```
*********************************************************************************************
* Name : 차윤성
* Student Id : 20163162
* Program Id : Hw#4
* Description
* - 키보드로부터 수식을 입력받고 Tree를 build한다.
* - build된 Tree로 부터 기본 traverse(inorder, preorder, postorder)를 진행한다.
* - build된 Tree로 부터 수식의 값을 도출한다.
* - build된 Tree를 그려낸다.
* Algorithm
* - 입력받은 수식의 연산자 또는 피연산자를 저장할 구조체 Node를 만든다. Node 내에는 data값이 피
*   연산자인지 연산자인지 판단할 기준이 되는 prio와 data의 child와 연결될 포인터 변수 left와 right
*   가 있다.
* - Tree를 구현할 class를 선언해준다. class에는 Tree의 상위 ancestor를 의미하는 포인터변수 root를
*   선언하고, 최초 tree객체를 생성하는 생성자 함수에서 tree가 비어있으므로 root를 0으로 설정해준다.
* - 모든 검사에 대해 root == 0 일 경우에는 tree 객체가 비어있다는 의미이다. 반복적으로 검사하는 경
*   우가 많으므로 isEmpty함수를 만들어준다.
* - data를 tree에 삽입 시에 먼저 node->prio를 검사하여 data가 연산자인지 피연산자인지를 검사한다.
* - data가 피연산자일 경우 node->prio를 4로 설정하고, 연산자일 경우 각 연산자의 우선순위가 설정되
*   어있는 배열 prec을 토대로 연산자에 맞는 우선순위를 설정해준다.
* - data가 피연산자일 경우 먼저 tree가 비어있을 경우 node를 root로 설정한다.
* - tree가 비어있지 않을 경우 가장 오른쪽 leaf에 node를 넣어준다.
* - data가 연산자일 경우 node->prio보다 root->prio가 클 경우 root를 node로 바꿔준다.
* - node->prio가 root->prio보다 클 경우 root->right에 새로운 node를 넣어주고 기존 root->right에
*   있는 node는 새로운 node의 left로 이동시킨다.
* - inorder는 tree객체의 root를 매개변수로 받은 후 tree의 left leaf를 먼저 출력한 후, 출력된 node
*   의 parent를 출력한 후, 출력된 parent의 right child를 출력한다.
* - postorder는 tree객체의 root를 매개변수로 받은 후 tree의 가장 높은 level의 left로 이동 후
*   node->left가 0일 경우 right로 이동한다.
* - 가장 높은 level의 node->right도 0일 경우 data를 뽑은 후 root로 이동한 뒤 가장 높은 level의
*   right에 있는 node들도 같은 방식으로 출력한다.
* - preorer는 tree객체의 root를 매개변수로 받은 후 tree의 root를 먼저 출력 후 level을 한 단계씩 올
*   리면서 node->left의 부분을 출력한 후에 node->right의 부분을 출력한다.
* - evalTree는 tree내의 있는 data들을 연산해주는 함수로, 먼저 tree가 비어있는지 아닌지 검사한다.
* - tree 객체 내에서 root->right의 leaf node들이 연산순위가 가장 높은 node들 이므로 먼저 연산하기
*   위해 recursive를 이용해 left를 evalTree(ptr->left), right를 evaltree(ptr->right)로 설정해준다.
* - 각각의 검사에 대해 node가 leaf node일 경우 value를 leaf node->data로 설정한다.
* - 현재 node->data값이 char형이므로 '0'을 빼주어 int형으로 형변환을 해준다.
* - left와 right가 설정 된 후, 두 피연산자 node의 parent 노드, 즉 연산자 node의 node->data를 검
*   사하여 value를 연산자에 맞는 연산결과로 저장한다.
* - 이 value값은 root node를 매개변수로 갖는 함수에서 right 또는 left 에 저장되므로 계속 연산을 진
*   행하면 최종 value값을 얻을 수 있다.
* - drawTree는 왼쪽에서부터 root node를 출력하며 tree 전체 구조를 출력하는 함수이다.
* - 먼저 ptr이 0이 아닐 때 까지 ptr->right로 이동하는데, 이동 할 때마다 level을 1씩 증가시킨다.
* - 그리고 ptr이 NULL값이 되면 level 수 만큼 공백을 넣어준다.
* - 그 이후 ptr->data를 출력한 뒤, ptr->right와 ptr->left가 0이므로 줄을 바꾼다.
* - 방금 출력한 ptr->data의 parent는 ptr->left와 ptr->right가 둘 다 0이 아니면 두 child 모두 값이
*   있다는 의미이므로 <를 출력하고 ptr->left만 값이 있는 경우에는 \를, ptr->right만 값이 있는 경우
*   에는 /를 출력하게 한다.
