**컴파일러 설계**

**프로젝트 3: Semantic analysis**

**2013012278 하경모**

**0. compilation environment**

- Ubuntu 18.04 LTS

- gcc 7.3.0

- report는 Microsoft word로 작성

**1. Semanic analyzer**

**1.1 목적**

- Semantic analysis를 통해 sematic error 찾기

**1.2 디자인**

- 두 번째 프로젝트 parser의 결과물에 맞게 심볼테이블 수정

- scope 개념 추가

- 그에 따른 symtab.c symtab.h analyze.c 파일 중점적으로 수정

- error detection에 초점이 맞춰져 있기에 symbol table print는 생략

- 그에 따른 Makefile 수정

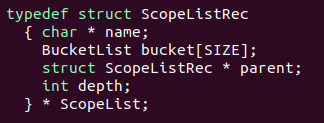
**1.3 구현**

- 기존 tiny에 없던 scope 관련 변수, 구조체 및 함수 추가

- .symtab.c와 syhmtab.h 수정

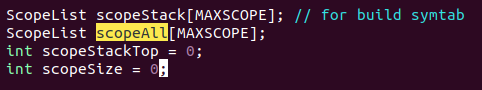
- 스택과 다뤄지는 방식이 알맞기에 스택으로 관리

- 스코프 구조체 ScopeListRec ScopeList



- 자신의 부모 스코프를 검색해야 하기에 parent 추가

- 관련 변수들

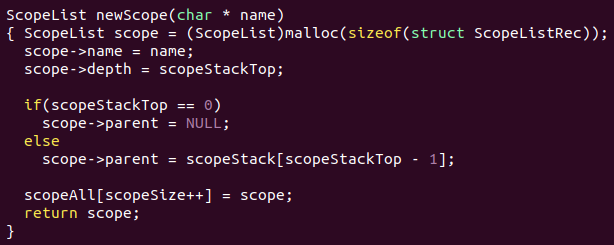


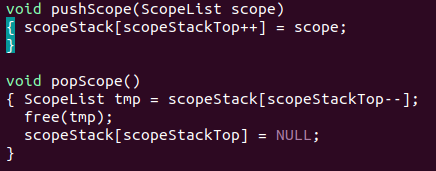
- 스택과 스택의 탑을 가리키는 변수 선언

- 스택 == 1이면 글로벌 스코프

- scopeAll과 scopeSize는 개발 도중 변경사항으로 인해 쓰이지 않음

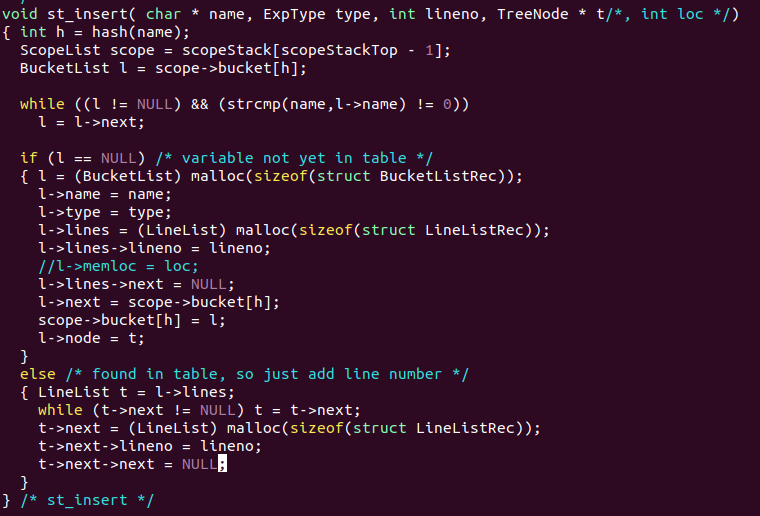
- 스코프 스택을 관리할 newScope, popScope, pushScope 등 구현

