Project

3D Cube

과 목 명	객체지향프로그래밍실습
학 과	컴퓨터정보공학부
학 번	2020202096
성 명	우성원
교 수 님	공영호교수님
제 출 일	2022.06.12

목차

◆ Introduction

3D Cube를 구현 후 Insert,Print,Delete,Find,Print_All,Turn,Exchange를 명령을 수행해야한다.
이때 Cube내 Block들은 Binnary Search Tree에 단어를 가지고 있어야하며
단어는 WordBook.txt 파일에서 읽어온다.Block들은 서로 Double Linked List로 연결되어있고
3차원이므로 각 Block 당 총 6개의 포인터를 가진다.Block의 6개의 포인터는 자신의 Block에서
바로 이웃하는 Block과 서로를 가리키킨다.
이 Block들을 줄세워 X,Y축으로 구성된 하나의 층으로된 2D List를 구현하고
이 층을 3개 겹쳐서 3D List를 구현한다.BST,Block,Linked List,2D List,3D List는 모두
계층 관계로 포인팅을 하며 포인팅을 위해 포인팅대상을 멤버변수로 가진다.
기본적인 알고리즘 구조는 포인팅하는 계층관계로 이루어진다.객체들은 BST와 Block을 제외하고
모두 Class Type이며 BST와 Block은 Struct Type을 지닌다.

000 001 002
100 101 102
200 201 202
010 011 012
110 111 112
210 211 212
020 021 022
120 121 122
220 221 222
Print 좌표

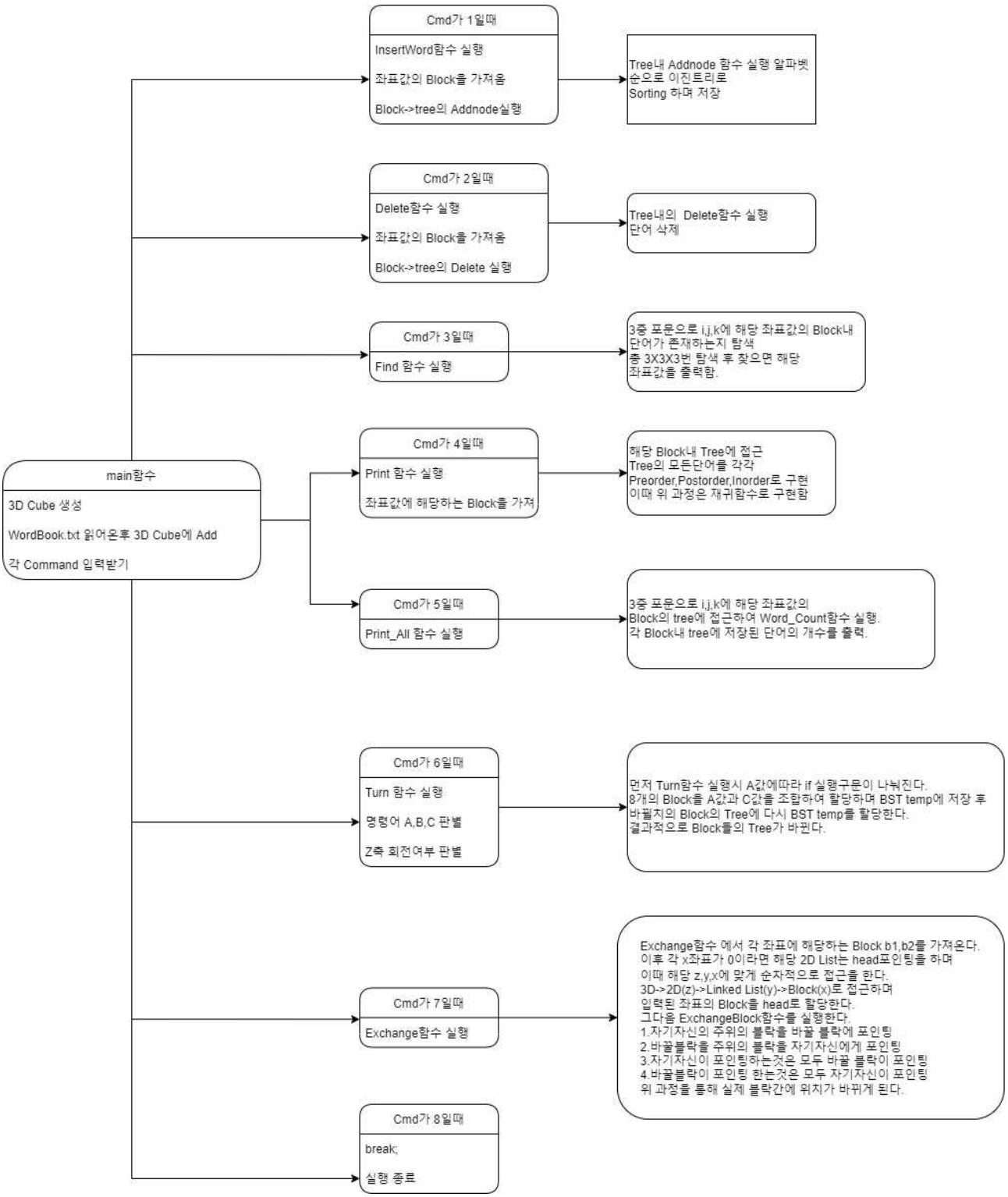
→ **0층 부분에 해당하며 Cube기준으로 전면에 해당한다.**
입력은 총 7가지 형태의 입력을 처리하며 Exchange에서 최대 7개까지의
입력변수를 받을 수 있다.
출력은 각 Command별로 다양한 형태로 나타나는데 기본적인 틀은 Print_All을 실행시
나타나는 좌표에 기인한다.
0층부터 층이 하나씩 내려갈때마다 Y축기준으로 Y값이 1씩 증가한다.
즉,Cube의 전면에서 하나씩 면이 증가한다.
이러한 구조는 Print_All,Turn에서 나타난다.

