**#2 수식 인터프리터 개발**

소프트웨어학과 201720715 박주현

# 서론

* 1. 개요

이번 과제는 저번에 만든 lexical 분석기를 바탕으로 먼저 token들을 token list에 저장하고 recursive descent parsing을 이용해 syntax tree를 만든다. 이후 syntax tree를 바탕으로 간단한 수식 인터프리터를 개발하는 것이다.

* 1. 수식 계산기 설명

구현된 부분:

* 기본적인 수식 3+5
* 복잡한 수식 –(-2)+5\*2
* Assign을 통한 변수에 값 지정 val = 10
* 이 외에도 기본적인 모든 수식은 구현.
* 주어진 오류 메시지 출력
* Syntax tree 구현

구현되지 않은 부분:

* Syntax tree의 출력
* 초기화된 변수(a)에 초기화 되지 않은 변수(b)를 assign해줄 경우 변수 b에 대해 오류 메시지가 뜨고 변수 a에 b에 들어있던 쓰레기 값이 들어가게 된다.
* 멀티라인처리를 하지 못하였다.

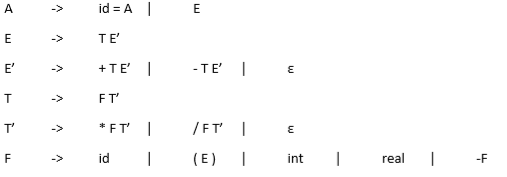
# 문제 분석

* 1. Grammar rule 분석

|  |  |
| --- | --- |
| **Rule** | **설명** |
| A -> idA’|F’T’E’ | Id가 나오면 A’를 부르고, 아닐 경우 F’T’E’를 차례로 확인한다 |
| A’ -> =A|T’E’ | Assign(=)이 나오면 A의 tree를 리턴하고, 아니면 T’E’의 tree를 리턴한다. |
| E -> TE' | E는 덧셈, 뺄셈 연산으로 순위가 가장 낮다. |
| E'->+TE' | -TE' | epsilon | parameter로 전달받은 tree와 F에서 반환된 tree를 두 child로 한 \* 노드  트리를 만들어 E’에 parameter로 전달  rhs E’이 반환한 tree를 반환 |
| T-> FT’ | F가 반환한 tree를T’의 parameter로 전달  rhsT’이 반환한 tree를 반환 |
| T'->\*FT' | /FT' | epsilon | parameter로 전달받은 tree와 F에서 반환된 tree를 두 child로 한 \* node를  만들어T’에 parameter로 전달  rhs T’이 반환한 tree를 반환 |
| F -> id | F는id 단말 node를 만들어 return 한다. |
| F -> F’ | F’가 반환한 tree를 return 받는다. |
| F’-> (E) | -F | INT | FLOAT | (), -, int, float, int노드를 반환한다. |

* 1. Grammar rule 분석

이때 제공받은 grammar을 처음 그대로 사용하지 않았다.



위의 사진이 처음 주어진 grammar인데 a=3이라는 입력이 주어질 경우 문제가 생긴다.

A->id=A에서의 id와 F->id에서의 id가 충돌이 생긴다. 따라서 위에 정의한 grammar

rule로 고쳐 진행하였다.

