

# Data Structure Project 1

컴퓨터정보공학부 2019202023 강병준



과목	데이터구조실습(수) 데이터구조설계(월 5 수 6)
교수	실습-공진홍 교수님 설계-이기훈 교수님
학번 및 이름	2019202023 강병준
전공	컴퓨터정보공학부
제출일자	2022/10/13

## Introduction

DS project 1 프로젝트는 2D Linked List(Loaded\_LIST), Binary Search Tree(Database\_BST), Stack, Queue 등의 자료구조를 이용하여 최종적으로 사진을 편집하는 프로그램을 구현하는 것을 목적으로 한다.

우선, 각 명령어와 자료 구조들과의 관계, 자료구조와 다른 자료구조와의 관계를 설명할 것이다.

### LOAD

LOAD 명령어는 파일 경로를 통해 csv 파일을 읽어오고, img\_files 라는 파일 선행 노드를 생성해 해당 파일 내부에 저장되어 있는 제목과 고유번호에 대한 정보를 Loaded\_LIST 라는 2 차원 연결 리스트 자료 구조에 저장한다. 저장되는 노드는 Loaded\_LIST\_Node 클래스 기반으로 생성하고, 각 노드들은 고유 번호, 폴더 이름, 제목에 대한 정보를 가지고 있다. csv 파일로부터 읽어와 Loaded\_LIST 에 저장된 노드의 개수가 100 개가 넘을 시에는 제일 먼저 들어온 노드, 즉 head 를 삭제하여 tail 에 새로운 노드를 추가함으로써 100 개가 넘지 않도록 유지한다. LOAD 명령어에 대한 csv 파일은 깃허브 repository 에 주어진 csv 파일을 디폴트로 할 것이기 때문에 파일 이름 수정에 대한 validation 은 고려하지 않아도 된다.

### ADD

앞서 설명한 LOAD 명령어와는 노드를 추가한다는 개념에서는 같지만, ADD 의 경우 새로운 폴더 이름이 담긴 파일 선행 노드와 해당 파일 선행 노드가 가리키는 정보들이 담긴 노드들을 저장한다. 이는 1 번째 인자인 폴더명과 2 번째 인자인 csv 파일명을 통해 경로 설정 및 csv 읽기를 통해 LOAD 와 마찬가지로 Loaded\_LIST 자료 구조에 저장한다. LOAD 명령어와 마찬가지로 ADD 에 의해 추가된 노드까지 총 노드의 개수가 100 개 이상이 되면 제일 먼저 들어온 노드를 삭제 후 삽입한다. 1 번째 인자와 2 번째 인자가 존재하지 않으면 파일 경로 및 csv 파일을 읽어올 수 없기에 error code 200 을 출력한다. 또한 ADD 는 기존에 추가하는 개념이기 때문에 Loaded\_LIST 가 없다면 마찬가지로 error code 200 을 출력한다.

### MODIFY

MODIFY 명령어는 LOAD 명령어와 ADD 명령어에 의해 구축된 Loaded\_LIST 자료 구조에 저장된 노드의 고유 번호를 수정하는 명령어이다. 1 번째 인자로 폴더명이 주어지고, 2 번째 인자로 제목이, 3 번째 인자로 해당 제목을 가진 노드의 수정된 고유 번호를 저장해주어야 하기 때문에 인자의 구성에 있어 인자가 부족하게 된다면 error code 300 을 출력한다. 또한 고유 번호를 수정하기 위해 사용하는 명령어이므로 인자의 개수 및 양식을 충족하였다고 하더라도 고유 번호가 중복된다면 수정하는 의미가 없기 때문에 마찬가지로 error code 300 을 출력한다.

### MOVE

MOVE 명령어는 LOAD 명령어와 ADD 명령어를 통해 구축된 Loaded\_LIST 에 저장된 모든 노드들을 Binary search tree 에 저장한다. BST 는 노드의 정량적인 특정 값에 의해 크고 작음의

구별에 따라 삽입되는 위치가 달라지며, 최대 2 개의 자식 노드를 갖는 트리 자료 구조이다. MOVE 에 의해 삽입되는 위치는 노드들의 고유번호의 크기에 따라 삽입 위치가 정해지게 된다. 따라서 Loaded\_LIST 자료 구조는 Database\_BST 에 구축을 위한 일종의 버퍼 역할을 수행하므로 Database\_BST 에 저장된 노드는 Loaded\_LIST 에서 제거된다. 최대 100 개의 노드를 가지는 Loaded\_LIST 자료 구조처럼 Database\_BST 자료 구조 또한 최대 300 개의 노드를 가질 수 있으며, 300 개가 넘게 되면 고유번호가 가장 작은 노드부터 삭제 후 새로운 노드를 추가하는 방식으로 구축해야 한다. 300 개가 넘었을 경우의 삭제 원리에 대해서는 추후에 Algorithm 부분에서 자세히 작성할 것이다. 버퍼에서 노드를 옮기는 것이기 때문에, 버퍼 역할인 Loaded\_LIST 에 아무 노드도 저장되어 있지 않은 상태라면 error code 400 을 출력한다.

## PRINT

PRINT 명령어는 Database\_BST 자료 구조에 저장되어 있는 모든 노드들의 정보를 출력하는 명령어이다. 노드들의 정보를 출력하는 방식은 BST 의 순회 방법에 따른 차이가 있다. BST 의 순회 방법은 크게 3 가지가 있는데 전위 순회, 중위 순회, 후위 순회가 있다. PRINT 명령어는 중위 순회를 통해 출력을 할 것이므로, 노드 고유번호의 오름차순 형식으로 출력될 것을 기대할 수 있을 것이다. 출력의 경우 폴더명 / "제목" / 고유 번호 의 양식으로 출력하며, Database\_BST 를 순회해야 하는데 존재하지 않는 경우 PRINIT 명령어를 사용할 수 없으므로 error code 500 을 출력한다.

## SEARCH

SEARCH 명령어는 인자로 주어진 문자열을 바탕으로 Database\_BST 에 저장되어 있는 제목과 비교해 해당 문자열이 들어가 있는 모든 노드들에 대한 정보를 출력하는 기능을 수행한다. 단순한 검색 알고리즘을 구축하는 것이 아닌, Iterative post-order 와 boyer-moore algorithm 을 사용할 것이다. Iterative post-order 는 Database\_BST 를 순회하며 자료 구조인 Queue 에 모든 노드들에 대한 정보를 저장한다. Queue 에 저장되어 있는 노드는 Queue 의 선입선출 방식을 고려하여 pop 을하고, 보이어-무어 알고리즘을 적용함으로써 찾는 문자열이 포함되어 있는지 검색을 할 것이다. 따라서 반복문으로 구현한 post-order 는 Database\_BST 의 정보를 Queue 에 저장하기 위해, 보이어-무어 알고리즘은 Queue 에 저장되어 있는 정보들을 pop 으로 하나씩 꺼내와 해당 알고리즘을 적용하여 찾는 문자열을 가진 노드의 제목과 고유번호를 출력함으로써 검색을 완료한다. 인자가 없게 되면 찾는 문자열이 없는 것이므로 해당 알고리즘을 적용할 수 없으므로 error code 600 을 출력해주어야 하며, BST 가 없게 되면 Iterative post-order 를 통한 Queue 에 노드를 저장하는 과정에서 아무 노드도 저장할 수 없기 때문에 마찬가지로 error code 600 을 출력한다.

## SELECT

SELECT 명령어는 이번 프로젝트의 핵심 기능인 사진 편집을 하는 명령어인 EDIT 을 위해 이미지의 경로를 찾고 저장하는 명령어이다. 1 번째 인자로 파일 고유 번호가 주어지기 때문에

해당 고유 번호를 pre-order traversal 을 이용하여 Database\_BST 에서 찾는다. 찾으면 해당 노드에 저장되어 있는 정보인 폴더 이름, 제목 등을 이용하여 경로를 구축하여 이미지를 해당 경로를 통해 찾아 EDIT 을 위한 저장을 한다. 파일 고유 번호로 파일을 찾아야 하기 때문에 인자가 존재하지 않거나 인자가 있더라도 고유번호가 없으면 마찬가지로 찾을 수 없기 때문에 error code 700 을 출력한다.

## EDIT

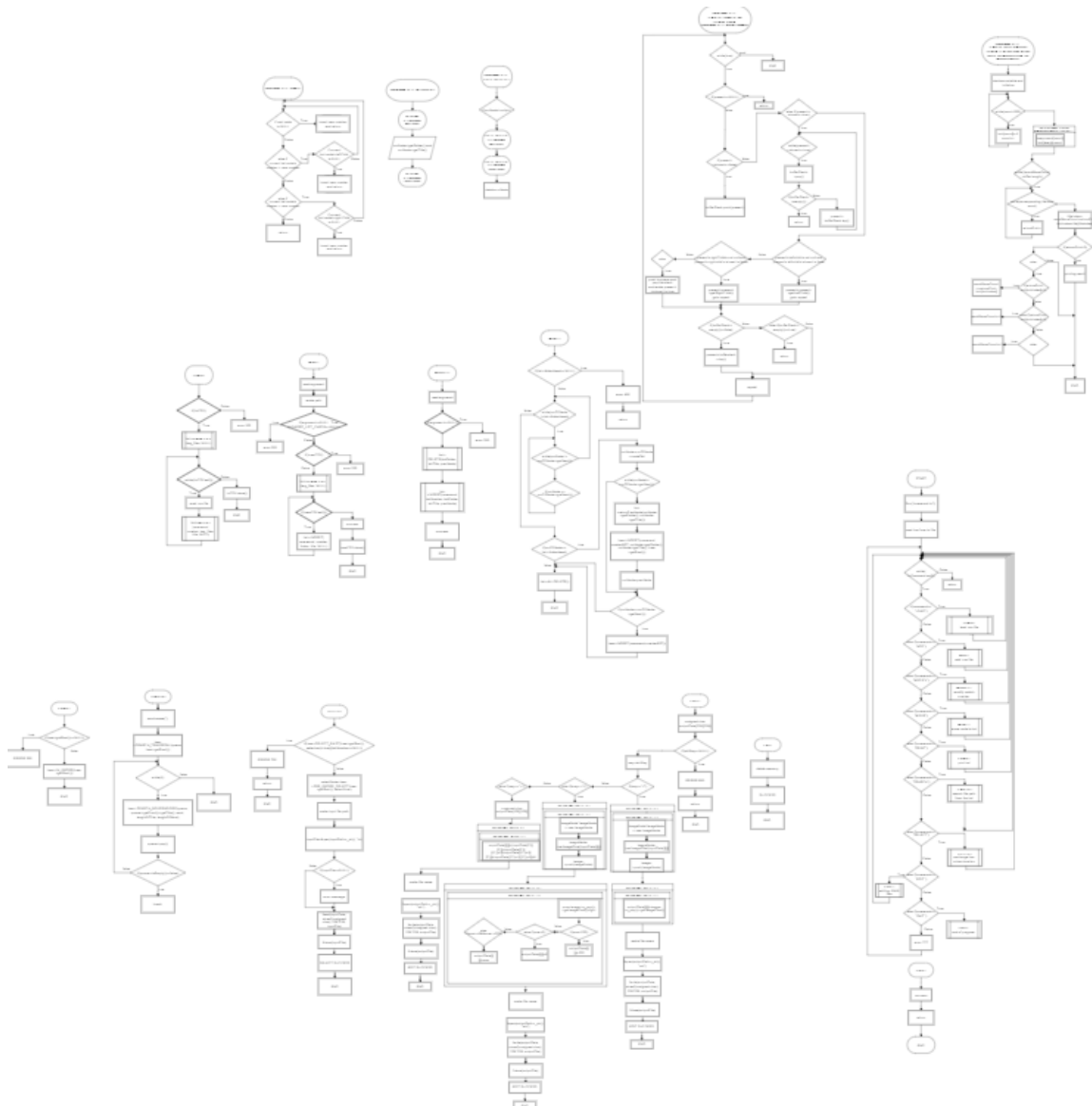
이번 프로젝트의 핵심 기능인 EDIT 명령어이다. 사진을 편집하는 방식은 총 3 가지가 있으며 각각 -f, -l, -r 로 input image 에 대해 점대칭 구현, input image 에 대해 밝기 조정, input image 에 대해 크기 조정의 편집을 수행한다. 이미지 점대칭의 경우 점대칭을 해야 하기 때문에 대각선을 기준으로 대칭적인 구조를 구현해야 하므로 임시로 픽셀을 저장할 버퍼가 필요하다. 점대칭의 경우 구현해둔 stack 구조를 이용할 것이며, 밝기 조정의 경우 구현해둔 Queue 구조를 이용할 것이다. 마지막으로 크기 축소의 경우 인접한 4 개의 픽셀에 저장되어 있는 크기값들의 평균을 내서 적용할 것이기 때문에 별도의 버퍼를 사용하지 않을 것이다. 각 기법을 적용하고 나온 output image 들은 Result 폴더에 저장되도록 경로를 찾는 logic 을 구현하여 EDIT 명령어를 수행해야 한다. 밝기 조정 옵션의 경우 2 번째 인자가 더하거나 빼기 위해 적용해 하는 값인데, 이것이 없다면 밝기 조정을 수행할 수 없으므로 error code 800 을 출력한다.