그 다음 Insert,Delete,Exchange는 Command에 대한 출력은 따로 없으며 실행 후 다음 명령을 받는다.
Find는 명령어의 번호와 찾을 단어를 입력시 해당 단어가 있는 Block의 좌표를 출력한다.
Print는 좌표를 기입시 좌표내 BST를 Preorder,Inorder,Postorder순으로 출력한다.
Turn은 Turn 실행 Command와 Turn 함수를 어떻게 실행할것인지를 나타내는 Command를 받는다.
형식은 6(Turn executing Command) A B C 과 같으며 A B C 에는 모두 각각 상수가 들어간다.
이때 A는 Cube의 회전축을 결정하며,B는 시계,반시계방향을
C는 하나의 층에서 회전축의 값을 나타낸다.

Command \Variables	A	B	C
0	Y축	시계방향	A축 위의 C점을 기준으로 회전.
1	Z축	반시계방향	
2	X축	X	

Exchange는 해당 함수 실행 명령어와 두 Block의 좌표값을 넣을시 두 Block을 바꾼다.
단순히 BST를 바꾸는 것이 아니라 한 Block이 포인팅하는 것,포인팅 받는것까지
모두 바꿔서 가상 Cube에서 실제로 바뀌게끔한다.

◆ Flowchart

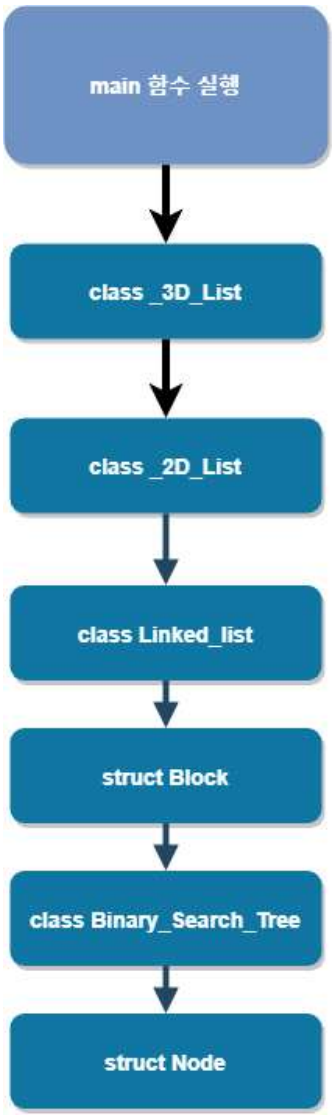

main함수
3D Cube 생성
WordBook.txt 읽어온후 3D Cube에 Add
각 Command 입력받기
Cmd가 1일때
InsertWord함수 실행
좌표값의 Block을 가져옴
Block->tree의 Addnode실행
Tree내 Addnode 함수 실행 알파벳 순으로 이진트리로 Sorting 하며 저장
Cmd가 2일때
Delete함수 실행
좌표값의 Block을 가져옴
Block->tree의 Delete 실행
Tree내의 Delete함수 실행 단어 삭제
Cmd가 3일때
Find 함수 실행
3중 포문으로 i,j,k에 해당 좌표값의 Block내 단어가 존재하는지 탐색 총 3X3X3번 탐색 후 찾으면 해당 좌표값을 출력함.
Cmd가 4일때
Print 함수 실행
좌표값에 해당하는 Block을 가져
해당 Block내 Tree에 접근 Tree의 모든단어를 각각 Preorder,Postorder,Inorder로 구현 이때 위 과정은 재귀함수로 구현함
Cmd가 5일때
Print_All 함수 실행
3중 포문으로 i,j,k에 해당 좌표값의 Block의 tree에 접근하여 Word_Count함수 실행. 각 Block내 tree에 저장된 단어의 개수를 출력.
Cmd가 6일때
Turn 함수 실행
명령어 A,B,C 판별
Z축 회전여부 판별
