# 자료구조 과제 보고서

## [연결리스트를 이용한 트리 구현]

12141579 윤찬미
010-9287-1679
cmmcme0604@gmail.com

# I. 개요

• 설계의 목적

일반 트리를 구현하여 삽입, 전위순회, 후위순회를  할 수 있게 하고,
일반 트리를 이진트리 형식으로 저장하고 좌표 출력을 할 수 있도록 한다.

• 요구 사항

- 노드의 값을 입력 받아 정해진 위치에 넣을 수 있도록 한다.
- 일반 트리에 대한 전위순회, 후위순회 함수를 수행할 수 있도록 한다.
- 일반 트리를 이진트리 형식으로 변환 하였을 때 해당 노드의 좌표를 출력할 수 있게 한다.

• 개발 환경

- Microsoft Visual Studio 2013
- 언어 : C++

# II.자료구조 및 기능

## •Normal Tree 를 구현하기 위한 세부 자료 구조

### 1.Tree Node

```
typedef struct Node
{
    Node* parent;    //부모를 가리킴
    Node* child;    //첫번째 자식
    Node* next;        //형제 노드
    Node* prev;          //전꺼를 저장함
    int num; //가지고 있는 데이터 번호
    Node() : parent(NULL), child(NULL), next(NULL), prev(NULL) {};       //각 값을 초기화 한다.
}Node;
```

- 구조체를 이용하여 Node를 생성해 주었다.
- Node 안에는 데이터 값인 num과
  부모,첫번째 자식, 형제, 자신의 바로 전 형제 노드를
  가리키는 포인터 값이 저장되어 있는
  Doubly LinkedList로 만들었다.

## 2. Tree

```
class Tree  {
public:
    Tree() : Root(NULL), Position(NULL) {}; //초기화
    void searchP(int _p, Node* s);
    void insert(int n, int p, int o);
    void Preorder(Node *s);
    void Postorder(Node *s);
    Node* getRoot() { return Root; }
    Node* getPosition() { return Position; }
    void resetPosition() { Position = 0; }

private:
    Node* Root;
    Node* Position;
};
```

- Class를 이용하여 일반 Tree를 생성해 주었다.
- Root 노드와 Search를 할때 위치를 저장하는 Position을 NULL로 초기화 한다
- SearchP ,Insert, Preorder , Postorder 함수를 정의해 주었다.
- getRoot, getPosition, resetPosition 함수를 정의해 값을 반환하거나 Position

## •Binary Tree 를 구현하기 위한 세부 자료 구조

## 3. BinaryTree Node

```
typedef struct BNode
{
    BNode* parent;  //바이너리로 바뀔을 때 부모 (트리노드에서 prev / 첫번째 자식일 경우 parent)
    BNode* left;    //왼쪽 자식 == 첫번째 자식 (child)
    BNode* right;   // 오른쪽 자식 == 형제노드 (next)
    int num;
    BNode() : parent(NULL), left(NULL), right(NULL), num(0) {};//각 값을 초기화 한다.

}BNode;
```

- 구조체를 이용하여 BNode를 생성해 주었다.
- BNode 안에는 데이터 값인 num과
  parent, left, right 를
  가리키는 포인터 값이 저장되어 있는
  LinkedList로 만들었다.

## 4. BinaryTree

```
class BinaryTree
{
public:
    BinaryTree() :x(1), count(1), Root(NULL), Position(NULL) {};
    void searchP(int _p, BNode* s);
    void insert(int n, int p, int o);
    void Inorder(BNode *s);
    int Depth(BNode *s);
    void resetX() { count = x = 1; }
    int getX() { return x; }
    BNode* getRoot() { return Root; }
    BNode* getPosition() { return Position; }
    void resetPosition() { Position = NULL; }

private:
    BNode* Root;
    BNode* Position;
    int x;
    int count;
};
```

- Class를 이용하여 BinaryTree를 생성해 주었다.
- Root 노드와 Search를 할때 위치를 저장하는 Position을 NULL로 초기화 한다x좌표

- x좌표를 저장하는 변수 x와 count를 선언하고, 그것을 구하는 Inoreder 함수를 선언해 주었다.
- y좌표를 구하기 위한 Depth 함수를 정의한다.
- Root, Position과 x값을 반환하는 함수와, x와 Position을 초기화 하는 함수를 선언하였다.