명령어들과 함께 프로젝트의 전체적인 내용을 introduction 하였다. 각 명령어에 대한 시각적인 flow 는 후에 기술할 Flowchart 에서 확인할 수 있으며, 기능 구현에 있어 핵심적인 알고리즘들은 후에 기술할 Algorithm 에 자세하게 작성하였다.

## +제출 파일 관련

1. 주어진 .RAW 파일은 img\_files 폴더 안에 filesnumbers.csv 파일과 동일한 위치에 있게 테스트하였습니다.
2. Result 폴더 안에 EDIT 명령어를 통해 얻은 편집된 이미지들을 저장하도록 구현하였습니다.
3. command.txt 와 출력 결과를 저장하는 log.txt 는 .cpp 와 .h 파일과 동일한 위치에 생성했었습니다.

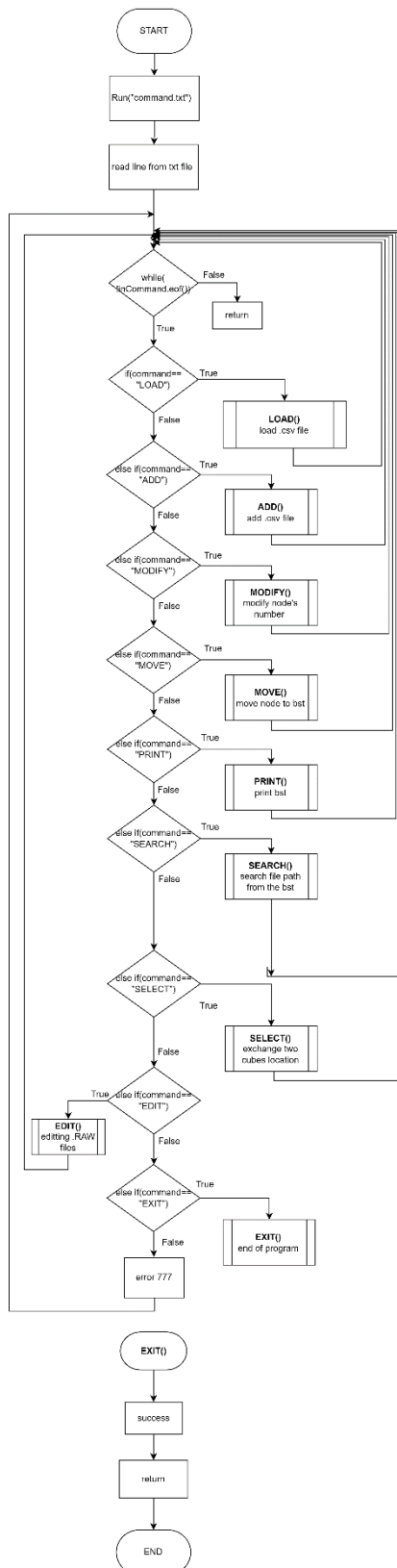
# Flowchart



(위 이미지는 draw.io 로 직접 그린 전체 Flowchart 입니다. 모든 함수를 포함해 세로로 flow 를 만들면 매우 길어지므로 서브루틴 기호를 사용하여 명령어 별로 각각 flowchart 를 작성했습니다.)  
아래는 제가 직접 작성한 draw.io flowchart 링크입니다.

<https://drive.google.com/file/d/1XsyXAt4LQ40spTbr1cBSxNfftCRLxLpc/view?usp=sharing>

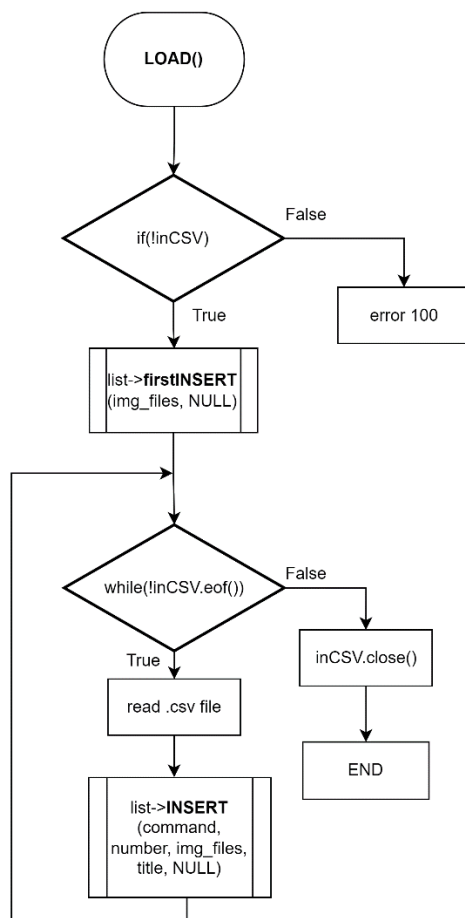
## 전체 흐름도-Run()



main.cpp 파일에서 제일 먼저 실행되는 Manager 클래스의 `Run()` 함수이다. `Run()` 함수는 txt 파일을 인자로 가지고 있기 때문에 `command.txt` 파일을 인자로 전달하여 실행한 것을

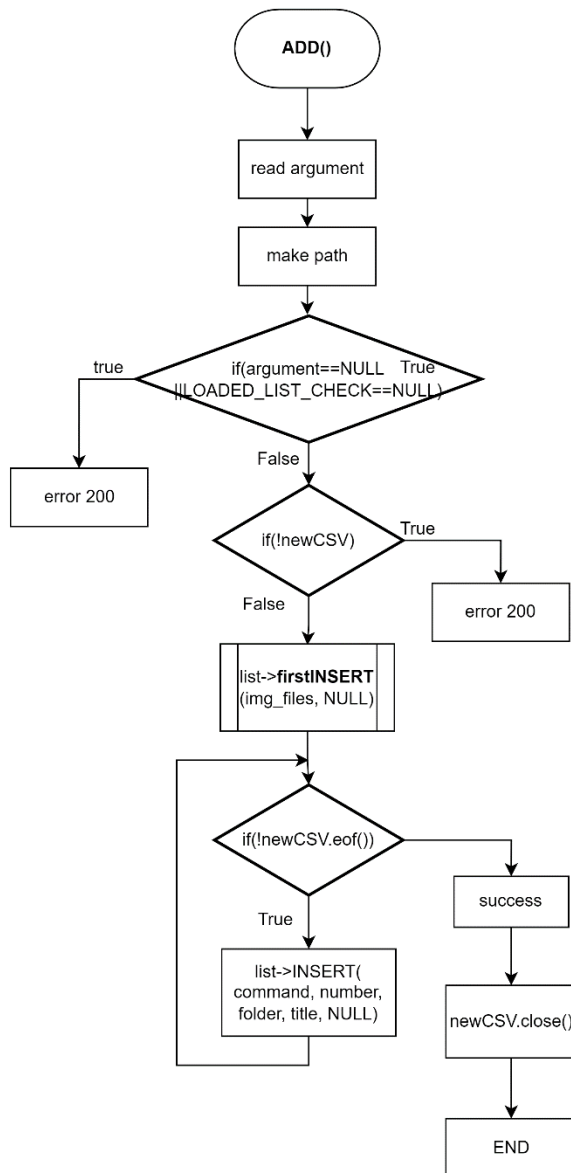
flowchart 에서 확인할 수 있다. command.txt 파일에서 getline 으로 읽어온 정보를 바탕으로 어떤 명령어가 실행될지 알 수 있다. command.txt 파일 안의 명령어는 하나만 존재하는 것이 아니므로 EXIT() 함수로 끝이 날 때까지 무한 루프문을 통해 지속적으로 명령어의 기능을 수행하게 되는 것이다. 각 명령어마다 서브루틴 기호로 그렸기 때문에 flowchart 상에서 각 명령어에 대한 함수와 연결된다. 이를 바탕으로 마지막으로 EXIT() 함수가 실행되면 종료되도록 flowchart 의 구성을 완료하였다.

## LOAD()



프로그램 실행상 제일 먼저 실행되는 명령어인 LOAD()이다. csv 파일을 읽는데, 열리지 않게 되면 해당하는 오류코드를 출력하고 함수가 종료되고, csv 파일을 있을 수 있으면 firstINSERT() 함수를 통해 파일 선행 노드(2 차원 링크드 리스트에서 세로 노드)를 삽입한다. csv 파일에서 읽은 고유번호와 제목은 csv 파일을 다 읽을 때까지 while 문을 수행하고, 읽어온 정보를 바탕으로 노드를 생성해 버퍼 개념의 자료구조로 사용한 2 차원 링크드 리스트인 Loaded\_LIST 에 INSERT() 함수를 실행해 지속적으로 삽입하는 것을 flowchart 를 통해 확인할 수 있다.