먼저 Turn함수 실행시 A값에따라 if 실행구문이 나눠진다. 8개의 Block을 A값과 C값을 조합하여 할당하며 BST temp에 저장 후 바뀔치의 Block의 Tree에 다시 BST temp를 할당한다. 결과적으로 Block들의 Tree가 바뀐다.
Cmd가 7일때
Exchange함수 실행
Exchange함수 에서 각 좌표에 해당하는 Block b1,b2를 가져온다. 이후 각 x좌표가 0이라면 해당 2D List는 head포인팅을 하며 이때 해당 z,y,x에 맞게 순차적으로 접근을 한다. 3D->2D(z)->Linked List(y)->Block(x)로 접근하며 입력된 좌표의 Block을 head로 할당한다. 그다음 ExchangeBlock함수를 실행한다. 1.자기자신의 주위의 블락을 바꿀 블락에 포인팅 2.바꿀블락을 주위의 블락을 자기자신에게 포인팅 3.자기자신이 포인팅하는것은 모두 바꿀 블락이 포인팅 4.바꿀블락이 포인팅 한는것은 모두 자기자신이 포인팅 위 과정을 통해 실제 블락간에 위치가 바뀌게 된다.
Cmd가 8일때
break;
실행 종료

◆ Algorithm

1. Cube탐색

먼저 Cube 탐색은 GetBlockbyIndex 함수를 이용한다.
GetBlockbyIndex함수는 3D에서 층을 가지는 2D List로 접근하며 for문을 통한 Z값에 따라
Z층에 접근한다. Z층에 접근 후 Z층이 가지는 하나의 행인 Linked_List로 접근하는데
이때 GetLinked_ListByY를 통해 X에 접근한다. 그다음 Linked_List의 Y행에서 Block*로 접근하는데
GetBlockbyX를 통해 Y행의 X번째 Block을 가져온다.
이처럼 3D->2D->Row->Block을 Class내에서 순차적으로 포인팅하면서 데이터를 가져온다.

2. 3D Linked List 구현(3D Cube)



기본적으로 상위 객체가 하위 객체를 Private으로 지니고
이를 포인팅하면서 하위객체 내의 method에 접근하여
생성하는 구조를 지닌다.

main함수가 _3D_List를 선언하면서 생성자를 실행시킨다.

Class _3D_List 내에서 생성자가 _2D_List의 생성자를 호출한다.
이때 _3D_List는 하나의 층을 만드는 _2D_List를
3회를 반복하여 Cube를 구현한다.

_2D_List에서는 Linked__list를 포인팅하는데
이때 마찬가지로 하나의 행을 만드는 Linked_list를
3회 반복하여 2차원의 하나의 층을 구성한다.

Linked_List는 Block을 포인팅하는데 역시 6개의 포인터를 가진 객체
Block을 3회 선언 후 이어주어 하나의 행을 만든다.

마지막으로 Block에서는 하나의 struct Block당 각 Binary_Search_Tree
포인팅을 할당하면서 Block내에 BST를 구현한다.

