데이터 구조 실습 보고서

학　　과: 컴퓨터정보공학

담당교수: 이기훈 교수님

학　　번: 2019202075

성　　명: 김효석

Introduction

해당 프로젝트는 이진 탐색 트리와 연결리스트, 큐(스택) 자료구조를 이용하여 간단한 사진 파일 편집 프로그램을 구현하는 것에 목적을 둔다.

이 프로그램은 특정 경로에 저장된 사질 파일에 대한 정보를 링크드 리스트(Linked List)로 저장한 후, 검색에 용이하게 트리 자료 구조로 저장한다.

이진 트리 탐색 알고리즘을 통해 사용자가 입력한 이미지 파일을 찾아서 해당 파일 경로를 읽어온다. 이후 큐(스택) 자료구조를 활용해 각각의 이미지 파일을 읽어오고, 동일한 디렉토리의 command.txt에 입력된 명령어에 맞게 이미지를 변형하는 기능을 추가하여 이미지 변형 프로그램을 구현한다.

이때 사용하는 이미지 파일의 크기(width, height)는 고정적이며, 동일하다.

본 프로젝트를 구현할 때의 주의점은 다음과 같다.

- 모든 명령어는 대문자로 입력한다.
- 명령어에 인자(parameter)가 모자라거나 필요 이상으로 입력 받을 경우 에러 코드를 출력한다.
- 예외처리에 반드시 에러코드를 출력한다.
- 출력은 제안서에 나와있는 출력형태를 반드시 따른다.
- 읽어야 할 파일이 존재하지 않는 경우 해당 파일을 생성한 뒤 진행하도록 한다.
- 링크드 리스트 내 자료가 100개 이상일 경우 삭제
- 트리 구조 내 자료가 300개 이상일 경우 삭제

본 프로젝트를 구현할 때의 반드시 선언해야 하는 class는 다음과 같다.

- Loaded_LIST
- Loaded_LIST_Node
- Database_BST
- Database_BST_Node
- Manager

위의 class는 각각 다음의 역할을 가지고 있다.

Loaded_LIST(Node): 링크드 리스트를 생성 후 변경 등등 수행

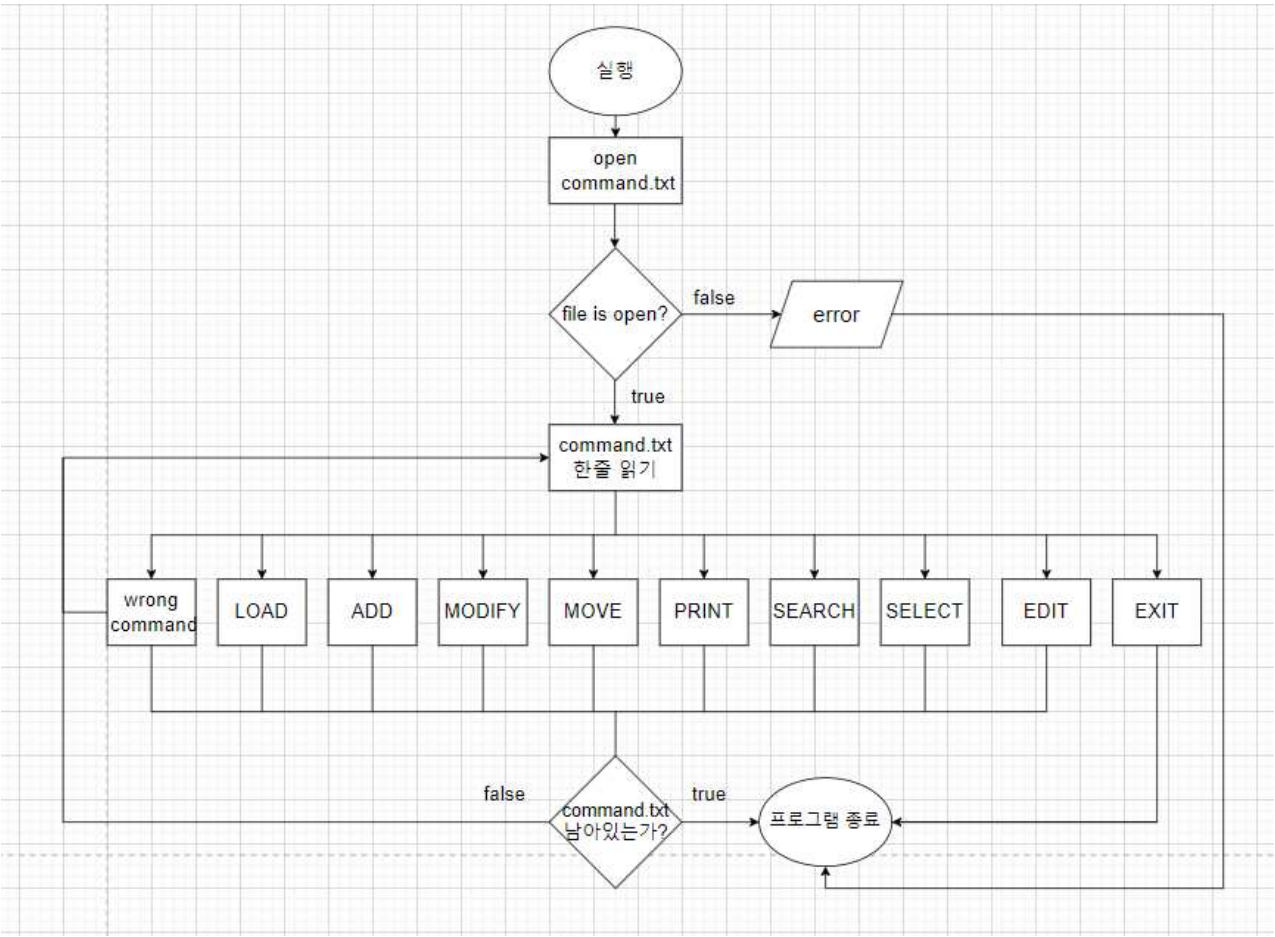
Database_BST(Node): binary search tree 생성 후 출력 및 변경 등 수행

