

 REPORT

자료구조 보고서

[연결리스트를 이용한 트리 구현]

과 목 명 : 자료구조

담당교수 : 심정섭 교수님

제 출 일 : 2015년 11월 2일 월요일

01041975132

yoohoogun114@naver.com

컴퓨터공학과 2학년 12121518 이름:유호균

**설계의 목적**

트리를 직접 설계하고 코딩함으로써 실제 구현 능력을 향상시키고 디버그능력을 키우는데에 있다.

**요구사항**

표준자료 입출력, Tree insert node, Search Algorithm(노드를 찾아서 넣기 위해 그 노드의 위치를 찾기 위해) ,Postorder Traversal, Preorder Traversal, compute and print binary X,Y coordinate

**개발 환경**

MacBook Pro OS X 10.11 엘케피텐 Xcode 7.1 에서 코드작성, 디버그 작업 후 메일로 코드를 옮겨 Windows 8.1 Visual Studio Professional 2013에서 디버그 및 테스트, 자료 입출력

**필요한 자료구조 및 기능**

자료구조: Node struct(child는 첫번째 자식을 가리키고, sibling은 바로 다음자식을 가리킨다), LinkedList(child, sibling을 포함한 노드로 구성되는), tree class

기능: Search, compute X, Y coordinate, pre,post order traversal, insert node(중간삽입 포함)

**과제를 구현하기 위해 사용한 세부 자료구조**

노드 설계:

struct Node { //Node의 구성

int elem; //노드의 번호

int child\_count; //자식의 갯수

Node\* child; //자식 노드의 위치

Node\* parent; //부모 노드의 위치

Node\* sibling; //형제 노드의 위치

};

클래스 설계:

class tree { //tree 구성

private: //tmp , temp가 붙은 값들은 모두 임시변수 입니다

Node\* root; //루트노드

Node\* tmp\_node; //search를 위한 위치저장용

int totalNum; //전체 노드의 수

int tempX; //X좌표 출력을 위한 임시변수

int tempY; //Y좌표 출력을 위한 임시변수

public:

tree(); // 생성자

//~tree(); //소멸자

void init\_tmp\_node(); // 임시변수 tmp\_node 초기화

void set\_tmp\_node(Node\*); // 임시변수 tmp\_node set

void search\_traversal(Node\*, int); // elem값을 받아서 해당함수의 위치를 찾음

void init\_tempX(); // X좌표위치를 찾기위한 임시변수 tempX 초기화

void add\_tempX(); // tempX 더하는 함수

void print\_tempX(); // tempX를 출력하는 함수

void init\_tempY(); // tempY 초기화

void add\_tempY(); // tempY 더함

void print\_tempY(); // tempY 출력

void initRoot(int); // Root 초기화

void insertNode(int, int, int); // element, 부모, 형제위치를 받아 노드추가

void addTotal(); // 트리의 노드의 총 개수 추가

void preTravle(); // 전위순회

void postTravle(); // 후위순회

void print\_recursionPreTravle(Node\*); // 전위순회하여 출력

void print\_recursionPostTravle(Node\*); // 후위순회하여 출력

void search\_coordinate(int); // x, y좌표 출력

void print\_X\_coordinate(Node\*, int); // x좌표 계산,출력

void print\_Y\_coordinate(Node\*, int); // y좌표 계산,출력

Node\* get\_tmp\_node(); // 임시tmp노드 반환

Node\* searchNode(int); // elem값을 받아서 노드의 위치를 search하여 반환

Node\* getRoot(); // root위치 반환

int get\_tempY(); // tempY 반환

void set\_tempY(int); // tempY set

};

**기능별 알고리즘 명세**

기능은 함수위주로 설명하겠습니다.