3. BST에 관한 전반적인 설명

Binary_Search_Tree는 크게 3가지 기능으로 나뉜다

-AddNode

-Delete

-Print

3-1. AddNode

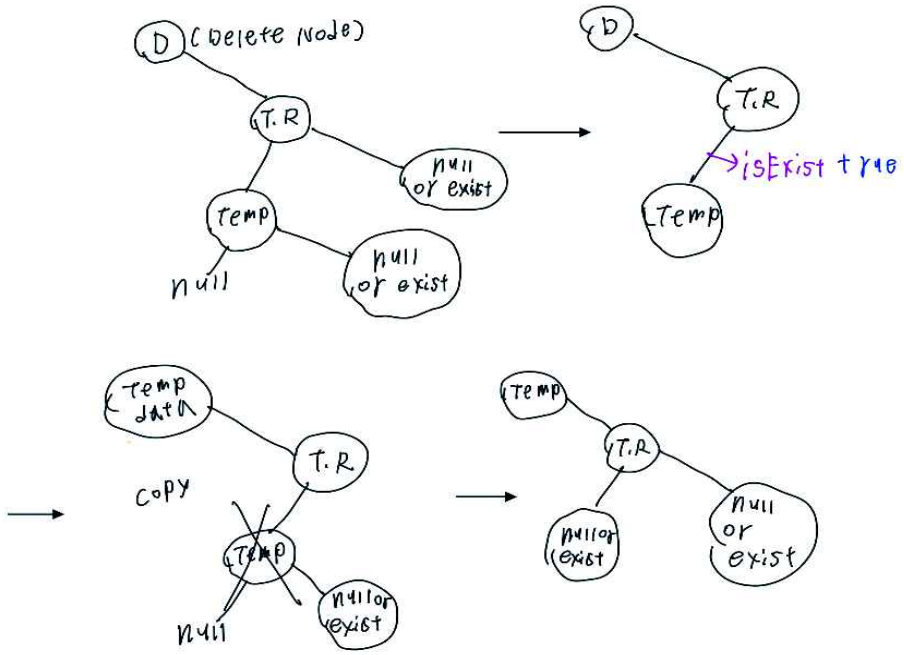
AddNode는 root node와 new node로 구분된다. 새로받은 data는 newnode에 할당하며 root가 null여부에따라 newnode의 삽입시퀀스가 달라진다. root가 null일 경우 newnode=root가 된다.
아니라면 tempRoot와 temp라는 node를 새로할당하며 temp가 nullptr이 아닐때까지 tree를 탐색한다. tree탐색은 isBig함수를 통해 newnode와 temp를 비교하며 newnode가 기존 node의 data보다 더 큰 ASCII값을 가질 때 오른쪽으로 아니라면 왼쪽으로 이동하며 탐색한다.tempRoot는 while문이 1회실행할때마다 temp와 같게해준다. 이때 temp가 nullptr일 경우tempRoot와 newnode를 isBig을 통해 비교하며 newnode의 자리를 temp에 할당한다. 마지막으로 tree에 할당된 node의 개수를 나타내는 size값을 증가시켜준다.

3-2. DeleteNode

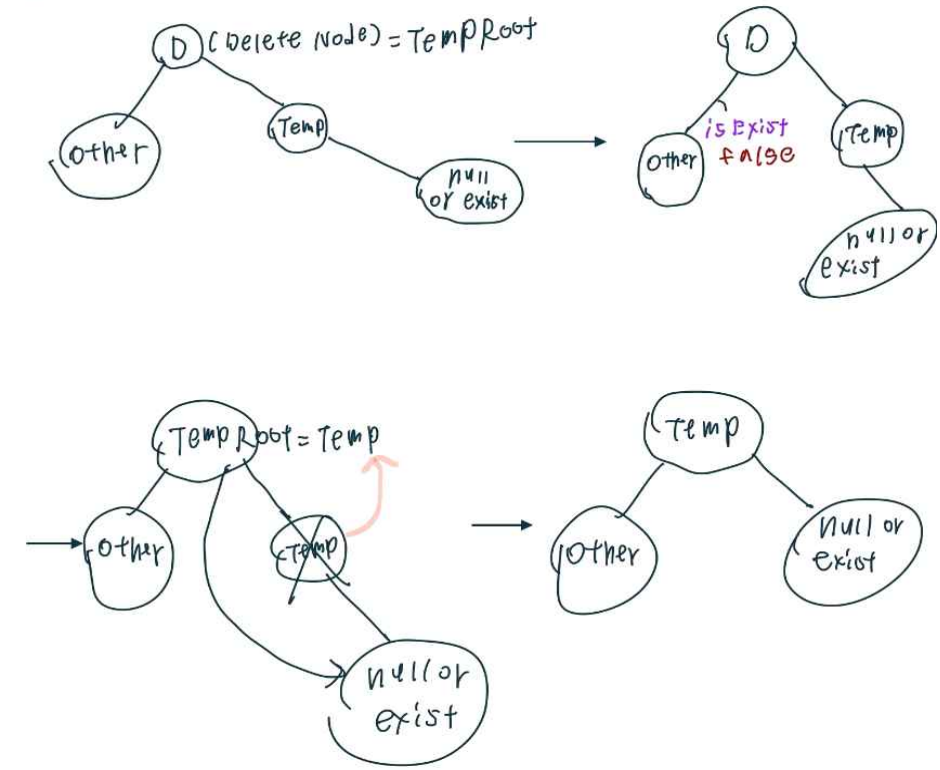
<table>
<tr><td rowspan="6">Delete Node</td><td rowspan="3">1.지워야할 node가 root</td><td colspan="2">1-1.자식이 없음</td></tr>
<tr><td rowspan="2">1-2.자식이 하나</td><td>1-2-1.자식이 왼쪽</td></tr>
<tr><td>1-2-2.자식이 오른쪽</td></tr>
<tr><td colspan="2">1-3.자식이 둘</td></tr>
<tr><td rowspan="3">2.지워야할 node가 root 아님</td><td colspan="2">1-1.자식이 없음</td></tr>
<tr><td rowspan="2">1-2.자식이 하나</td><td>2-2-1.자식이 왼쪽</td></tr>
<tr><td>2-2-2.자식이 오른쪽</td></tr>
<tr><td colspan="2">1-3.자식이 둘</td></tr>
</table>