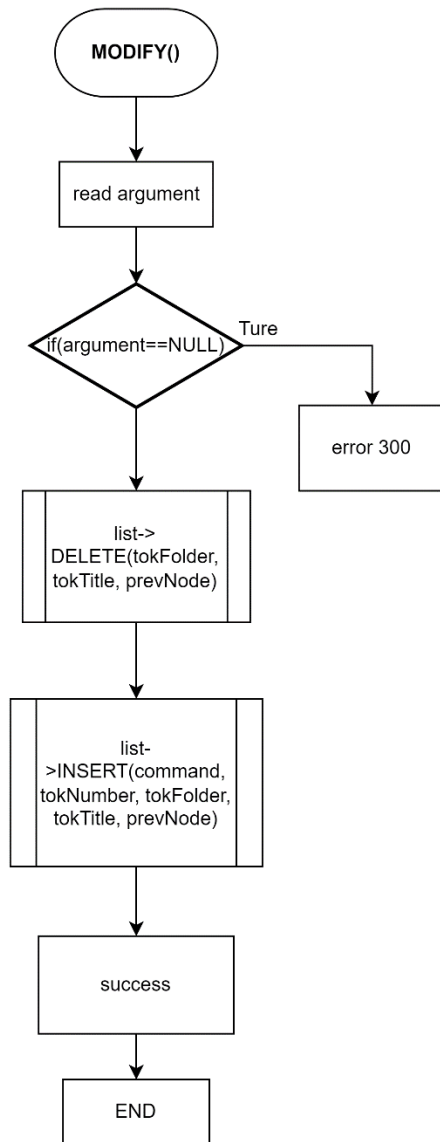
## ADD()



LOAD()로 만들어진 Loaded\_LIST 에 덧붙여지는 링크드 리스트이다. Loded\_LIST 가 존재하지 않거나 인자들이 없는 경우 error 200 을 출력하며, csv 파일이 열리지 않는 경우에도 error 200 이 출력되도록 validation 처리한 것을 확인할 수 있다. 파일이 제대로 열린다면 파일 선행 노드를 삽입 후 LOAD()와 마찬가지로 노드를 지속적으로 Loded\_LIST 에 삽입한다. 이후 과정은 LOAD()와 동일하므로 동일한 설명은 생략하였다.

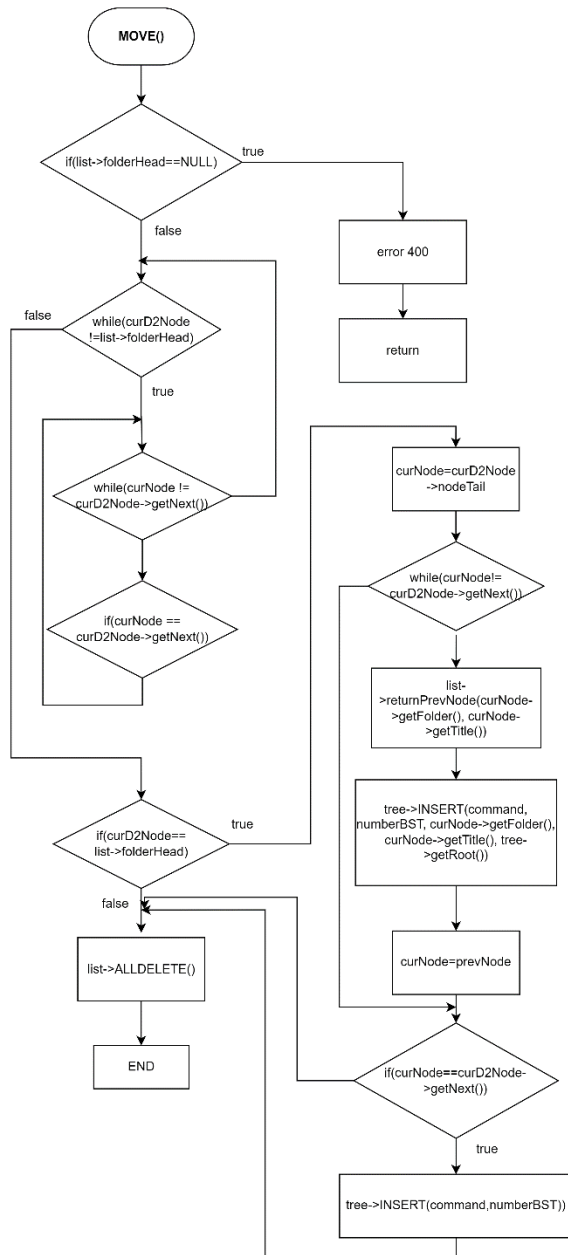


## MODIFY()



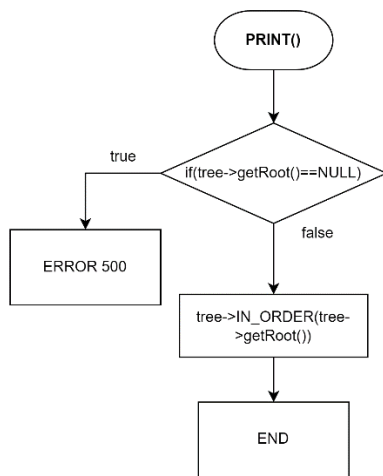
`MODIFY()` 함수는 인자들을 바탕으로 해당 노드를 찾아 삭제 후 바뀐 고유번호를 바탕으로 그 자리에 다시 삽입된다. 인자가 충분하지 않을 경우 `error 300` 을 출력하도록 validation 을 설정해주었으며, 인자가 조건을 만족할 경우 폴더 이름, 제목, 고유 번호로 삭제할 노드를 찾아 `DELETE()` 함수를 실행하여 지운다. 이어서 새로운 고유번호와 함께 폴더 이름과 제목을 같이 삽입하여 `prevNode` 를 이용해 삭제한 노드의 위치를 찾아 그 자리에 다시 삽입하였다.

## MOVE()



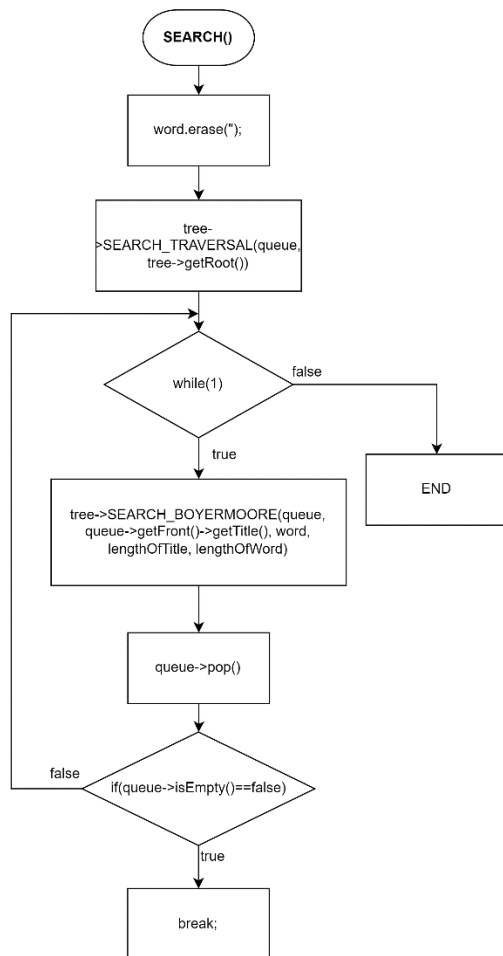
앞선 명령어들로 Loded\_LIST 에 저장되어 있는 노드들을 Database\_BST 에 삽입하는 MOVE() 함수이다. Loaded\_LIST 가 없을 경우 error 400 을 출력하고 종료되도록 validation 처리를 해주었으며, flow 가 복잡해 보이지만 마지막 노드들인 nodeTail 을 바탕으로 끝에서부터 삭제해 nodeTail 의 위치를 이동하고, nodeTail 과 nodeHead 의 위치가 같아지면 해당 가로 줄의 노드는 파일 선행 노드를 제외하고 하나가 남게 된다. 이 또한 삽입 후 지우게 되면 파일 선행 노드 외의 나머지 노드는 존재하지 않으므로 파일 선행 노드 또한 삭제해주었다. 이 과정을 Loaded\_LIST 의 모든 노드들에 적용함으로써 성공적으로 BST 에 노드들을 옮기고, 옮긴 노드들을 Loaded\_LIST 로부터 삭제하는 flow 을 확인할 수 있다.

## PRINT()



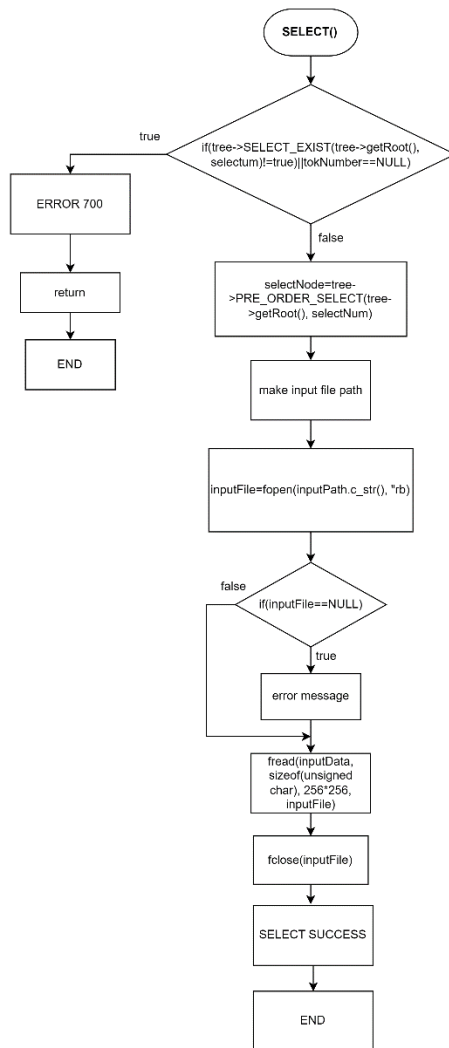
PRINT() 함수는 Database\_BST 자료구조에 저장되어 있는 모든 노드들을 IN\_ORDER() 순회로 출력한다. 그 이전에 BST 에 아무 노드도 없다면 출력한 노드가 없기 때문에 error 500 을 출력하는 것을 flowchart 를 통해 확인할 수 있다. BST 가 NULL 이 아니라면, 중위 순회 함수를 실행하고 해당 함수 안에서 재귀를 하며 노드의 정보를 출력한다.