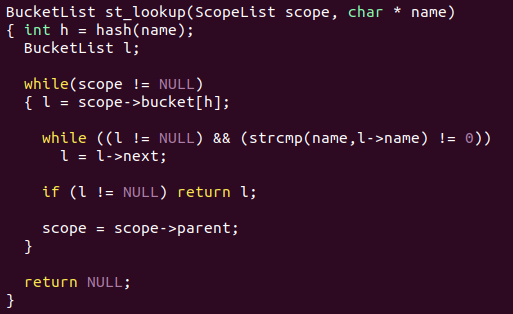


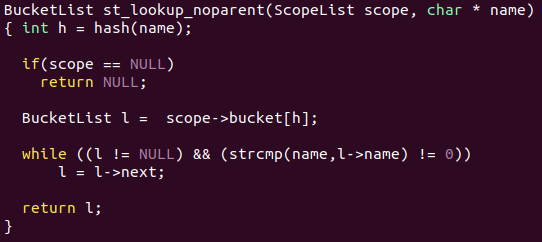


- 기존 tiny의 함수들을 프로젝트 2의 결과물에 맞게 수정

- 심볼테이블 추가와 검색 함수 수정







- 현재 스코프 스택이 위치한 스코프 내에서의 검색만 하는 함수도 추가

- symtab.c의 수정에 맞게 symtab.h도 수정

- 기존 tiny의 Bucketlist 구조체도 현 프로젝트에 맞게 수정

- 심볼테이블 구성 시 에러를 체크하고 구성하는 것으로 변경

- 즉, 기존 tiny의 insertNode와 checkNode를 insertNode 하나로 합침

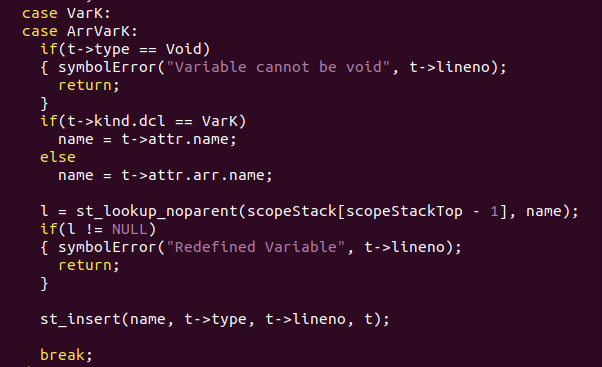
- 에러가 발생한 노드를 아예 심볼테이블에 삽입하지 않음

- 노드를 아예 추가하지 않기에 연쇄적으로 다른 에러가 발생할 수 있음

- 하지만, error detection이 초점이고 애초에 에러가 발생하면

입력된 코드 실행이 불가능하기에 이 방식으로 변경

- 수정된 insertNode 함수의 일부 첨부

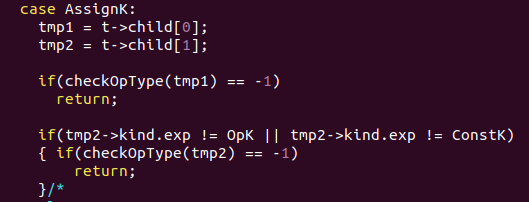


- 변수 선언 부분

- 보이드로 선언될 수 없으며 스코프 내에 같은 변수가

이미 선언되어 있으면 안되기에 error detection 추가

- 오류가 없다면 심볼테이블에 추가

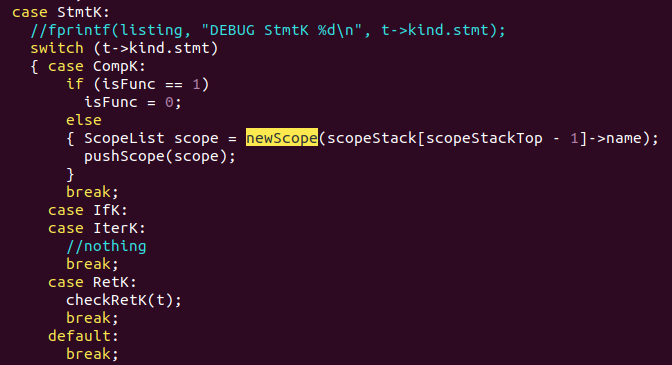


- 값 할당 부분

- checkOpType을 통해 타입체크

- checkOpType은 후술

- 선언 부분이 아니기에 심볼테이블에 추가하지 않음



- statement 부분