* - 이렇게 root까지 recursive를 통해 right를 먼저 출력한 후, left도 같은 방식으로 출력하면
*   drawTree가 완성된다.
* Variables
* const char const prec - 연산자의 우선순위를 저장하고 있는 배열이다.
* struct Node - 입력받은 한 개의 Node값을 저장해주는 구조체 변수이다.
* char data - Node값 중 연산자 또는 피연산자를 저장하는 변수이다.
* int prio - Node값 중 연산자 우선순위를 결정해주는 변수이다. 피연산자의 경우 이 변수의 값은 4로
*            설정된다.
* Node *left - 트리 구조 중 왼쪽으로 이어지는 Node를 가리키는 포인터 변수이다.
* Node *right - 트리 구조 중 오른쪽으로 이어지는 Node를 가리키는 포인터 변수이다.
* class Tree - 트리 구조를 갖추는 객체이다.
```

```
* Node *root - 트리 객체에서 가장 상위 Node를 가리키는 포인터 변수이다.
* Tree() - 트리 객체를 만들어주는 생성자 함수이다.
* bool isEmpty() - 트리 객체가 비어있는지 아닌지 검사하는 함수이다.
* void inorder(Node *ptr) - 트리 객체 내 Node들을 중위 표현으로 출력하는 함수이다.
* void postorder(Node *ptr) - 트리 객체 내 Node들을 후위 표현으로 출력하는 함수이다.
* void preorder(Node *ptr) - 트리 객체 내 Node들을 전위 표현으로 출력하는 함수이다.
* void buildTree(char input) - 매개변수로 받은 input값으로 Node를 만들고, 연산자인지 피연산자인
*                              지 판단해주는 함수이다.
* void Operand(Node *ptr) - buildTree에서 새로 만든 Node가 피연산자일 경우 트리에 넣어주는 함
*                           수이다.
* void Operator(Nofe *ptr) - buildTree에서 새로 만든 Node가 연산자일 경우 트리에 넣어주는 함수
*                            이다.
* int evalTree(Node *ptr) - 트리 객체 안에 있는 Node값들을 연산해주는 함수이다.
* void drawTree(Node *ptr, int level) - 트리 형태를 그대로 출력해주는 함수이다.
* Node* getRoot() - 트리 객체의 root를 Node 포인터 형식으로 반환하는 함수이다.
* char input[] - 입력받을 수식을 저장할 char형 배열이다.
*********************************************************************************
#include <iostream>

using namespace std;

const char const prec[4][2] = { '*',2,'/',2,'+',1,'-',1 }; //연산자의 우선순위를 저장하고 있는 배열로
                                                            각 행에 대한 열이 우선순위이다.
struct Node {
        char data;
        int prio;
        Node* left;
        Node* right;
        Node(char value) {
                data = value;
                prio = 4;       // 연산 우선순위의 기본값은 4로 설정해준다.
                left = 0;
                right = 0;
        }
};

class Tree {
private:
        Node *root;
public:
        Tree() {
                root = 0;
        }
        bool isEmpty();
        void inorder(Node *ptr);
        void postorder(Node *ptr);
        void preorder(Node *ptr);
        void buildTree(char input);
        void Operand(Node *ptr);
        void Operator(Node *ptr);
        int evalTree(Node *ptr);
        void drawTree(Node *ptr, int level);
        Node* getRoot();
};
```

```
*********************************************************************************************
* function : isEmpty
* description : isEmpty의 사용목적은 Tree의 root를 검사하여 Tree가 비어있는지 아닌지 검사하는 것
*               이다
*********************************************************************************************
bool Tree::isEmpty() {
        if (root == 0)
                return true;
        else
                return false;
}
*********************************************************************************************
* function : inorder
* description : inorder의 사용목적은 tree를 중위 순회하는 것이다.
* variable : ptr - 매개변수로 받는 Node형식의 포인터변수로, 매개변수로는 root를 받는다.
*********************************************************************************************
void Tree::inorder(Node *ptr) { // 트리의 왼쪽으로 갔다가 데이터를 출력하고 오른쪽으로 이동한다.
        {
                if (ptr) {
                        inorder(ptr->left);
                        cout << ptr->data << " ";
                        inorder(ptr->right);
                }
        }
}
*********************************************************************************************
* function : postorder
* description : postorder의 사용목적은 tree를 후위 순회하는 것이다.
* variable : ptr - 매개변수로 받는 Node형식의 포인터변수로, 매개변수로는 root를 받는다.
*********************************************************************************************
void Tree::postorder(Node *ptr) { //트리의 가장 왼쪽으로 갔다가 오른쪽으로 가고 데이터를 출력한다.
        if (ptr) {
                postorder(ptr->left);
                postorder(ptr->right);
                cout << ptr->data << " ";

        }
}

*********************************************************************************************
* function : preorder
* description : preorder의 사용목적은 tree를 전위 순회하는 것이다.
