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, assignment operator 등을 포함하며 괄호를 허용한다. 별도로 명시되지 않으면 C언어의 문법과 의미를 가진다.

1) operators
   - binary operator: '+', '-', '*', '/', '='(assignment)
   - unary operator: '+', '-'
     e.g. -5 + -10 == -5 - +10 == -(5+10)
   - operators의 우선순위는 C언어에서의 연산자 우선순위를 따른다.
   - string expression과 혼합되어 사용될 경우에도 연산자의 우선순위는 동일함
     e.g. 3 + "hello" * 3 => 3 + ("hello" * 3)
     3 + sub("hello",2,2) * 3 => 3 + (sub("hello",2,2) * 3)

2) constants
   - integer
   - real number (지수형 제외)
   - string

3) variables
   - 영문자로 시작하고 영문자와 숫자가 반복적으로 나타날 수 있음
     e.g. variables, compil2r, exampl2
   - 앞 10자리로만 구분함
   - integer, real number, string 값을 저장할 수 있음

4) numeric expression
   - int op int => int
   - int op real => real
   - real op real => real
     e.g. 3+3 => 3, 3+3.5 => 6.5, 3.5+3.5 => 7.0

5) string expression
   - string1 + string2 => string concatenation
     e.g. "abc"+"abc" => "abcabc"
   - string * n => n은 integer 값만 가능하며, string의 n번 연속적인 concatenation
     e.g. "abc" * 3 => "abcabcabc"
   - string1 / string2 => string2가 string1에 앞에서부터 반복적으로 나타나는 횟수

e.g. "abcabcabc" / "abc" => 3 , "abcabc123"/"abc" => 2

- string + integer (혹은 integer + string) => integer를 string을 변경한 후 concatenation
  e.g. "abc" + 123 => "abc123"
- string + real (혹은 real + string) => real을 string으로 변경한 후 concatenation
  e.g. "abc" + .123 => "abc.123"
- **sub**(string, n1, n2) => string의 n1 번째 문자부터 n2 개수만큼의 substring
  주의) 1차 과제와 다름. **sub**가 operator임
  e.g. sub("abc123",3,3) => "123" // ₩0을 붙여줌

5) assignment expression
   - var = expression
     e.g. hello3 = "hello" * 3

6) type checking
   - parsing에서는 변수 및 수식의 type을 체크하지 않는다.
   - 수행시 type을 체크하여 변환이 필요하면 변환한다.
     e.g. 3 + 3.5 => 3.0 + 3.5
     e.g. "abc" + 2.1 =>"abc" + "2.1"
   - type이 일치하지 않아 계산이 불가능하면 오류 메시지를 출력한다.
     e.g. 3 * "hello" => 오류 메시지 출력

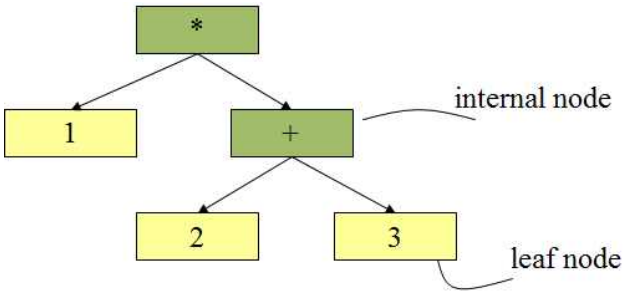
7) ( ) expression
   - '(', ')'로 묶인 expression
   - 괄호 속의 연산을 주변 연산보다 먼저 수행하도록 함
   - 괄호 안에는 모든 expression이 나타날 수 있다.
     e.g. -5*(3+4),, "abc"+("abc"), (sub((("abc123"))),(3),(5)))

[인터프리터 구조]

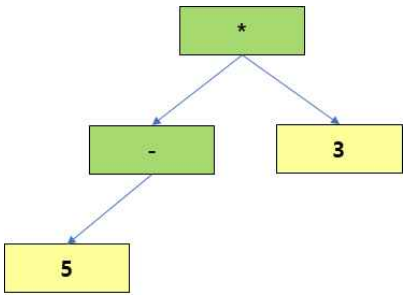
인터프리터는 2단계로 이루어져 있다.,

- 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



[오류처리]

1. 수행 중 오류가 발생하면 runtime error라고 표시하고 구체적인 원인을 출력함
   - 초기화되지 않은 (즉, 처음으로 나타난) 변수를 사용하면 "undefined variable"이라고 출력
   - 수식의 type이 일치하지 않으면(변환이 허용되지 않은 경우)
     e.g. "abc" * "3" => runtime error : string과 string은 *이 불가능함
     3 * "abc" => runtime error : 정수에 string을 곱하는 것은 불가능함