• 프로그램의 기능

### 1. void searchP(int _p, Node *s);
### void searchP(int _p, BNode *s);

-> _p라는 데이터 값을 가지고 있는 노드의 위치를 반환하는 함수이다.
첫번째 자식(Tree에서 child , BinaryTree에서 left)를 먼저 확인한 후,
형제들을 확인한다(Tree에서 next, BinaryTree에서 right)

## 2 . void insert (int n, int p, int o);

### Tree

-> 추가할 Data 값과 부모 노드의 Data 값, 몇번째 자식인지 결정 지을 값을 입력받아 searchP 함수를 사용하여 부모노드를 찾고 새 노드를 선언하여 첫번째 자식인지 아닌지 확인하고 첫번째 자식일 경우 비어있는지 비어있지 않은지 확인하여 삽입.
첫번째 자식이 아닐 경우 o번째 위치의 자식으로 가서
비어있는지 비어있지 않은지 확인하고 삽입한다.

### BinaryTree

-> 추가할 Data 값과 부모 노드의 Data 값, 몇번째 자식인지 결정 지을 값을 입력받아 searchP 함수를 사용하여 부모노드를 찾고 새 노드를 선언하여 첫번째 자식인지 아닌지 확인하고 첫번째 자식일 경우 비어있는지 비어있지 않은지 확인하여 왼쪽에 삽입.
첫번째 자식이 아닐 경우 처음엔 첫번째 자식위치 (left로 간 후)
그것의 Right(형제)를 따라가며 O번째에 다다르면, 비어있는지 비어있지 않은지 확인하고 삽입한다.

## 3. void Preorder(Node *s);

-> 출력(visit)를 먼저 수행하고 자식이 있는지 확인하고 재귀를 하여 방문하는 순회이다.

## 4. void Postorder(Node *s);

-> 자식이 있는지 확인하고 재귀하고, 자식이 비어있다면 형제노드를 확인한 후 재귀하고 출력(visit) 한다.

## 5. void Inorder(BNode *s);

-> x좌표를 알아내기 위해 사용하는 Inorder 함수이다. 왼쪽지 비어있지 않은지 확인한 후 재귀를 실행 하고, 비어있거나 재귀가 끝나면 내가 찾는 노드가 맞는지 확인한다. 맞다면 x에 count를 대입하고 아니라면 count를 더해준다. 후에 오른쪽 자식도 검사해보며 재귀를 한다.

## 6. int Depth(BNode *s);

-> y좌표를 찾는 함수. 루트노드의 y좌표가 0이므로 depth를 0으로 초기화 하고 현재 노드가 루트노드가 될때 까지 반복한다. 루트노드가 아니면 1을 더하고 현재 노드를 그 노드의 부모노드로 바꾸고 반복문이 끝난 후 depth 값(y좌표)을 리턴해준다.

## 7. void GetX();

-> x의 값을 리턴 해준다.

## 8. void resetX();

->x의 값과 count 값을 1로 초기화 해준다.

## 9. (B)Node *getRoot();

-> Root의 위치를 반환해준다

## 10.(B)Node* getPosition();

-> Position의 위치를 반환해준다.

## 11. void resetPosition();

-> Position의 위치를 초기화해준다.

