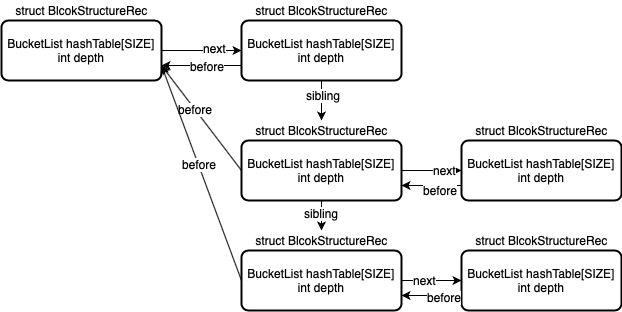
Phase 3: Design and Implementation of Semantic Analyzer

제출일: 2019. 5. 28.  
개발자: 14조 20131547 김한길 20131612 최대운

1. 개발 목표  
   - 프로젝트 2에서 구현했던 c- 파서를 이용하여 semantic analyzer를 구현한다.  
   - Semantic analyzing은 크게 두 단계로 나뉜다. 첫번째 단계에서는 Symbol table을 생성하여 함수나 변수의 중복된 선언을 체크하고 변수의 Scoping error를 확인하며 메모리 메핑을 진행한다. 그리고 생성된 변수, 함수와 파라미터의 type이나 파라미터에 대한 정보 Array에 대한 정보를 저장한다.  
   - 두번째 단계에서는 Type Checking을 진행한다. Type Checking은 명세서에 제시된 부분을 검사한다.
2. 개발 범위 및 내용
   1. 개발 범위  
      C- 언어에 대한 Semantic Analyzer 을 만든다. C- 언어 파일을 입력받으면 Scope 별로 symbol table을 형성하고 두번째 단계에서는 semantic error을 확인한다.

2번째 프로젝트에 analyze.c,h symtab.c,h 그리고 globals.h 를 확장하고 변형하여 구현한다.

* 1. 개발 내용



<scope 별로 심볼테이블을 구성하기 위한 Blockstructure>

* 먼저 HashTable의 top에 최상위 scope를 위한 HashTable을 하나 만든다. 그 이후 compound statement 를 만나면 scopeIn 함수를 호출하여 depth를 더 깊게하고 before를 통해 연결해두어 scopeOut 함수를 통해 이전 scope로 이동할 수 있다. 뿐만 아니라 scopeIn 함수를 호출하였을 때 먼저 생성된 HashTable이 존재할 경우에는 HashTable 생성 후 sibling를 통해 연결한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

<HashTable 정보를 갖는 구조체와 Symbol 정보를 저장하기 위한 구조체>

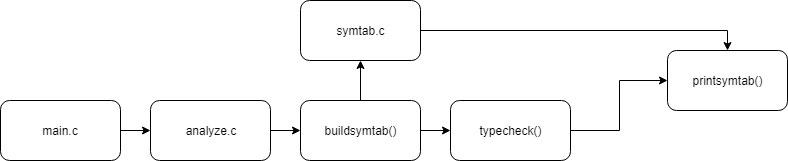
* BucketListRec 구조체는 주어진 구조체로 Symbol 정보를 저장하기 위해 info 변수를 갖도록 추가한다. SymbolInfoRec은 Type Checking에 필요한 정보들을 저장한다.
* 최종적으로 Symbol Table을 만들고 Type Checking 하여 에러가 없는 경우 Symbol Table을 출력한다.
* Analyze.c/h 에서는 AST를 기반으로 symbol table을 만들고 type check를 위한 부분이 구현된다.
* Symtab.c/h 에서는 symbol table을 만들기 위한 함수들이 구현된다.

1. 추진 일정 및 개발 방법
   1. 추진 일정

|  |  |
| --- | --- |
| 일정 | TODO |
| 05. 19.~ 05. 27 | 프로젝트 명세서 파악 및 개발 |
| 05. 27. | 보고서 작성 |
| 05. 28. | 제출 |

* 1. 개발 방법  
     - Analyze.c/h, Symtab.c/h를 적절히 수정하여 개발한다.

