**Compiler project2(Parser) Report**

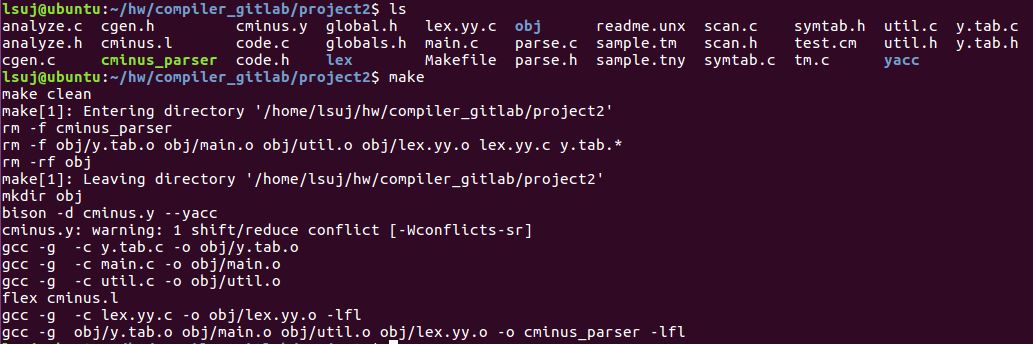
2013011112 소프트웨어전공

이 수 종

● **Compilation method and environment**

**<Compilation Environment>** - **OS** ( Ubuntu 16.04 - 64bit ) **gcc** 5.4.0

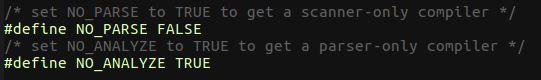
**<Compile method>**



project2 폴더에서 make하면 자동적으로 make clean을 먼저 수행하고 make을 수행하여 cminus\_parser 파일이 생성됩니다. 만일 처음 make을 수행하여 make clean할 파일이 없다면 무시하기 때문에 문제가 발생하지 않습니다.

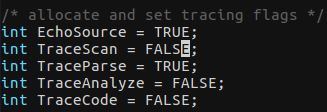
● **Explanation about how to implement and how to operate**

**<main.c>**



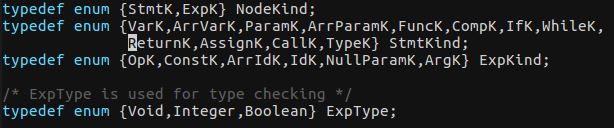
parser-only compiler를 만드는 것이 목표이므로 NO\_PARSE를 FALSE로 define하고 NO\_ANALYZE를

TRUE로 define하였습니다.

****

parse의 결과만 얻으면 되므로 TraceScan을 FALSE로, TraceParse를 TRUE로 변경하였습니다.

**<globals.h>**



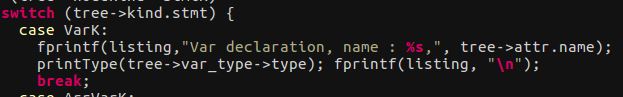
Cminus로의 변경에서 추가된 grammar를 위해 새로운 Kind를 추가하였다. 기존의 큰 틀을 유지

하기 위해 StmtKind와 ExpKind구조를 그대로 사용하였습니다. 생성 시점에 해당 kind에 대한 명

확한 값이 정해지는 경우는 ExpKind로, 그렇지 않은 경우에는 StmtKind로 분류하였습니다.

