Compiler Project  
-parser-

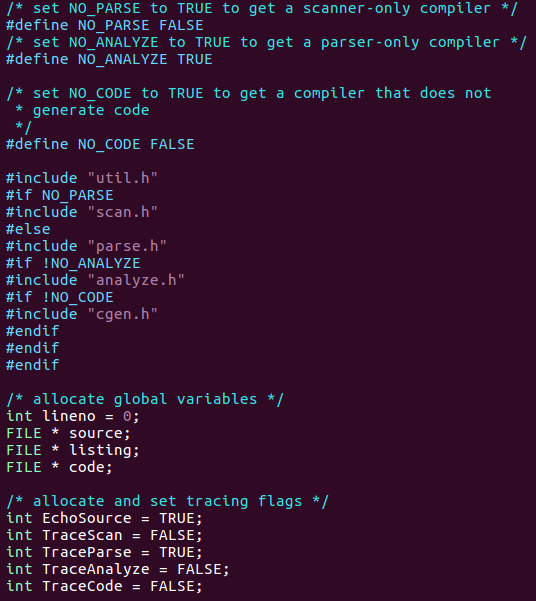
컴퓨터전공

2012003330 김선욱

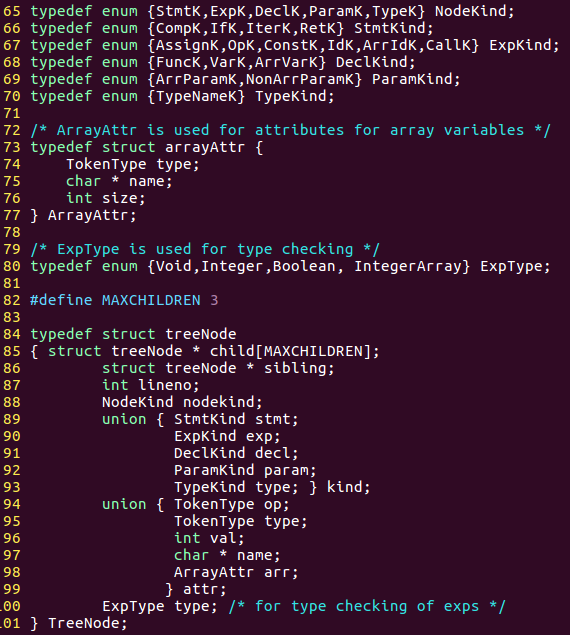
1. 목적

C-minus컴파일러의 파서(syntax) 부분에 대한 구현을 실습하며 파서에 대해 익힌다.

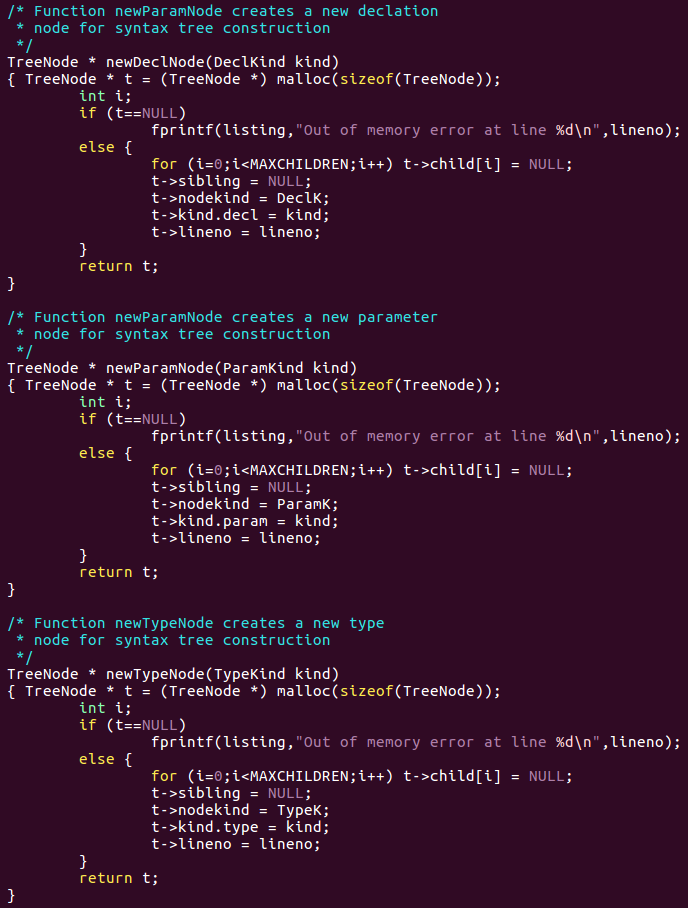
1. 환경
   1. OS : Ubuntu 16.04.1-desktop-amd64
   2. Compiler: cc 5.4.0
   3. Yacc/bison (GNU Bison) 3.0.4
   4. Makefile을 통해 컴파일
2. Tiny Compiler의 코드 수정
   1. Main.c