1. 연구 결과



<프로그램 구성도>

이전 프로젝트에서 생성된 abstract syntax tree를 이용하여 analyze.c의 buildsymtab 에서 symbol table을 작성하고 한번 더 Type checking하여 semantic error를 확인한 후 printsymtab를 통해 symbol table을 출력한다.

* + 1. 분석 내용

Symtab의 생성과 Type checking은 함수의 재귀호출을 통해 구현한다.

함수 memory location은 가장 먼저 나오는 함수가 0을 가지고 이후에 나오는 함수들은 memory location이 1씩 증가한 값을 갖는다.

변수 memory location은 지역 변수의 경우 return address가 저장될 -4 이후로 저장된다. 예를 들면, 변수 a와 b가 선언되었을 때, a는 -8, b는 -12의 memory location을 갖는다. 전역 변수의 경우 앞서 선언된 전역 변수의 memory 사이즈 다음부터 저장된다. 예를 들면, a[10], b[2]이 선언되었을 때, a는 40, b는 48의 memory locaiton을 갖는다.

파라미터 memory locaiton은 가장 오른쪽 파라미터부터 memory location 값을 받는다. 만약 a와 b가 파라미터로 선언된 경우 b는 4, a는 8의 memory location을 갖는다.

* + 1. 제작 내용

메인 함수에서는 스켈레톤 코드에서 TraceAnalyze Flag를 True로 만들어 buildSymTab함수와 typeCheck함수를 호출한다.

Symtab.c/h 파일에 다음과 같은 구조체를 추가한다.  
BlockStructureRec: 제공된 소스코드의 BucketList 구조체와 Scope의 깊이 정보를 저장하는 구조체로 앞서 제안한 Symbol table의 형태를 구현한다.  
SymbolInfo: Symbol이 BucketList에 저장될 때, 가지고 있어야 하는 정보들을 갖는다. 노드의 종류, 선언 종류(변수, 파라미터, 함수), expType, Array 정보, 파라미터의 정보(Symbol이 함수일 경우), 리턴 되는 값의 타입(Symbol이 함수일 경우) SymbolInfo는 BucketList 구조체에 포함된다.  
ParamInfo: SymbolInfo 구조체에 포함되는 구조체로 Symbol이 함수일 경우 파라미터에 대한 정보를 갖는다. 파라미터의 이름과 파라미터의 타입(int, array)을 저장한다.

Symtab.c/h 파일에 주어진 함수를 변형한 부분은 다음과 같다.   
st\_insert: 이 함수는 변수, 함수, 파라미터를 선언하거나 참조할 때 호출되는데, 이 때 name을 기반으로 Symbol table을 검색할 때 현재 Scope의 Symbol table 뿐만 아니라 before symbol table까지 검색함으로써 전역 변수나 현재 depth보다 더 낮은 depth에 선언된 변수를 참조하여 정확한 위치의 Hash Table을 찾아 Symbol 노드를 생성하거나 line number에 대한 정보를 추가한다.

Symbol table의 생성은 AST의 Root부터 시작해서 재귀호출하여 생성한다. 이 때 .y파일에서 정의한 rule에 따라 treenode의 child로 재귀 호출 하거나 sibling으로 이동하는 방식으로 노드를 생성하게 된다.

