# 객체지향프로그래밍설계

## 프로젝트


담 당 교 수 : 공 영 호

학 과 : 컴퓨터정보공학부

학번 : 2019202021

성 명: 정 성 엽

# OOP Project 보고서

## 1. Introduction

- Problem 1

  ■ 2D Linked List를 쌓아서 올린 3x3x3크기의 3D 큐브를 만들며 이 큐브는 각 면이 회전할 수 있고 Block끼리 교환할 수 있도록 한다. 또한 큐브의 각 Block은 내부에 Binary Search Tree(BST)를 가지고 있어 BST를 이용해 단어의 저장, 삭제, 탐색, 출력이 가능하다. 가장 중요한 것은 각 노드는 상, 하, 좌, 우, 앞, 뒤로 연결되어 있는 구조로 구성하며, 각 노드가 서로 쌍방향 연결이 되도록 한다. 3D Linked List를 만들기 위해서는 먼저 1D Linked List를 쌍방향 연결하고 이 1D Linked List를 쌓아서 2D Linked List를 만들고 각 노드가 쌍방향 연결되도록 한다.. 이제 만들어진 2D Linked List를 쌓아서 3D Linked List를 만들고 쌍방향 연결되도록 하면 문제에서 요구하는 3D 큐브를 만들 수 있다.

- Problem 2

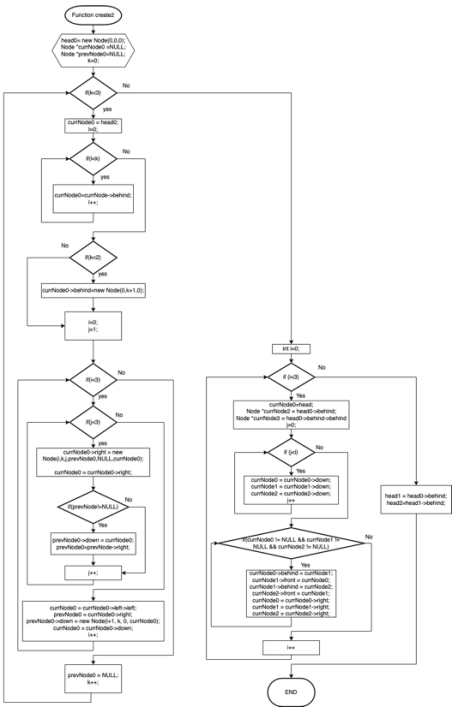
  ■ 3D 큐브를 만든 후에는 WordBook.txt 파일을 읽고 큐브의 각 Block에 BST구조로 저장한다. WordBook.txt 파일은 줄마다 'z좌표, y좌표, x좌표 단어'로 구성되어 있으며 각 줄에 적혀있는 좌표에 위치한 블럭의 BST 구조에 단어를 사전식 정렬 순으로 저장한다. 프로젝트는 3D 큐브와 BST에 대해서 Insert, Delete, Find, Print, Print_All 명령어를 수행할 수 있어야한다. Insert는 좌표와 단어를 입력 받고블럭에 단어를 사전식 정렬을 기준으로 삽입한다. 만약 같은 단어가 있을 경우에는 삽입하지 않는다. Delete는 좌표와 단어를 입력 받고 해당 좌표의 블럭에 단어가 있을 경우 삭제하며 없을 경우 명령을 무시한다. 단어가 삭제될 경우에도 BST의 사전식 정렬은 유지되어야 한다. Find는 단어를 입력 받고 해당 단어를 가지고 있는 모든 큐브의 좌표를 출력한다. 이때 좌표의 출력 순서는 임의로 한다. Print는 좌표를 입력 받고 해당 좌표에 있는 모든 단어를 출력한다. 이때 좌표에 존재하는 단어의 출력 순서는 Pre-order, Post-order, In-order 3가지 방식 모두 사용하여 출력한다. 마지막으로 Print_all은 모든 Block에 각 저장하고 있는 단어의 개수를 출력한다.

- Problem 3

  ■ 3D 큐브를 Turn 명령어에 따라 각 면을 회전시키고 Exchange 명령어에 따라서 Block의 위치를 교환한다. Turn은 Table 1에서 설명하고 있는 바와 같이 전면, 윗면, 측면에 대해서 시계방향, 반시계 방향으로 면의 회전을 수행해야 하며, Exchange는 2개의 Block의 위치를 교환해야 한다. 또한 Turn과 Exchange의 수행 후에도 3D Cube의 모든 Block간 연결은 올바르게 연결되어 있어야 하며 반복해서 수행해도 오류가 없어야 한다. 이 때문에 모든 함수는 블럭의 z, y, x값에 따라 작동하는 것이 아닌 각 블럭의 link를 이용해서 만들도록 해야 한다. Turn은 큐브의 한 면을 회전하는 것으로, 큐브의 회전의 경우의 수는 전면, 측면, 윗면과 각 면의 양방향 회전 6개의 경우가 있으며 각 면에 대해서 몇 번째 면인지를 고려하면 기존 6개의 경우에 각 3개씩 있으므로 18개의 경우를 처리해야 한다. Exchange는 큐브의 2개의 Block의 위치를 교환하는 것으로, 이 때 Block의 내부의 BST가 아닌 큐브의 Block을 교환해야 한다. Exit는 명령의 입력은 '8'이며 프로그램을 종료한다.

