**Design and Development of Compiler**

**for C- Language**

과목명 : 기초 컴파일러 구성 : [CSE4120]

담당 교수 : 서강대학교 컴퓨터공학과 정성원

개발자 : 20120121 이희락

개발기간: 2019. 04.29 - 2019. 05. 03

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

**Phase 2 : Design and Implementation of LALR Parser**

**제출일 : 2019년 5월 3일**

**개발자 : 이희락**

**1. 개발 목표 :**

C- 문법을 바탕으로 LALR parser를 만들고, parsing 결과로 만들어지는 Abstract Syntactic Tree 출력하는 PrintTree() 함수를 구현한다.

**2. 개발 범위 및 내용**

**가. 개발 범위**

i) C- 언어를 LALR parsing Tree를 만드는 tiny.y 파일 작성.

ii) printTree()함수를 구현을 포함하고 있는 util.c 파일 수정.

**나. 개발 내용**

C- 언어를 타깃으로 삼는 Parser를 만든다. 입력은 Lexical Analyzer 호출로 얻은 C- token들이며, 결과로서 Abstract Syntactic Tree를 만든다. printTree()함수를 구현하여 해당 AST를 traversing하여 tree 구조를 콘솔에 출력한다.

**3. 추진 일정 및 개발 방법**

**가. 추진 일정**

4월 29일 : C- 문법 파악

4월 30일 ~ 5월 2일 : tiny.y 파일 작성, printTree() 구현

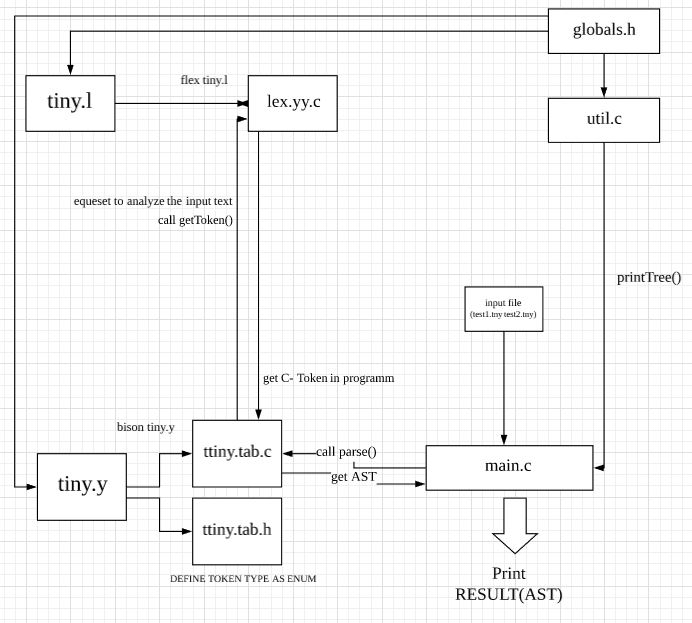
5월 3일 : 최종 test 및 debugging

**나. 개발 방법**

c- 문법 표현 및 문법에 따른 action을 명세하고 있는 tiny.l 파일을 작성한 후 이를 bison을 통해 compiler 한다. 이를 통해 lexical analyzer 호출을 통해 token을 받아 AST를 만드는 parser를 구현한다. util.c에 정의한 printToken()을 이용하여 만들어진 트리를 보여줄 수 있도록 했다.

**4. 연구 결과**

**가. 합성 내용**

****

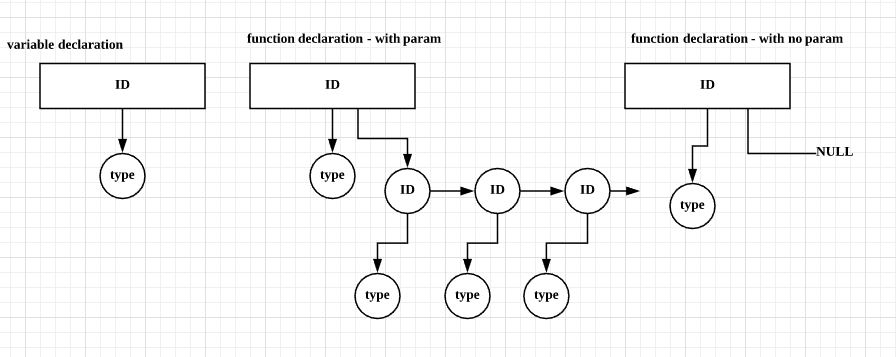
**<그림1. 프로그램 구성도>**

bison을 이용하여 tiny.y로부터 생성된 tiny.tab.c는 기술된 action과 문법에 따라 AST를 산출하는 routine을 가지고 있다. AST 생성은 token을 필요로 하는데, 이는 lexical analyzer로부터 얻을 수 있다. getToken() 함수는 호출로 token을 얻을 수 있다. 결과로서 만들어진 AST를 출력하기 위해 main함수는 util.c에 정의된 printTree()함수를 호출하며, parser로부터 얻은 AST를 인자로 넘겨주어 이러한 기능을 수행할 수 있다.