## SEARCH()



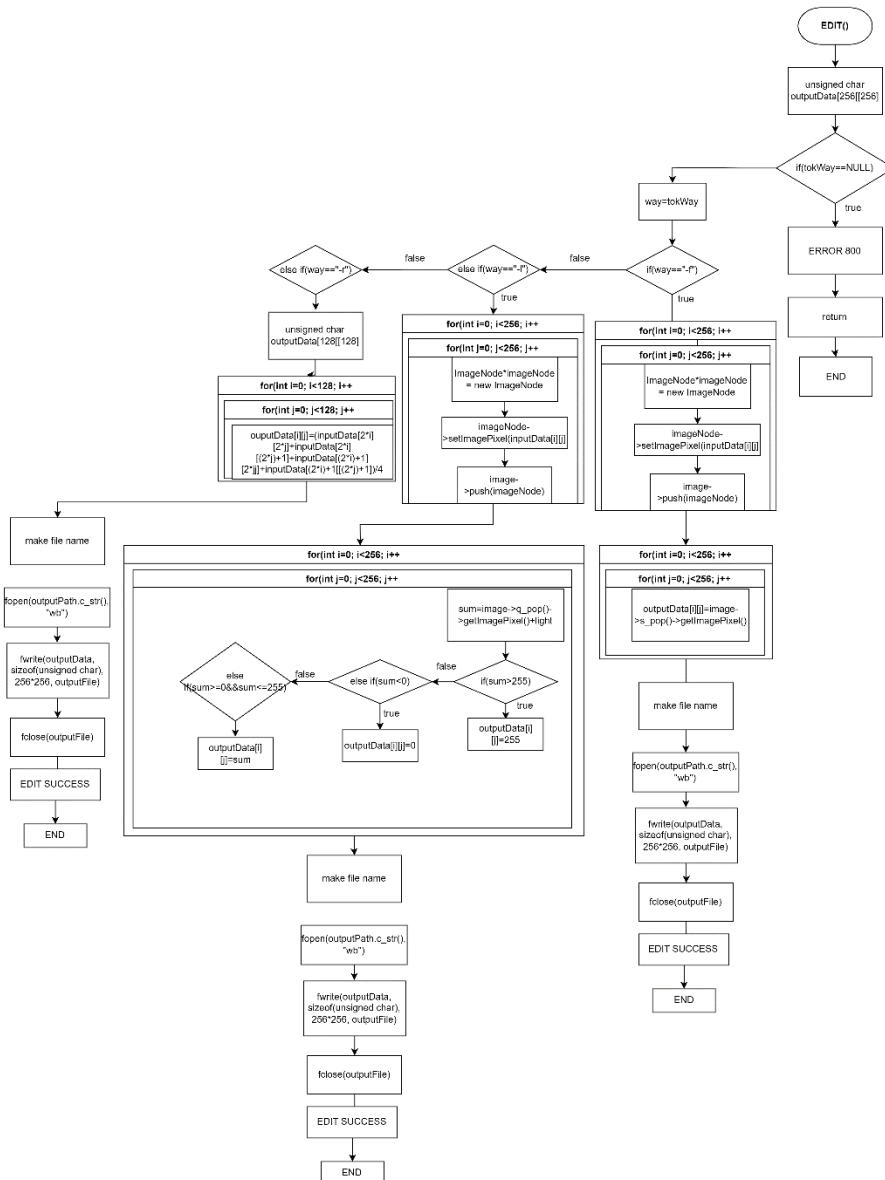
SEARCH() 함수는 BST 에 저장된 노드들에 대해 찾고자 하는 단어가 제목에 들어가 있는 노드들의 정보를 출력하는 기능을 한다. SEARCH() 함수 구현 과정에서 재귀 함수를 사용한 post-order 가 아닌, 반복문을 사용한 post-order 를 사용함으로써 Queue 자료 구조에 저장한다. Queue 자료구조의 선입선출 특성을 이용해서 Queue 에서 노드들을 하나씩 pop 함으로써 보이어-무어 알고리즘을 사용한 함수를 통해 bad char method 를 적용하여 해당 단어를 제목에 가진 노드들의 정보를 출력해주었다. Queue 가 비게 되면 더 이상 보이어 무어 함수를 적용할 수 없으므로 break 를 하여 루프를 빠져나감으로써 모든 노드들에 대해 search 가 적용되도록 flow 을 구성하였다.

## SELECT()



`SELECT()` 함수는 `EDIT()` 함수를 위한 선행 함수이다. 따라서 편집할 이미지에 대한 경로를 만들고, 가져와야 한다. 처음 함수가 실행되고 인자가 없거나 고유번호가 존재하지 않으면 함수의 기능을 실행할 수 없기 때문에 error 700 으로 validation 처리를 해주었다. error 700 이 나오지 않도록 해당 조건들을 만족하면 pre-order 를 입력 받은 고유 번호를 바탕으로 실행하여 해당 노드에 대한 정보를 가져와 노드에 해당하는 파일을 찾기 위해 경로를 구성하였다. 이어서 `fopen` 함수를 실행해 rb, 즉 읽기 방식으로 파일의 경로를 바탕으로 열고, `fread` 함수를 통해 이미지 편집에 필요한 정보들을 전달해주었다. 작업을 마쳤으므로 `fclose` 함수를 통해 안전하게 파일 입출력 함수를 다루는 것을 flow 를 통해 확인할 수 있다.

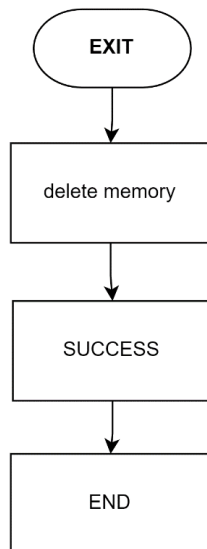
## EDIT()



EDIT() 함수는 SEARCH() 함수 실행 후 해당 이미지를 편집하는 함수이므로 출력 데이터 형인 unsigned char 를 256x256 size 로 선언해주었다. 어떤 작업을 수행할 지 인자가 전달되지 않으면 error 800 을 출력하였다. 인자가 제대로 전달된 경우 3 가지 작업 중 하나를 수행하게 되는데 점대칭 편집 동작인 -f 는 2 중 for 문으로 각 pixel 에 대한 정보를 가지고 있는 ImageNode 형 노드를 new 로 생성해주었으며, Queue 와 Stack 의 기능을 수행할 수 있는 Image 클래스를 통해 stack 구조에 순서대로 push 하고 난 후 늦게 들어온 정보가 먼저 나가게 되는 동작을 수행하고 난 뒤 output file 에 대한 확장자 설정, fopen 과 fwrite 를 통한 output file 을 경로에 저장함으로써 점대칭 동작을 수행할 수 있게 해주었다. 점 대칭에 대한 첫 번째 조건문에 해당하지 않는다면 2 번째 조건문으로 flow 가 이동한다. 만약 -1 이라면 밝기 편집 동작을 수행하게 되는데 점대칭 편집 동작과 마찬가지로 2 중 for 문으로 각 pixel 에 대한 정보를 가진

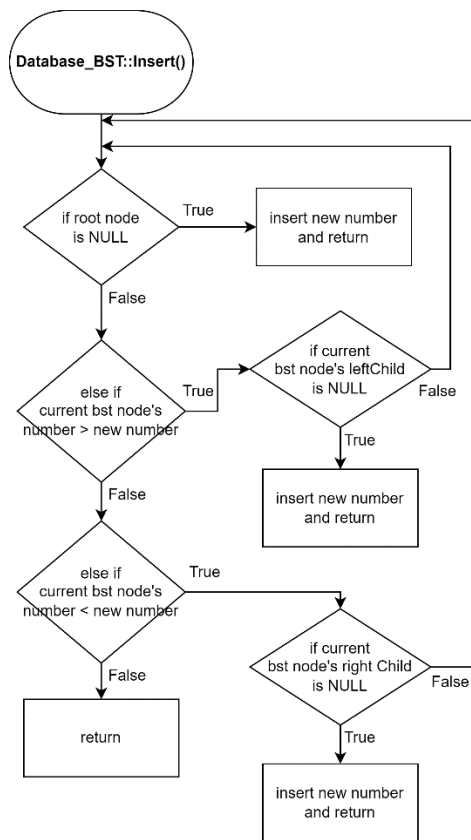
노드를 생성하여 queue 구조에 순서대로 push 하고 난 후 제일 먼저 들어온 정보가 제일 먼저 나가게 되는 동작을 수행하고 난 뒤 밝기 편집 동작을 해주어 output file 을 경로에 저장한다. 마지막 조건문인 -r 과 동일한 인자를 받을 경우 크기를 1/4 로 줄이는 이미지 크기 조정 편집 동작을 수행한다. 마찬가지로 2 중 for 문을 통해 동작을 수행하되, 앞의 두 동작들과 다르게 스택이나 큐 자료구조를 사용하지 않고 바로 적용하여 output file 을 경로에 저장함으로써 flow 를 구성하였다.

## EXIT()



자료 구조를 사용하며 allocate 했던 메모리들에 대해 메모리 누수를 방지하기 위한 deallocate 를 수행 후 프로그램을 종료하는 마지막 명령어로서 기능을 수행하는 flow 를 확인할 수 있다.

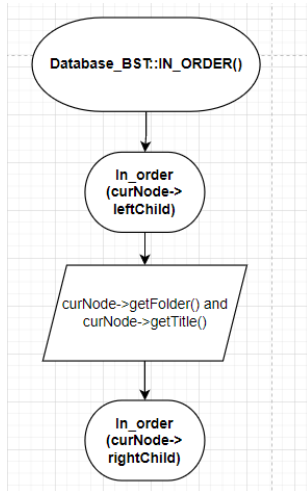
## Database\_BST::Insert()



Database\_BST class 의 Insert() 함수. 즉 Database\_BST\_Node 를 삽입하는 함수이다. 노드가 삽입될 때 벌어지는 모든 case 들을 고려해서 flow 를 구성했다. 먼저, root node 가 없으면 BST 노드가 아무것도 없다는 것이므로 root node 에 첫 노드를 저장한다. root node 가 이미 저장되어 있다면 현재 고유 번호와 크기를 비교해 bst 의 특성을 고려하여 분기를 따라 이동한다. 각 조건이 true 가 되면 노드의 크기에 따라 left child 혹은 right child 로 새롭게 삽입된다.

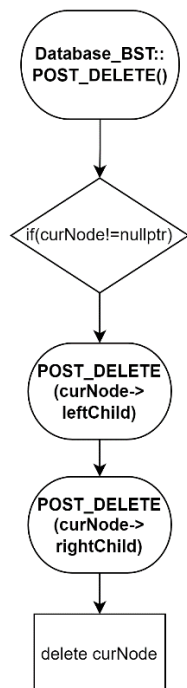


## Database\_BST::IN\_ORDER()

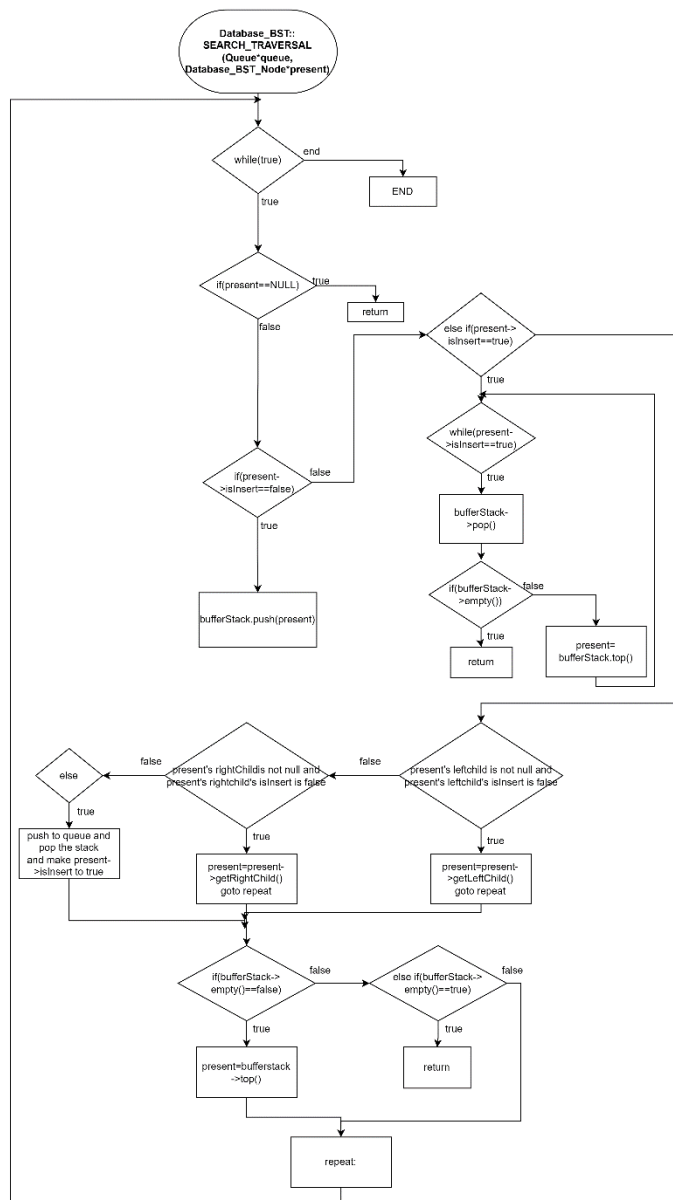


PRINT() command 의 기능을 수행하기 위해 중위 순회로 탐색하며 노드의 정보를 출력하는 IN\_ORDER()함수이다. 중위 순회이므로 재귀문(방문)-출력-재귀문(방문) 순으로 flow 를 구성한 것을 확인할 수 있다.