Manager: 다른 모든 클래스들의 동작을 관리하여 프로그램 전체적으로 조정하는 역할 수행

Flow chart

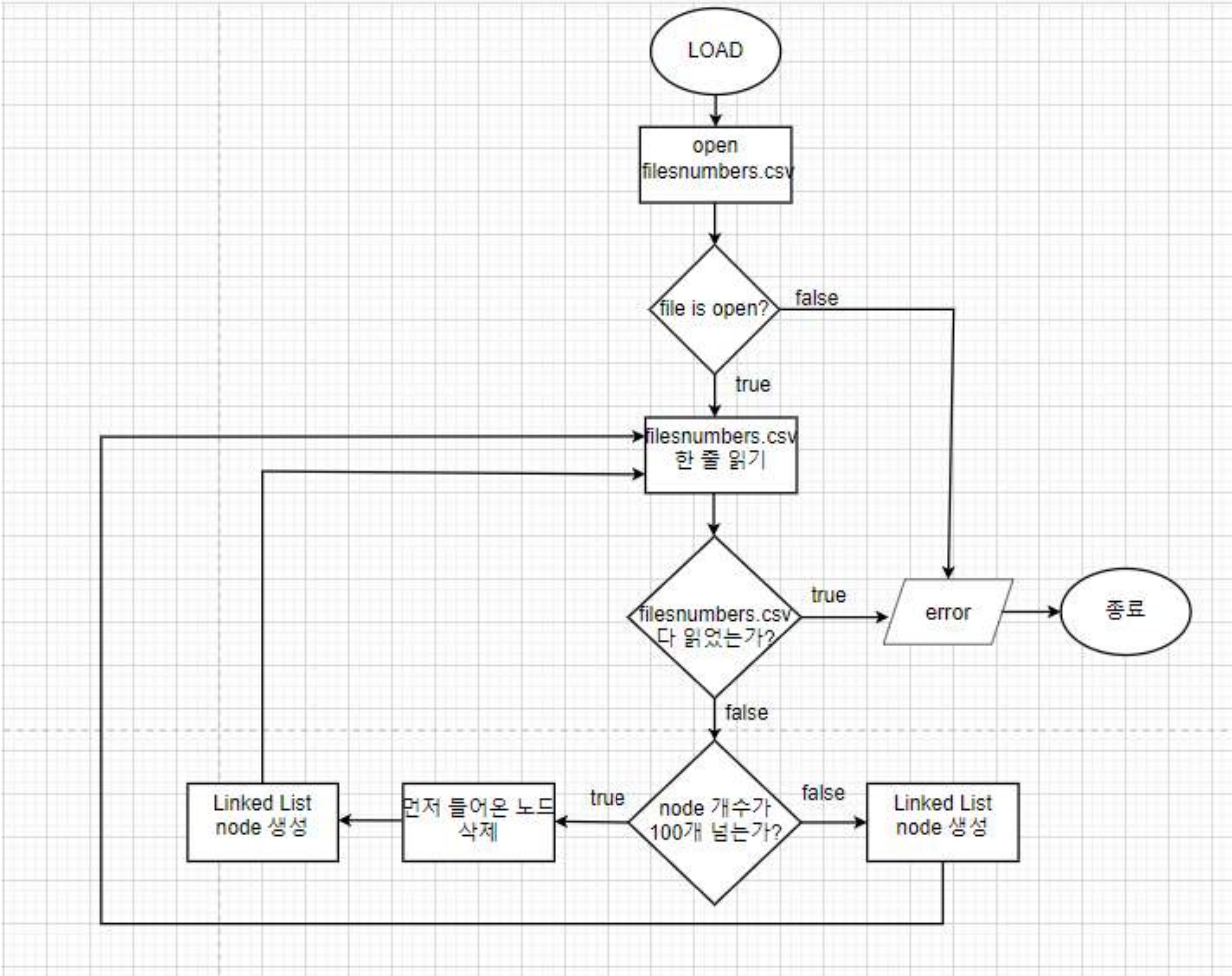
manager

프로그램을 전반적으로 관리해주는 class이다. Manager class는 프로그램이 시작하면 command.txt 로부터 명령어를 읽어와 해당하는 동작을 수행한다.