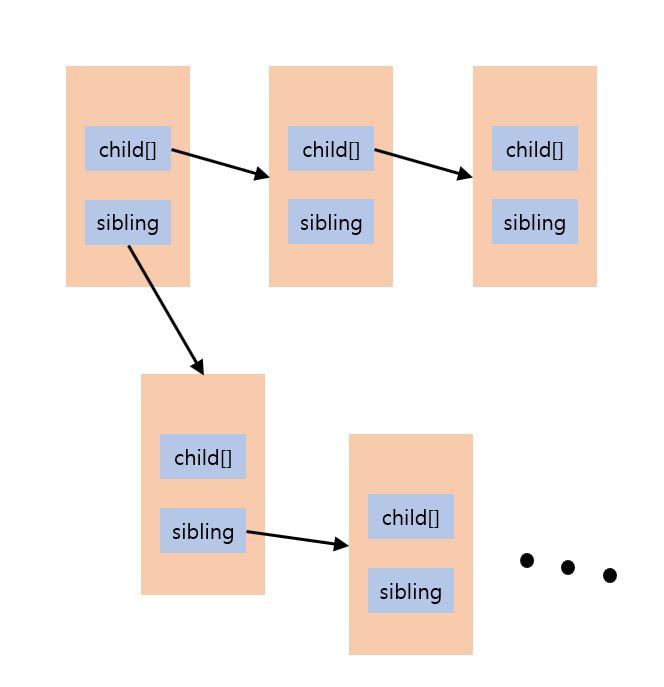
**<util.c>**

생성된 tree를 terminal에 print하기 위해 기존의 printTree함수에 기능을 추가하였습니다.

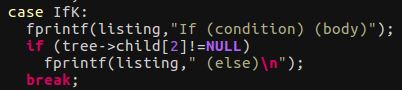
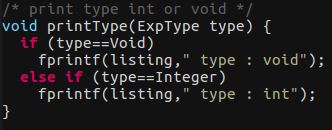


기본적인 구조는 위와 같습니다. tree의 각 node들은 StmtKind와 ExpKind로 분류되는 kind값을

가집니다. 이에 따라 적절한 형식에 맞춰 정보들을 출력합니다.



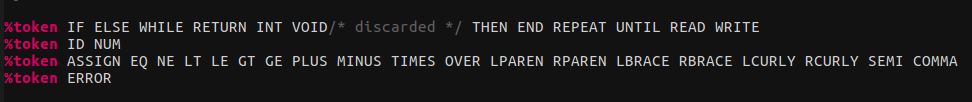
생성된 트리는 기본적으로 이와 같은 모습을 하고 있습니다. 한 node에서 child[]는 해당 node를 구성하기 위해서 필요한 요소들이고, sibling은 큰 구문들을 이루는 node들입니다. 즉 sibling이 NULL일 때 종료됩니다. 이를 위해서 printTree함수는 하나의 node에 대한 출력을 완료하면 모든 child[]에 대해서 탐색을 하고 다음 sibling으로 넘어가는 행동을 반복합니다.



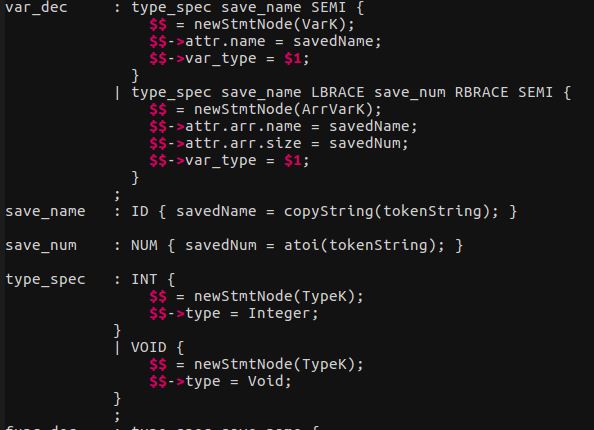
이처럼 위와 같은 방식으로 해결하기 힘든 경우 tree내의 정보에 직접 접근하여 node에 대한 정

보를 출력하도록 하였습니다.

**<cminus.y>**



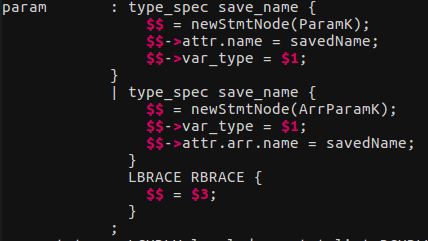
이전 project1에서 추가했던 것과 같이 bison에서 사용할 token들을 정의하였습니다.



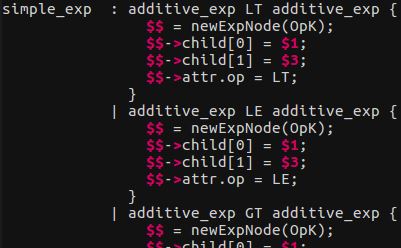
ID의 tokenString을 가져올 때 ID { get\_string } SEMI의 형식으로 가져오니 계속해서 ID다음의

token에 대한 string을 가져오는 오류가 발견되었습니다. 이를 해결하기 위해서 ID를 save\_name

으로 단계를 나눠 tokenString을 가져오도록 하였습니다.