## Database\_BST::POST\_DELETE()

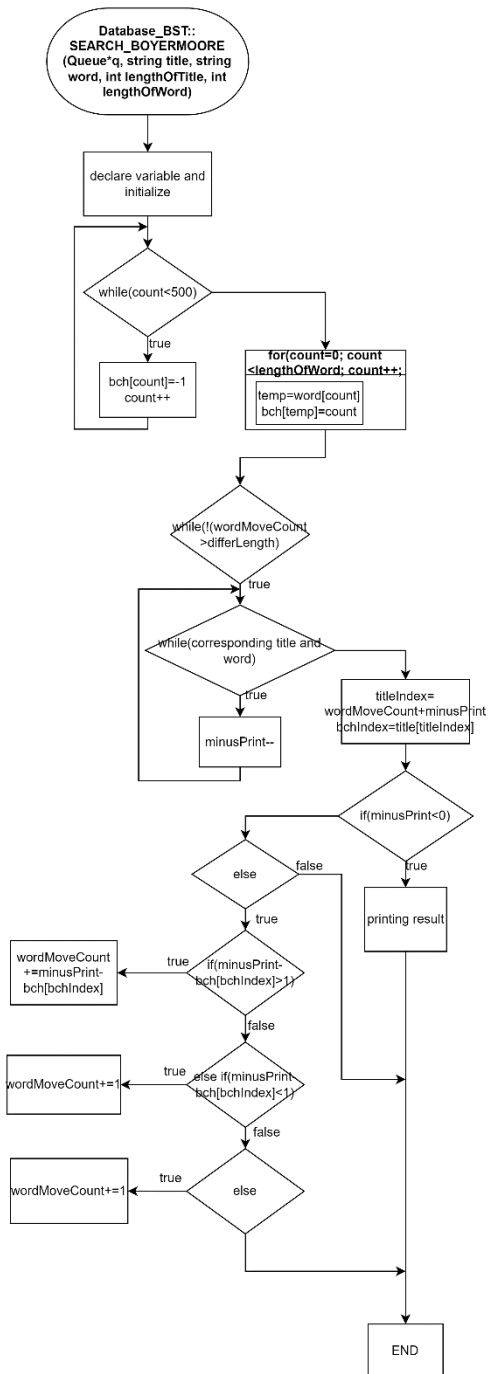


EXIT 명령어에서 메모리 할당 해제를 해주는데, BST 의 메모리 할당 해제를 해주는 Flow Chart 이다. 메모리 해제에 대한 방식은 자유이므로 recursive post-order 방법을 사용해 재귀적으로 노드를 delete 를 하도록 flow 를 구성한 것을 확인할 수 있다.



SEARCH 명령어 구현 중 iterative post-order 의 함수이다. 자세한 설명은 추후 Algorithm 에서 기재하였으므로 flow chart 에 대한 세부 설명은 생략하겠다. 전체적으로 크게 분석하면 bst 노드를 이동하면서 stack 에 차례대로 쌓게 되고 top 노드를 pop 하여 보이어-무어 알고리즘 적용을 위한 Queue 자료 구조에 저장하는 방식의 flow 이다.

## Database\_BST::SEARCH\_BOYERMOORE()



SEARCH 명령어 구현 중 찾고자 하는 문자가 노드들의 제목의 일부를 구성하고 있으면 출력하는 보이어-무어 알고리즘을 적용한 함수이다. 자세한 설명은 추후 Algorithm 에서 기재하였으므로 flow chart 에 대한 세부 설명은 생략하겠다. 전체적으로 크게 분석하면 bad char method 를 사용하여 제목의 각 index 와 맞춰보고 오른쪽 끝고 비교 제목과 틀리다면 제목의 문자를 bad char 에서 찾고, 해당 index 만큼 이동을 하는 식으로 flow 를 구성하였다.

# Algorithm

## 파일 읽기 후 명령어 및 인자들을 구분하는 Algorithm

파일로부터 getline 으로 읽어오기 때문에 string 으로 저장된다. 명령어와 인자들을 구분하기 위해 해당 string 을 조각 내서 자르는 방법을 사용했다. 그 방법은 strtok 함수를 사용하는 것인데, 이 strtok 함수는 char\* 자료형에만 사용이 가능해 string 형에 .c\_str()를 적용함으로써 char\*형으로 변환해 strtok 를 " " 공백으로 구분해서 각 변수들에 저장해 인자들을 구분하였다.

## filennumbers.csv 의 첫 번째 숫자가 잘 나오지 않는 현상을 처리하는 Algorithm

조교님께서 깃허브에 업로드 되어 있는 .csv 파일을 기준으로 채점한다고 말씀해주셨다. 이 csv 파일은 인코딩이 UTF-8 인데 이를 읽으면 제일 앞 고유 번호 앞에 이상한 문자가 같이 출력되는 현상이 발견되었다. UTF-8 에서 ANSI 로 인코딩 형식을 변경하면 해결할 수 있지만 유연한 코드를 작성하기 위해 첫 고유번호에 대해 .erase 함수를 사용해 Wxef, Wxbb, Wxbf 를 제거해주었다. 이 값들의 존재는 디버깅을 통해 인덱스 0 부터 2 까지 들어가 있는 것이 확인되어 제거해주는 코드를 작성할 수 있었다.

## LOAD()와 ADD()의 노드 삽입 Algorithm

LOAD 명령어와 ADD 명령어를 통한 노드의 삽입 과정은 동일하다. 우선 처음 명령어가 실행되면 폴더 이름을 가진 파일 선행 노드를 삽입한다. 이어서 csv 파일에 저장되어 있는 정보를 바탕으로 파일 선행 노드와 해당 노드들을 연결함으로써 2 차원 링크드 리스트 구조를 갖추게 된다. 파일 선행 노드가 가리키는 가로 노드가 정보가 담긴 노드들 중에서 nodeHead 가 되고 제일 끝에 있는 노드는 nodeTail 이 되도록 해주고, 파일 선행 노드들 또한 파일 선행 노드 중에서 가장 먼저 삽입된 파일 선행 노드를 folderHead, 제일 늦게 삽입된 파일 선행 노드를 folderTail 이라는 변수를 설정하여 사용함으로써 Loaded\_LIST 관련 작업 시에 접근하기 유용하도록 algorithm 을 구현하였다.

## MODIFY()를 통해 삭제한 노드 자리에 새로운 노드를 삽입하는 Algorithm

MODIFY 명령어는 Loaded\_LIST 에 이미 존재하는 노드에 대하여 고유 번호를 수정하는 명령어이지만 단순히 값을 변경하는 것이 아니라 해당 노드를 삭제 후 그 자리에 그대로 생성해야 한다. 수정하는 방식은 인자로 함께 주어진 폴더명과 제목을 이용하여 고유번호를 바꾸고자 하는 노드를 찾고 이 노드를 삭제 후에 고유 번호만 바꾸고 제목은 유지되는 새로운 노드를 만들어 삭제된 자리에 삽입한다.

### **노드의 개수가 300 개 이상이 된 Database\_BST 의 노드를 삭제하는 Algorithm**

기존의 BST 에서 노드를 삭제하는 case 는 상황에 따라 여러가지가 있다. 하지만 고유번호가 가장 작은 노드를 삭제한다는 조건이 있으면 delete 로직 구현이 굉장히 간단해진다. 우선 노드가 insert 될 때마다 counting 변수를 1 씩 증가시켜주었다. 노드가 300 개가 되어 counting 변수가 300 이 되어 꼭 찬 상태이고 이 상황에서 하나의 노드를 더 추가하면 BST 에서 고유 번호가 가장 작은 노드를 삭제 후 counting-1 후에 새로운 노드를 삽입하여 counting+1 로 300 이하를 유지한다. 고유 번호가 가장 작은 노드를 삭제하는 case 는 2 가지가 있다. root 노드 기준으로 제일 leftChild 가 child node 를 가지고 있지 않은 경우 해당 노드만 제거하면 된다. 하지만 root 노드 기준으로 제일 leftChild 가 rightChild 를 갖는 경우가 존재할 수 있다. 이 경우에서 그냥 delete 를 해버리면 제일 leftChild 의 rightChild 는 그대로 잃어버리게 되기 때문에 제일 leftChild 의 parent node 가 제일 leftChild 의 rightChild 를 가리키게 하고 난 후에 제일 leftChild 였던 노드를 삭제함으로써 구현하는 algorithm 을 작성하였다.

### **PRINT() 함수에서 중위 순회로 노드의 정보들을 출력하는 Algorithm**

PRINT 함수는 Database\_BST 에 저장되어 있는 모든 노드들을 출력해주어야 한다. 출력 방식은 IN\_ORDER 로 중위 순회이다. 중위 순회 방식은 방문-출력-방문이기 때문에 재귀함수-출력-재귀함수 형태로 알고리즘을 구성하면 된다.

### **SEARCH() 함수에서 iterative post-order 로 Queue 에 노드를 저장하는 Algorithm**

기존의 재귀적인 순회 방식과는 다르게 반복문을 사용해서 구현한 iterative post-order 의 algorithm 설명이다. 기존의 재귀적인 순회 방식은 bst 의 특성상 최대 2 개의 자식 노드를 가지고 있고, 재귀적으로 함수를 호출함으로써 중복없이 특정 값 출력 등의 작업이 가능했었다. 하지만 반복문을 사용하여 post-order 로 구현하게 되면 노드의 수가 많아 질수록 중복되는 노드 반복적으로 순회하면서 중복적으로 스택에 들어간 노드들이 다시 Queue 에 push 되면 안 되므로 boolean 변수로 Queue 에 push 되었는지 안 되었는지를 판단해서 중복에 대한 처리를 한다. 이를 위해 사용하는 것이 stack 자료구조와 중복을 막기 위한 boolean 변수인 isInsert 를 선언하여 사용하였다. iterative post-order 를 통해 우리가 기대하는 것은 post-order 방식을 유지하되, 거친 노드들에 대해 중복 없이 Queue 에 push 해주는 것이다. Queue 에 저장하는 이유는 후에 기술할 보이어-무어 알고리즘을 사용한 문자열 검색을 통해 최종적으로 SEARCH command 의 기능을 수행할 수 있기 때문이다. 내가 작성한 코드를 바탕으로 설명을 해보자면 먼저 queue 에 모든 값을 넣는 것이 목표이므로 무한 루프문인 while(true)를 통해 block 을 시작한다. 내가 만든 iterative post-order 는 SEARCH\_TRAVERSAL 이라는 함수명과 2 개의 매개변수인 Queue\* queue 와 Database\_BST\_Node\*present 를 받는다. 첫 번째 매개변수는 Manager.cpp 에서 이 함수에게 순회하면서 queue 에 노드를 저장하기 위해 넣어주었고, 두 번째 매개변수는 Manager.cpp 에서

루트 노드를 넘겨주어 순회를 하기 위함이다. 그래서 while 문 안에 들어오게 되면 처음에 넘겨준 노드가 NULL 이라면 return 하게 한 이유는 즉, BST 가 없는 경우에서 발생하게 된다. BST 가 없다는 것은 root 노드 또한 없다는 뜻이므로 밖에서 root 를 넘겨준 것이다. 그 다음에 queue 에 저장하기 전 중복을 막아주기 위한 버퍼 역할을 수행하는 stack 인 bufferStack 이 push 를 통해 스택에 쌓이도록 구성하였다. 그리고 while 문을 돌면서 queue 에 들어가게 되면 isInsert 가 true 값을 갖게 할 것인데, 이를 통해 isInsert 에 대한 true false 여부로 true 인 경우(이미 queue 에 들어간 적이 있는 노드) bufferStack 에서 pop 을 하여 버린다. while 문으로 이 또한 처리하는 것이기 때문에 이미 다 넣어버린 bufferStack 이 빈 경우가 있게 되면 return 을 하여 함수를 종료한다. 그 후 bst 가 순회를 하면서 왼쪽 자식 노드, 오른쪽 자식 노드로 조건에 따라 present 노드가 바뀌게 된다. 노드가 바뀌었으므로 goto 문을 사용해 while 문을 처음부터 돌 수 있도록 코드를 작성해주었다. else 로 분기를 해주었기 때문에 위의 조건에 어긋나게 되면 순회에서의 실행 위치에 해당하기 때문에 queue 에 노드를 push 해준다. push 를 해주었으므로 bufferStack 에서는 pop 해버림과 동시에 현재 노드의 isInsert 변수를 true 로 바꾸어준다. (queue 에 들어갔으므로) 마지막으로 bufferStack 이 비어있지 않으면 현재 노드를 bufferStack 의 top 노드를 가지게하고, bufferStack 이 비어 있다면 더 이상 queue 에 저장할 노드가 없다는 의미이므로 return 을 통해 함수를 종료한다. 이 과정을 루프문을 통해 반복하면서 반복적으로 post-order 를 수행하되, queue 자료구조에 모든 노드가 정상적으로 저장될 수 있게 구현해주었다.