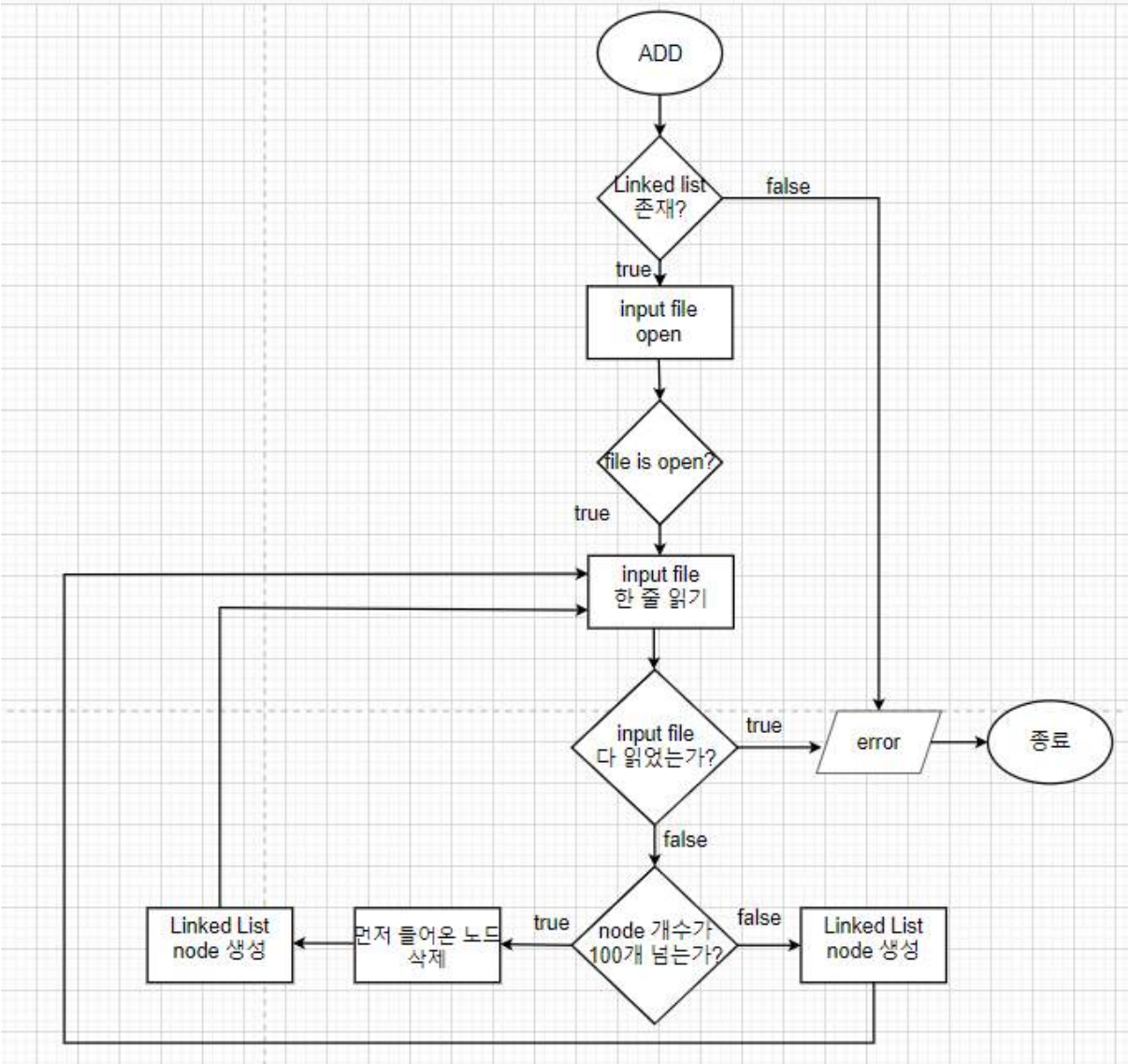
LOAD

LOAD 명령어는 "filesnumbers.csv"에 저장되어 있는 데이터 정보를 불러오는 명령어이며, csv파일에 데이터 정보가 존재할 경우 csv파일을 읽어 링크드 리스트에 모두 저장한다. 만약 저장할 때 자료의 개수가 100개가 넘을 경우 먼저 들어왔던 노드를 삭제 후 새로운 데이터를 추가한다.
LOAD를 실행할 때의 파일 경로는 "img_files/filesnumbers.csv"으로 고정적이다.



ADD

ADD 명령어는 새로운 디렉토리를 탐색 후 데이터를 추가하는 명령어이다. Loaded_LIST에 새로운 디렉토리의 파일 데이터들을 추가한다. 데이터를 추가할 때 기존 Linked list가 존재한다면 2차원 링크드 리스트로 구성하고, 만약 기존 Linked List가 존재한다면 에러 코드를 출력한다. 만약 Linked List 전체 노드 개수가 100개가 넘을 경우 먼저 들어왔던 노드 삭제 후 새로운 데이터를 추가한다.