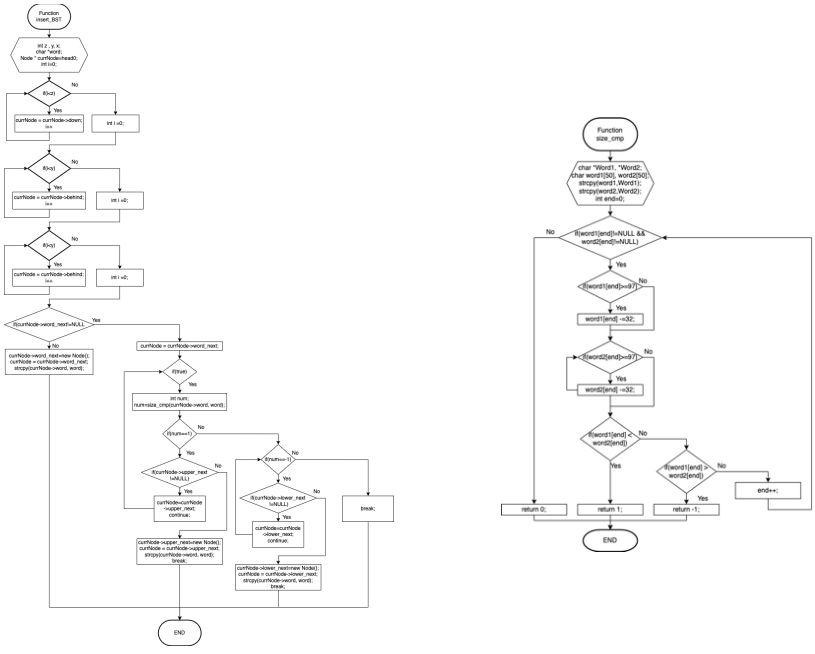
## 2. Flowchart

<Problem 1 3D Linked List로 구성된 3D 큐브 생성>



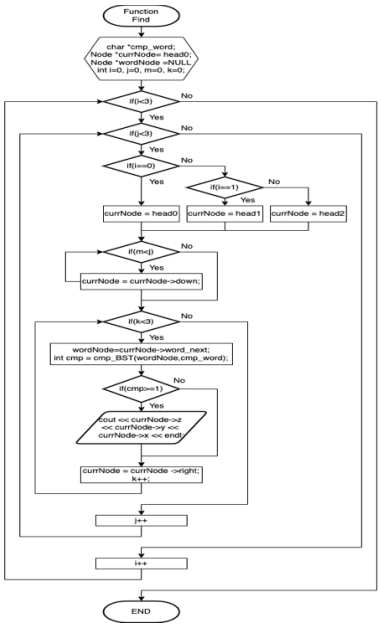

프로그램 시작 시 3x3x3 3D 큐브를 생성하기 위한 함수로 Node 생성 시 z, y, x의 값과 up, down, left에 대한 정보를 바로 입력한다. 이를 이용해 2D linked list 3개를 생성하고 이를 생성한 후에는 3개의 2D linked list를 연결하여 3D linked list로 만든다.

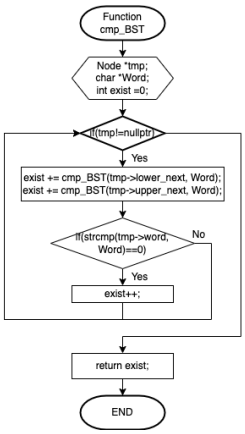
<Problem 2 Binary Search Tree Insert와 이에 쓰인 Function size_cmp >




3D 큐브에서 해당 좌표까지 이동하고 Binary Search Tree 방식으로 저장이 되는 함수이다. Function size_cmp를 통해 이동하면서 크기를 비교하며 클 경우, 1 작을 경우 -1, 같을 경우 0을 반환한다.