* variable : ptr - 매개변수로 받는 Node형식의 포인터변수로, 매개변수로는 root를 받는다.
*********************************************************************************************
void Tree::preorder(Node *ptr) { //데이터를 출력한 뒤 트리의 왼쪽으로 갔다가 오른쪽으로 이동한다.
        if (ptr) {
                cout << ptr->data << " ";
                preorder(ptr->left);
                preorder(ptr->right);
        }
}
```

```
*****************************************************************************************
* function : buildTree
* description : buildTree의 사용목적은 매개변수로 받은 하나의 값의 우선순위를 판별하고 tree에 넣는
*               것이다.
* variable
* input - 매개변수로 받은 char형 값으로, 이 값을 tree에 넣어준다.
* *temp - 매개변수로 받은 char형 값을 data로 가지는 Node형 포인터변수로, 최초 우선순위는 4로 설
*         정되어있다.
* Operand() - input 값이 피연산자일 경우 tree에 넣어주는 함수이다.
* Operator() - input 값이 연산자일 경우 tree에 넣어주는 함수이다.
*****************************************************************************************
void Tree::buildTree(char input) {
        while (input != NULL) {
                Node *temp = new Node(input);

                for (int i = 0; i < 4; i++) { // data값을 검사하고 data의 우선순위를 결정한다.
                        if (temp->data == prec[i][0]) {
                                temp->prio = prec[i][1] - '0';
                                break;
                        }
                        else {
                                temp->prio = 4;
                        }
                }

                if (temp->prio == 4) { // 우선순위가 4일경우 피연산자
                        Operand(temp);
                        break;
                }
                else { // 그 외의 경우 연산자
                        Operator(temp);
                        break;
                }
        }
}
*****************************************************************************************
* function : Operand
* description : Operand의 사용 목적은 매개변수로 받은 ptr의 data값이 피연산자 일 경우 tree에 넣어
*               주는 것이다.
* variable
* *ptr : 매개변수로 받은 Node형 포인터변수로, ptr->data는 buildTree에서 매개변수로 받은 input값으
*        로 가진다.
* *p : ptr을 tree에 알맞은 위치에 넣기 위한 비교용 Node형 포인터변수이다.
*****************************************************************************************
void Tree::Operand(Node *ptr) {
        Node *p;
        if (isEmpty()) {
                root = ptr;
        }
        else {
                p = root;
                while (p->right != NULL) {
                        p = p->right;
                }
                p->right = ptr;
        }
}
```

```
*****************************************************************************************
* function : Operator
* description : Operator의 사용 목적은 매개변수로 받은 ptr의 data값이 연산자 일 경우 tree에 넣어
*               주는 것이다.
* variable
* *ptr : 매개변수로 받은 Node형 포인터변수로, ptr->data는 buildTree에서 매개변수로 받은 input값으
*        로 가진다.
*****************************************************************************************
void Tree::Operator(Node *ptr) { //피연산자가 트리에 있기 때문에 isEmpty를 검사 할 필요가 없음
        if (root->prio >= ptr->prio) { // 새로 들어온 data의 우선순위가 root보다 낮을 경우
                ptr->left = root;         새로 들어온 data가 root가 되고 기존 root는 새로 들어온
                root = ptr;               data의 왼쪽으로 이동한다.
        }
        else {
                ptr->left = root->right; //새로 들어온 data의 우선순위가 root보다 높을 경우
                root->right = ptr;        기존에 root 오른쪽에 있는 data를 새로 들어온 data의
        }                                 왼쪽에 넣어주고 그 자리를 차지한다.
}
*****************************************************************************************
* function : getRoot
* description : getRoot의 사용 목적은 tree객체의 root를 포인터 형으로 반환하는 것이다.
*****************************************************************************************
Node* Tree::getRoot() {
        return root;
}
*****************************************************************************************
* function : evalTree
* description : evalTree의 사용 목적은 tree안에 있는 값들을 연산하는 것이다.
* variable
* value : evalTree에서 최종적으로 반환하는 값이며, 연산의 결과 값이 저장되는 변수이다.
* left : tree에서 각 node의 왼쪽 값을 저장해주는 변수이다.
* right : tree에서 각 node의 오른쪽 값을 저장해주는 변수이다.