이전 tiny-C와는 다르게 Cminus에서는 param, param[]의 두 가지 형식이 모두 지원됩니다. 이를 위해 ParamK와 ArrParamK로 kind를 나눠주었습니다.



기존 Appendix A.2에는 simple\_exp : additive\_exp relop additive\_exp | additive\_exp의 형식으로 정

의 되어 있습니다.

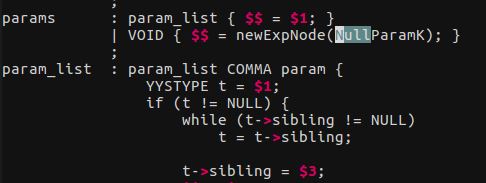


만약 기존의 정의대로 구현한다면 relop은 이런 두 가지 방식 중 한 가지 방식으로 구현이 될 것

입니다. 첫 번째 방법은 따로 $1이 token이기 때문에, 두 번째 방법은 LE가 non-pointer이기 때문

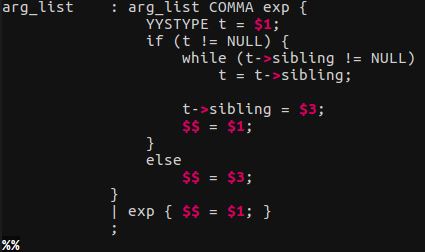
에 오류가 납니다. 또한 OpK의 kind로 node를 만들었기 때문에 해당 node는 op에 대한 정보를

담고 있는 것이 맞다고 생각해서 이렇게 디자인하였습니다.



function에서 parameter에 void가 들어오는 경우가 있습니다. 이런 상황에 대해서 정보를 출력하

도록 해주고 싶었고 이를 위해 NullParamK라는 kind를 새로 추가하였습니다.



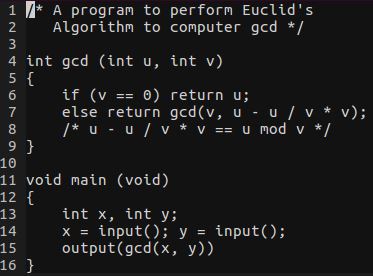
재귀적으로 자기 자신으로 호출하는 grammar입니다. 위와 같이 while문을 돌면서 sibling에 계속

적으로 요소를 추가시켜주는 것을 볼 수 있습니다. 이런 경우에는 계속해서 파생되는 node들이

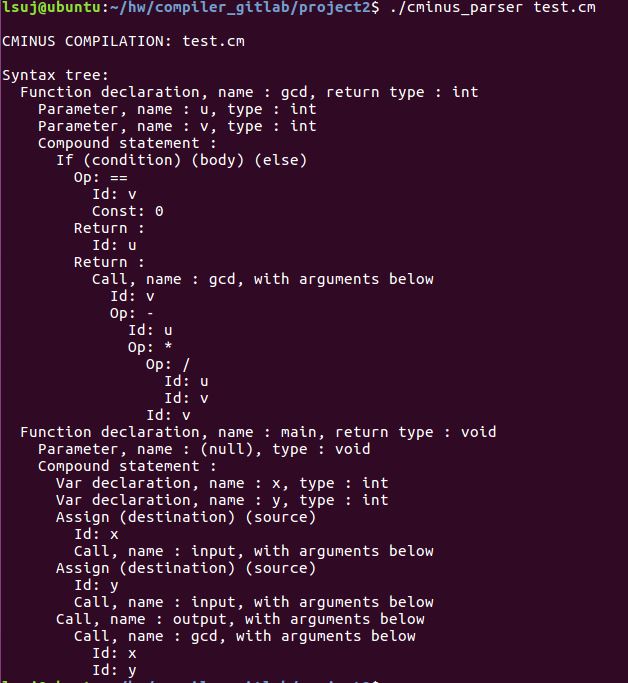
parent와 child의 관계가 아닌 동등한 관계에 있기 때문에 sibling에 추가시킵니다.

● **Example and Result Screenshot**

**<Example>**

****

**<Result>**

****