Symbol table의 depth를 관리하기 위해 hashTableTop, hashTableCurrent 변수를 생성한다. hashTableTop은 만들어진 Symtab의 top을 가리키고 hashTableCurrent는 현재 scope의 Symtab을 가리킨다. 위 두 변수를 관리하는 함수는 st\_scopeIn, st\_scopeOut 두 함수이다.

st\_scopeIn 함수는 처음 호출 될 때, hashTableTop이 NULL이라면 hashTableTop에 Symtab을 생성한다. 그리고 hashTableCurrent는 이 Symtab을 가리킨다. 그 이후 호출 될 땐 hashTableCurrent의 next가 NULL이라면 hashTableCurrent의 next에 새 Symtab을 생성하고 hashTableCurrent가 새로 생성된 Symtab을 가리키는 방식으로 depth가 증가한다. 만약 next가 NULL이 아니라면 next의 Sibling에 계속적으로 새 Symtab을 생성하고 새로 생성된 Symtab을 hashTableCurrent가 가리킨다. 새로운 Symtab은 before로 hashTableCurrent를 가리킴으로써 상위 depth의 Symtab을 가리킨다.

st\_scopeOut 함수는 hashTableCurrent를 before를 통해 상위 depth로 이동시킨다.

Symtab의 생성이 완료되면 hashTableCurrent는 depth 0의 Symtab을 가리키게 된다.

TypeCheking은 Symtab 생성과 마찬가지로 재귀호출하여 진행한다. 이때 Scope에 맞는 Symtab을 참조하기 위해 st\_scopeMove 함수를 이용한다. st\_scopeMove함수의 파라미터는 이동량을 갖고 0일 땐 hashTableCurrent의 next로 이동하고 0보다 큰 값인 경우 그 값만큼 next의 sibling으로 이동한다. If나 while statement 같이 compound statement가 나오는 경우에 sibling값을 증가시키는 방식이다. 함수 선언이나 compound statement가 단독으로 나오는 경우 st\_scopeMove함수의 파라미터를 0으로 주어 next Symtab을 참조하게 한다. TypeChecking에 사용되는 checkNode 함수는 해당 node의 결과가 어떤 expType인지 리턴한다. 자세한 type checking 방식은 명세서에서 주어진 조건과 함께 하단에서 설명한다.

SymTab의 출력은 hashTableTop부터 출력하기 때문에 depth 0 부터 next SymTab, sibling SymTab 순서로 재귀호출하여 출력된다.

Semantic error 들

- 선언되지 않은 변수나 함수는 사용 불가

-> buildSymtab과정 중 Exp node에서 Identifier를 참조할 때 hashTableCurrent 부터 그 상위(before) Symtab에 Identifier가 없을 때 에러가 출력된다.

- 변수나 함수, 함수 파라미터 선언 시 중복된 이름 check

-> buildSymtab과정 중 Dec node에서 hashTableCurrent 부터 그 상위(before) Symtab에 중복된 이름이 있을 때 에러가 출력된다.

- 변수나 parameter 선언 시 void type으로 선언할 수 없음.  
 -> buildSymtab과정 중 Dec node에서 Symbol Info의 expType이 Void 인 경우 에러가 출력된다.

- 변수나 parameter 선언 시 array 변수인지 확인해야 함.  
 -> .y 파일에 rule에서 array 타입으로 선언될 시 treenode에 isArray 변수를 통해 확인한다.

- 변수에 값을 assign 하는 경우 type에 대한 check  
 -> TypeChecking 과정 중 assign statement 에서 두 자식 노드의 checkNode 함수의 리턴 값을 비교하여 다를 경우 에러를 출력한다.

- 변수가 array인 경우 array index가 int가 아닌 경우

-> TypeChecking 과정 중 exp node의 ID kind에서 이 node가 자식 노드가 있을 때, 자식 노드의 checkNode함수 리턴 값이 int형인지 체크한다.

- array가 아닌 변수를 array처럼 사용하려 하는 경우

-> TypeChecking 과정 중 exp node의 ID kind에서 Symbol Info의 isArray가 FALSE 이고 이 node가 자식 노드가 있을 때 에러를 출력한다.

- 함수 호출 시 parameter의 개수와 type이 일치하는지 check

-> TypeChecking 과정 중 exp node의 FuncCall kind에서 Symbol Info의 파라미터 정보와이 노드의 자식 노드와 비교한다.