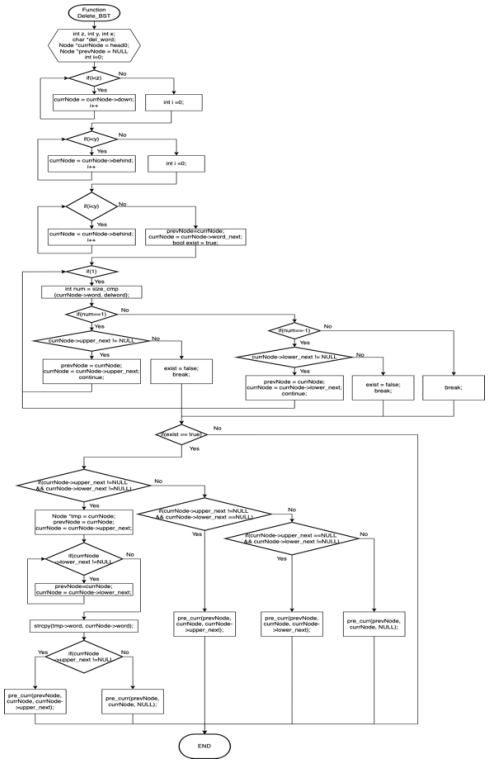
<Problem 2 Find 구현과 이에 쓰인 Function cmp_BST>

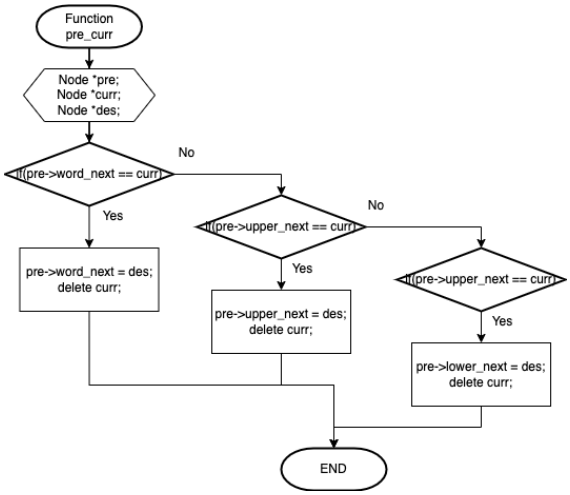



모든 block을 거쳐서 해당 BST에 원하는 단어가 있는지 찾는 함수로 Function cmp_BST를 통해서 단어가 있을 경우 1 이상을 반환하여 단어의 유무를 판단한다. 이 비교는 재귀함수를 통해 nullptr일 때까지 진행된다.

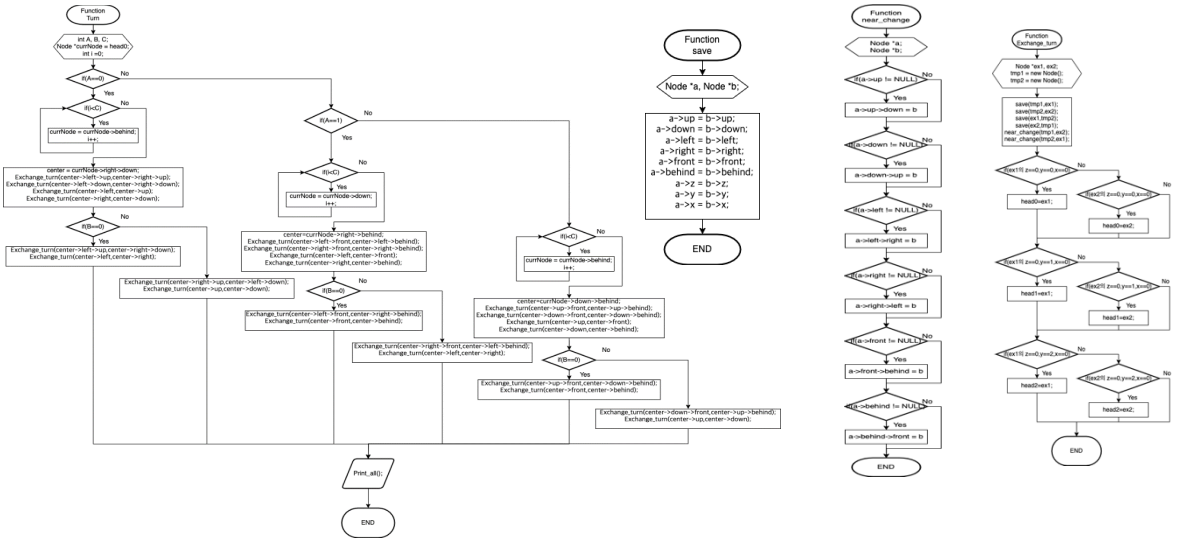
<Problem 2 Delete 구현과 이에 쓰인 Function pre_curr>