void init\_tmp\_node() // tmp노드를 초기화하는 함수, O(1)

void set\_tmp\_node(Node\*); // tmp노드를 set하는 함수, O(1)

void search\_traversal(Node\*, int); // elem을 인자로 받아 해당노드의 위치를 temp에 set하는 재귀함수 => 실제로 search수행, O(n)

void init\_tempX(); // tempX변수를 초기화하는 함수, O(1)

void add\_tempX(); // tempX변수를 더하는 함수, O(1)

void print\_tempX(); // tempX변수 출력하는 함수, O(1)

void init\_tempY(); // tempY변수 초기화하는 함수, O(1)

void add\_tempY(); // tempY변수 더하는 함수, O(1)

void print\_tempY(); // tempY변수 출력하는 함수, O(1)

void initRoot(int); // Root를 초기화하는 함수, O(1)

void insertNode(int, int, int); // 노드를 트리에 삽입하는 함수, 들어갈 함수의 위치를 검색하는 과정을 포함한다, 입력받은 형제노드순서의 위치에 노드가 미리 있는지 검사하고 그 노드를 나의 다음노드로 지정하고 부모가 나를 child로 지정하는 과정도 포함한다. (★ 중간삽입 및 수업시간에 설명한 매다는 과정 포함) 해당 함수의 부모를 찾는 시간O(n) 그 노드의 자식이 있는지, 중간에 삽입하는지 처음에 삽입하는지 검사하는 과정O(1), 중간에 삽입할때 자신의 앞에 들어갈 노드를 찾고 매달고 삽입하는 과정O(n) 도합 O(n)

void addTotal(); // totalNum 즉 노드의 전체 수에대한 변수를 증가시키는 함수 O(1)

void preTravle(); //루트노드를 찾고 임시주소에 넣고 print\_recursionPreTravle 을 담고있는 함수, 루트노드 반환과정 O(1) , print\_recursionPreTravle 호출하여 실행 O(n) 도합O(n)

void postTravle(); //루트를 찾고 임시주소에 넣고print\_recursionPostTravle을 담고있는 함수, 루트노드 반환과정 O(1), recursionPostTravle 호출하여 실행 O(n) 도합 O(n)

void print\_recursionPreTravle(Node\*); //실제로 preorder Traversal하는 함수, 재귀호출하여 O(n)

void print\_recursionPostTravle(Node\*); //실제로 postorder Traversal하는 함수, 재귀호출하여 O(n)

void search\_coordinate(int); //좌표값을 초기화 O(1)하고 찾아 서 print\_X\_coordinate는 Postorder순서로 traversal 해야 하므로 O(n), print\_Y\_coordinate는 preorder traversal을 재귀호출 하므로O(n)

void print\_X\_coordinate(Node\*, int); // 일반트리를 이진트리로 변환해 나오는 X좌표는 일반트리를 postorder traversal한 순서임을 알 수 있다 O(n)

void print\_Y\_coordinate(Node\*, int); //Preorder traversal하며 Y좌표값을 계산한다

// 규칙성을 찾아보았다. 일반트리를 이진트리로 변환해 Y 좌표값을 찾아보면

// preorder로 부모에서 child로 갈때는 1이 늘어나고 자식으로 갈때도 1이 늘어난다

// 그러나 자식에서 두번째 자식으로는 바로 접근이 불가능하다(노드 설계 때문에)

// 아이디어는 preorder 순회하며 child로 갈때 1이 늘어나고 자식갈때 1 y좌표를 늘리고 O(n)

// 노드 자식에서 부모를 다 찾고 올라왔을때까지 search를 못 하면 y좌표값을 자식 수만큼 차감한다 O(1)

도합 O(n)

Node\* get\_tmp\_node(); // tmp\_node의 위치를 반환하는 함수 Node형 포인터

Node\* searchNode(int); // elem을 인자로 받아 해당 노드를 반환하는 함수 => 실제로 수행하는 함수는 아니지만 search\_traversal을 호출하여 실행하므로 O(n)

재귀호출로 최악시간 n번 순회

Node\* getRoot(); // Root의 위치를 담는 Node형 포인터 O(1)

int get\_tempY(); //y좌표를 담는 임시변수 tempY를 return하는 함수 O(1)

void set\_tempY(int); //y좌표를 넣을 임시변수 tempY를 set하는 함수 O(1)

**인터페이스 및 사용법**

인터페이스는 올려주신 과제자료.PDF 에 있는 인터페이스와 동일하게 만들었습니다. 실행되었을때 화면에 아무것도 나오지 않으나 이때부터 입력을 받습니다. 예를들어 I 5 -1 -1 이렇게 하면 루트노드가 생성되고 I 3 5 1을 하면 루트노드에 3인 자식이 만들어집니다. 과제자료에 설명해 주신대로 I 4 5 1을 하면 5의 첫째자식이 1이되고 3은 두번째 자식이 됩니다. 과제자료에 올려주신 모든 기능이 됩니다.