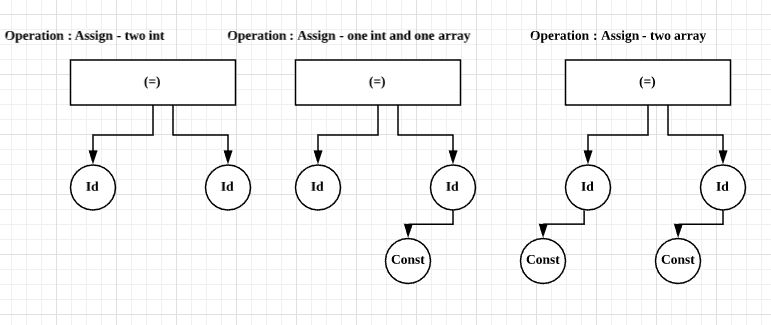
**나. 분석 내용**

**(i) AST를 구성하는 각 Node의 모습**

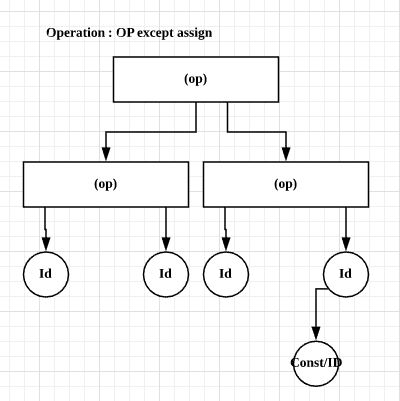
**<Declaration Node>**

****

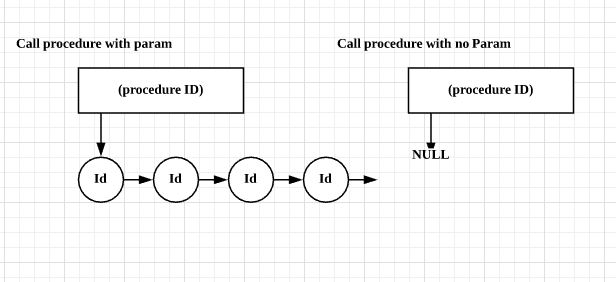
**<Expression Node - assign>**

****

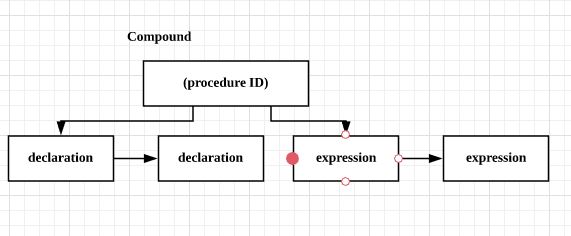
**<Expression Node – other operation>**

****

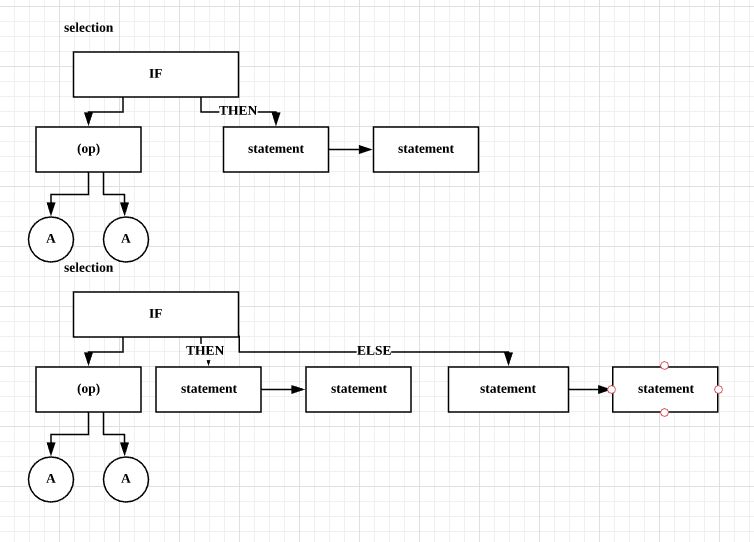
**<Expression Node – Call>**

****

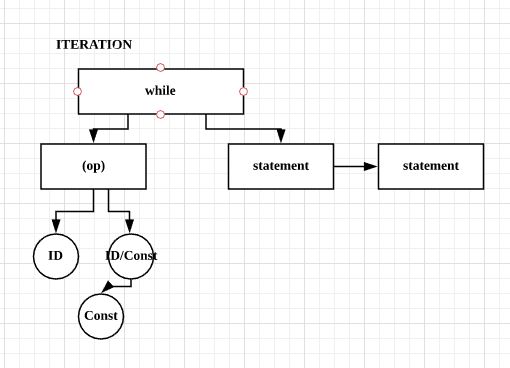
**<Statement - compound>**

****

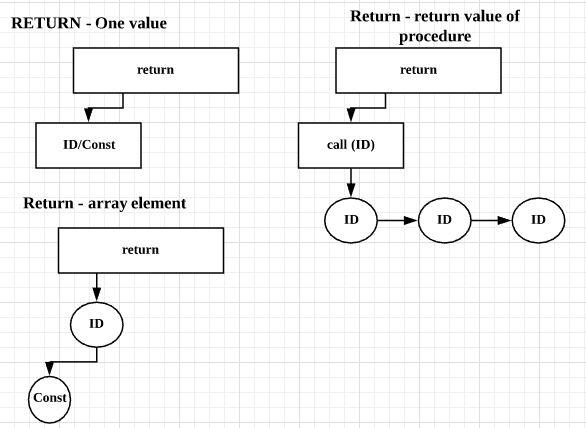
**<Statement - selection>**

****

**<Statement - Iteration>**

****

**<Statement - return>**

****

**(ii) printToken()의 동작** - printToken()의 동작 방법은 다음과 같다.

**(기본 원칙)**

(a) 각 노드의 sibling에 해당하는 node에 관한 출력은 indentation을 더하지 않는다.

(b) 각 노드의 child에 해당하는 node에 대해서는 indentation을 더하여 출력한다.

(c) 자식은 left child -> right child 순으로 출력한다.

