**Design and Development of Compiler**

**For C- Language**

**(Design and Implementation of Semantic Analyzer)**

**결과 보고서**

**과목명: [CSE4120] 기초 컴파일러 구성**

**담당교수: 서강대학교 컴퓨터공학과 정 성 원**

**개발자: 15조 (20141540 심규진, 20141602 황기덕)**

**개발기간: 2019. 05. 23 ~ 2019. 05. 27**

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

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

**제출일 : 2019. 05. 27**

조원 : 20141540 심규진, 20141602 황기덕

**I. 개발 목표**

- C-Minus 언어를 위한 semantic analyzer를 구현한다.

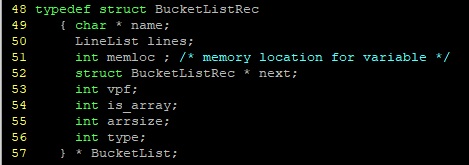
**II. 개발 범위 및 내용**

1. **개발 범위**

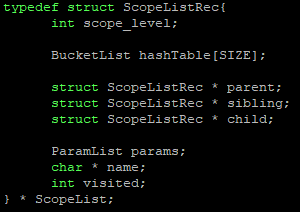
* 프로젝트 2에서 만든 syntax tree를 바탕으로 C-Minus 언어에 대한 semantic analyzer를 구현한다.

1. **개발 내용**

* C-Minus 언어에 맞는 semantic analyzer를 만들기 위해서는 추가적인 자료구조가 필요하다.
* symtab.c에 있는 BucketListRec 자료구조에는 다음의 변수들이 추가되었다.
  + - int vpf : variable, parameter, function를 구분하는 변수
    - int is\_array : array인지 아닌지를 확인하는 변수
    - int arrsize : array size의 정보를 갖는 변수
    - int type : type의 정보를 갖는 변수



* symtab.c에 있는 ScopeListRec 자료구조를 선언해주었다. 이 자료구조는 각 scope level에서의 symbol들을 저장하기 위한 Bucketlist와 scope에서 필요한 정보를 갖고 있다. 또한, 함수인 경우 parameter들의 정보를 저장하기 위한 ParamList를 가지고 있다. scope level에 따라 parent와 child로 연결하였고 같은 scope인 경우 sibiling으로 연결해주었다.
  + - int scope\_level : scope level의 정보를 갖는 변수
    - BucketList hashTable[SIZE] : symbol들을 저장하기 위한 hash table
    - struct ScopeListRec \*parent, \*sibling, \*child : scope level에 따른 연결
    - ParamList params : parameter의 정보를 갖는 구조체



**III. 추진 일정 및 개발 방법**

1. **추진 일정**

-2019/05/23 : 명세서 확인 및 개발목표 수립

-2019/05/24 : symbol table 제작 코드 작성

-2019/05/25 : type checking 코드 작성

-2019/05/26 : 코드 testing 및 보고서 작성

-2019/05/27 : 프로젝트 제출

1. **개발 방법**

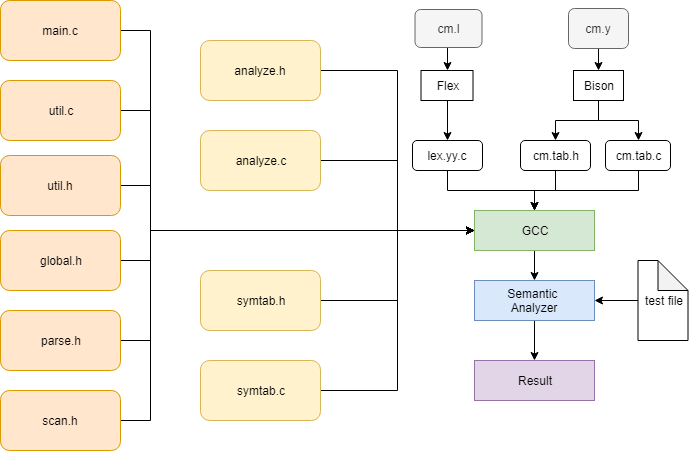
교재에 나와있는 symtab.h, symtab.c를 참고하여 semantic analyzer 역할을 수행하는 analyze.c 파일을 작성한다. 먼저 C-Minus언어에 대한 symbol table을 제작하는 코드를 개발한다. 이 단계에서는 scoping error에 대한 에러를 check한다. 두번째로 variable과 function에 대한 type checking을 수행하는 코드를 작성한다. C-Minus언어에서는 void, integer, array, function에 대한 checking만을 수행한다. type checking을 수행하는 과정에서 type checker는 name equivalence를 이용하여 두개의 expressions가 같은 type을 나타내는지 확인하여야 한다. 이 단계에서는 type check뿐만 아니라 그 외 semantic error들에 대한 checking도 수행한다.