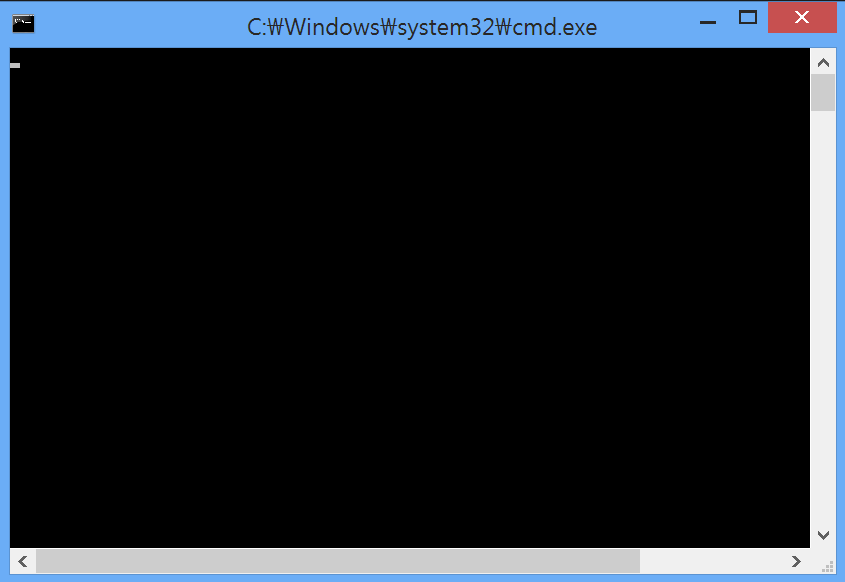


그림 1인터페이스는 없지만 이때부터 입력받음

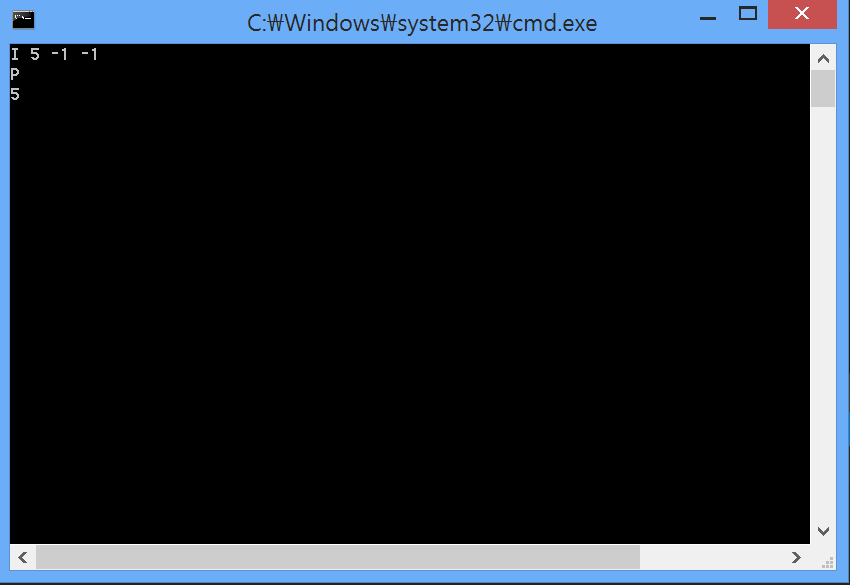


그림 2입력방법

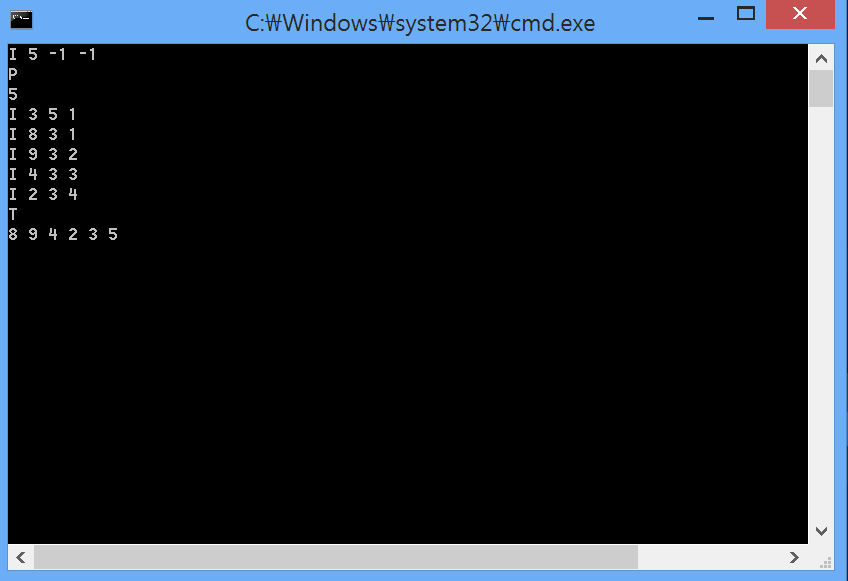


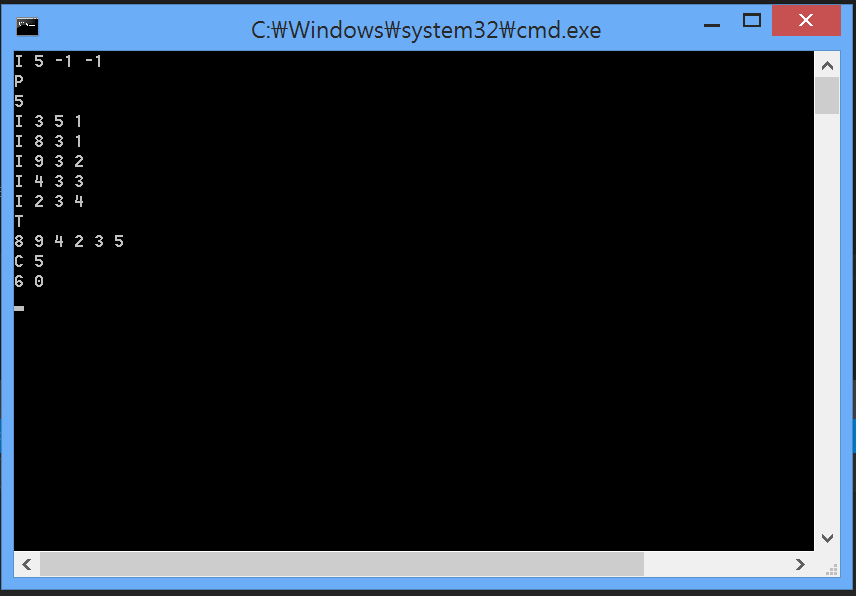
그림 3Preorder Postorder 순회

그림 4(x,y)좌표 출력

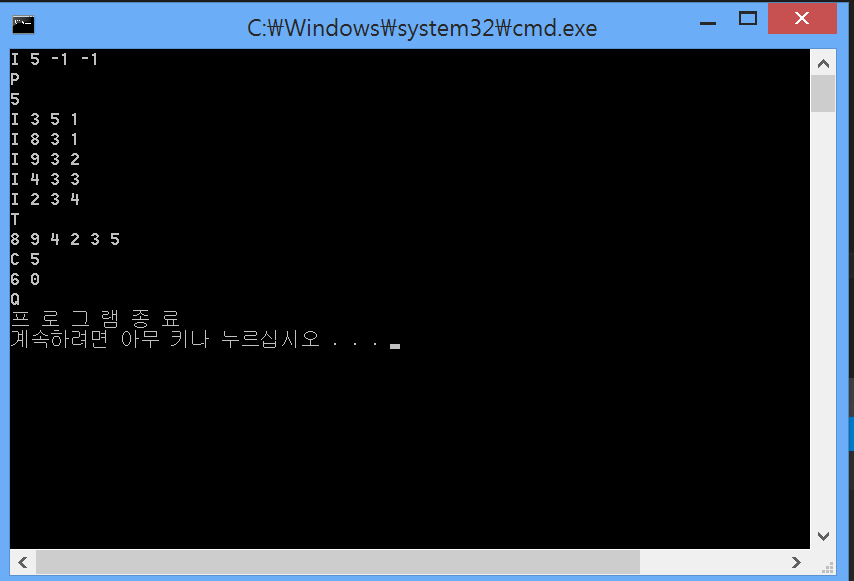


그림 5 Q를 입력하여 프로그램 종료

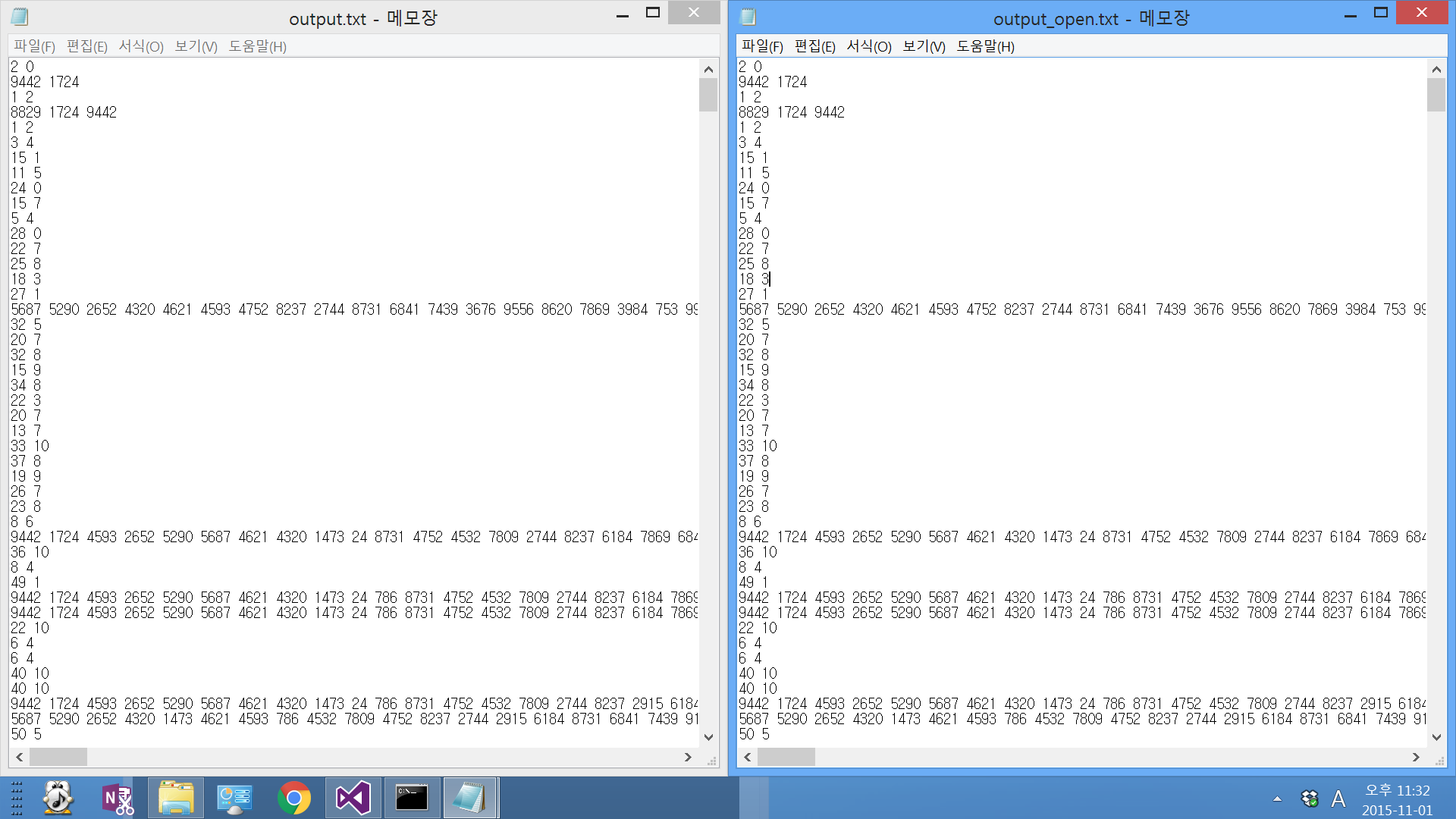


그림 6좌측의 제 프로그램의 출력, 오른쪽이 정상적 output

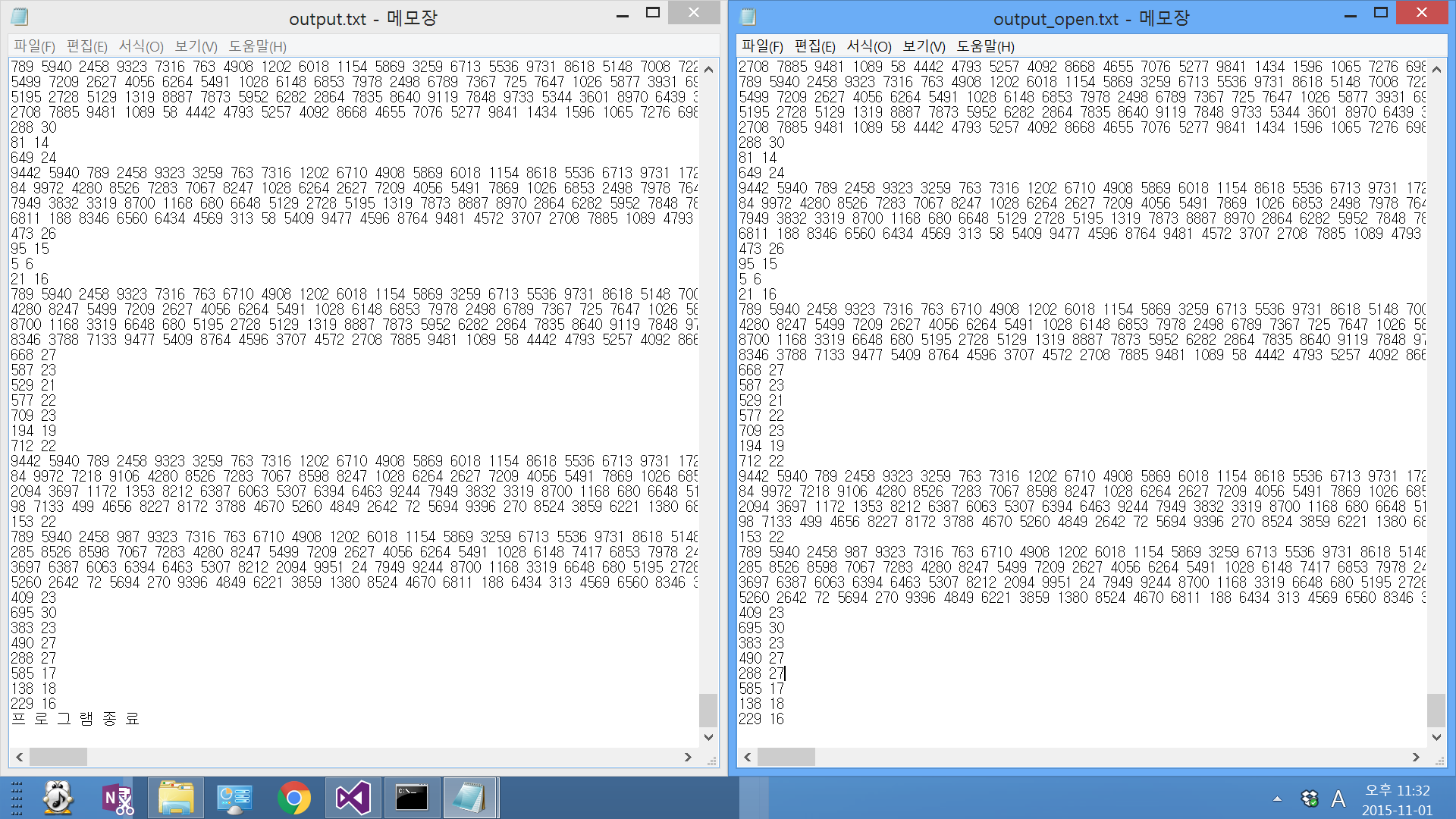


