Programming Project #2: 수식 인터프리터 개발

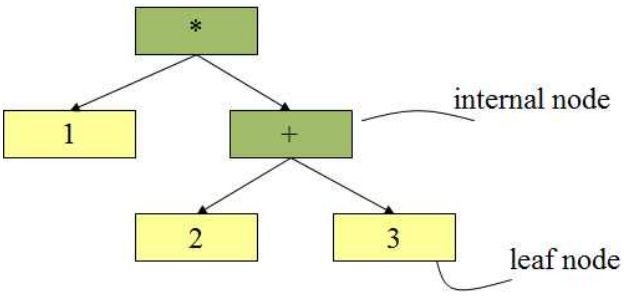
[문제]

Recursive Descent Parsing 기법을 이용하여 간단한 expression을 위한 인터프리터를 개발한다. expression은 아래와 같이 constants, variables, binary operators, unary operators, 지정수식(assignment expression)을 포함하며 괄호를 허용한다.

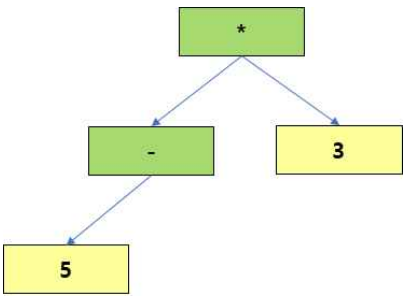
1) operator
   - binary opertor로 +, -, *, /, =(assignment)
   - unary operator: -
     e.g. -5 + -10 ==> -15
          --(5+10) ==> 15
          -(-(5+10)) ==> 15
2) 상수
   - integer
   - real number
3) assignment operator(=)는 right-associativity를 가진다.
   e.g. a = b = 10 ==> a = (b = 10) (결과 값은 10)
4) 변수, 상수 등 토큰의 패턴은 1차 과제에서와 동일하다.

인터프리터는 2단계로 이루어져 있다.,
- 1단계: Parsing 하면서 syntax tree를 생성한다.
- 2단계: 생성된 syntax tree를 계산하여 그 값을 출력한다.

예시1) 1 * (2 + 3)를 위한 syntax tree



예시2) -5 * 3를 위한 syntax tree



[입력형식 및 실행 예]

```
>                       # >은 식을 입력받기 위한 prompt임
>
> 10 + 5                # 수식 입력
15                      # 15는 실행 결과임
```

```
> 10
10
> val = 10              # value는 변수임. value의 값을 저장함.
10                      # 지정문의 변수 값을 출력
> val                   # 변수의 저장된 값을 출력
10
> val + 10              # 앞에서 저장된 변수의 값을 사용함.
20
> i = j = 20
20
> val = val + i         # 왼쪽 val의 값을 출력
30
> (val + val) * 3
180
> -val - 100
-130
> abc + 10
error: abc는 정의되지 않음
> value + + i
error: syntax error
> value << 10
error: lexical error
```

※ 하나의 수식은 하나의 라인에 입력되는 것으로 가정한다. 단, 멀티라인을 지원하고 싶다면 라인의 끝에 ₩를 붙여 연속적인 수식이라는 것을 표시하여 사용함(선택사항).

예) (a + b) * c를 두 라인에 입력한 예

```
> ( a + b ) ₩
  * c
```

[오류처리]

1. 초기화되지 않은 (즉, 처음으로 나타난) 변수를 사용하면 "정의되지 않음"이라고 출력
2. syntax가 틀리면 "syntax error"라고 출력
3. 어휘가 틀리면(즉, 잘못된 token이 사용되면) "lexical error"라고 출력

[힌트]

1. expression의 문법은 별도 공지한 문서를 참조한다.
2. 과제 1에서 만든 lexical analyzer를 축소하여 사용할 수 있다. 즉, 필요한 token 만 남겨 사용하면 됨.
3. 변수의 값은 symbol table을 확장해서 저장하면 됨.
4. 모든 것을 한꺼번에 해결하려고 하지 말고 점진적으로 확장하길 바람.
   단계 1) lexical analyzer를 개발

* parsing을 하면서 토큰을 하나 씩 만드는 것 보다 하나의 수식 라인을 입력받아 토큰 리스트로 모두 만든 후 parsing을 시작 하는 것이 더 편리

단계 2) recursive descent parser를 개발

단계 3) syntax tree생성하는 부분을 parser에 추가함

단계 4) evaluator를 개발함

4-1) 숫자만 입력하여 출력

4-2) 가감승제 수식 처리

4-3) assignment expression을 처리

5. 음수를 나타내는 부호(-)를 연산자로 처리하는 것이 쉬움

[보고서 구조]

1. 서론
   - 과제 소개
   - 구현된 부분과 구현되지 않은 부분을 명확하게 명시할 것.
2. 문제 분석
   - grammar rule 분석
   - recursive-descent parsing을 이용한 수식 계산기의 기본 개념정리
3. 설계
   - 주요 자료구조(syntax tree 등)
   - 프로그램 module hierarchy 및 module에 대한 설명
     (가능하면 modularization할 것)
4. 수행 결과 (화면 캡쳐)
   - 다양한 수식을 포함하도록 실행
   - syntax error가 있는 경우를 포함

주의) 소스코드는 첨부하지 않아도 됨

[제출방법 및 제출일]

- "hw2_학번" 디렉토리를 만들어 소스코드(*.c, *.h, *.l)와 보고서를 넣는다. 단, lex 컴파일러를 수행하여 나온 코드(lex.yy.c)는 넣지 않는다. **(주의: 실행파일 아님)**
- 아주Bb 과제게시판에 "hw2_학번" 디렉토리 전체를 압축하여 올릴 것. (파일명: hw2_학번.zip)
- 제출일: 2019년 5월 9일(목)자정 (보고서 출력본: 5월 9일(목) 수업시간에 제출)
- **주의: 제출기한 하루 초과시 5%감점. 제출기한 2일을 초과하는 경우 0점 처리**