*****************************************************************************************
int Tree::evalTree(Node *ptr) {
        int value, left, right;
        if (!isEmpty()) { // 트리의 가장 밑에 있는 data는 항상 피연산자이다.
                if (ptr->left == 0 && ptr->right == 0)
                        value = ptr->data - '0';
                else {
                        left = evalTree(ptr->left);
                        right = evalTree(ptr->right);
                        switch (ptr->data) {
                        case '+':
                                value = left + right;
                                break;
                        case '-':
                                value = left - right;
                                break;
                        case '*':
                                value = left * right;
                                break;
                        case '/':
                                value = left / right;
                                break;
                        }
                }
        }
```

```
        else {
                cout << "Empty Tree" << endl;
                return 0;
        }
        return value;
}
*****************************************************************************************
* function : drawTree
* description : drawTree의 사용 목적은 tree객체를 tree형태로 그려서 출력하는 것이다.
* variable
* ptr - 매개변수로 함수에 입력되는 Node형 포인터변수로, root값이 들어간다.
* level - tree의 층을 나타내는 변수이다.
*****************************************************************************************
void Tree::drawTree(Node *ptr, int level) {
        if (ptr != 0) {
                drawTree(ptr->right, level + 1);
                for (int i = 1; i <= level - 1; i++)
                        cout << "     ";          // 트리의 level 만큼 공백을 넣어준다.
                cout << ptr->data;
                if (ptr->left != 0 && ptr->right != 0)
                        cout << " < " << endl;
                else if (ptr->right != 0)
                        cout << " / " << endl;
                else if (ptr->left != 0)
                        cout << "\\" << endl;
                else
                        cout << endl;
                drawTree(ptr->left, level + 1);
        }

}

int main() {
        Tree tree;
        char input[100];

        cout << "Enter expression : ";

        cin >> input;

        int i = 0;
        while (input[i] != 0) {
                tree.buildTree(input[i]);
                i++;
        }

        cout << endl;
        cout << "   Inorder : ";
        tree.inorder(tree.getRoot());
        cout << endl;

        cout << endl;
        cout << "   Postorder : ";
        tree.postorder(tree.getRoot());
        cout << endl;

        cout << endl;
        cout << "   Preorder : ";
        tree.preorder(tree.getRoot());
        cout << endl;
```

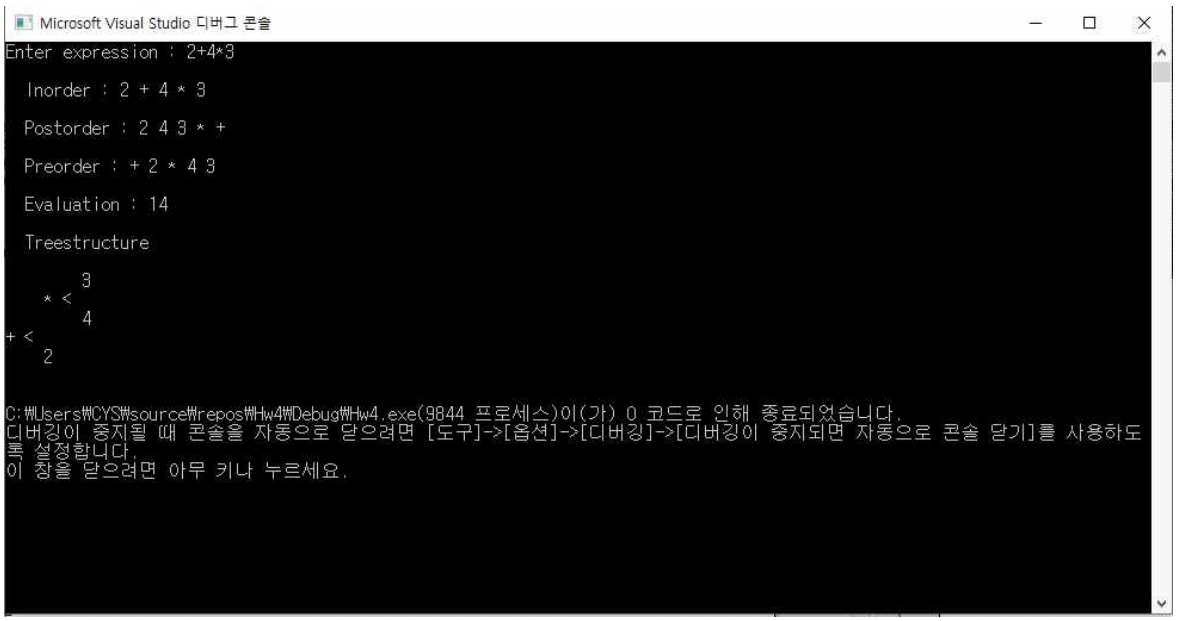
```
        cout << endl;
        cout << "   Evaluation : ";
        cout << tree.evalTree(tree.getRoot()) << endl;

        cout << endl;
        cout << "   Treestructure" << endl;
        cout << endl;
        tree.drawTree(tree.getRoot(), 1);
        cout << endl;
}
```

****

* Output

*



*

****