- 함수 호출 시 호출된 것이 함수가 맞는지 (즉 변수 등이 아니었는지) 확인해야

-> TypeChecking 과정 중 exp node의 FuncCall kind에서 Symbol Info의 decKind가 Function kind가 아닌 경우 에러를 출력한다.

- 함수의 return 값과 return type이 일치하는지 check

(함수의 return type이 void일 경우 return이 없어야 함)

-> TypeChecking 과정 중 statement node의 return kind에서 호출된 함수의 Symbol Info에 return 된 데이터 타입을 저장한다. 이후 dec node의 function kind의 끝(재귀 호출이 완료된 상태)에서 리턴 타입을 비교한다.

- main함수는 가장 마지막에 선언되어야 함

-> TypeChecking 과정 중 dec node의 function kind에서 노드의 이름이 main이면서 이 노드의 sibling이 존재하는 경우 마지막에 선언된 것이 아니므로 에러를 출력한다.

- main함수는 반드시 void로 선언되어야 함

-> TypeChecking 과정 중 dec node의 function kind에서 노드의 이름이 main이면서 Symbol Info의 expType이 Void가 아닌 경우 에러를 출력한다.

Memory Location은 Symtab을 생성하는 과정에서 진행되며 지역 변수 생성으로 인한 메모리 변화를 유지시키기 위해 insertNode 함수에 localVarLoc 변수를 만들어 함수 끝에서 그 값을 리턴한다. 이후 compound statement에서 지역 변수를 생성하는 부분인 child[0]의 insertNode가 진행되고 난 후 리턴 값을 child[1]의 insertNode를 진행할 때 반영하도록 했다.

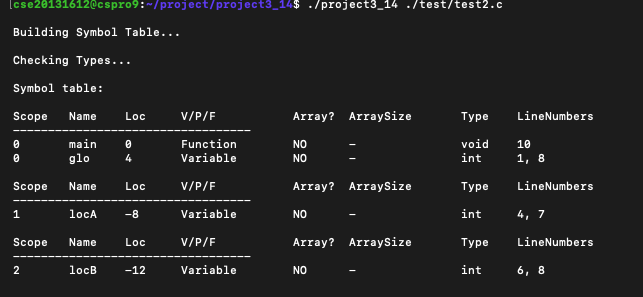
* + 1. 시험 내용

실행하는 방법은 다음과 같다.

* + - 1. Make
      2. ./project3\_14 [testFile]

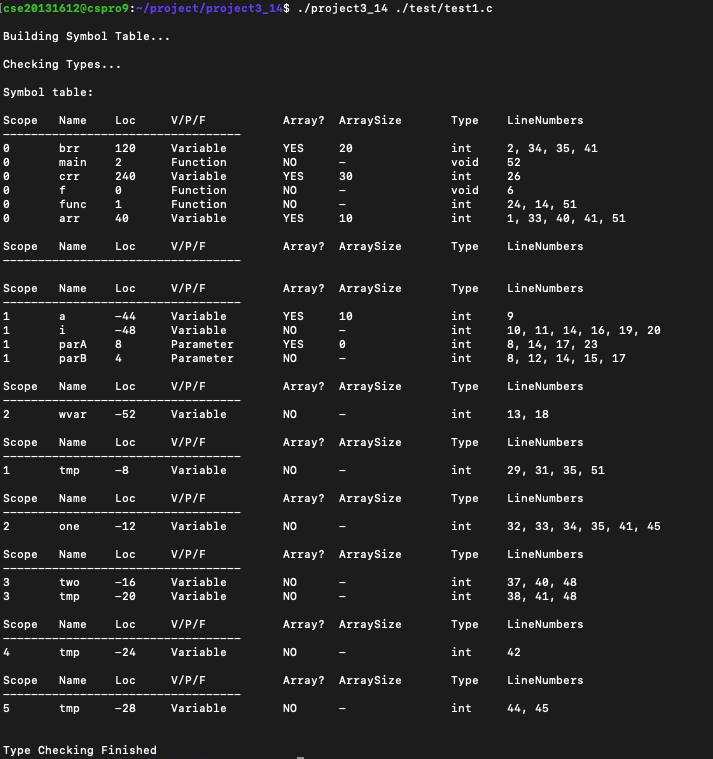
제시된 입력 결과에 대한 결과는 아래와 같다.