# III.기능별 알고리즘 명세

## 1. Algorithm searchP(int _p, Node* s)

| | 시간복잡도 |
|---|---|
| if s->num = _p then | 1 |
| Position <- s; | 1 |
| else then | 1 |
| if ㄱs->child = NULL then | 1 |
| searchP(_p, s->child); | n |
| if ㄱs->next = NULL then | 1 |
| searchP(_p, s->next); | n |

---

O(n)

## 2. Algorithm insert(int n, int p, int o)

//Tree와 Binary Tree에서의 구현 방법은 동일하기에 하나만 정의

| | 시간 복잡도 |
|---|---|
| if p = -1 then | 1 |
| new Node.num <- num | 1 |
| Root <- new Node | 1 |
| else then | |
| searchP(p, Root) | n |
| if position <- NULL then | 1 |
| print(노드가 없습니다) | 1 |
| else then | |
| new Node.parent <- Position | 1 |
| if(o = 1) then | 1 |
| if(Position.child = NULL) then | 1 |
| Position.child <- new Node | 1 |
| else then | |
| Position.child.prev <- new Node | 1 |
| new Node.next <- Position.child | 1 |
| Position.child <- new Node | 1 |

```
 new Node.num <- n                                  1


else then

Position <- Position.child                          1
new Node.num <- n                                   1
o <- o-1                                            1
while(o to o=0 o <- o--)                            n
  if( o = 0 & Position.next = NULL) then            1
     Position.next <- new Node                      1
     newN.prev <- Posirtion                         1

  else if ( o = 0 & Position.next != NULL) then     1
 new.next <- Position.next                          1
 newN.prev <- Position                              1
      Position.next.prev <- new Node                1
      Position.next <- newN                         1
  Position <- Position.next                         1
```

---

O(n)

## 3. Algorithm Preorder(Node *s)

시간복잡도

```
       if ㄱ s = NULL then                              1

          print( s->num)                                1

           if ㄱ s.child = NULL then                    1

               Preorder(s.child)                        n

           if ㄱ s.next = NULL                          1

               Preorder(s.next);                        n
```

---

O(n)

## 4. Algorithm Postorder(Node *s)

| | 시간복잡도 |
|---|---|
| if ㄱ s = NULL then | 1 |
| if ㄱs. child = NULL then | 1 |
| Postorder(s.child) | n |
| print(s.num) | 1 |
| if ㄱ s.next = NULL then | 1 |
| Postorder(s.next) | n |
| | O(n) |

## 5. Algorithm Inorder(BNode *s)

| | 시간 복잡도 |
|---|---|
| if ㄱ s.left = NULL then | 1 |
| Inorder(s.left) | n |
| if s = Position then | 1 |
| x = count | 1 |
| else then | 1 |
| count++ | 1 |
| if ㄱ s.right = NULL then | 1 |
| Inorder(s.right) | n |
| | O(n) |

## 6. Algorithm Depth(BNode *s)

| | 시간 복잡도 |
|---|---|
| int depth <- 0 | 1 |
| while (ㄱ s = Root) | n |
| depth <- depth + 1 | 1 |
| s <- s.parent | 1 |
| return depth; | |
| | O(n) |

7.

```
Algorithm resetX() { count = x = 1; }
Algorithm getX() { return x; }
Algorithm getRoot() { return Root; }
Algorithm getPosition() { return Position; }
Algorithm resetPosition() { Position = NULL; }
```