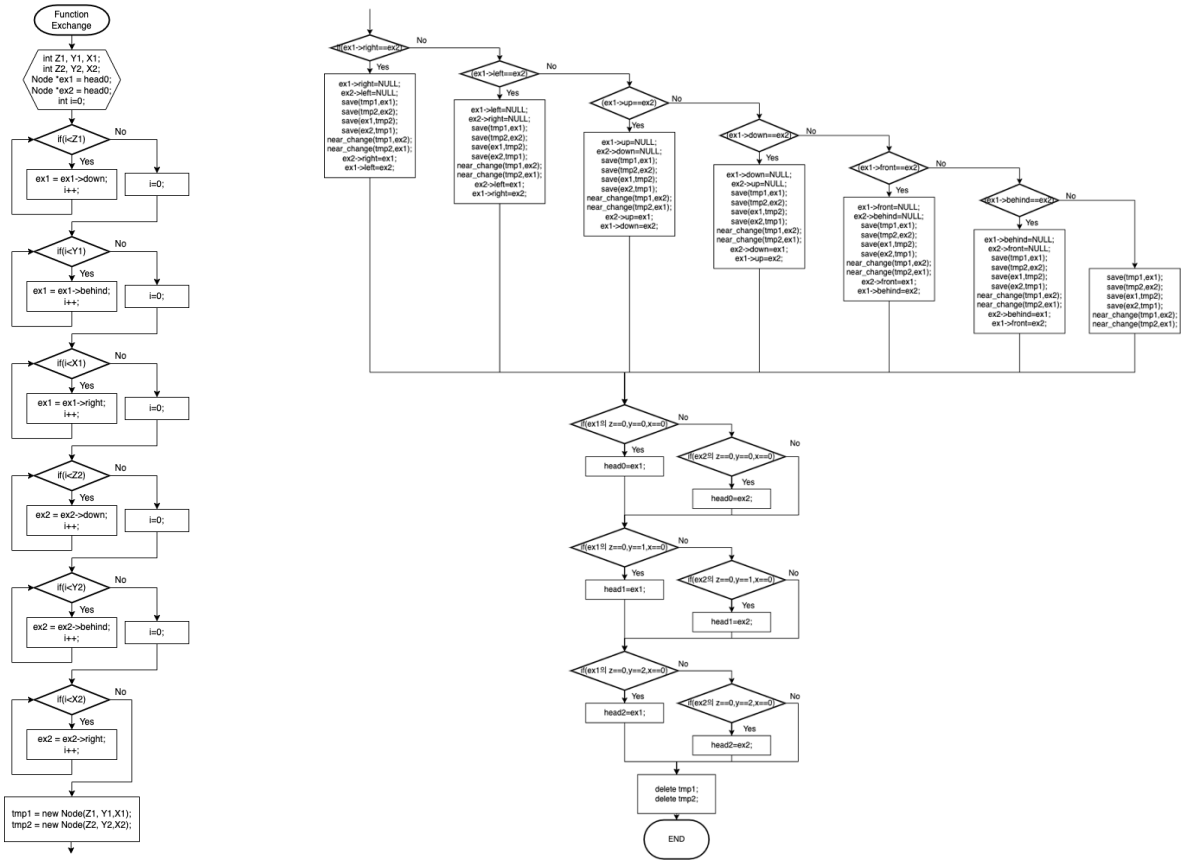

원하는 좌표의 단어를 삭제하는 함수로 삭제 방법은 BST의 삭제 방법에 따라 설계하였으며, Function pre_curr을 통해 이전 노드와 현재 노드와의 관계를 통해 삭제 후의 연결을 관리하였다.

<Problem 3 Turn 구현과 이에 쓰인 Function Exchange_turn, save, near_change



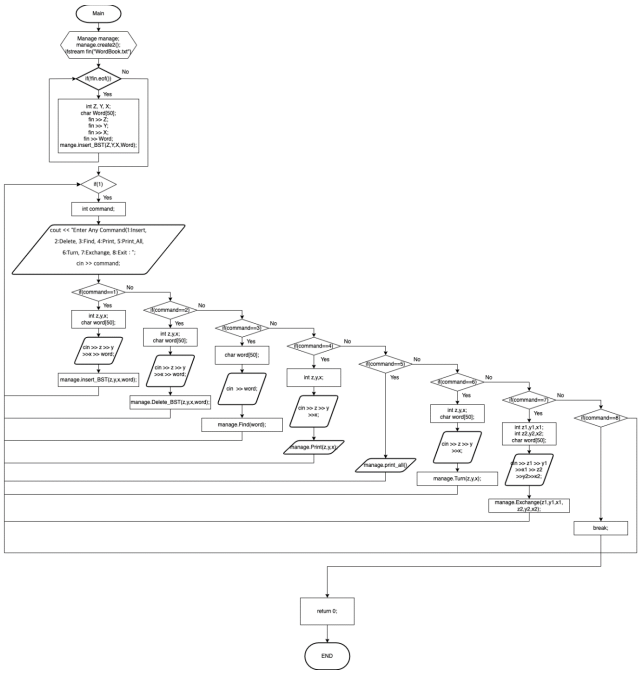

Block을 회전시키는 함수로 A의 값에 따라 면의 방향의 경우를 3가지로 나누고 C의 값에 따라 순서의 경우를 3가지로 나눈다. 마지막으로 B의 값에 따라 회전 방향의 경우를 2개로 나누어 진행한다. 마지막으로 전에 사용한 Print_all 함수를 사용하여 모든 블록의 각 단어 개수를 출력한다.