## SEARCH() 함수에서 Queue 를 활용한 boyer-moore 문자열 검색 Algorithm

앞서 iterative post-order 를 통해 Queue 에 저장한 BST 의 노드들을 바탕으로 각 노드의 제목의 인덱스와 특정 단어의 인덱스를 비교하여 해당 단어가 제목에 들어있는지 판단하고, 해당 단어가 제목에 있으면 해당 노드의 정보를 출력해주는 boyer-moore algorithm 을 구현한 동작을 설명하겠다. 우선, boyer-moore algorithm 은 Good suffix 방식과 bad character 방식이 존재한다. 둘 다 모두 문자열과 찾고자 하는 단어를 검출한다는 목표는 동일하지만 찾는 과정이 다르다. 두 방식에 대한 설명 후 내가 선택한 방식과, 선택한 이유에 대해서 설명하겠다. 보이어-무어 방식은 단어를 찾고자 하는 제목의 오른쪽부터 비교한다. 따라서 good suffix 방식과 bad character 방식 모두 오른쪽부터 비교한다는 점이 base 이다. bad character 부터 탐색 방식에 대해 설명하자면, 제목과 단어의 인덱스를 맞춰두고, 단어 인덱스의 맨 오른쪽부터 왼쪽 방향으로 제목의 인덱스와 일치하지 않는 부분을 찾는다. 일치하지 않는다면 해당 인덱스의 문자가 bad character 이다. 이 제목의 bad character 가 단어에 존재하는지 맨 오른쪽부터 왼쪽 방향으로 찾아나간다. 만약에 bad character 가 단어에 존재한다면 해당 단어 index 가(제목의 bad character 와 동일) 제목의 bad character 가 저장된 index 로 위치를 이동한다. 이 과정을 반복하여 제목의 끝 index 까지 단어가 bad character 에 맞춰 위치를 이동함으로써 하나의 노드의 제목에 대한 boyer-moore algorithm 의 적용이 끝나게 된다. 이어서 good suffix 방식은 제목과 단어의 인덱스를 초기에는

맞춰둔다. 맞춘 상태에서 오른쪽에서 왼쪽 방향으로 제목 index 에 저장되어 있는 문자와 단어 index 에 저장되어 있는 문자가 일치하도록 탐색한다. 일치하게 되는 부분은 good suffix 가 된다. 즉, 일치하는 가장 왼쪽 index 의 왼쪽 index 는 불일치하게 되는 것이다. 이어서 불일치했던 부분에서 good suffix 와 동일한 부분을 찾아나간다. 찾은 부분이 있으면 그 찾은 부분이 good suffix 와 일치하도록 오른쪽으로 단어를 이동시킨다. 이 과정을 반복하면서 최종적으로 온전한 단어가 해당 노드의 제목에 존재하는지 판단할 수 있다. 이어서 두 방법을 비교했을 때 나쁜 부분을 기준으로 처리하는 쪽이 거르는 인덱스가 더 많아져 구현 시에 생각할 요소가 착한 부분에 비해 줄어든 것 같다는 생각이 들어 bad character 를 선택하였다. 이를 구현하는데 있어 가장 중요한 요소는 제목과 단어의 인덱스 부분이므로, 이를 위해 SEARCH\_BOYERMOORE() 함수를 실행하는 부분에서 매개변수로 큐, 제목, 단어, 제목의 길이, 단어의 길이를 받아주었다. 그리고 단어를 제목의 끝까지 비교해야 하므로 이를 위한 제목의 길이와 단어의 길이를 뺀 값을 가지는 differLength 변수를 선언하였다. 이 변수는 앞으로 얼마나 이동할 수 있는 지에 대한 정보가 되어줄 것이다. 이어서 bad character 에 대한 array 의 모든 index 에 저장된 값들을 -1 로 초기화해준다. 이는 뒤에서 출력을 위한 변수와 비교하는 과정에 사용할 것이고, while 문을 통해 앞서 설명한 differLength 변수를 바탕으로 이동할 수 있는 최대 길이와 단어가 움직인 인덱스의 수를 counting 하는 wordMoveCount 가 differLength 보다 작으면 while 문을 반복적으로 실행이 가능하게 하도록 함으로써 제목의 끝까지 단어를 비교할 수 있게 로직을 구성해주었다. 이어서 가장 중요한 로직인 minusPrint 변수의 감산 과정이다. minusPrint 변수는 제목에서 문자를 찾은 노드의 정보를 출력하기 위해 사용할 것인데 이 변수의 감산이 되는 조건은 while 문을 사용하여 bad character index 를 지나가는 수만큼 감산을 해준다. 따라서 감산 조건에 의해 0 미만으로 감소하게 된 순간이 제목에서 해당 단어를 찾은 것이므로 해당 제목을 가지고 있는 노드의 정보를 출력함으로써 구현하였다.

## SELECT() 함수에서 경로를 만드는 Algorithm

SELECT() 함수에서 읽을 파일의 경로를 만들기 위해서는 고유 번호를 바탕으로 찾은 노드의 폴더명과 제목이다. inputPath 라는 string 형 변수를 이용해 append 함수를 이용하여 문자열을 붙여주었다. 첫 번째로 선택한 노드의 폴더이름, 두 번째로 경로 구분을 위한 "/", 세 번째로 선택한 노드의 제목, 네 번째로 확장자인 ".RAW"를 append 함수를 이용하여 경로를 생성해주었다.

## Database\_BST 의 동적 할당 해제 Algorithm

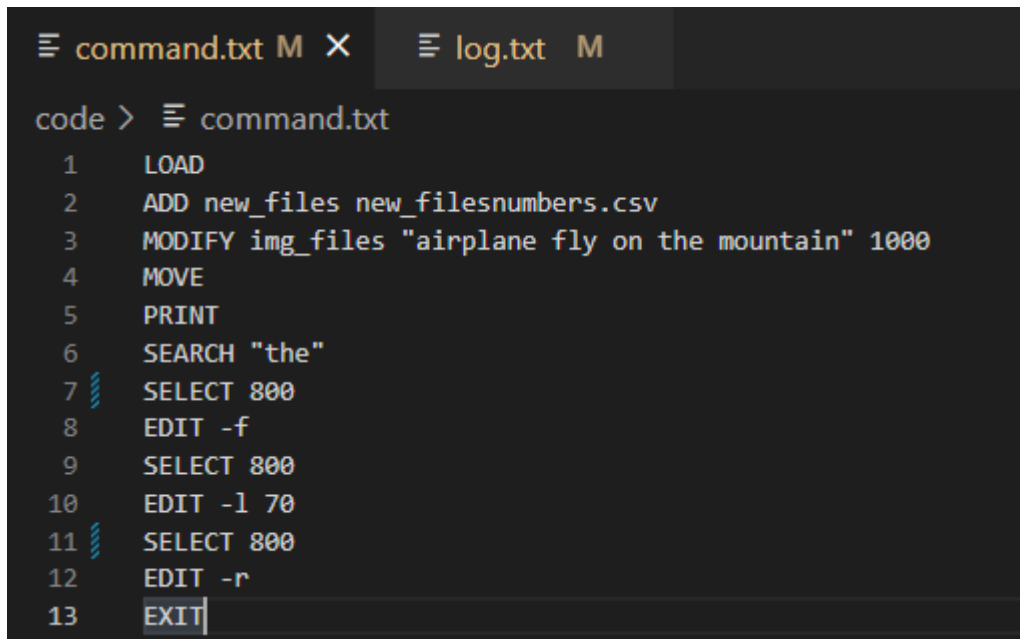
Database\_BST 의 경우 노드를 new 로 메모리를 할당하였기 때문에 사용이 끝나면 delete 를 해줌으로써 메모리 누수를 방지해야 한다. 동적 할당 해제 방법은 순회를 사용하면 된다. 모든

노드를 한 번씩 방문해서 delete 를 해야 하므로 post, in, pre 어떠한 순회를 사용해도 상관없이 재귀적으로 호출하여 delete 를 하게 되면 모든 노드의 동적 할당을 해제할 수 있다.

## Result Screen

작성에 앞서 Result Screen 들은 과제 spec 에 나와있는 출력 포맷을 따랐습니다.

### command.txt 의 test case



```
code > command.txt
1   LOAD
2   ADD new_files new_filesnumbers.csv
3   MODIFY img_files "airplane fly on the mountain" 1000
4   MOVE
5   PRINT
6   SEARCH "the"
7   SELECT 800
8   EDIT -f
9   SELECT 800
10  EDIT -l 70
11  SELECT 800
12  EDIT -r
13  EXIT
```

log.txt 에 출력된 결과를 얻기 위해 작성한 command.txt 의 test case 는 위와 같다.



## LOAD Result Screen

```
1  =====LOAD=====
2  there are lots of people in the park/100
3  the man is gorgeous/111
4  the woman is smiling/200
5  the man takes a picture/222
6  three peppers are big/300
7  the building is like a pyramid/333
8  river is under the bridge/400
9  The house has windows and doors/444
10 the boat sails a big river /500
11 The celebrity posed for the picture/555
12 there are lots of ariplane/600
13 The woman is looking at the man /666
14 lena is famous person/700
15 the woman is wearing a hat/777
16 there are cars and people/800
17 airplane fly on the mountain/888
18 there are numbers and chinese characters/900
19 fence is near trees/999
20 =====
```

위는 command.txt 의 line 1 을 실행한 LOAD 의 Result Screen 이다.

현재 프로젝트 코드와 동일한 level 에 있는 img\_files/filesnumbers.csv 경로를 통해 성공적으로 filesnumber.csv 파일 내부의 정보들을 노드로 만들어 Loaded\_LIST 2 차원 링크드 리스트 구조에 잘 삽입되어 filesnumbers.csv 의 19 개의 정보들과 동일하게 출력된 것을 확인할 수 있다. 앞서 Algorithm 에서 설명한, 인코딩 방식을 바꾸지 않고도 csv 파일의 첫 번째 문자가 깨지지 않도록 검증까지 함께 하였다.