그림 7좌측의 제 프로그램의 출력, 오른쪽이 정상적 output

**평가 및 개선 방향**

- 본 결과의 장점 및 단점 - 향후 개선 방향

Search 알고리즘과 print\_Y\_coordinate(2진트리로 변환했을때 y좌표 검색,출력)부분을 짜면서 이렇게 하면 분명 좋은 알고리즘은 아니지 않을까 하는 생각이 들었지만 우선 결과가 나오는게 중요하다고 생각되어 제출한 프로젝트에 있는 코드대로 작성하였다

Search는 이정도면 어느정도 시간내에 찾는다고 생각되지만 print\_Y\_coordinate

은 정말 오래 걸리는 것 같다. 왜냐하면 재귀함수를 계속 호출하는데 Y좌표를 출력한 후에도 임시변수를 계속 수정하느라 원하는 결과가 나온 후에도 계속 함수가 실행된다. 이를 개선하려면 임시로 Flag 변수를 만들어 IF 문을 만들어 찾았을때 Flag를 true 로 설정하고 Flag가 true가 되었을때 재귀함수를 종료하는 방법을 사용해야 될 것 같다.

**그 외의 특이사항이 있으면 정리.**

1. Mac Xcode 7.1에서 코드를 완성 후 자료를 출력하였을 때는 자료가 정상적으로 출력되었다

그러나 Windows 8.1 VisualStudio 2013으로 옮겨서 자료를 출력하니 좌표를 출력하는 값에 쓰레기값이 들어있었다. 디버그를 하면서 자료값을 추적하니 어느순간 이후로 자료값이 initialize가 되지 않아서 쓰레기값이 들어있는 것이었다. Windows와 mac은 C++ compiler 가 다르기 때문일까?

같은 코드라도 어디(OS, compiler)에서 실행되느냐가 조금 신기했다.

구글링하여