<Problem 3 Exchange 구현




원하는 block 개의 위치를 바꾸는 함수로 두 block이 상, 하, 좌, 우, 앞, 뒤로 붙어있거나 아닌 경우 총 7개로 두어 block의 위치를 바꾸어 예외 사항을 처리하였다.

<Main 구현 및 전체적 흐름>

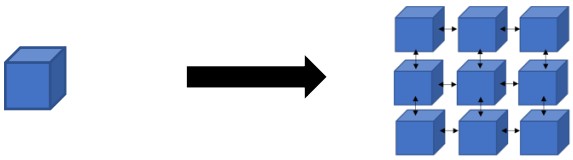



지금까지의 함수들을 사용할 main으로 3D cube를 생성하고 "WordBook.txt"로 부터 좌표 및 단어를 받아서 저장한다. 이후 command 숫자에 따라 class에 있는 멤버 함수를 호출하여 프로그램을 진행하고 command가 8인 경우 반복문을 탈출하고 프로그램을 종료한다. 이 main 함수가 프로그램 전체적 흐름을 나타내고 각 함수에 대한 흐름은 위에서 나타낸 flowchart와 같다.

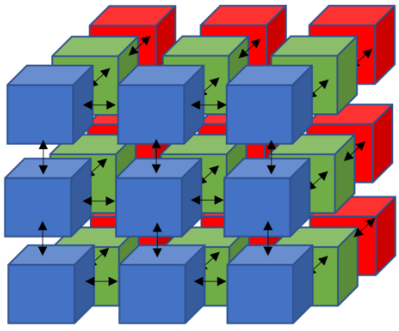
## 3. Algorithm

<Cube 생성 방법>

0, 0, 0 좌표를 모든 것의 시작이라고 생각한다. 그 후 전면에 대한 2D Linked List를 생성한다. 이때 노드는 상, 하, 좌, 우로 연결되어있다.

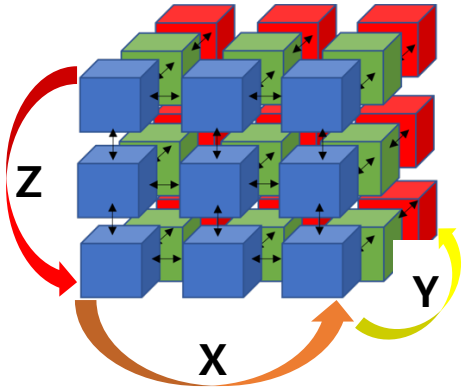


같은 방법으로 0, 1, 0과 0, 2, 0 을 시작으로 2D Linked List를 생성하고 생성된 2D Linked List들 3개를 이제 앞, 뒤로 이어서 3D 큐브를 완성한다.



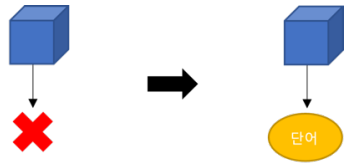
<Binary Search Tree 생성 방법>

- Cube 탐색 방법
  - 0, 0, 0을 시작으로 입력 받은 z만큼 아래로, y만큼 뒤로, x만큼 오른쪽으로 각각 반복하여 이동한다.



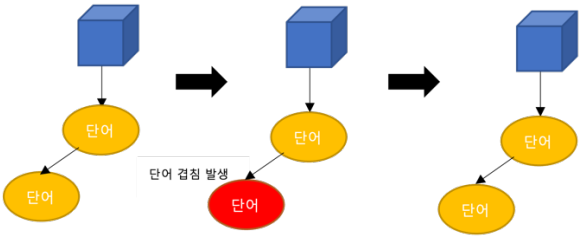

- Binary Search Tree에 저장
  - 이진 트리에 단어 저장하는 과정은 해당 단어의 순서가 사전식 정렬 기준으로 비교 단어보다 뒤인 경우 오른쪽, 앞인 경우 왼쪽으로 이동한다. 만약 이동하기전 이동할 위치가 NULL인 경우 해당 위치에 단어를 저장한다. 만약 처음부터 단어가 없으면 처음에 단어를 저장한다. 있는 단어라면 무시한다.

