Lab8 Tree Build, Traverse & Evaluation

1. Node Creation:

```
template <class T> class Tree;
template <class T>
class Node {
private:    T data;
            int priority;
            Node<T>* left;
            Node<T>* right;
public:       Node(T value) : data(value),
              priority(4), left(0), right(0) {}
friend class Tree<T>;
};
```

```
template <class T>
class Tree {
public:
   Node<T>* root;
   void insertAsOperator(Node<T>* node);
   void insertAsOperand(Node<T>* node);
   Tree();    ~Tree();
   int evaluationPostOrder(Node<T>* node);
   void insert(T data);
   void inOrder(Node<T>* node);
   void postOrder(Node<T>* node);
   void preOrder(Node<T>* node);
   void buildTree(string expression);
};
```

2. Precedence Table(연산자 우선순위 테이블)

char prec[4][2] = { '*', 2,   '/', 2,   '+', 1,   '-', 1};

prec[i][0]	*	/	+	-
prec[i][1]	2	2	1	1

3. Main Program

1) Get mathmatical expression in numbers/characters (ex: 2+4*3, a*b-c/d)

2) Build Tree (expression)

3) Traverse tree (Inorder, Preorder, Postorder)

4) Output: Tree Expression

4. Details

1) Get math expression: program 에서 입력.

string exp1 = “8+9-2*3   string exp2 = “A/B*C*D+E”

2) Build Tree(expression)

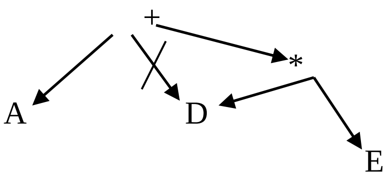
```
while (expression[i] != NULL) {   insert(expression[i]); i++ }
level = i;     // to print Tree
```

3) Procedure insert(data){

```
. create new-node
. for i=0 to <4   (if new-node-> data == prec[i][0])   then new-node->prio = prec[i][1])
. if (i==4)   then   call Operand(new-node)
              else   call operator(new-node)}
```

```
4) procedure Operand(new-node){
      if (root == NULL)    then      {root = new-node      return}
      P = Head
      while (p->right !=NULL)      p=p->right
      p->right = new-node
  }
```

```
5) procedure Operator (new-node){
        if (root->prio    >=    new-node->prio)
                  new-node->left = root
                  root = new-node
        else
                  new-node->left = root->right
                  root -> right = new-node
```



6) Traverse (Tree traverse algorithm 참조): Inorder, Preorder, Postorder

7) Tree Evaluation

```
procedure evalTree (Node* p)          {
      if    (p!=NULL)      {
            if (p->data in [0..9])    then    value = p->data-'0'
            else {
                  left = evalTree(p->left)
                  right=evalTree(p->right)
                  switch (p->data)    {
                      case '+':    value=left+right;
                      case '-':    value=left-right;
                      case '*':    value=left*right;
                      case '/':    value=left/right;
                  }
              }
          }
      return value;
  }
```

8) Tree Expression

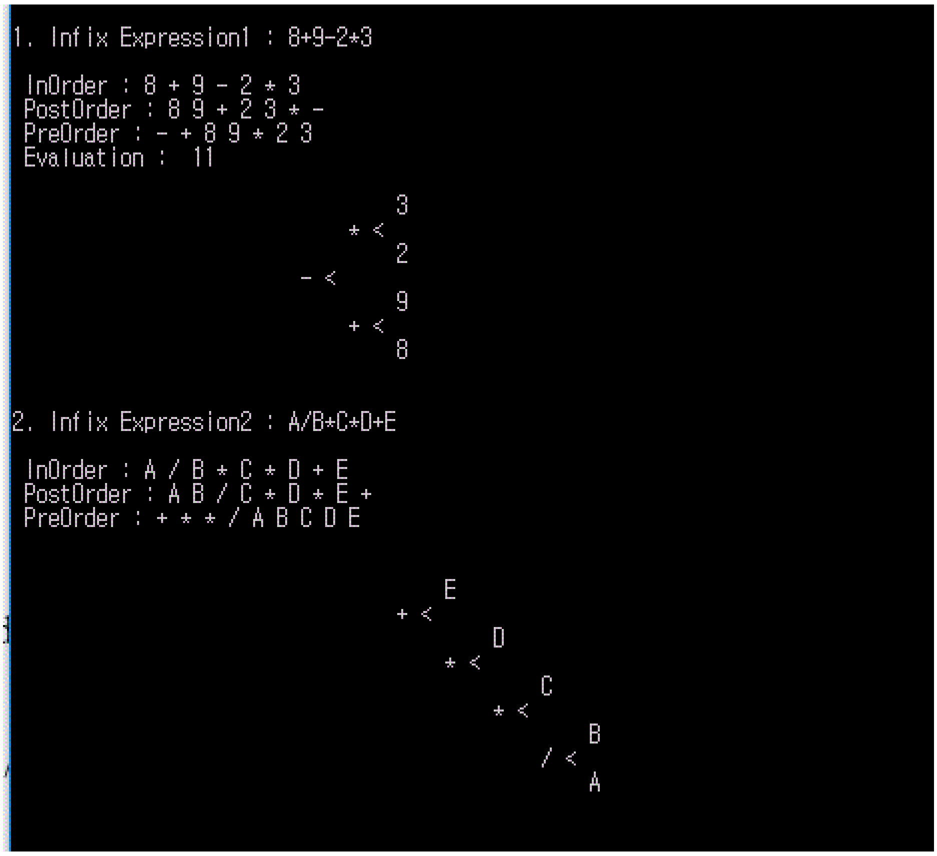
```
void Tree::PrintTree(Node* P, int level) {
   int j = 1;

   if (P != NULL) {
         PrintTree(P->right, level + 1);                    //Space over (skip levels)
         while (j++ < level)     cout << "        ";          // Print data
         cout << P->data;

         if (P->left != 0 && P->right != 0)    cout << " <";        //two child
         else    if (P->right != 0)                   cout << " /";          //only right child
         else    if (P->left != 0)                     cout << " \\";         //only left child
         cout << endl;

         PrintTree(P->left, level + 1);
   }
}
```

5. Output:



```
1. Infix Expression1 : 8+9-2*3

 InOrder : 8 + 9 - 2 * 3
 PostOrder : 8 9 + 2 3 * -
 PreOrder : - + 8 9 * 2 3
 Evaluation :  11

                                3
                            * <
                                2
                        - <
                                9
                            + <
                                8


2. Infix Expression2 : A/B*C*D+E

 InOrder : A / B * C * D + E
 PostOrder : A B / C * D * E +
 PreOrder : + * * / A B C D E


                                    E
                                + <
                                        D
                                    * <
                                            C
                                        * <
                                                B
                                            / <
                                                A
```