- compound statement 같은 경우 함수 선언 후 바로 나오는

중괄호는 아무 것도 하지 않기에 이를 스코프 추가 시

나오는 중괄호와 구분하기 위해 조건문 추가

- return statement 같은 경우 checkRetK를 통해 체크

- checkRetK는 후술

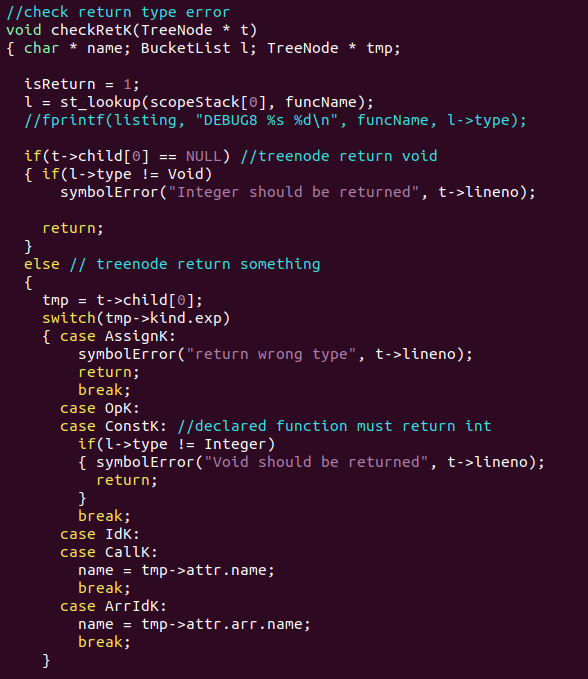
- checkOpType()



- 연산 시 적절하지 않은 타입이 연산되는지 검사

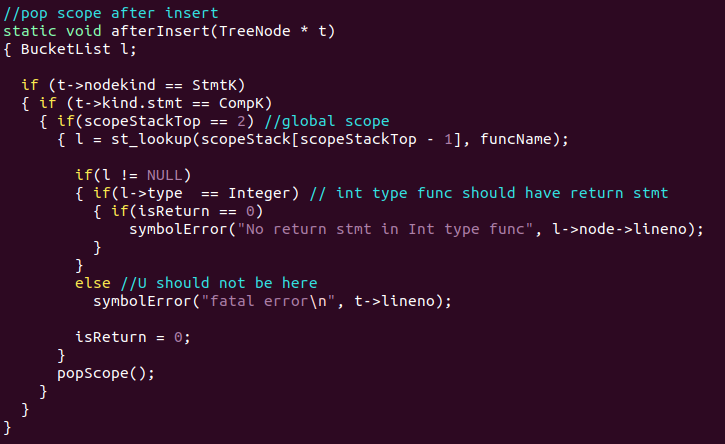
- checkRetK()

- 함수가 길어 일부만 발췌



- 적절하지 않은 리턴구문을 검사

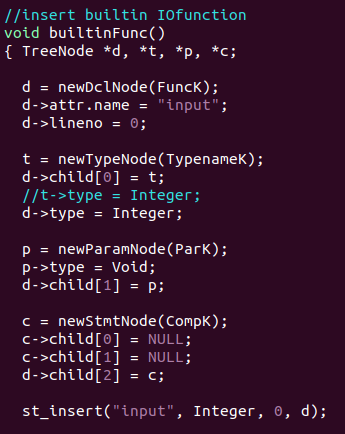
- 함수 하나가 끝나면 스코프를 pop해야 하기에 traverse 함수에서 insertNode 후 실행



- int 타입의 함수에는 return 구문이 있어야 하기에 검사하는 조건문 추가

- 이미 선언되었다고 가정하는 input()과 output() 함수 추가

- 함수가 길어 일부만 발췌



- 이 외의 몇 가지 함수를 주어진 조건 하에 적절히 수정

- 수정된 c파일에 맞게 헤더 파일도 수정

- 오류가 발견되면 해당 오류의 종류를 발생 위치와 함께 출력

**1.4 결과**

- 실행 방법

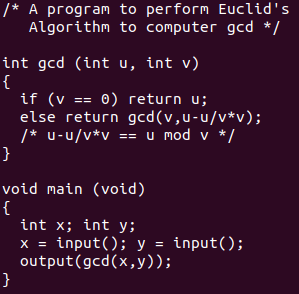
- make => cminus\_analyzer 실행파일 제작

- ./cminus\_analyzer sample/test.cm

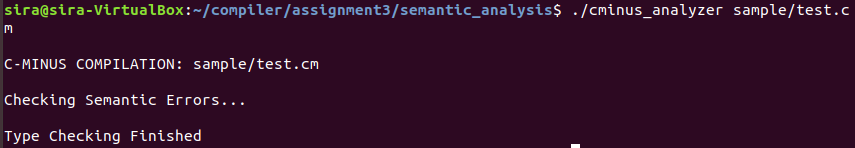
- test 파일들이 많은 관계로 폴더 내 sample 폴더에 위치

- 일부 예제 코드들의 실행 결과 첨부

- 예제 코드 (test.cm)