---

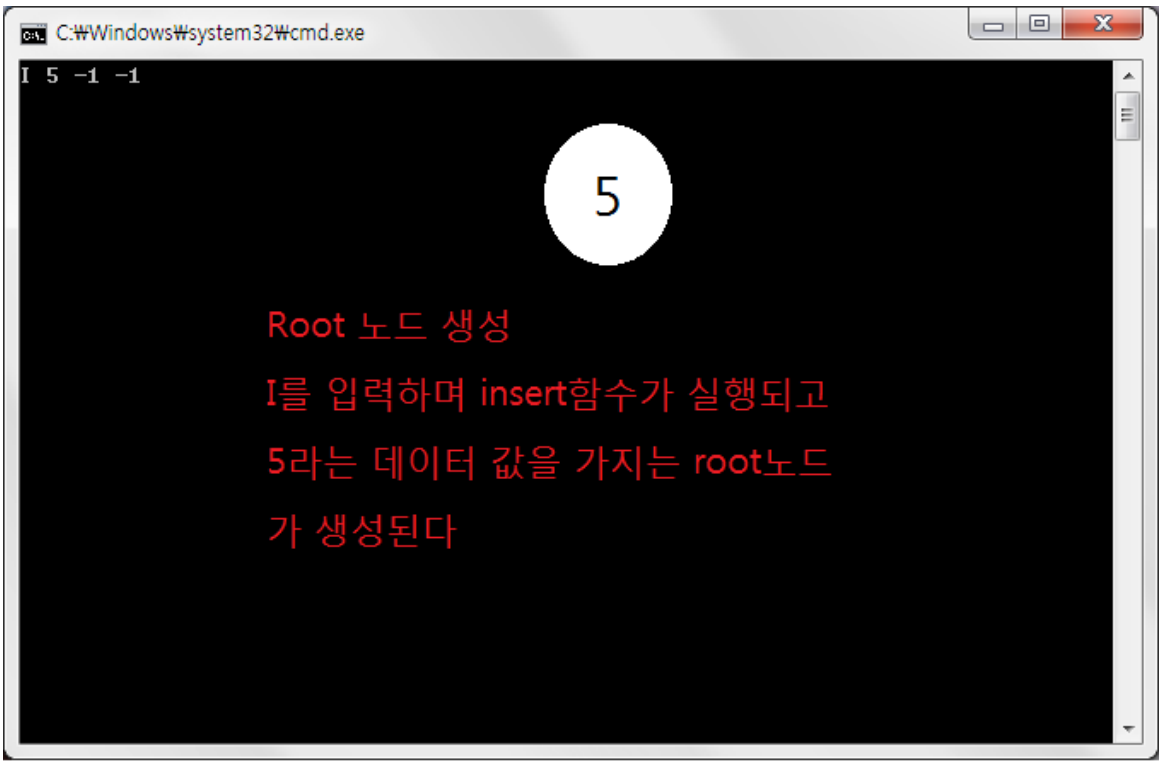
시간 복잡도 O(1)

* 재귀함수 사용시 Callbyvalue로 넘어간다. 즉, 함수에 대한 새로운 메모리를 할당해주고 스택에 쌓기 때문에 따라서 n번의 공간할당이 이루어진다.