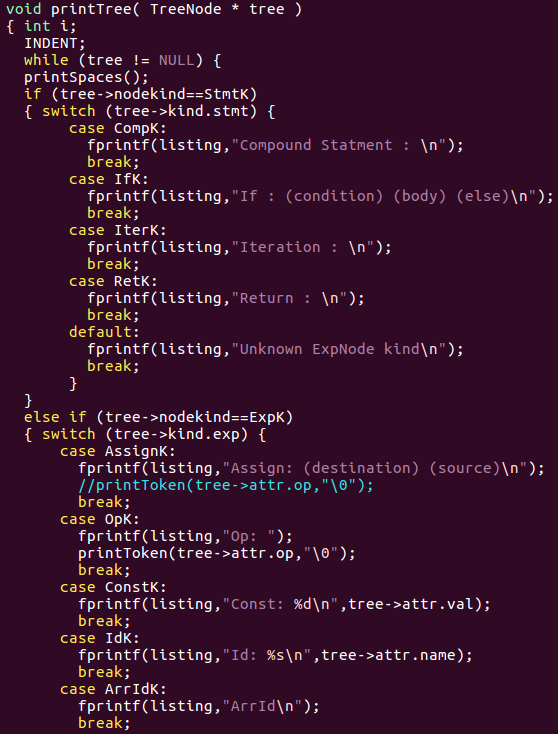
1. NO\_PARSE의 값을 FALSE로 수정
2. TraceScan 값을 FALSE로 하여 스캔 출력 안하도록 설정.
3. TraceParse 값을 TRUE로 하여 파싱 된 트리 출력 하도록 설정.
   1. Globals.h



1. 선언 형식에 따라 필요한 syntax tree의 형식을 예약한다.
   1. Statement
   2. Expression
   3. Declaration
   4. Parameter
   5. Type
2. 배열 선언 시 배열의 선언 상태를 담을 구조체 선언
3. Expression 타입에 따른 값을 예약한다.
4. 트리노드에 구성되는 요소들을 새로 선언한 syntax tree의 형식에 맞추어 수정하여준다.
   1. Util.c



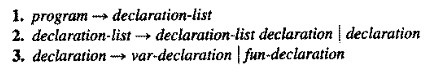
* + 1. 트리 구성에 새롭게 추가된 선언, 파라미터 등에 대해 TreeNode를 생성하는 코드를 추가하여 준다.

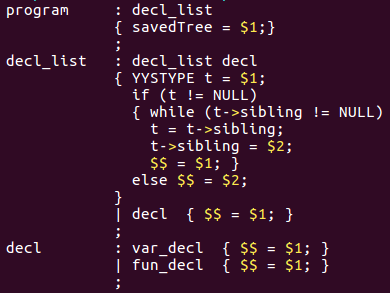


* + 1. printTree 함수 부분에 파싱된 트리가 원하는 형식으로 출력되도록 수정한다.  
       (캡쳐는 이하 생략)
       1. 각 노드 타입에 맞게 분기되어 타입 내의 종류로 다시 분기 되어 출력

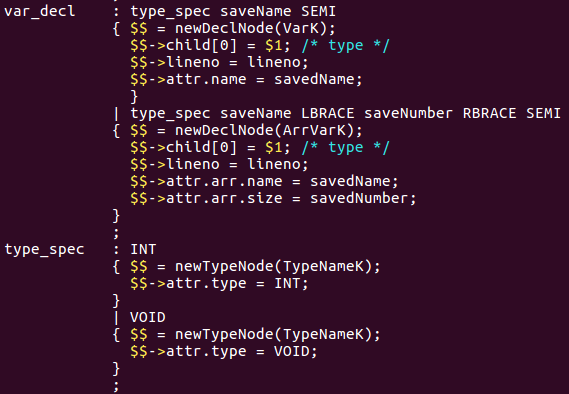
1. yacc활용을 위한 cminus.y 파일 생성

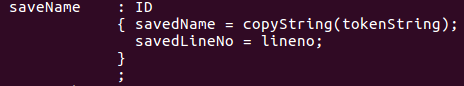
주어진 c-minus 문법에 맞게 코드를 생성한다.