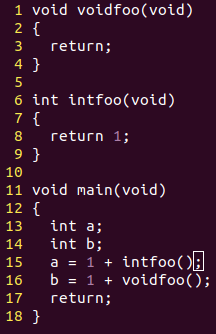


- 실행 결과

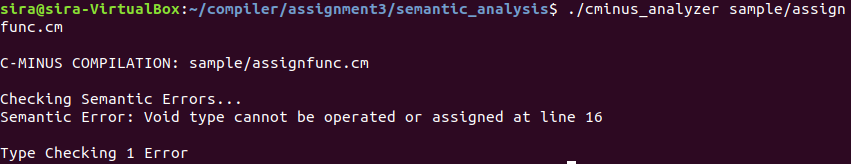


- 오류가 없는 코드이기에 오류를 출력하지 않음

- 예제 코드 (assignfunc.cm)



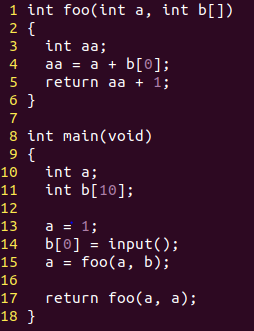
- 실행 결과



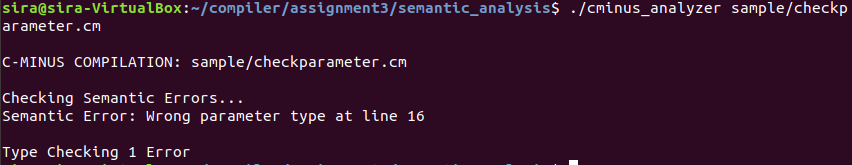
- line 15의 intfoo는 int를 반환하기에 연산이 가능하지만

Line 16의 voidfoo는 void를 반환하기에 연산이 불가능해 오류 발생

- 예제 코드 (checkparameter.cm)

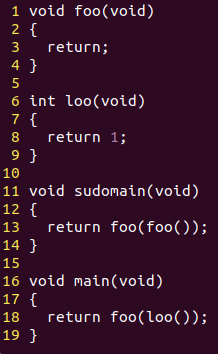


- 실행 결과

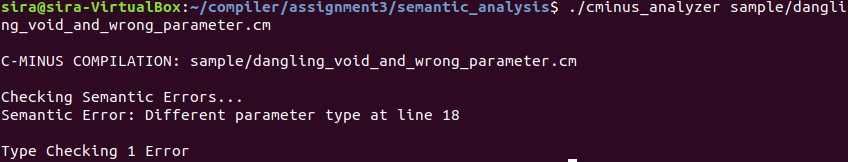


- line 15와 다르게 line 17은 배열을 인자로 전달하지 않아 오류 발생

- 예제 코드 (dangling\_void\_and \_wrong\_parameter.cm)



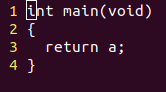
- 실행 결과



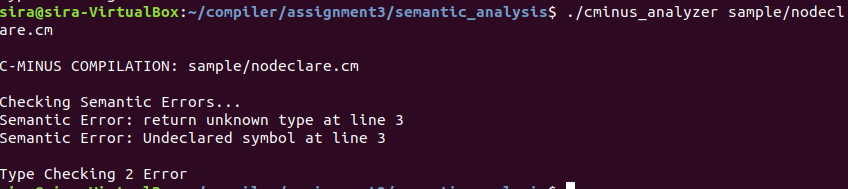
- line 13은 void타입의 인자와 리턴 값을 전달해 오류가 없지만

Line 18은 int 타입을 인자로 전달해주어 오류 발생

- 예제 코드 (nodeclare.cm)



- 실행 결과

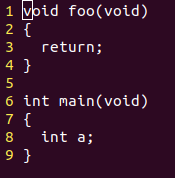


- 오류가 2개가 잡힘

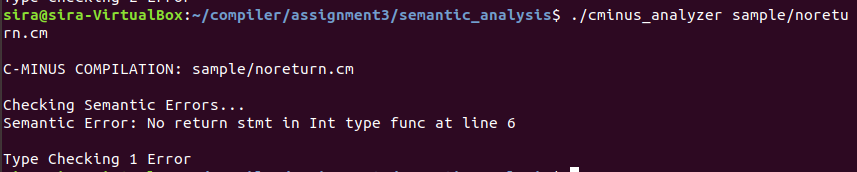
- 선언되지 않은 a라는 변수를 리턴 구문에 활용하고, 리턴 안의 구문에서도

활용했기에 오류가 추가로 발생

- 예제 코드 (noreturn.cm)



- 실행 결과



- line 6의 main함수는 int 타입이어서 return이 있어야 하지만 없어 오류 발생

- 이 외 여러 예제 코드로 실행 결과 적절한 error detection 수행