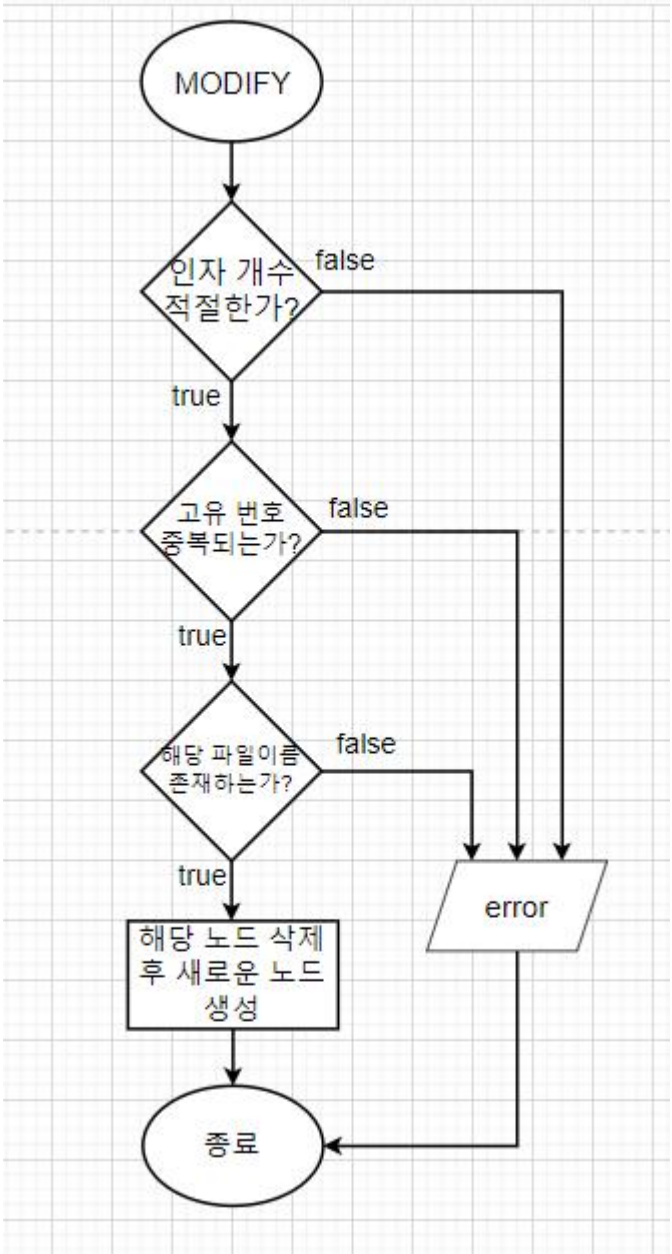


MODIFY

MODIFY 명령어는 Loaded_LIST에 존재하는 노드의 파일 고유번호를 수정하기 위한 명령어이다.

파일 고유번호는 중복되지 않으며 만일 중복된다면 에러 코드를 출력한다.

또한 인자가 하나라도 없거나 파일 이름에 대한 노드가 존재하지 않는다면 에러 코드를 출력한다.