**(선언부 - Declaration)**

변수 선언의 경우 *ID : 변수이름* 을 출력한다. 이후 indentation을 더해 변수의 type을 *Type : 타입* 형식으로 출력한다. 정수형 배열에 대해서는 다음줄에 *Const : 배열크기* 형식으로 배열임을 알려주며 그 크기 또한 출력한다.

함수 선언의 경우 *Function : 함수이름*을 출력한다. 이후 indentation을 더해 함수의 반환 타입을 *Type : 타입* 형식으로 출력한다. 함수의 인자에 대해서는 *Parameter : 인자 이름* 형식으로 이름을 출력한 뒤, indentation을 더해 각 인자의 자식에 해당하는 type을 *Type : 타입*형식으로 출력한다.

**(수식부 - Expression)**

*Op : 수식*(=, >=, == 등)을 출력하여 현재 노드가 어떠한 연산을 수행하는 구문인지 출력한다. 이후 indentation을 더해 자식으로 가지고 있는 두 Operand의 이름(상수의 경우 값)을 차례대로 출력한다. 만일 배열 타입의 자식을 가지고 있어서 한 배열의 원소에 대한 계산이라면 indentation을 더하여 자식으로 가지고 있는 해당 Operand의 index를 출력한다.

**(호출부 – Calling procedure)**

*Call Procedure : 함수이름*을 출력하여 현재 해당 이름의 함수가 출력되고 있는 부분임을 알려준다. 이후 indentation을 자식으로 지니고 있는 argument를 차례대로 출력하며 출력형식은 *ID : 인자 이름* 과 같다.

**(Statement – Compound Statement)**

*Compound Statement* 출력 후 줄 바꿈 하여 해당 부분부터는 compound statement 안의 내용임을 알린다. indentation 이후 compound statement 내부의 여러 statement를 앞선 규칙에 따라 출력한다. compound statement가 끝나면 indentation을 다시 줄인다.