The Intel C++ Compiler is compatible with popular compilers including Visual C++\* (Windows\*) and GCC (Linux\*, OS X\* and Android\*)

출처 -https://software.intel.com/en-us/intel-compilers

라는 자료를 찾게되었다. Windows는 대체로 비쥬얼 c++ , OS X는 GCC를 쓰는구나 알게되었다.

2. C++을 군대가기 전에 했으니 약 3년이 다 되어가는데 개념이 희미하여 많이 애먹었다. 그동안 JAVA, javascript, ruby등을 하다보니 특히 입력받는 부분(인터페이스)에서 많이 애먹었다. C++보다 높은 level 의 언어는 대체로 String split이란 메소드를 지원하여 String에 있는 값들을 잘라서 배열에 넣어주는데 c++에는 이 메소드가 없어서 만들고 split하여 이리저리 변환하는 코드를 넣었는데 String을 int , char로 변환도 해야하니 인터페이스 부분 코드가 많이 지저분했다. 그러던중 cin >> char >> int >> int >> int 를 하면 된 다는것을 알게되었다. Cin은 space를 만나면 입력을 받지 않으니 스페이스로 구분이 가능하다는걸 구글링해서 알게되었다.

3. 일반트리에서는 elem을 받아 search하는법, x,y 좌표를 계산하는 방법이 상당히 복잡하다.

코드를 다 짜고보니 일반 트리를 binary tree로 변환하여 binary tree에서의 계산 결과를 보여주는 방법도 좋은 방법이라고 생각한다. 비록 tree를 binary tree로 계산하는 과정이 추가되기는 하지만 x,y좌표 계산 부분이 상당히 편할 것 같다.