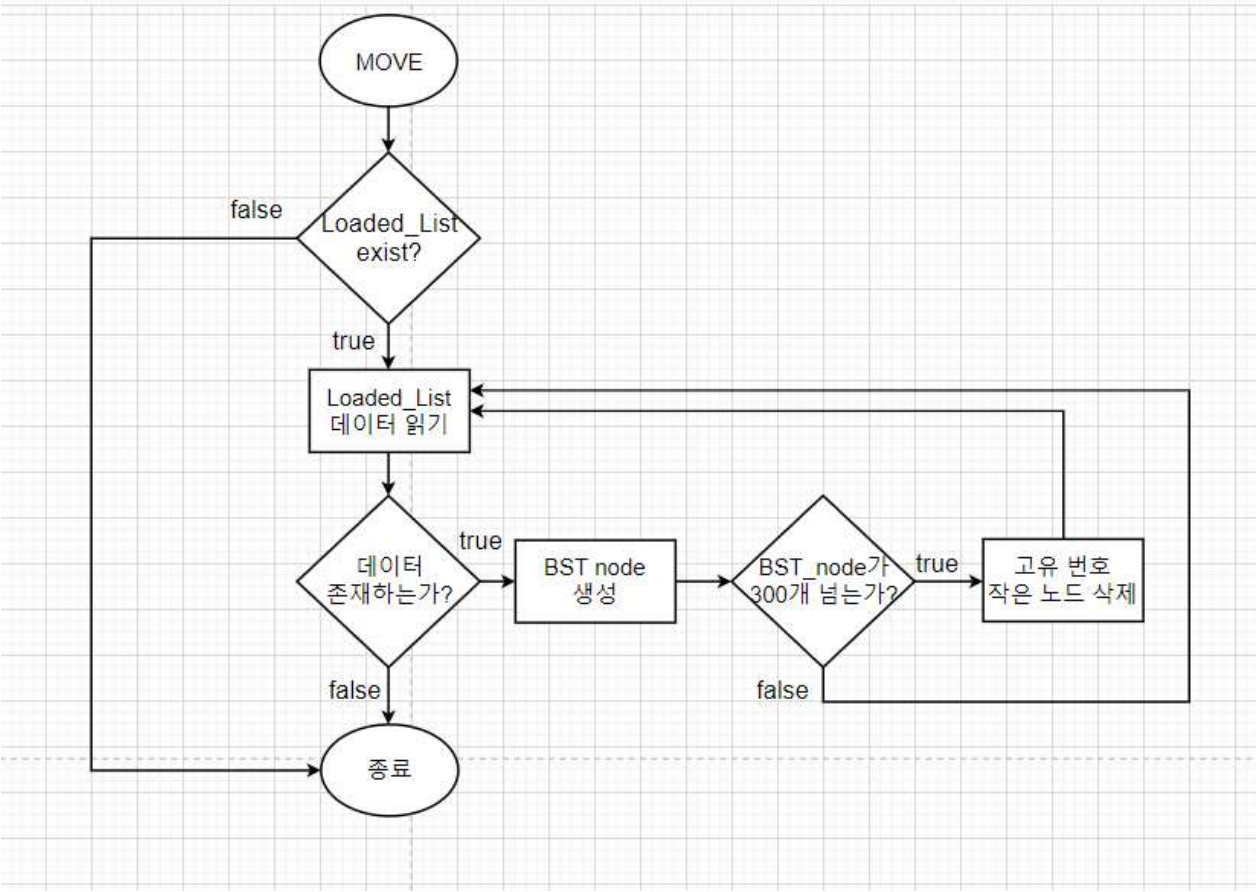


MOVE

MOVE 명령어는 Loaded_List 안에 있는 데이터들을 Database_BST로 옮기는 역할을 하는 명령어이다. 파일 고유번호와 파일 이름을 이용하여 BST노드를 