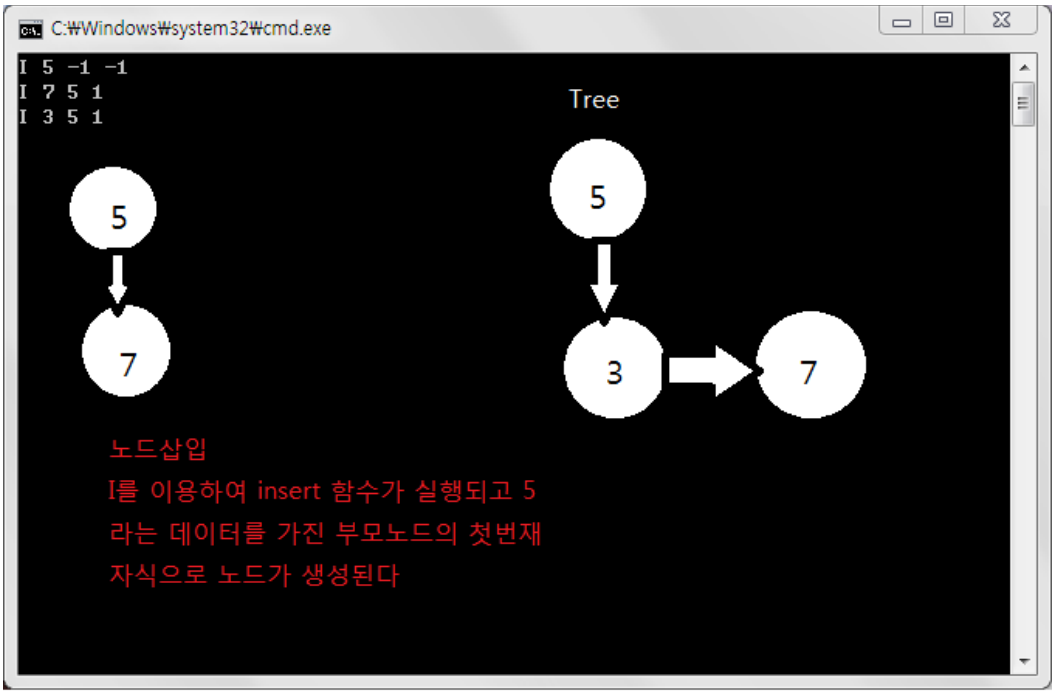
# IV.인터페이스 및 사용법

- 실행화면 스크린샷과 기능 설명

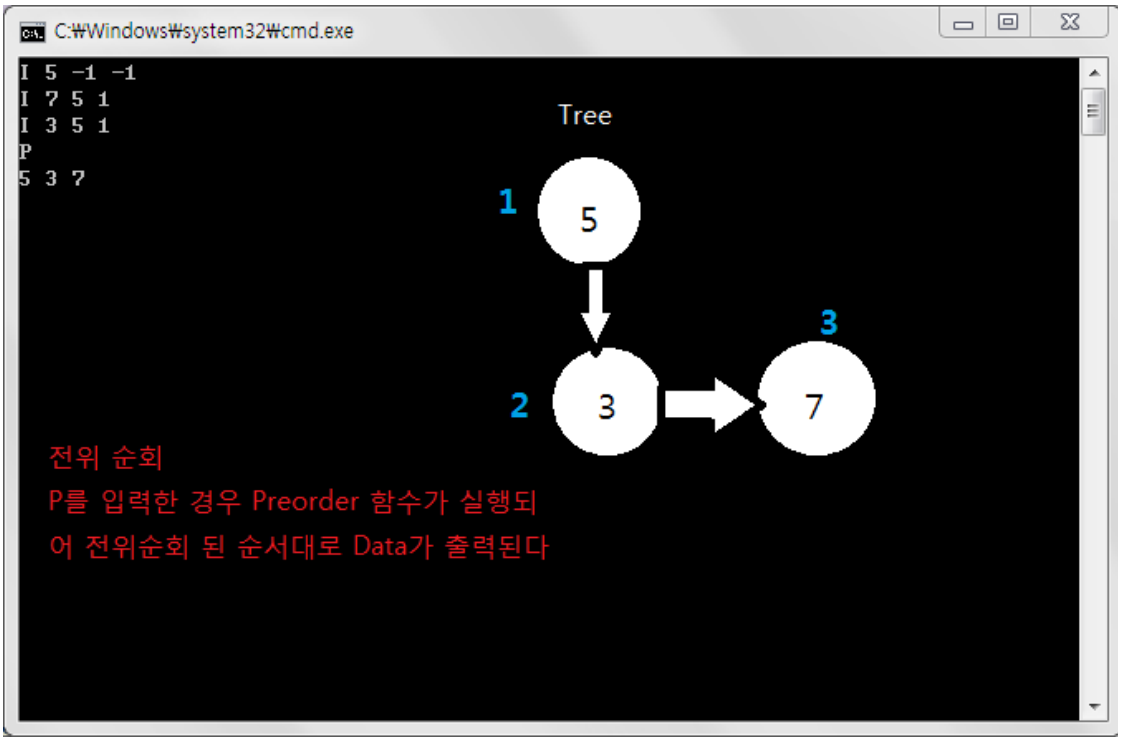
## 1) 노드 생성



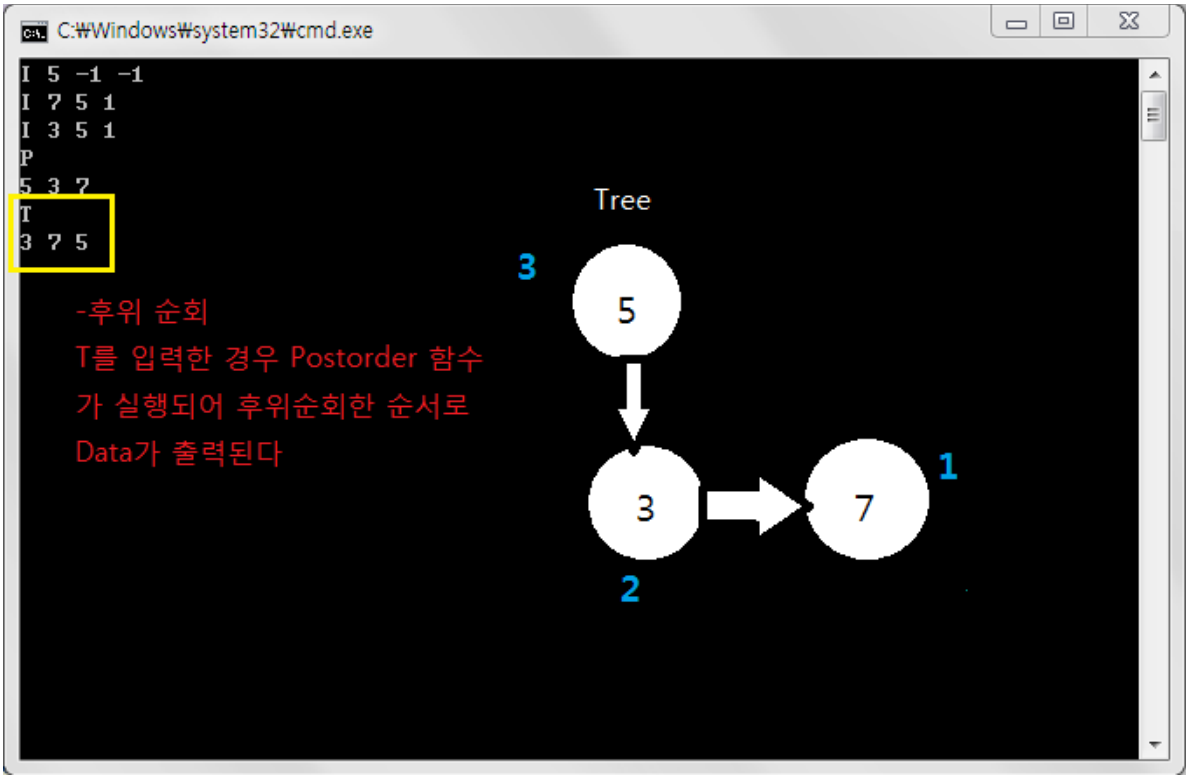
## 2) 노드 삽입


C:₩Windows₩system32₩cmd.exe
I 5 -1 -1
I 7 5 1
I 3 5 1
Tree
5
7
5
3
7
노드삽입
I를 이용하여 insert 함수가 실행되고 5
라는 데이터를 가진 부모노드의 첫번재
자식으로 노드가 생성된다