**(Statement - Iteration)**

*While*을 출력하여 이후 이어지는 내용은 while 반복문에 부수적인 statement임을 알려준다. indentation을 더하여 곧이어 조건부를 출력하며 조건부는 수식부의 규칙과 동일하게 출력한다. 이어지는 while문 내부( {-내용- } )의 statement를 앞서 설명한 규칙에 따라 *compound* 출력 후 내용을 출력한다.

**(Statement - selection)**

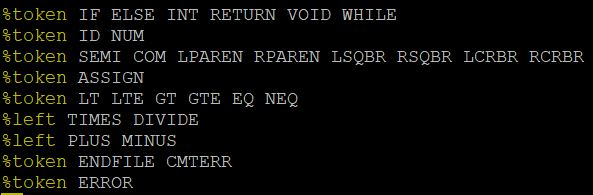
*If* 를 출력한 후 indentation을 더하여 수식부의 규칙에 맞추어 조건부를 출력한다. 이후 compound statement 출력 후 조건에 맞은 경우 수행되는 statement를 출력한다.

else문이 있는 경우 then 파트와 같은 indentation 레벨에서 else 부분의 statement를 출력한다.

**(Statement - return)**

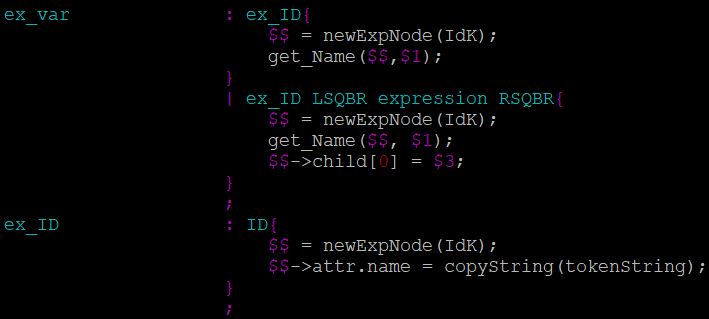
반환부의 경우 *return*을 출력하여 반환 부임을 알리고 indentation을 더하여 자식 노드에 있는 statement를 규칙에 맞추어 출력한다. 상수 혹은 변수 반환의 경우 *ID : 상수/변수이름*을 출력한다. 함수 호출을 통해 함수 반환 값을 반환하는 경우는 함수 호출 규칙에 맞추어 출력한다. 배열 원소 출력의 경우도 앞서 수식부의 배열 원소에 관한 처리와 같이 출력한다.

**(iii) 연산자 우선 순위 및 연산자 결합 법칙**

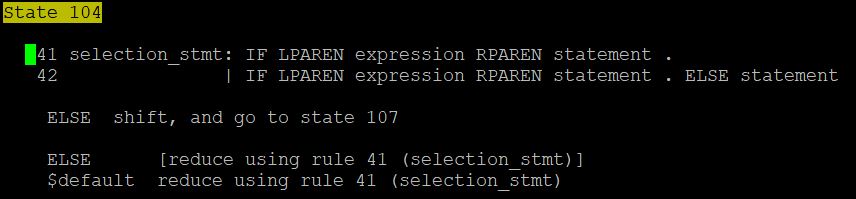
****

대입 연산자를 다른 비교 연산자보다 우선순위에 두어 대입 연산 이후의 값을 비교하도록 설정했다. 또한 일반적인 수학 계산 방법에 따라 곱셈과 나눗셈을 덧셈과 뺄셈보다 높은 우선순위에 두었으며 각각은 왼쪽부터 계산하도록 left directive를 두었다.

**(iv) conflict resolution**



ex\_ID 대신 ID를 두어 copyString을 한 경우 변수의 이름이 아닌 [이나 ;이 저장되어 원하는 값을 tree의 노드에 두지 못했었다. 이는 lookahead를 parsing하기 때문에 발생한 문제로 새로운 syntax symbol을 두어 제대로 된 값을 node에 두도록 설정하여 문제를 해결했다.



if 문과 if else 구문 사이에 shift reduce가 발생한 상황이다.

**다. 제작 내용**

**(a) error가 없는 경우. (b) error가 있는 경우(type 하나에 여러 변수선언)**

|  |  |
| --- | --- |
|  |  |
|  |

**라. 시험 내용**

가독성이 다소 떨어질 수 있으나 일관적인 규칙에 따라 출력하였으므로 매뉴얼을 확인한다면 출력결과를 통해 AST 구성을 충분히 파악할 수 있다. 또한 유효하지 않은 문법에 대해서는 앞선 결과처럼 syntax error를 출력하여 안정성을 높였다.