    1) Block에 저장된 단어가 없는 경우



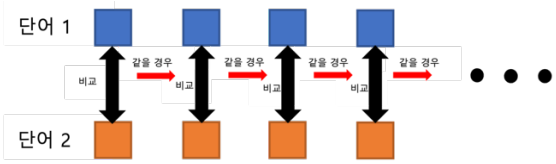

    2) 입력할 단어가 앞, 뒤인 경우


    3) 단어가 겹치는 경우




- 단어 크기 비교 과정

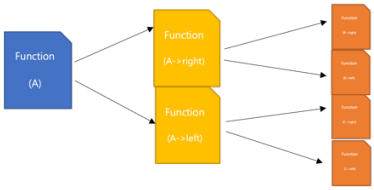
■ 단어를 첫 문자부터 비교해 가며 같을 경우 뒤의 단어를 비교하며, 크기가 크면 1을 반환, 작으면 -1을 반환하고 끝까지 같으면 0을 반환한다.




<Binary Search Tree에 저장된 단어 삭제 >

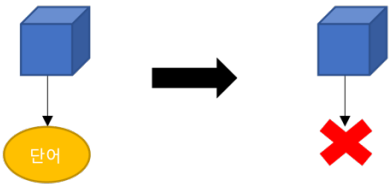
- 해당 Block의 BST에 저장된 모든 단어 비교하며 같은 단어가 있는 경우 삭제한다

■ 모든 단어 탐색은 Recursion을 하도록 한다.

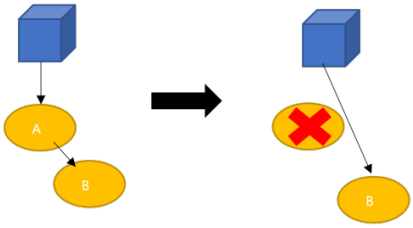



■ 위 방법은 Binary Search Tree 탐색 또는 출력 그리고 삭제에 사용한다.

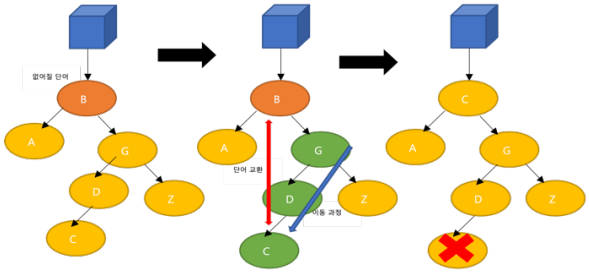
■ Child가 없는 단어인 경우 단어도 삭제하고 이전의 연결도 NULL로 바꾼다.




■ Child가 1개인 경우 단어를 삭제하고 이전 연결과 Child를 연결해준다.



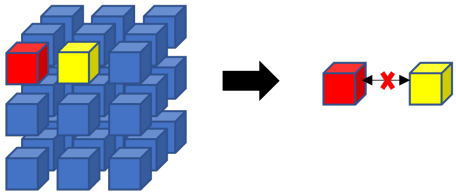

■ Child가 2개인 경우 해당 단어의 오른쪽으로 이동 후 가장 작은 단어와 바꾸고 바꾼 노드를 삭제하고 Child가 있으면 이전 노드와 Child를 연결하고, 없으면 NULL과 연결한다.



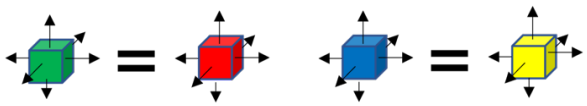

<Block 교환 알고리즘>

- Block 두 개의 관계 확인

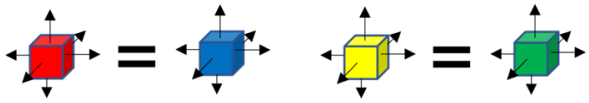
■ 두 Block이 붙어 있는 경우 두 Block 간의 연결을 해제 한다. 이 연결을 두 노드를 바꾼 후에 연결한다. 붙어 있지 않는 경우 그대로 진행한다.



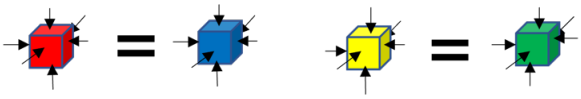
■ 각 Block의 상, 하, 좌, 우, 앞, 뒤 연결 정보를 임시 노드에 각각 저장한다.



■ 임시로 저장된 연결정보를 서로 바꾸어 저장한다.



■ 임시로 저장된 연결정보에서 주변 Block에 따라 연결을 바꾸어 저장한다.