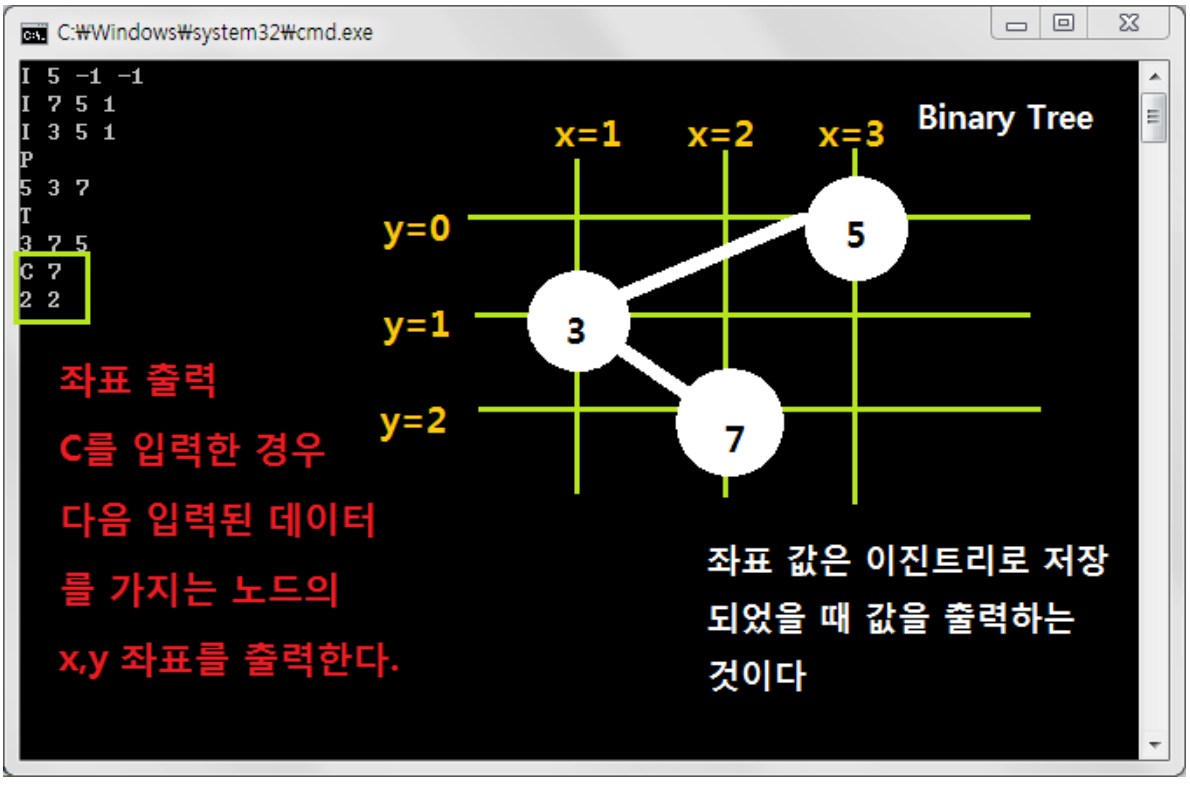
## 3) 전위순회


C:₩Windows₩system32₩cmd.exe
I 5 -1 -1
I 7 5 1
I 3 5 1
P
5 3 7
Tree
1
5
2
3
3
7
전위 순회
P를 입력한 경우 Preorder 함수가 실행되
어 전위순회 된 순서대로 Data가 출력된다

4) 후위순회


C:₩Windows₩system32₩cmd.exe
I 5 -1 -1
I 7 5 1
I 3 5 1
P
5 3 7
T
3 7 5
-후위 순회
T를 입력한 경우 Postorder 함수
가 실행되어 후위순회한 순서로
Data가 출력된다
Tree
3
5
3
7
1
2

5) 좌표출력


C:₩Windows₩system32₩cmd.exe
I 5 -1 -1
I 7 5 1
I 3 5 1
P
5 3 7
T
3 7 5
C 7
2 2
좌표 출력
C를 입력한 경우
다음 입력된 데이터
를 가지는 노드의
x,y 좌표를 출력한다.
x=1
x=2
x=3
Binary Tree
y=0
y=1
y=2
5
3
7
좌표 값은 이진트리로 저장
되었을 때 값을 출력하는
것이다

# V.평가 및 개선 방향

## – 본 결과의 장점 및 단점

### 장점

-> class로 만들었기 때문에 객체 생성을 여러번 하여 다른 종류의 여러 개의  Tree를 생성 할 수 있다는 점이 있다.
Root를 반환해주는 함수를 따로 만들었기 때문에 main함수에서 객체의 Root에 접근하여 class에 있는 함수를 쉽게 사용 할 수 있게 만들었다는 것이다.

### 단점

-> insert 함수에서 많은 조건문으로 인하여 코드의 가독성이 떨어지고,
BinaryTree에서도 삽입을 하는 알고리즘이기 때문에 메모리 차지가 커졌다고 할 수 있다.

## –향후 개선 방향

-> insert 함수에서 루트노드 삽입 경우와, 첫번째 자식에 삽입하는 경우의 조건문을 제외한 삽입 알고리즘을 통해 시간복잡도가 같은 O(n)이라도 빠른 알고리즘을 구현 할 수 있을 것이다.
-> 노드의 데이터가 int 만 받을 수 있도록 설게되어 있는데 template를 이용하여 다른 자료형의 데이터도 받을 수 있게 만들면 쓰임이 더 많아질 것이라고 생각한다.