## ADD Result Screen

```
22  =====ADD=====
23  SUCCESS
24  =====
```

위는 command.txt 의 line 2 를 실행한 ADD 의 Result Screen 이다.

LOAD 로 이미 Loaded\_LIST 에 노드가 존재하기 때문에 ADD 명령어가 성공적으로 실행된 것을 확인할 수 있다. command.txt 에 의하면 new\_files 폴더 안의 new\_filesnumber.csv 파일 내부의 있는 정보들을 바탕으로 노드를 만들어 Loaded\_LIST 에 삽입했을 것이다. 출력 포맷을 따르기 위해 노드의 정보를 직접 출력하지는 않았으나 MOVE 명령어 후에 PRINT 를 통한 확인으로 자연스럽게 검증될 것이다.

## MODIFY Result Screen

```
26  =====MODIFY=====
27  SUCCESS
28  =====
```

위는 command.txt 의 line 3 을 실행한 MODIFY 의 Result Screen 이다.

인자로 전달한 img\_files 라는 폴더명과 제목을 통해 해당 노드를 잘 찾은 상황에서 인자로 전달한 고유 번호가 Loaded\_LIST 안에 존재하지 않기 때문에 기존의 노드 삭제 및 그 자리에 새로운 노드 삽입 후 출력되는 SUCCESS 문구가 확인되었으나 직접 확인을 위해서는 PRINT 명령어를 통한 확인으로 MODIFY 기능에 대한 검증이 완료될 것이다..

## MOVE Result Screen

```
30  =====MOVE=====
31  SUCCESS
32  =====
```

위는 command.txt 의 line 4 을 실행한 MOVE 의 Result Screen 이다.

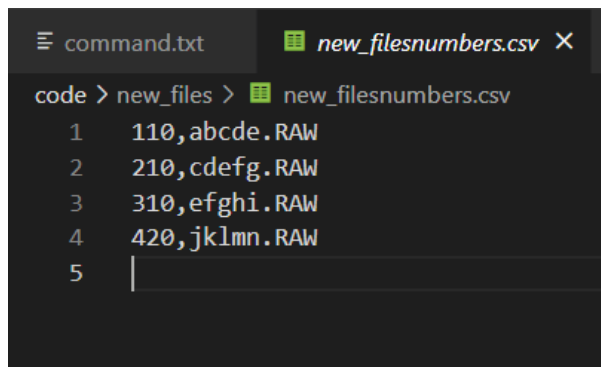
Loaded\_LIST 에 존재하는 모든 노드들을 Database\_BST 라는 bst 자료 구조에 저장 후 저장된 노드의 경우 Loaded\_LIST 에서 전부 삭제하면 SUCCESS 가 출력되도록 구현한 상태에서 SUCCESS 가 출력된 것을 확인할 수 있다. 제대로 BST 에 insert 되었는지 확인하기 위해 PRINT 명령어의 결과로 MOVE 에 대한 검증이 가능할 것이다.

## PRINT Result Screen

```
34  =====PRINT=====
35  img_files / "there are lots of people in the park" / 100
36  new_files / "abcde" / 110
37  img_files / "the man is gorgeous" / 111
38  img_files / "the woman is smiling" / 200
39  new_files / "cdefg" / 210
40  img_files / "the man takes a picture" / 222
41  img_files / "three peppers are big" / 300
42  new_files / "efghi" / 310
43  img_files / "the building is like a pyramid" / 333
44  img_files / "river is under the bridge" / 400
45  new_files / "jklmn" / 420
46  img_files / "The house has windows and doors" / 444
47  img_files / "the boat sails a big river " / 500
48  img_files / "The celebrity posed for the picture" / 555
49  img_files / "there are lots of ariplane" / 600
50  img_files / "The woman is looking at the man " / 666
51  img_files / "lena is famous person" / 700
52  img_files / "the woman is wearing a hat" / 777
53  img_files / "there are cars and people" / 800
54  img_files / "there are numbers and chinese characters" / 900
55  img_files / "fence is near trees" / 999
56  img_files / "airplane fly on the mountain" / 1000
57  =====
```

위는 command.txt 의 line 5 을 실행한 PRINT 의 Result Screen 이다.

In-order 방식 즉, 트리의 중위 순회 방식으로 탐색하여 정보를 출력하기 때문에 고유 번호의 크기 순대로 정상적으로 출력된 것을 확인할 수 있다. 또한 이를 통해 앞서 SUCCESS 출력문만 보고는 정확한 검증이 힘든 명령어인 ADD, MODIFY, MOVE 에 대해서 함께 검증을 시작하도록 하겠다. 우선 ADD 의 인자로 전달해준 csv 파일의 내부 정보들은 아래와 같다.



```
command.txt  new_filesnumbers.csv X
code > new_files > new_filesnumbers.csv
1 110,abcde.RAW
2 210,cdefg.RAW
3 310,efghi.RAW
4 420,jklmn.RAW
5 |
```

ADD 로 추가한 노드들 또한 정상적으로 Database\_BST 에 추가된 것을 보아 Loaded\_LIST 에 이미 정상적으로 insert 가 완료되었다는 것이므로 ADD 명령어에 대한 기능 검증을 완료하였다.

MODIFY 명령어는 위의 command.txt 를 통해서 확인할 수 있다시피 아래와 같다.

MODIFY img\_files "airplane fly on the mountain" 1000

기존에 고유번호가 888 번이었던 해당 제목을 가진 노드의 고유번호가 1000 으로 수정되어 56 번째 줄에서 정상적으로 출력된 것을 통해 기존에 고유 번호 888 을 가진 img\_files 의

"airplane fly on the mountain"는 정상적으로 삭제되었으므로 출력이 되지 않았고, 새로 삭제된 자리에 추가한 고유 번호 1000 을 가진 img\_files 의 "airplane fly on the mountain"가 정상적으로 출력되었으므로 MODIFY 에 대한 검증 또한 완료되었다.

마지막으로, 마찬가지로 SUCCESS 출력만 되었던 MOVE 에 대한 검증의 경우, 트리의 중위 순회 방식을 써서 출력한 PRINT 명령어가 정상적으로 수행되어 반영된 것을 통해 같이 검증이 가능하다.

## SEARCH Result Screen

```
59  =====SEARCH=====
60  "there are lots of people in the park" / 100
61  "the man is gorgeous" / 111
62  "the woman is smiling" / 200
63  "the man takes a picture" / 222
64  "the building is like a pyramid" / 333
65  "river is under the bridge" / 400
66  "the boat sails a big river " / 500
67  "The celebrity posed for the picture" / 555
68  "there are lots of ariplane" / 600
69  "The woman is looking at the man " / 666
70  "the woman is wearing a hat" / 777
71  "there are cars and people" / 800
72  "there are numbers and chinese characters" / 900
73  "airplane fly on the mountain" / 1000
74  =====
```

위는 command.txt 의 line 6 을 실행한 SEARCH 의 Result Screen 이다.

command.txt 에 의하면 인자로 "the"를 전달해줬으므로 아래 이미지와 같이 BST 에 현재 "the"라는 단어가 포함된 14 개의 노드들에 대한 정보가 성공적으로 출력된 것을 확인할 수 있다.

```
34  =====PRINT=====
35  img_files / "there are lots of people in the park" / 100
36  new_files / "abcde" / 110
37  img_files / "the man is gorgeous" / 111
38  img_files / "the woman is smiling" / 200
39  new_files / "cdefg" / 210
40  img_files / "the man takes a picture" / 222
41  img_files / "three peppers are big" / 300
42  new_files / "efghi" / 310
43  img_files / "the building is like a pyramid" / 333
44  img_files / "river is under the bridge" / 400
45  new_files / "jklmn" / 420
46  img_files / "The house has windows and doors" / 444
47  img_files / "the boat sails a big river " / 500
48  img_files / "The celebrity posed for the picture" / 555
49  img_files / "there are lots of ariplane" / 600
50  img_files / "The woman is looking at the man " / 666
51  img_files / "lena is famous person" / 700
52  img_files / "the woman is wearing a hat" / 777
53  img_files / "there are cars and people" / 800
54  img_files / "there are numbers and chinese characters" / 900
55  img_files / "fence is near trees" / 999
56  img_files / "airplane fly on the mountain" / 1000
57  =====
```

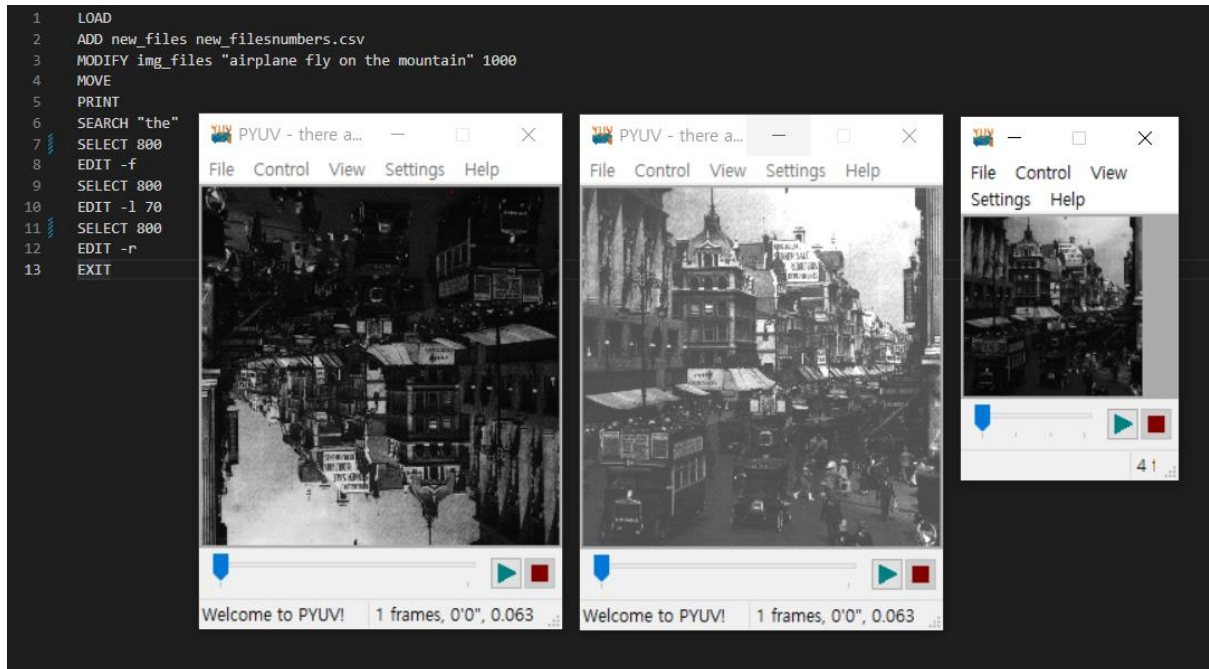
## SELECT Result Screen, EDIT Result Screen

```
76 =====SELECT=====
77 SUCCESS
78 =====
79
80 =====EDIT=====
81 SUCCESS
82 =====
83
84 =====SELECT=====
85 SUCCESS
86 =====
87
88 =====EDIT=====
89 SUCCESS
90 =====
91
92 =====SELECT=====
93 SUCCESS
94 =====
95
96 =====EDIT=====
97 SUCCESS
98 =====
```

위는 command.txt 의 line 7, 9, 11 을 실행한 SELECT 와 command.txt 의 line 8, 10, 12 를 실행한 EDIT 의 결과가 동시에 출력된 Result Screen 이다. SELECT 와 EDIT 은 같이 실행되어야 하기 때문에 일부러 교차하여 command.txt 또한 작성하였다.

```
7  SELECT 800
8  EDIT -f
9  SELECT 800
10 EDIT -l 70
11 SELECT 800
12 EDIT -r
```

SELECT 명령어를 3 개 실행하는데 다 같은 고유 번호를 SELECT 한 이유는 같은 이미지여야 3 개의 EDIT 결과로 나온 이미지들이 정상적으로 편집되었는지 시각적으로 쉽게 판단이 가능하기 때문이다. 1 번째 SELECT 하여 점대칭 수행(-f), 2 번째 SELECT 하여 이미지 밝기 조정(-l), 3 번째 SELECT 하여 이미지의 크기 조정 수행(-r) 후에 조교님들께서 제공해주신 PYUV 프로그램을 통해 확인한 이미지 편집 결과는 아래 이미지에서 확인할 수 있다. 순서대로 -f, -l, -r 의 결과이다.