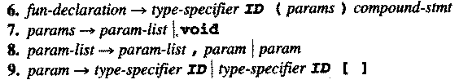


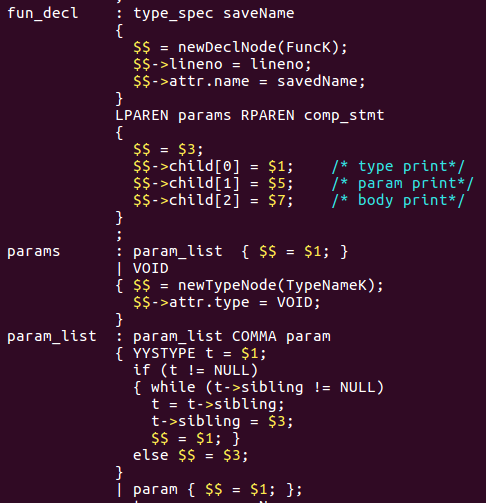


프로그램과 첫 선언문에 대해 도입부를 설정한다. Decl\_list 선언 종류를 decl에서 변수인 var\_Decl과 함수 선언인 fun\_decl로 나눈다.



변수 선언에 해당되는 구문으로 일반 변수선언과 배열 변수선언을 구분하여 사용한다. ID의 경우 바로 받을 경우 토큰의 값이 밀려 받아지므로 savename이란 추가 토큰을 통해 미리 문자열을 받아둔다.  
  
타입 설정으로는 INT와 VOID에 분기되어 값이 설정된다.

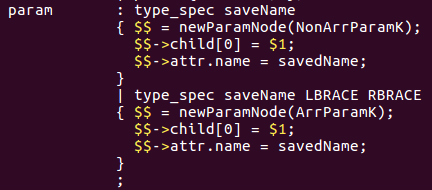




함수 선언 부에 대해 함수 이름이 토큰으로 받아질 때 밀려 받아지는 경우가 있으므로 saveName에서 한번 분기하여 name을 저장하고 다음 토큰을 받는다. Function node의 자식 노드로 순서대로 타입, 파라미터, 바디를 담는다.

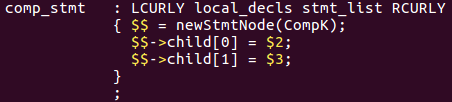
파라미터는 파라미터 리스트 또는 void, 없는 형태 두가지로 분류된다.

파라미터 리스트는 다시 COMMA를 통해 파라미티들이 구분되어 넣어진다.

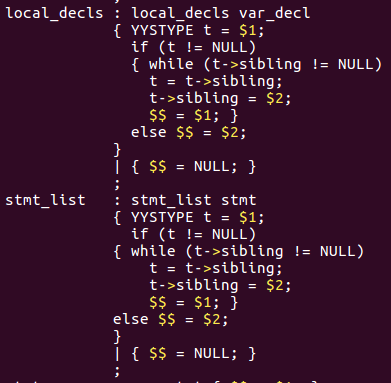


파라미터들은 일반적인 변수형태 또는 배열의 형태로 받아지므로 해당 파라미터에 알맞게 노드를 생성하고 설정해준다.

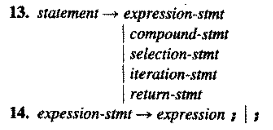


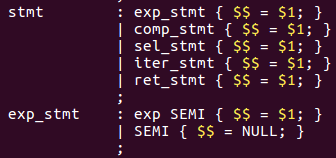


함수의 몸체가 되는 compound-statement로 중괄호로 감싸여진 형태를 갖는다.

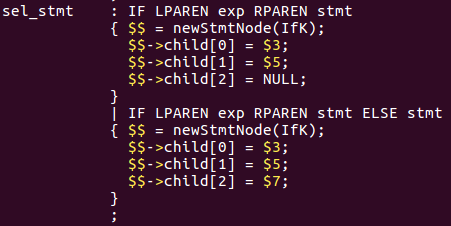


지역변수 선언이 되는 부분과 실제 로직이 실행되는 statement 들의 집합으로 이루어진다.

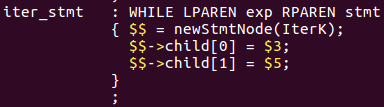




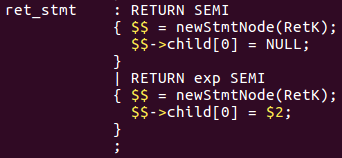
Statement는 각 expression, compound, selection, ieration, return statement로 나누어 지고 이중 expression statement는 “expression;” 형태로 나타난다.