<큐브 회전 알고리즘>

- 결국 회전이라 함은 2D linked list 사이에서의 이동이라고 보면 된다. 이 때 3*3의 면에서 가운데 있는 block은 회전하지 않는다. 이동 과정은 아래의 표를 이용해서 보인다.

| | | |
|---|---|---|
| A | B | C |
| H | X | D |
| G | F | E |

[회전할 면]

■ 먼저 A, C를 교환하고 G, E를 교환한다. 그리고 B, H를 교환하고 D, F를 교환한다.

| | | |
|---|---|---|
| C | H | A |
| B | X | F |
| E | D | G |

[중간 과정]

■ 이제부터 시계 방향과 반시계 방향인 경우 2개로 나누어서 진행된다. 모든 경우가 위의 중간 과정을 가진다.

■ 시계 방향인 경우 C, G를 교환하고 B, F를 교환한다.

| | | |
|---|---|---|
| G | H | A |
| F | X | B |
| E | D | C |

[시계 방향 회전]

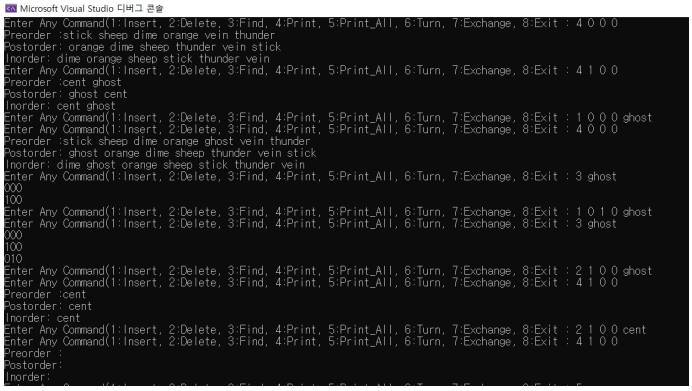
- 반시계 방향인 경우 H, D를 교환하고 A, E를 교환한다.

| | | |
|---|---|---|
| C | D | E |
| B | X | F |
| A | H | G |

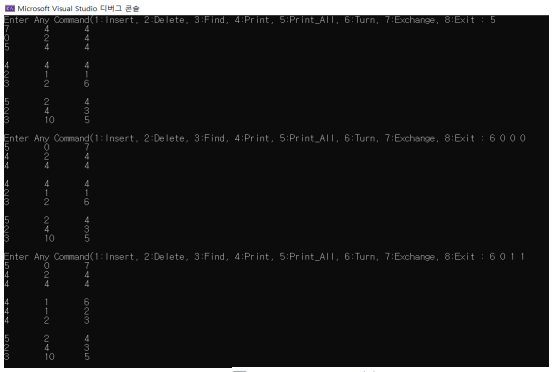
[반시계 방향 회전]

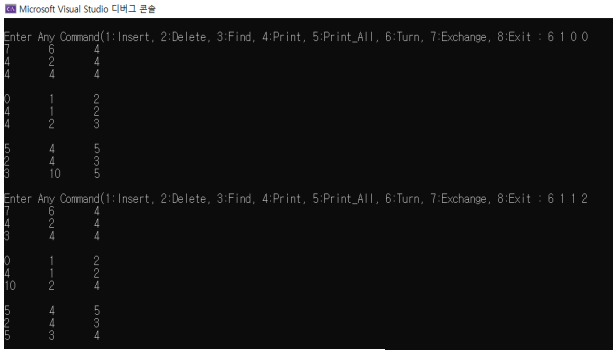
- 위 방법을 이용해 전면, 윗면, 측면 모두 회전 가능하다.
- 이때 사용하는 block 교환은 위에서 설명한 block 교환 알고리즘을 사용한다. 이때 교환하는 블록은 절대 붙어있지 않으므로 각 Block 연결 저장 후 바꾸어 원래 Block에 저장, 그리고 주변 Block의 연결 저장한다.