1. Test2.c



<제공해주신 2번테스트케이스 출력>

1. Test1.c



<제공해주신 1번 테스트케이스 출력>

1. Test3.c 선언되지 않은 변수의 사용 에러



<제공해주신 3번 테스트케이스 출력>

1. Test4.c 같은 이름의 변수 선언 에러



<제공해주신 4번 테스트케이스 출력>

1. Test5.c Array subscript type 에러

텍스트이(가) 표시된 사진

자동 생성된 설명

<제공해주신 5번 테스트케이스 출력>

1. Test6.c 연산자 좌우의 피연산자 type match 에러

텍스트이(가) 표시된 사진

자동 생성된 설명

<제공해주신 6번 테스트케이스 출력>

1. Test7.c void function에 return statement가 있는 에러

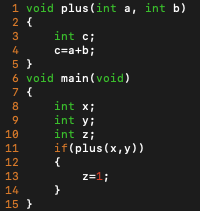
텍스트이(가) 표시된 사진

자동 생성된 설명

<제공해주신 7번 테스트케이스 출력>

다음은 임의로 만든 입력 결과에 대한 결과이다.

1. If statement의 조건문에 void 값이 들어가는 경우



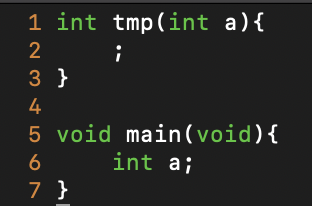
<실험을 위해 만든 테스트케이스 입력>

텍스트이(가) 표시된 사진

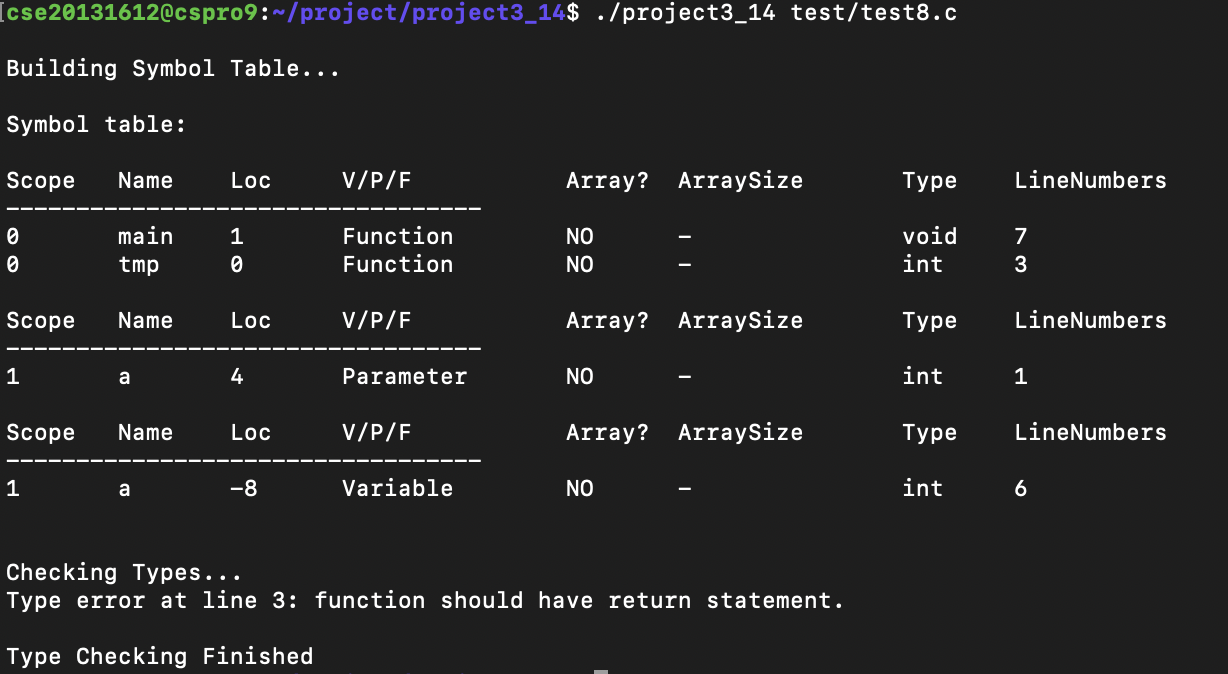
자동 생성된 설명

<실험을 위해 만든 테스트케이스 출력>

1. Int function에 return statement가 없는 경우

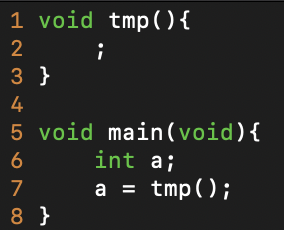


<return statement가 없는 경우 입력>



<return statement가 없는 경우 출력>

1. Int function에 return statement가 없는 경우

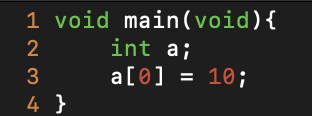


<변수에 값을 assign 하는 경우 type에 대한 check 입력>

