Assignment 09: Arithmetric

Date: 2024. 11. 12
Student ID: 201924451
Name: 김태훈

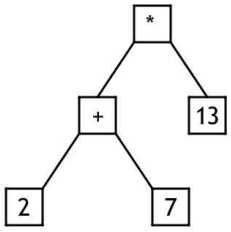
1. 산술식을 지원하기 위한 Lex 입력과 설명

```
%{
  #include   <stdio.h>
  #define    ILITERAL   301
  #define    SYMBOL     302
%}
iliteral     [-+]?[0-9]+
symbol       [a-zA-Z\+\*\-\/]
space        [ \t\n]+
comment      ;[^\n]*\n
%%
{iliteral}  { return ILITERAL;  }
{symbol}    { return SYMBOL;    }
"("         { return '(';    }
")"         { return ')';    }
{space}      ;
{comment}    ;
%%
int yywrap() {
    return 1;
}
```

literal 토큰은 숫자를 인식하고, 앞에 - 또는 +가 붙을 수 있으며, 0-9 숫자 여러개로 구성되어있다.
symbol 토큰은 연산자를 인식하고, +,*,-,/를 인식한다. 각 기호는 lex에서 특수기호이므로, 앞에 escape 문자(\)를 붙여 정의한다.
space는 공백( ), 탭(\t), 줄바꿈(\n)이 여러번 반복되는 것을 의미하며 이 토큰은 무시한다.
comment는 세미콜론(;) 뒤에 줄바꿈을 제외한 문자가 여러개 오고 마지막에 줄바꿈이 오는것으로 정의하고, 이 토큰은 무시한다.
또한 여는 괄호(() 와 닫는 괄호())를 인식하여 해당 토큰이 인식되었음을 알려준다.

(+ 3 5) 는 (+ (3,(5, NIL))) 로 처리되므로, 연산자는 result.pval->d[0].sval로 접근할 수 있고
첫 번째 피연산자는 result.pval->d[1].pval->d[0].ival,
두 번째 피연산자는 result.pval->d[1].pval->d[1].pval->d[0].ival 로 접근할 수 있다. 괄호가 중첩된 경우에는 재귀를 통하여 연산자와 피연산자를 구해야할 것이다. 여기서는 연산자 1개와 피연산자 2개로 이루어진 식 하나만 주어진다고 가정하였다. 연산자가 + 이면 두 피연산자를 더하고 - 이면 빼서, * 이면 곱해서, / 이면 나눠서 한 줄에 출력하였다.

2. 산술식을 나타내는 다른 방법[1]



위 그림과 같이 Tree를 사용하여 표현할 수 있다. 해당 트리는 아래 3과 같이 3가지 방식으로 해석할 수 있다.

3. 수식을 나타내는 세 가지 방식의 비교 표[2]

표현 방법	트리에서 해석	예	장점	단점
prefix	전위 순회 (루트 - 왼쪽 - 오른쪽)	* + 2 7 13	괄호가 필요없고, 중첩된 괄호가 있는 표현식을 처리할 수 있다. 연산자를 스택에 넣고, 피연산자가 나오면 연산자를 스택에서 꺼내어 계산할 수 있다.	사람이 이해하기 어렵다.
infix	중위 순회 (왼쪽 - 루트 - 오른쪽)	(2 + 7) * 13	사람이 읽고 이해하기 쉽다.	연산 순서를 지정하려면 괄호가 필요하다.
postfix	후위 순회 (왼쪽 - 오른쪽 - 루트)	2 7 + 13 *	괄호가 필요없다. 피연산자를 스택에 넣고, 연산자가 나오면 스택에서 꺼내어 계산할 수 있다.	사람이 이해하기 어렵다. 복잡한 수식의 경우 계산 추적이 힘들다.

References

[1] http://learn.hfm.io/expressions.html

[2] https://www.geeksforgeeks.org/infix-postfix-prefix-notation/