1 번째 점대칭 -f 의 경우 상하좌우가 정상적으로 반전된 것을 확인하였으므로 EDIT -f 에 대한 검증이 완료되었다. 2 번째 점대칭 -1 의 경우 밝기를 +70 만큼하여 더 밝게 보이는 것을 양 옆 같은 이미지의 명암과 비교하면 시각적으로 확연히 다르다는 것을 확인할 수 있으므로 검증이 가능하다. 마지막으로 3 번째로 이미지의 크기를 1/4 로 축소하는 -r 의 경우 unsigned char outputData 의 크기를 [128][128]로 하여 작업을 하고, PYUV 프로그램을 통해서 출력 크기를 128x128 로 해주었을 때 잘리지 않고 정상적으로 딱 맞게 출력된 것을 통해 EDIT -r 에 대한 검증을 끝으로 ADD 부터 EDIT 까지의 기능을 구현 완료 및 출력 결과를 통한 검증으로 성공적으로 구현되었다는 것을 확인할 수 있다.

## EXIT Result Screen

```
100  =====EXIT=====
101  SUCCESS
102  =====
```

위는 command.txt 의 line 13 을 실행한 LOAD 의 Result Screen 이다. 프로그램 상의 모든 메모리를 해제해야 하므로, 지금까지 해제하지 않았던 유일한 자료구조인 Database\_BST 의 모든 노드들에 대한 메모리를 delete 로 해제하면 된다. 해제 과정에서 오류가 없고 다 delete 되었으면 SUCCESS 문구를 출력하도록 함으로써 EXIT 를 끝으로 모든 기능에 대한 검증이 완료되었다.

## Consideration

이번 DS 1 차 프로젝트를 통해 stack 과 queue, 2D linked list, bst 등의 자료 구조를 직접 구현하고 이를 서로 연결함으로써 사진 편집 프로그램을 과제 spec 에 만족하도록 성공적으로 모든 명령어를 구현 완료 및 예외 처리를 하였다. 하지만 모든 기능을 구현 완료하기까지 간단한 오류부터 학부생 수준인 내 수준에서는 파악하기 매우 어려운 시행착오 또한 많이 겪었다. 고찰은 객프와 데구설 과목의 다른 환경에 의한 차이점과 파악하기 매우 어려웠던, 기억에 남는 에러 사항 순으로 작성하겠다. 알고리즘에 대한 고찰과 해결하기 어려웠던 에러는 후에 기술하고, 2-2 데이터구조설계 과목과 2-1 객체지향프로그래밍설계 과목을 수행하면서 느꼈던 가장 큰 변화점들을 정리하자면 아래와 같다.

1. 운영체제의 변경 (window -> ubuntu linux 18.04)
2. 기존과는 다른 툴, 다른 컴파일러, 다른 디버깅 방법, 다른 빌드 방법
3. 같은 오류 상황에 대한 다른 오류 메시지 출력

첫 번째로 운영체제의 변경이었다. 현재 나는 window 운영체제 환경에서만 작업을 해왔었다. 하지만 리눅스 환경을 사용해야 하기 때문에 이를 window 운영체제 위에서 실행하기 위해 조교님들이 주신 강의자료를 따라 vmware 로 ubuntu 18.04 가상 환경을 구축하였다. 처음 vmware 에 로그인하고 우분투 환경을 접했을 때, 2-1 때의 객체지향프로그래밍설계 과목에서의 구축 환경과는 GUI 또한 많이 차이가 났기 때문에 우분투 리눅스 18.04 버전 환경에 익숙해지는데 많은 시간이 필요할 것 같다는 생각이 들었다.

두 번째와 세 번째는 기존과는 다른 툴, 컴파일러, 디버깅, 빌드, 서로 다른 오류 메시지 출력이다. 객프 과목에서는 윈도우용 visual studio 2022 를 설치해서 주차별 과제 및 프로젝트를 수행했다. 또한 vs 가 default 로 설치하자마자 한글이 적용되어 에러 파악도 수월했던 경험이 있었다. 하지만 아직 운영체제에 대한 개념이 부족해 큰 차이가 없을 것이라고 생각하고, 기존에 사용했었던 윈도우 환경의 visual studio 2022 에 프로젝트를 생성해서 진행했었다. 윈도우에서 기능을 몇 개 구현하고 오류가 없는 것을 확인하고, 이후에 ubuntu 환경의 vim 에서 컴파일을 했을 때 컴파일 에러가 발생하는 것을 보고 매우 놀랐었다. 익숙한 윈도우 환경에서 기능 구현 후 우분투로 옮겨 매번 확인하는 작업도 번거로울 뿐더러 옮긴 후에 에러가 발생한다면 추가적인 수정 작업을 거쳐야 하기 때문에 그 간극에 대한 차이 때문에 이러한 방법은 매우 비효율적이라고 판단했다. 그래서 우분투 리눅스 18.04 환경을 구축하되, visual studio 처럼 익숙한 툴로 작업할 수 있는 방법이 있는지 구글링을 통해 여러 방법을 찾은 결과, 결과적으로 윈도우 환경에 visual studio code 프로그램을 설치하고, wsl2 로 우분투 리눅스 18.04 환경에서 작업하는 방법을 선택하였다. 툴과 환경에 대한 불편함이 어느 정도 해소되었지만, gcc 라는 다른 컴파일러를 사용한다는 점에서 오류를 다루는 측면에서는 여전히 익숙하지 않았다. 게다가 default 가 영어로 되어 있으나, 일부러 한글로 바꾸지 않고 영어 버전을 사용하고자 마음먹은 내

고집 또한 보태진 결과였다. 이대로 사용하면서 느꼈던 것은, 윈도우 환경에서의 성공적인 컴파일이 리눅스 환경에서 컴파일 에러를 발생시키는 것이 아닌, 컴파일러의 차이에 의한 것이라는 점을 새로 알게 되었다. 게다가 같은 에러에 대해 두 컴파일러가 다른 오류 메시지를 출력하여 에러에 대한 구글링을 할 때에도 오류 레퍼런스를 넓게, 많이 보아야 한다는 단점이 있었다. 하지만 이와 같은 문제는 vscode wsl2 를 사용하면서 프로젝트에 할애한 수많은 시간에 의해 어느 정도 해소되었으나 여러 에러를 통합해서 segmentation fault 에러나 aborted 에러로 보여준다는 느낌으로부터는 아직 익숙하지 않았으나 2 차, 3 차 프로젝트를 진행하면서 오류 처리에 대한 프로세스를 진행하는데 좋은 경험이 될 수 있겠다는 생각을 하게 되었다. 마지막으로, 파악하기 매우 어려웠던 기억에 남는 에러이다. 상황에 앞서 오류 메시지는 아래와 같다.

```
c++ exception has occurred aborted
```

이 오류가 앞서 언급했던 범용적으로 출력하는 느낌의 에러이다. 이 오류는 Loaded\_LIST 의 노드를 삭제하는 과정에서 나타났다. MOVE command 나 MODIFY command 가 수행되기 위해서는 Loaded\_LIST 의 노드를 삭제하는 과정이 필요하다. 그런데 노드를 삭제하는 과정에서 해당 오류가 발생해 디버깅을 F11 을 눌러가며 하나하나 다 확인하였는데도 그 원인을 알 수 없었다. 파악이 용이하게 하기 위해 노드들을 순차적으로 insert 된 순서대로 delete 하는 함수를 구현해 실행해보았다. 그 결과 100, 111, 200, 222, 300, 333, 400 까지는 잘 삭제되었으나 444 를 삭제하는 과정에서 위의 오류가 다시 그대로 출력되면서 컴파일 에러가 발생하였다. 삭제하는 코드를 내가 잘 작성하지 않았을 가능성이 높아 3 번을 새롭게 작성하고 다른 로직도 적용해보았으나 똑같이 444 를 삭제하던 중 오류가 발생하였기 때문에 해당 가능성은 배제하였다. 디버깅을 해보면 삭제되었던 노드들과 다르게 444 delete 코드에서 갑자기 소멸자로 이동하더니 aborted 에러가 발생하는 것이었다. 더욱 알 수 없었던 점은 head 노드나 tail 노드를 삭제하는 과정에서 발생한 오류라면 충분히 발생할 수 있을 것 같다는 생각이 들었는데 중간을 삭제하다가 이런 경우는 처음 겪고, 주위에서 이런 상황을 경험해본 사람이 없어서 조언을 구하기도 매우 힘들었다. 해당 에러로 구글링을 하면 지워진 주소를 다시 참조하려고 하기 때문에 생기는 오류라고 하여서 insert 함수를 잘못 작성했는지 검증했으나 bst 에도 잘 들어가고, next 로 다음 노드의 주소 또한 각자 다 다르게 잘 할당된 것을 확인하여 insert 에도 이상이 없음을 확인하였다. 따라서 오류에 대한 레퍼런스도 도움이 되지 못 하는 상황이고, 아무리 insert, delete 함수를 수정하고, 디버깅을 해도 소용없는 상황을 겪게 되었다. 모든 기능을 구현했으나 이 오류 때문에 edit 에서 256x256 크기로 노드를 할당하는 코드에서 저장 범위를 넘었다는 이상한 오류까지 보게 되었다. 최후의 수단으로 조교님들께 메일을 보내어 감사히도 도움을 받을 수 있게 되었다. 찾아주신 오류의 원인은 command.txt 의 ADD 명령어의 추가 인자를 받아오면 string 형인데, strtok 를 사용하기 위한 과정에서 char\* 형을 new 로 동적으로 할당하여 사용한 후 delete 로 바로 할당해제를 해주지 않아서 생긴 오류였다. 자세한 원인은 모르겠으나 개인적으로



노드의 new 크기와 char\* 동적할당의 new 크기가 달라 할당하는 과정에서 충돌이 나 unalign 이 발생해 생긴 오류라는 추측을 해보았다. 해결 방법은 char\*형으로 동적 할당하는 것 대신 char 형으로 정적으로 할당해주면 오류가 해결되고, 노드의 delete 기능 또한 정상적으로 해제 되는 것을 확인할 수 있었다. 지금까지 동적 할당 해제를 하면서 동적 할당 요소를 제거를 해야만 할 수 있는 기능에 대한 해제만 고려하여 작업해주었는데, 이전에 동적 할당한 요소가 후에 동적 할당한 요소에 영향을 줄 수 있는 상황은 처음 겪게 되었다. 이번 프로젝트를 진행하면서 가장 생각나는 시행착오이며, 할당 후 사용하지 않는 메모리 공간은 바로 바로 확보를 해주어야 겠다는 교훈을 얻은 소중한 경험이었다.