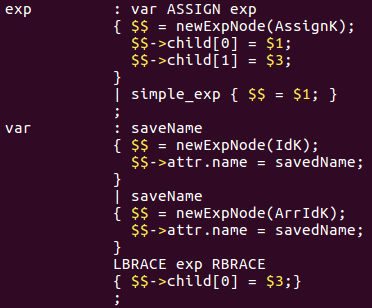
Selection statement는 기존의 IF문과 같은 형태로 If문으로 끝나는 형태와 else문이 이어져서 나타나는 형태 두가지로 구성할 수 있다. 이때 else문이 있는가에 따라서 트리 노드의 자식이 달라진다.



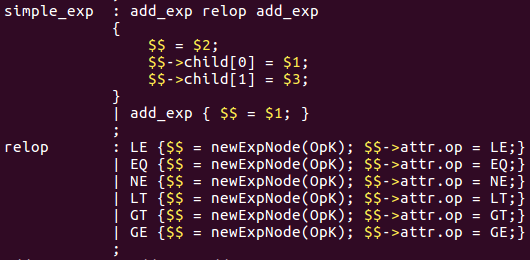
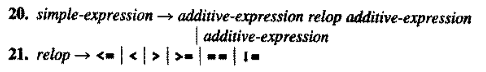
Iteration statement 는 while문으로 이루어진 반복문 형태이다.



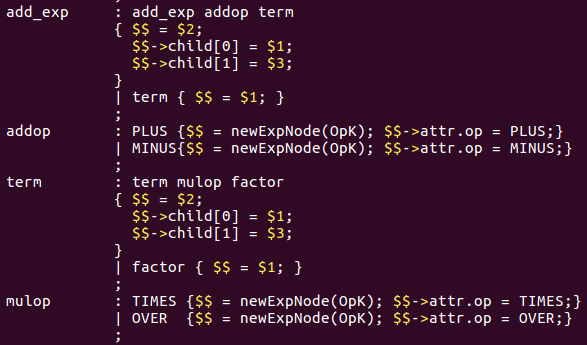
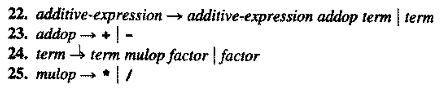
Return statement는 일반적인 return의 형태를 갖으므로 expression이 있을 수도 없을 수도 있다.



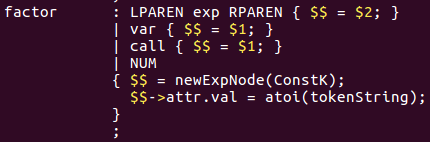
Expression이 구성되는 형태와 대입식으로 되어있는 ASSIGN 구문에 대해 일반 변수와 배열 변수를 다르게 표현한다.



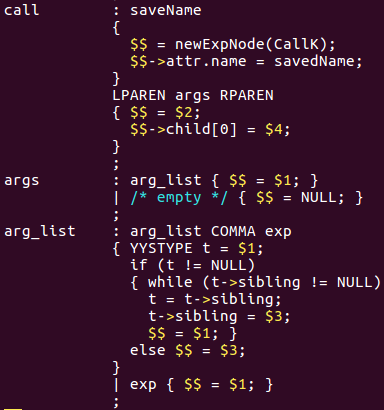
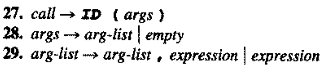
Simple-expression은 간단한 비교구문으로 이루어져 있다. Relop에 해당하는 비교 연산자에 따라 구분된다.



Add expression으로 이루어진 문법의 경우 plus와 minus만 있고 term 이란 부분에서 mulop 연산자 곱하기와 나누기 연산자를 처리해 주는데 이렇게 따로 처리하는 것은 이후 연산자 우선순위 문제와 연관이 있다고 생각된다.



Factor는 expression의 결과, 함수호출, 상수 등의 표현에 사용되는 문법이다.



Call은 함수 호출에 해당하는 문법으로 함수 호출 명과 인자 부분을 따로 처리하는 것은 토큰이 밀리는 것을 방지하기 위함이고 CallK에 해당하는 expression 트리노드를 생성한다.

인자에 해당하는 args는 다시 인자 리스트 또는 인자없이 처리되는데 이는 함수 선언과 마찬가지로 인자 리스트는 COMMA를 통해 구분 된다.

1. 실행결과
   1. Makefile을 통하여 Make 하면 cminus 프로그램이 output으로 나온다.