**IV. 연구 결과**

1. **합성 내용**

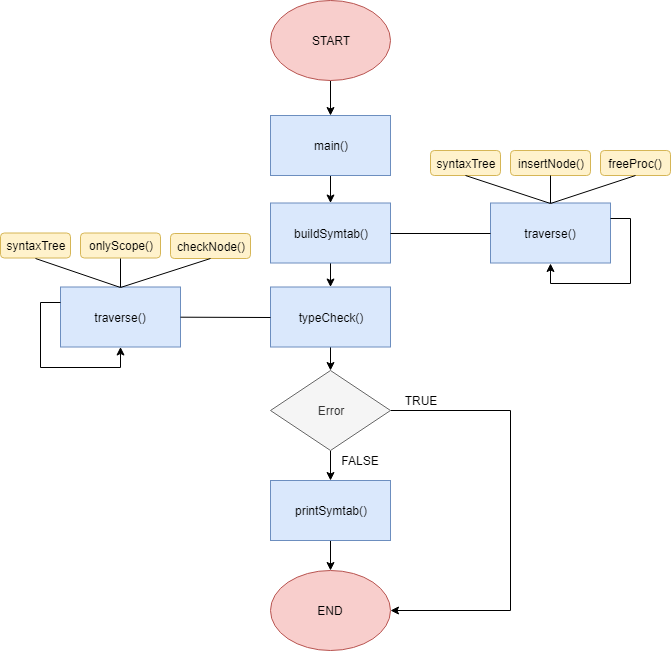
프로젝트2번 LALR Parser로 만든 syntax tree를 기반으로 Symbol table 생성 및 semantic analysis를 진행한다.

**[Semantic Analyzer의 파일 구성도는 다음과 같다]**



1. **분석 내용**

**[Semantic Analyzer의 구성도는 다음과 같다]**



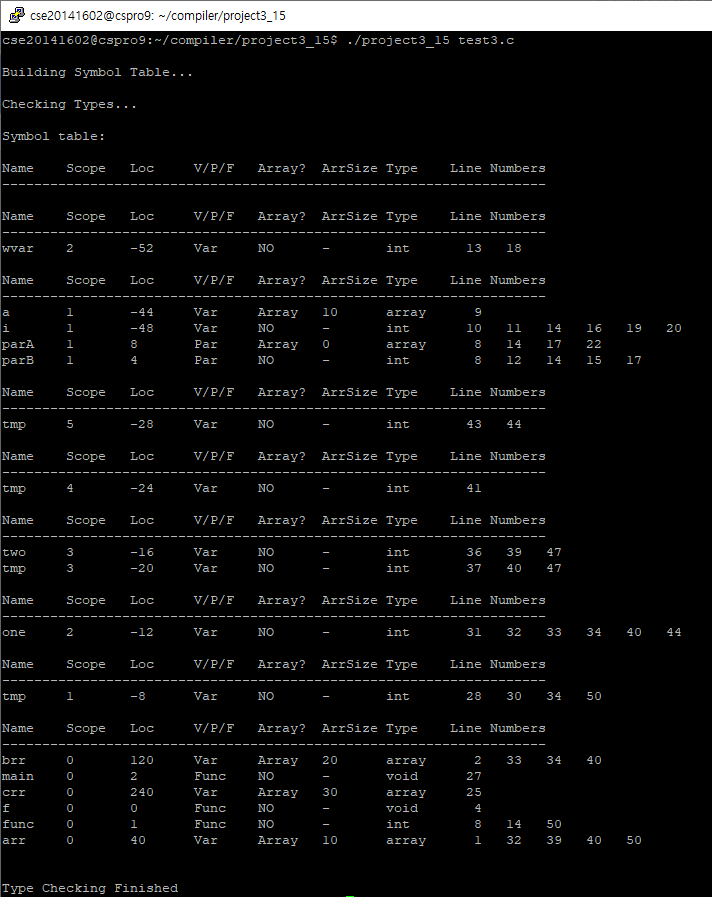
* 우선 main 함수에서 2차 프로젝트에서 만든 syntax tree를 buildSymtab함수로 넘겨준다.
* traverse 함수는 그림과 같이 recursive하게 호출된다. 첫번째 argument로 syntax tree를 받고, 2, 3번째 arugment로 함수포인터를 인자로 받아서 해당 함수를 진행한다.
* buildSymtab() 함수에서는 preProc으로 InsertNode 함수를, postProc으로 nullProc 함수를 넣는다.
* typeCheck() 함수에서는 preProc으로 onlyScope(scope을 계산하는 함수), postProc으로 checkNode 함수를 넣는다.
* Error가 발생하지 않으면 symbol table을 출력하고 프로그램을 종료한다.

1. **제작 내용**

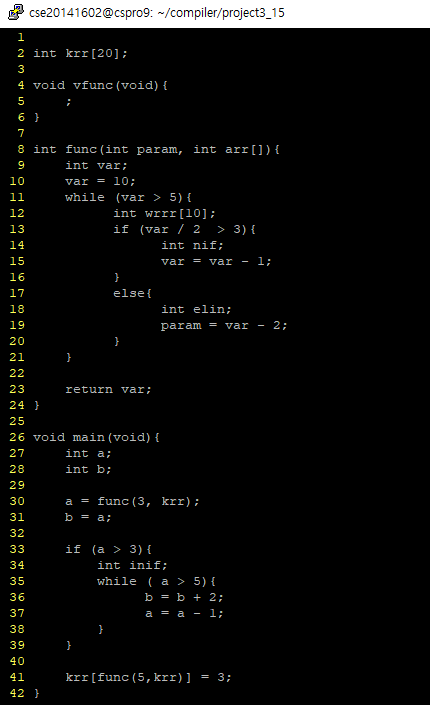
* **예시 1번**



**<<예시1번 semantic analyzer 수행 결과>>**



* **test.tny (자체 제작 코드)**



**<<자체 제작 코드 semantic analyzer 수행 결과>>**