2. parsing 중 오류가 발견되면 "syntax error"라고 표시하고 구체적인 원인을 출력함
   e.g. sub(hello3, 5, 10 +
   => syntax error: sub(string, n1, n2)의 형식이 올바르지 않음

3. 어휘가 틀리면 "lexical error(위치)"라고 출력하고 해당 문자를 출력
   e.g. 3 + hello_3 - 3abc
   => lexical error(10): _ or lexical error(5): hello_3
   lexical error(15): 3 or lexical error(15): 3abc

[조건]

1. lexical analyzer는 lex(혹은 flex)를 이용하여 생성한다.
2. parsing은 recursive descent parsing 방법을 이용한다.
3. 다음과 같은 parser directive($문자로 시작)를 구현한다.
   > $ast #직전 하나의 올바른 문장의 abstract syntax tree를 출력한다. 올바른 문장이 없다면 아무런 출력을 하지 않음.
   > $symbol #직전 문장까지의 symbol table을 출력한다.

AST의 출력방식

- 트리를 root 레벨부터 시작하여 레벨 순으로 출력한다.
- 한 레벨의 노드들은 왼쪽부터 오른쪽 순으로 하나의 행에 출력한다.
- 하나의 노드는 하나의 문자열로 출력한다.
  . operator의 경우 operator와 arity(operand 수)값을 묶어 출력 (*2, =2, -1, 등)
  . constant는 그대로 출력한다. (120, "hello", 3.14, 등)
- 예시1)

```
> 1 * (2 + 3)
5
> $ast
*2
1 +2
2 3
>
```

- 예시2)

```
> -5 * 3
-15
> $ast
*2
-1 3
5
>
```

Symbol Table의 출력방식

- 각 symbol을 하나의 행에 출력한다.
- name, type, value를 출력한다.
- 예시)

```
> $symbol
name      type      value
val       int       30
i         int       20
sum       real      20.5
hello     string    "hello"
hello3    string    "hellohello"
>
```

[입력형식 및 실행 예]

```
>                       # >은 식을 입력받기 위한 prompt임
>
> 10 + 5                # 수식 입력
```

```
15                          # 15는 실행 결과임
> 10
10
> val = 10
10                          # 지정문에서 저장한 변수 값을 출력
> val                       # 변수의 저장된 값을 출력
10
> val + 10                  # 앞에서 저장된 변수의 값을 사용함.
20
> i = j = 20
20
> val = val + i
30
> (val + val) * 3
180
> -val - 100
-130
> hello = "hello"
"hello"
> hello3 = hello*3
"hellohellohello"
> hello3/"hello"
3
> sub(hello3, 5, 10)
hellohello
> 3 * hello
runtime error: number * string은 정의되지 않음
> abc + 10
runtime error: abc는 undefined variable
> i + j = val
syntax error : assignment의 왼쪽이 variable가 아님
> val << 10
lexical error: <<
>
>
> $ast       // sub(hello3, 5, 10)를 위한 syntax tree. sub 수식을 binary tree로 만들었다면
sub3         // 그 형식대로 출력하면 됨.
hello3 5 10
```

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

예) (a + b) * c를 두 라인에 입력한 예

```
> ( a + b ) ₩
  * c
```

[보고서 구조]

1. 서론
   - 과제 소개
   - 구현된 부분과 구현되지 않은 부분을 명확하게 명시할 것.
2. 문제 분석
   - grammar rule 분석
3. 설계
   - 주요 자료구조(syntax tree 등)
   - 프로그램 module hierarchy 및 module에 대한 설명
4. 수행 결과 (화면캡쳐)
   - 다양한 수식을 포함하도록 실행
   - syntax error 등의 error가 있는 경우를 포함

주의) 소스코드는 첨부하지 않아도 됨

[제출방법 및 제출일]

- "hw2_학번" 디렉토리를 만들어 소스코드(*.c, *.h, *.l), Makefile, 보고서를 넣는다. 단, lex 컴파일러를 수행하여 나온 코드(lex.yy.c)는 넣지 않는다.
- **Makefile의 수행결과로 생성되는 실행 파일의 이름을 "parser_학번.out"으로 한다.**
- 아주Bb 과제게시판에 "hw2_학번" 디렉토리 전체를 압축하여 올릴 것. (파일명: hw2_학번.zip)
- 제출일: 2022년 11월 17일(목) 자정 (보고서 출력본: 11월 17일(목) 수업시간에 제출)
- **주의: 제출기한을 하루 초과시 5%감점. 제출기한 2일을 초과하는 경우 0점 처리**