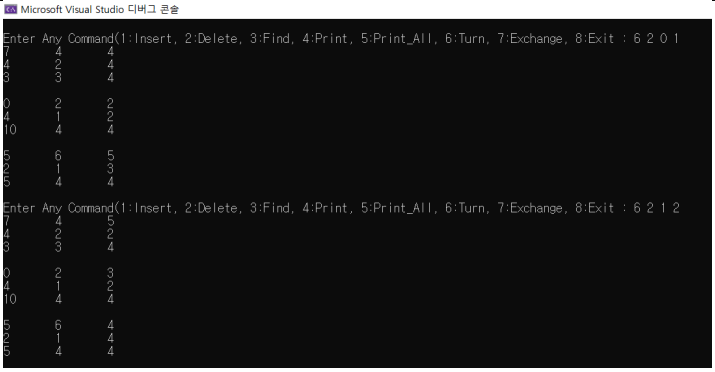
# 4. Result & Verification



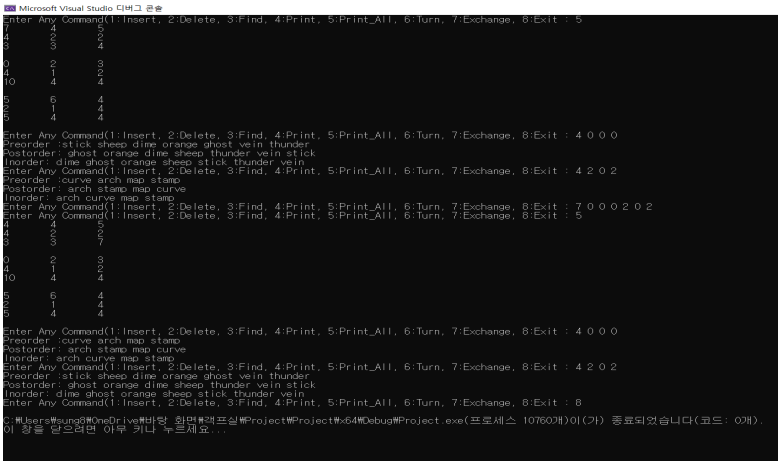

**[사진. Insert, Delete, Find, Print 작동 확인]**



**[사진. Print_All, Turn 작동 확인]**




**[사진. Exchange, Exit 작동 확인]**

# 5. Conclusion

- 처음 3D 큐브 생성 시 2D Linked List를 반복문이 아닌 각각 따로 만들도록 했다. 후에 이 코드를 다시 읽고 반복문으로 정리하여 코드를 줄여서 제작하고자 하였고 이때 만들은 3D 큐브 생성 함수를 새로운 버젼으로 정리하였다. 이 반복문으로 만들어진 2D Linked List 3개는 앞, 뒤로 연결해 마무리한다. 추가적으로 클래스 소멸 시 노드가 잘 사라졌는지 확인하기 위해 임의의 함수로 각 노드를 출력하는 과정에서 큐브 생성할 때 실수로 3,0,0 노드, 3,1,0 노드, 3,2,0 노드가 생성되었음을 알게 되었고 생성 함수를 수정하였다.

- Binary Search Tree를 생성하는 것은 이전 과제를 참고하여 제작하였으며, 추가 삽입, 찾기, 출력 또한 이전 과제를 참고하였다. 다만 원하는 Block에 저장된 모든 단어를 출력할 때 pre-order, post-order, in-order 방식 총 3가지 방법으로 출력하는데 이는 재귀함수로 이동하면서 출력하되 재귀함수 호출과 출력 함수 위치 관계에 따라 달라짐을 확인하여 설계하였다. 추가적으로 모든 재귀함수는 단어의 왼쪽부터 간다. Binary Search Tree에서 단어를 삭제하는 것도 이전 과제를 참고하여 제작하되 그 때 고찰로 생각한 이전 노드와 현재 노드에 대한 관계를 따로 저장하여 후에 연결을 다시 하도록 했다.

- 3D 큐브의 회전 알고리즘을 생각할 때, 처음에는 Block을 떼서 원하는 위치에 붙이고 원래 있던 것을 떼서 이동하는 것을 반복하여 말 그대로 회전을 하도록 하려고 했다. 이 방법은 tmp에 임의로 저장하여 계속 이어 나갈 수 있지만 계속 떼었다 붙이며 이동할 때 해제되어 있는 Block의 관리가 어렵다고 생각했다. 그리고 이에 대한 새로운 함수를 제작이 필요했다. 그래서 먼저 만든 Exchange 함수를 활용하여 시계 방향 회전과 반시계 방향 회전을 만들었고 해당 알고리즘은 위 보고서 내용과 같다. 이 방법을 이용했기에 더 간단하고 쉽게 Turn을 구현하였다.

- 프로그램 종료 후 소멸자가 진행될 때 Binary Search Tree에 새로 할당된 Node를 삭제할 때는 출력할 때와 같이 재귀함수를 통해 진행했고, 이를 활용하여 3D 큐브의 Node를 해제할 때도 재귀함수를 통해 진행했다. 삭제하는 노드에 함수로 오른쪽, 아래, 뒤에 대해 진행하여서 2,2,2부터 삭제가 진행되도록 했다. 이때 삭제하기 전 Exchange 함수에서 사용했던 near_change 함수에서 매개 변수는 삭제할 노드와 NULL로 하여 삭제할 노드의 주변 모든 노드와의 연결을 NULL로 바꾼다. 그래서 Recursion을 사용할 때 이미 삭제한 노드에 대해서 문제가 생기지 않는다.