전체 알고리즘은 먼저 지워야할 이름을 char* name으로 지정후 그 이름의 위치를 Compare함수를 통해 탐색 node내 이름과 비교하여 탐색한다. 탐색 알고리즘은 상단의 표와 같다.
1-1.먼저 지워야할 node가 root이고 자식이 없는 경우 root를 지워버린다.
1-2-1.지워야할 node가 root이고 root->right이 null인 경우 root->left를 새로운 root로 할당한다.
1-2-2.지워야할 node가 root이고 root->left가 null인 경우 root->right를 새로운 root로 할당한다.
1-3.지워야할 node가 root이고 자식이 두 개인 경우 tempRoot와 temp로 두 개의 root node를 할당 후 tree내에서 먼저 temp를 root의 right으로 1회 이동후 temp node의 left가 null일 때 까지 temp를 이동시켜준다. 즉,root의 오른쪽 tree중 가장 좌측 node에 temp를 위치시킨다. 이때 두가지 경우이다.

① Temp가 TempRoot의 왼쪽일경우



② Temp가 TempRoot의 오른쪽일경우



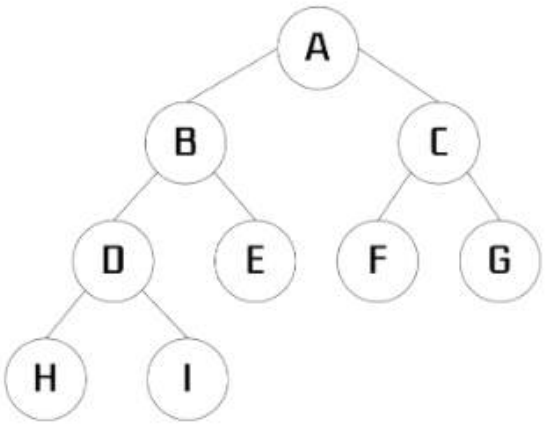
위 과정중 isLetf함수는 TempRoot의 Left에 temp가 있으면 true를 반환하는 함수이다.

위와 같은 알고리즘으로 지워야할 node가 root가 아닌 경우도 이루어진다.

다만 기준 node가 root에서 입력한 name과 같은 node로 바뀌는 것 뿐이다.

3-3. BST Print

Tree내부 Print는 총 3가지 방법을 통해 구성된다.



1) Preorder 운행 : Root → Left → Right 순으로 운행한다.
A B D H I E C F G

2) Inorder 운행 : Left → Root → Right 순으로 운행한다.
H D I B E A F C G

3) Postorder 운행 : Left → Right → Root 순으로 운행한다.
H I D E B F G C A

운행법은 위와 같이 세가지로 구성되며 모두 재귀함수로 구성하였다.

4. Turn

Turn은 Turn 실행 Command와 Turn 함수를 어떻게 실행할것인지를 나타내는 Command를 받는다.
형식은 6(Turn executing Command) A B C 과 같으며 A B C 에는 모두 각각 상수가 들어간다.
이때 A는 Cube의 회전축을 결정하며,B는 시계,반시계방향을
C는 하나의 층에서 회전축의 값을 나타낸다.