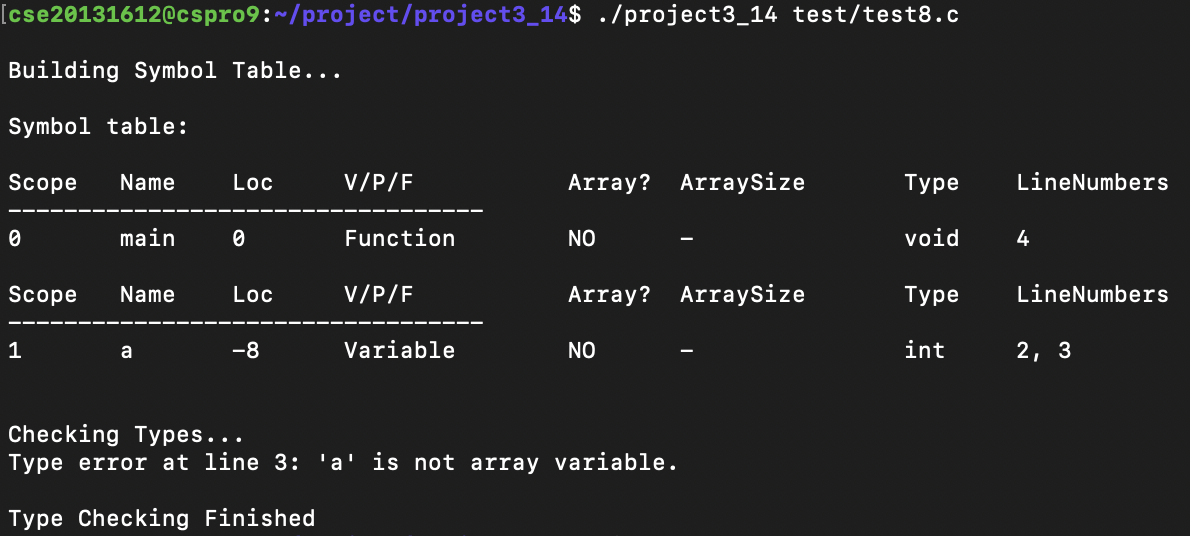


<변수에 값을 assign 하는 경우 type에 대한 check 출력>

1. 배열이 아닌 변수를 배열처럼 사용하는 경우

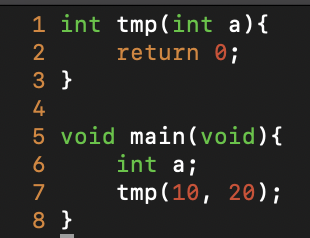


<배열이 아닌 변수를 배열처럼 사용하는 경우 입력>

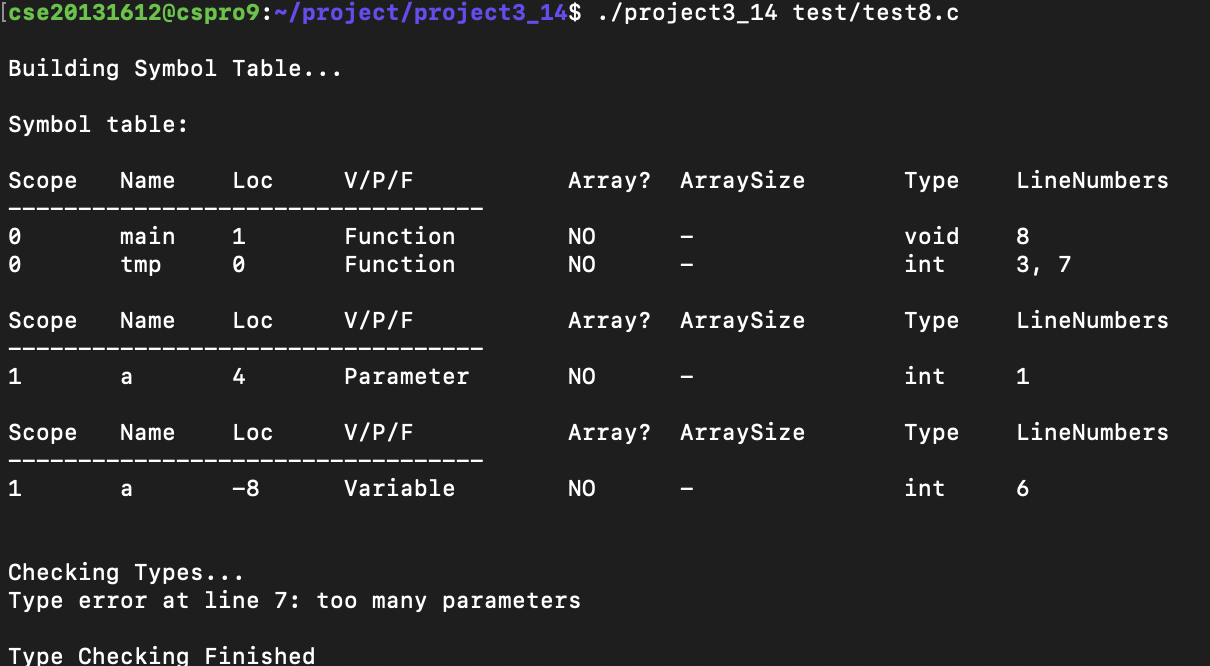


<배열이 아닌 변수를 배열처럼 사용하는 경우 출력>

1. 함수 호출 시 parameter의 개수와 type이 일치하는지 check

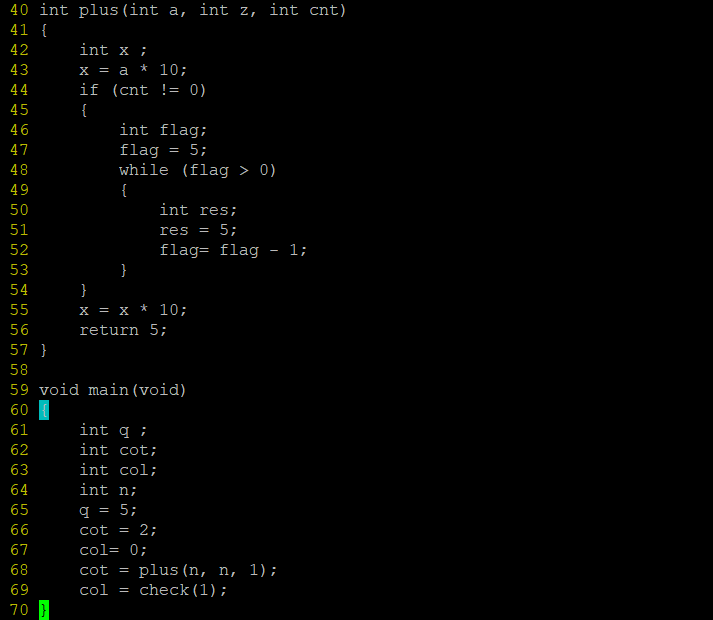


<함수 호출 시 parameter의 개수와 type이 일치하는지 check 입력>

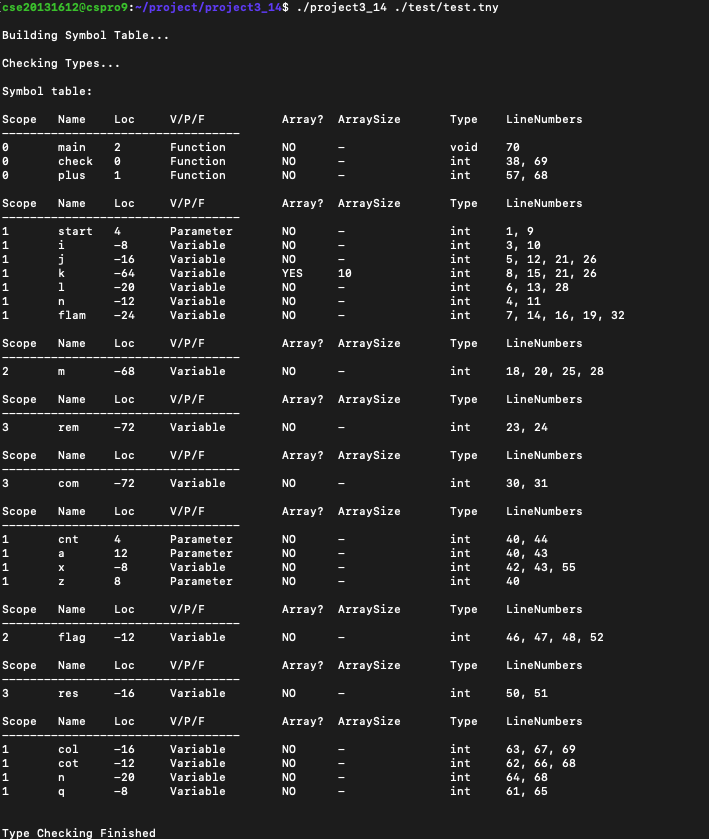


<함수 호출 시 parameter의 개수와 type이 일치하는지 check 출력>

1. Non-trivial 한 3 scope depth를 가진 error free 임의의 코드 test.tny

<실험을 위해 만든 예제테스트케이스 test.tny 입력>



<실험을 위해 만든 예제테스트케이스 test.tny 출력>

1. 기타
   1. 자유 기술

Symtab을 교재에서 제시된 BlockStructure로 만들기 위해 구조체들끼리의 연결 관계를 구성하는 부분에 신경을 많이 썼다. 그리고 Scope control을 위해 HashTableTop과 HashTableCurrent를 만들어 관리한 것이 편했다. 뿐만 아니라 type checking이나 여러 semantic error등을 해결하면서 발생할 수 있는 오류들을 더 명확하게 알게되고 3개의 프로젝트를 진행하면서 컴파일러의 전반적인 구조를 이해할 수 있었다.

* 1. 팀원의 역할 분담

김한길(50%), 최대운(50%): symtab의 구조를 잡는 것은 회의를 통해 같이 고민하고build symtab과 type checking을 나누어 개발했습니다. 보고서도 같이 작성했습니다.