Data Structure Homework 8

B1100731 潘永牧

1. tree

- 在解構的指令當中，我利用 level order 的方式將 Node 依序存入節點，再將其依序 Pop 出來 delete 掉。

```
~Tree() {
    if (!root) return;

    queue<TreeNode<T>*> q;
    q.push(root);

    while (!q.empty()) {
        TreeNode<T>* current = q.front();
        q.pop();

        if (current->LeftChild) q.push(current->LeftChild);
        if (current->RightChild) q.push(current->RightChild);

        delete current;
    }
}
```

- Insert 是用來加入元素的，我利用了兩個指標 q、p 來找到正確的插入位置，q 用來存取目前找到節點的位址，而 p 用來繼續往下找的指標，當 p 是 NULL 時就會停止找尋，代表已經到最底部了。當要插入的數值大於目前的節點時，p 就會往右邊的孩子找，而反過來時，要插入的數值小於目前的節點，p 就會往左邊找。找到位置後就依數值大小插入到右邊或左邊。

```
void Insert(T x) {
    TreeNode<T>* p = root;
    TreeNode<T>* q = NULL;
    while (p) {
        q = p;
        if (x == p->data) return;
        if (x < p->data) p = p->LeftChild;
        else p = p->RightChild;
    }
    p = new TreeNode<T>;
    p->LeftChild = NULL;
    p->RightChild = NULL;
    p->data = x;
    if (!root) {
        root = p;
    }
    else if (p->data > q->data) {
        q->RightChild = p;
    }
    else {
        q->LeftChild = p;
    }
};
```

- 利用遞迴的方式依序造訪 left-subtree、data、right-subtree。

```
void Inorder() {
    Inorder(root);
    cout << endl;
};
void Inorder(TreeNode<T>* CurrentNode) {
    if (CurrentNode != NULL) {
        Inorder(CurrentNode->LeftChild);
        cout << CurrentNode->data << " ";
        Inorder(CurrentNode->RightChild);
    }
};
```

- 利用遞迴的方式依序造訪 data、left-subtree、right-subtree。

```
void Preorder() {
    Preorder(root);
    cout << endl;
};
void Preorder(TreeNode<T>* CurrentNode) {
    if (CurrentNode != NULL) {
        cout << CurrentNode->data << " ";
        Preorder(CurrentNode->LeftChild);
        Preorder(CurrentNode->RightChild);
    }
};
```

- 利用遞迴的方式依序造訪 left-subtree、right-subtree、data。

```
void Postorder() {
    Postorder(root);
    cout << endl;
};
void Postorder(TreeNode<T>* CurrentNode) {
    if (CurrentNode != NULL) {
        Postorder(CurrentNode->LeftChild);
        Postorder(CurrentNode->RightChild);
        cout << CurrentNode->data << " ";
    }
};
```

- 利用 queue 儲存造訪的順序，從 root 開始，再來左邊的孩子，右邊的孩子，左邊的孩子的左邊的孩子，左邊的孩子的右邊的孩子，以此類推。

```
void Levelorder() {
    queue<TreeNode<T>*> q;
    TreeNode<T>* CurrentNode = root;
    while (CurrentNode) {
        cout << CurrentNode->data << " ";
        if (CurrentNode->LeftChild) q.push(CurrentNode->LeftChild);
        if (CurrentNode->RightChild) q.push(CurrentNode->RightChild);
        if (q.empty()) {
            cout << endl;
            return;
        }
        CurrentNode = q.front();
        q.pop();
    }
};
```