Command \Variables	A	B	C
0	Y축	시계방향	A축 위의 C점을 기준으로 회전.
1	Z축	반시계방향	
2	X축	X	

가령,6 0 0 0 라는 Command를 입력시 A,B,C는 각각 Y축,시계방향,Y=0인 면을 기준으로 회전을 뜻한다.
Print_All에서 설명한 것과 같이 출력절차는 Y축면을 기준으로 순차적으로 출력한다.
Turn함수에서 A가 0이라면 Block 8개를 Y=0인 면에서 해당 좌표값을 받아와 GetBlockByIndex를 통해 총 8개의 Block을 불러온 후 각 Block들의 Tree를 Tree temps[8]에 각각 할당한다.
그 다음에 *blocks[(i+2)%8]=temp[i] ,i는 0부터 7까지 증가하는데.이때 블록들의 시계방향 시퀀스 좌표에 따라 각각의 block들의 다음 포인터가 들어가게끔 설계하였다.

Current Block (축 제외)	Next Block (축 제외)
00	02
01	12
02	22
12	21
22	20
21	10
20	00
10	01

이처럼 A값의 축을 제외한 좌표들이 시계방향으로 90도 회전할 때 이러한 시퀀스를 지닌다는걸 알게되었으며 2개의 블록단위 만큼 시계방향으로 움직일 때 축을 제외한 좌표값은 모두 동일하다는 것을 이용하였다.
그래서 축부분은 C값을 좌표값으로 할당하였다.

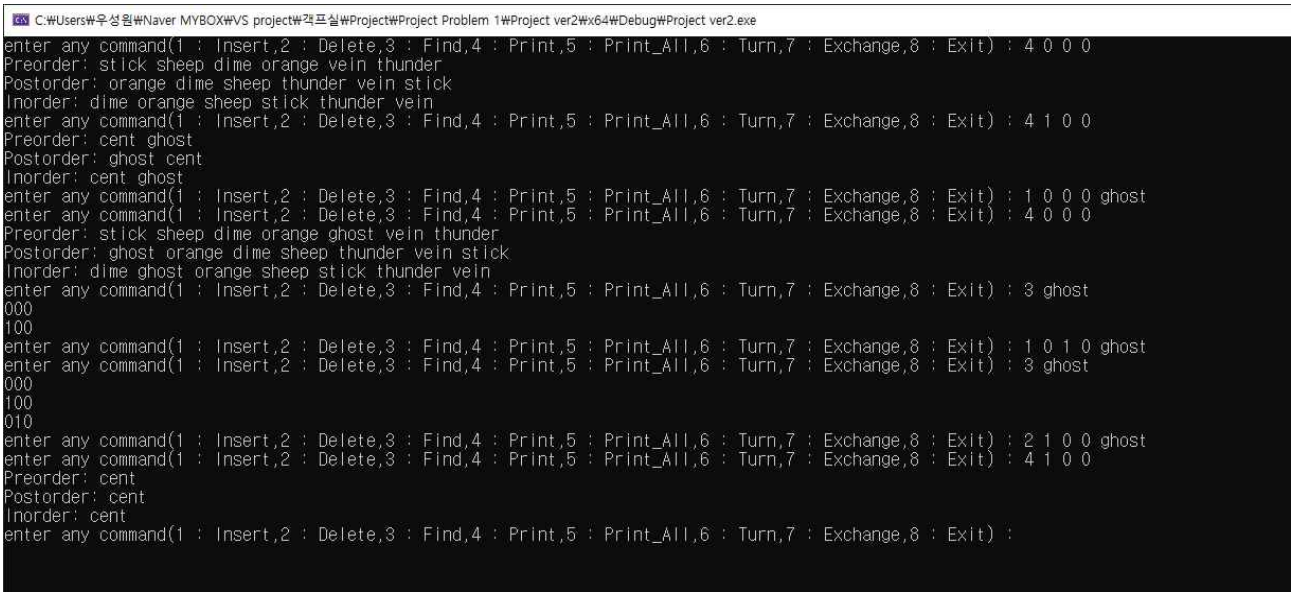
```
if (A == 0) {//y축
    blocks[0] = GetBlockByIndex(x:0, y:C, z:0);
    blocks[1] = GetBlockByIndex(x:0, y:C, z:1);
    blocks[2] = GetBlockByIndex(x:0, y:C, z:2);
    blocks[3] = GetBlockByIndex(x:1, y:C, z:2);
    blocks[4] = GetBlockByIndex(x:2, y:C, z:2);
    blocks[5] = GetBlockByIndex(x:2, y:C, z:1);
    blocks[6] = GetBlockByIndex(x:2, y:C, z:0);
    blocks[7] = GetBlockByIndex(x:1, y:C, z:0);
    Binary_Search_Tree temps[8];
    for (int i = 0; i < 8; i++) {
        temps[i] = *(blocks[i]->tree);
    }
    for (int i = 0; i < 8; i++) {
        *(blocks[(i + 2) % 8]->tree) = temps[i];
    }
```

나머지 축도 모두 동일하게 알고리즘으로 움직이며 B가 반시계방향일 경우 시계방향으로 3번 움직이게 설계하였다. 다만,A=1 즉,Z축인 경우 시계방향이 반대로 움직이기에 반대로 움직이게끔 설계하였다.