* 1. Recursive-descent parsing을 이용한 수식 계산기의 기본 개념

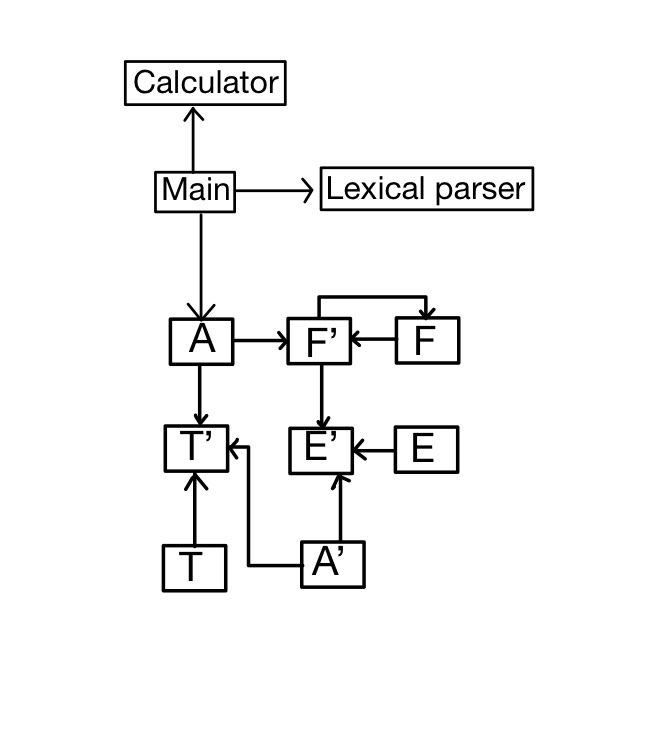
먼저 syntax analysis란 어휘 분석 과정에서 인식된 합성 문자 토큰의 그룹을 단위로 하여 구문의 타당성을 검사하는 과정이다. 과제의 기본 조건은 Recursive-descent parsing은 top-down형식의 parsing이다. 주어진 문법의 출발문자인 A로부터 시작한다. 즉 문장을 좌측에서 우측으로 한 문자씩 읽고 이를 leftmost derivation을 거쳐 구문을 분석한다.

먼저 덧셈과 뺄셈의 경우, 트리를 만들 때 뒤의 term을 하나 더 보고 operator를 두 term의 root로 삼는다. 곱셈과 나눗셈의 경우엔 제일 최근의 term의 값을 해당 operator의 왼쪽 자식으로 삼고, 다음 term을 오른쪽 자식으로 삼는다. 또한 계속 derivation을 거쳐 terminal을 leaf node에 놓는다. 이 방식을 사용하여 grammar rule의 operator의 precedence에 따라 구현하였다.

-F의 경우에서는 -를 root로 하여 왼쪽 자식에는 0의 값을 가진 노드가 오른쪽엔 F에서 생성된 tree를 자식으로 두어 -2 = 0-2로 구현하였다.

# 설계

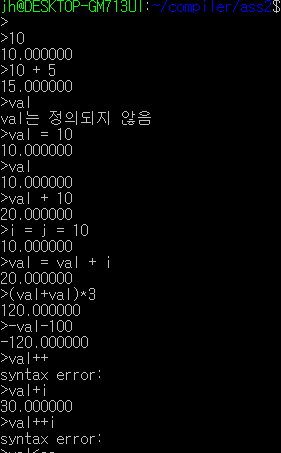
* 1. 주요 자료구조
* Abstract syntax tree를 사용하였다. tree에서는 leaf만 값을 가지고 나머지 node는 operator를 나타낸다. operator를 나타내는 경우에 child 두 개를 연산하여 해당 node의 값을 얻는다. 최종적으로 가장 root인 곳의 value값이 result값이 된다. 강의노트를 참조하였는데, 프로시저들이 tree를 리턴해주면 해당 프로시저는 자신이 호출한 프로시저들로부터 받은 tree를 합쳐주었다.
* token리스트와 symbol리스트를 배열로 선언하였고, 제공한 코드를 사용하였다.
* Makenode와 savetree함수를 이용해 노드를 생성하고 트리에 합치는 과정을 진행하였다.
  1. 프로그램 module 설명



먼저 main에서 lexical analysis를 수행한다. 이 단계를 통해 token\_list와 symbol\_table이 생성된다. 그 이후 A()를 호출한다. Token\_list의 순서대로 lookahead를 삼아 찾아나가며, 프로시저들을 호출하며 진행한다. 선들을 보면 어떤 함수가 어떤 함수를 호출하는지에 대한 dependency를 알 수 있다. 예를 들어 A는 A’과 F’T’E’를 호출하므로 연결되어 있다. 모든 syntax tree 생성 작업을 마치면 main으로 돌아와 error를 출력하고, calculator를 호출한다.

# 수행 결과

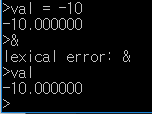
1. 아래는 과제에서 주어진 입력을 사용하였다.



1. 복잡한 input

1. Error 처리

# 고찰

저번 과제로 flex가 충분히 익숙해져서 괜찮았던 것 같다. 이번 과제를 통해 중간고사까지 배웠던 내용을 정리할 수 있었다. 이론으로만 접했던 top-down parsing을 직접 구현해보면서 부족했던 개념을 채울 수 있었다. 어려웠지만 배워가는 것이 많은 과제였다고 생각한다.