Title : Expression Binary Tree

Name :

Date :

수식트리 규칙 ① 연산자 = tree, subtree의 Root Node

② 피연산자 = Leaf Node

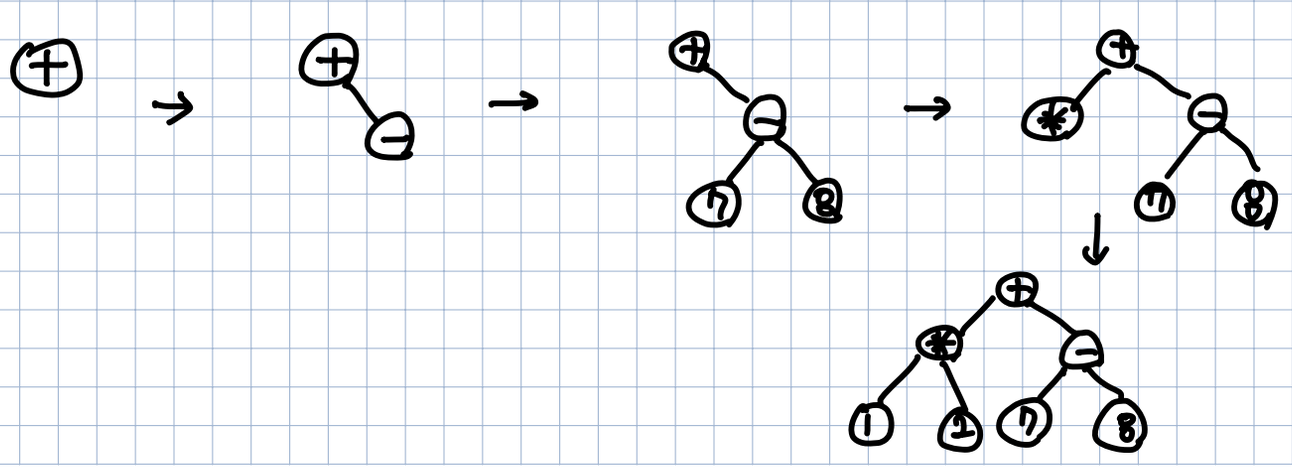
주로 후위표기식을 통해 수식 트리를 생성한다.

만드는 과정

1. 후위표기식 입력
2. 맨 뒤 문자 ~ 맨 앞 문자까지 조사.
   이때 가장 맨 뒤에 있는 연산자 = Root Node
3. 뒤에서 앞으로 진행하면서
   만약 token = 연산자 → 가지노드 (Root, SubRoot) 생성
   연산자가 등장하면 그 앞에 피연산자가 2개 있거나
   그 앞에 연산자가 존재함.
   ↳ 이 경우, 자식 노드로 불러버리고
   자식 노드로 이동해서 토큰 검사

후위표기식 반대방향으로 넣어야 하니까 Root → R → L

ex) 1 2 * 7 8 - +



Title : Name :

Date :

수식트리 생성

```
void BuildExpressionTree(후위표기식, Tree의 RootNode*){

    int len = strlen(후위표기식);
    char token = 후위표기식[len-1];
    후위표기식[len-1] = '\0'; //재귀로 구해야 해서 배열 두정해야 함

    switch(token){
        case +, -, /, *: //연산자인 경우
            (*Node) = 새 노드 생성(token)
            BuildExpressionTree(후위표기식, &(*Node)->Right);
            //Root → Right → Left 순서로 노드를 채워나감
            BuildExpressionTree(후위표기식, &(*Node)->Left);
            break;
        default: //피연산자 인 경우
            (*Node) = 새 노드 생성(Token);
            break;
```

앞에있는 피연산자 2개를 Build

여기에서 노드 생성해서 넣어주는거임!

Title : _____ Name : _____

Date : _____

수식트리 계산

수식 트리 생성시 Root → Right → Left 로 집어넣었으므로

계산 시 후위 순회로 돌리면 된다.

↳ 재귀 때문에 return value : Double

```
double Calc (ETNode* Tree) {
                ↳ Tree 전달
    char Temp[2]; // 계산

    double Left; // 왼쪽 value
    double Right; // 오른쪽 value
    double Result;

    if (tree == NULL) → 0 반환
    switch (Tree → Data):
        case +, -, /, * : //연산자인 경우 재귀
            Left = Calc (Tree → Left);
            Right = Calc (Tree → Right);

            연산자에 따라 연산 후 break;

        default:
            memset (Temp, 0, sizeof (Temp)); // Temp array 할당, 0으로 초기화
            Temp[0] = Tree → Data
            Result = atof (Temp) // char → float 변환 (double)
            break;

    return Result;
```