5. Exchange

Exchange의 경우 총 4가지 경우의 수로 이루어지는데,아래 4가지 절차이다.
1.자기 자신을 포인팅하는 주위 블록을 상대가 포인팅
2.상대를 포인팅하는 주위 블록을 자신을 포인팅
3.자기가 포인팅하는 주위 블록을 상대가 포인팅
4.상대가 포인팅하는 주위 블록을 자기가 포인팅
먼저 Exchange실행시 2개의 Block 좌표를 가져온다.
해당 좌표값의 Block을 가져오고 좌표중 x가 0일 경우 Linked_List가 x=0 인 블록을 head 로 포인팅 하고 있는데 이 head를 가져온 b1,b2로 각각 할당해주는 작업을 실행한다.
위 작업을 해주지 않으면 head 포인팅 다음에 이어주는 Block이 없어 Cube가 붕괴된다.
그 다음 ExchangeBlock함수를 통해 위의 4가지 절차를 실행한다.
TempBlock을 선언한 이유는 3번과정을 실행하면 Block 1의 포인터가 바뀌기 때문에 이를 바꾸기전에 저장하지않으면 4번과정을 의도대로 하지못하기 때문이다.

◆Result & Verification



먼저 예시와 동일하게 Print Command를 출력하였다.

Preorder,Postorder,Inorder순으로 잘 작동한다.

그리고 ghost command를 삽입하였다. 보다시피 ghost Command가 잘 작동하는 모습이다.

ghost를 delete하였다. 좌표를 출력하고 ghost가 삭제되는 모습을 볼 수 있다.

010 좌표에 ghost를 삽입하고 ghost를 Find했을 때 Block을 탐색하면서 탐색된 좌표를 출력한다.

다만,이 프로젝트는 000부터 블록->행->층->큐브순서 즉,Z가 1 증가하면서 다음 층 순서로

탐색하기 때문에 000->010->100으로 탐색 순서를 출력한다.

그 다음은 ghost를 지우는 모습이다. ghost를 지우고 100을 Print할시 ghost가

지워진 모습을 볼 수 있다.

이제 Turn을 실행시켜 보겠다.

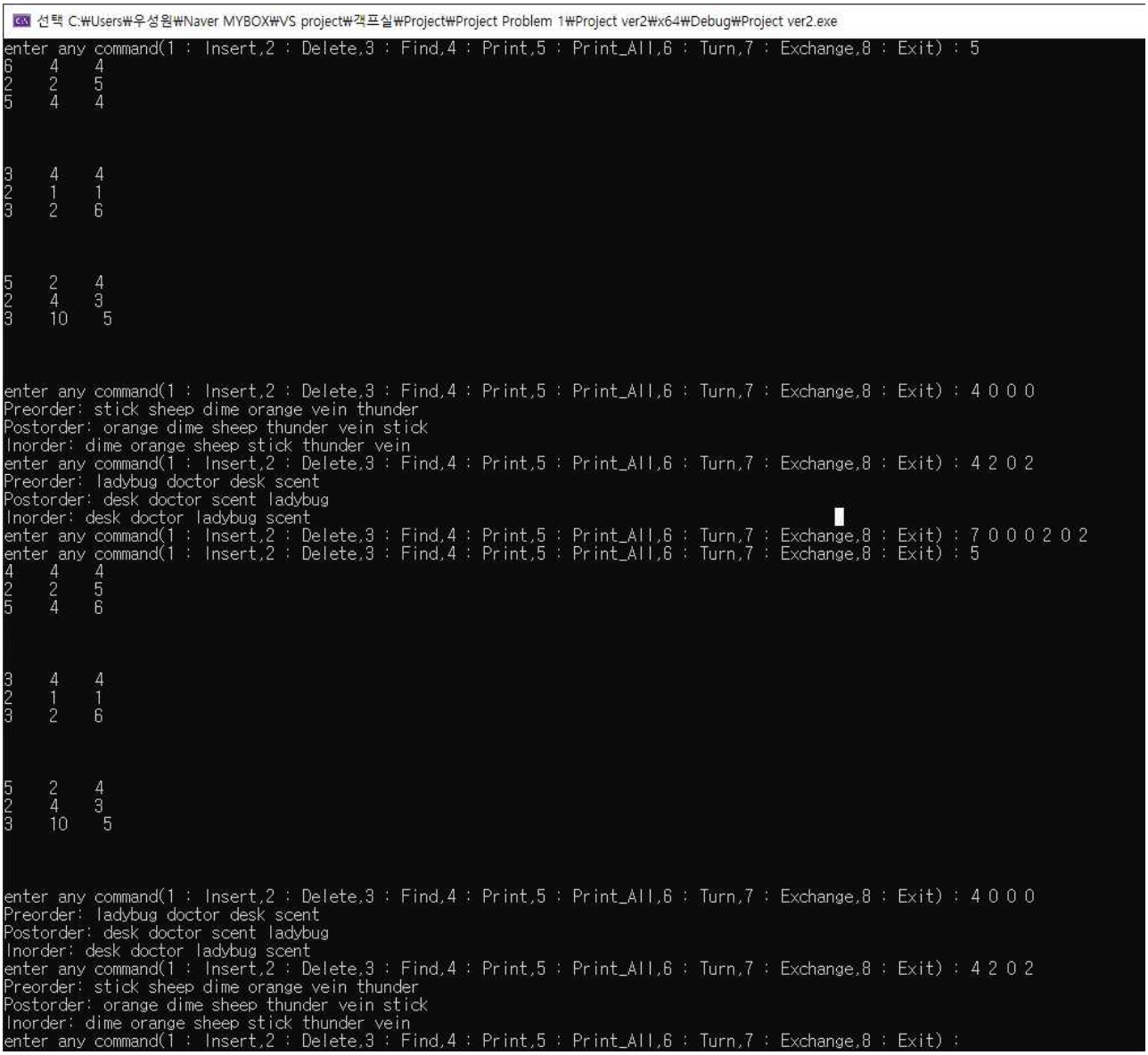
6 0 0 0을 기입시 Y=0을 회전면으로 지정하고 시계방향으로 돌리는 것을 볼 수 있다.

6 1 0 0을 기입시 Z=0을 회전면으로 지정하고 시계방향으로 돌리는 것을 볼 수 있다.

6 2 0 0을 기입시 X=0을 회전면으로 지정하고 시계방향으로 돌리는 것을 볼 수 있다.

예시와 다른 것은 개인적인 생각으로 예시가 오류라고 생각한다.

왜냐하면 Print_All에도 (Z,Y,X)순으로 단어가 표기,Insert에도 Z,Y,X순으로 들어가는 것으로 보아 모든 좌표계가 (Z,Y,X)를 이용한다는 것을 알 수 있다.



마지막으로 000 좌표와 202좌표를 Exchange할 것이다.

두 Block내의 단어의 개수도 바뀌었고 tree자체가 바뀐 것을 확인할 수 있다.

포인터를 이용해 가상 Cube내에서 실제로 바뀐 것이다.

◆ ConClusion

일단 기본적인 구조설계부터 어려웠다.그리고 BST내에서 Delete 실행시 NULL을 만나 계속 꺼지는 문제가 있었는데 Tree 설계가 잘못되었다는 것을 알고 다시 공부 및 설계해서 도움이 되었다.
그리고 기본 좌표계가 x,y,z가 아닌 z,y,x이기에 설계하는데 많은 공을 들여야했다.
또한 예시랑 내 프로젝트의 Turn값이 일치하지않아 문제를 파악하는데 시간이 걸렸다.
특히,GetBlockbyIndex는 참 잘 설계한 것 같다.Cube 생성 원리를 이용해 Index만 기입시 바로 해당 좌표의 Block을 반환하게끔 하는 이 함수 덕분에 설계가 수월했다.
하지만 Turn은 정말 어려웠다.그래도 회전하는 축을 제외한 나머지 좌표값들의 공통적인 이동 시퀀스를 찾아내 그것을 적용한 것은 잘했다고 생각한다. 그리고 Exchange를 할 때 4가지 절차를 거치는데 이때 따로 포인터를 temp에 담아서 다시 사용할수있게 하여 문제를 해결한 것은 잘한것같다.
전반적으로 코드양이 길고 간결하지 못한느낌이 있는데..eNum으로 자주쓰이는 반복명령을 할당하거나 조금 더 읽기 편하게 문맥적인 코드로 통일성 있게 구성했으면 가독성이 좋을 것 같다.