생성하며 만약 BST노드가 300개 이상이 된다면 고유번호가 가장 낮은 BST노드부터 삭제한다.

다음은 BST 연결 규칙이다.

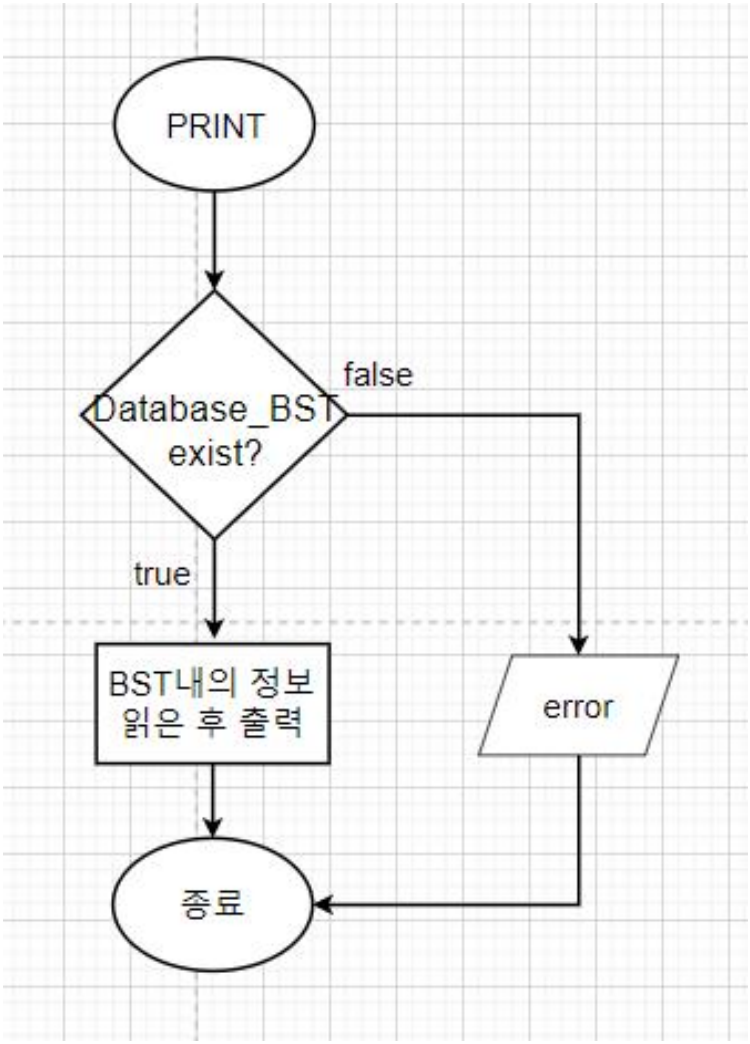
1. 부모 노드보다 파일 고유번호가 작은 노드는 왼쪽, 큰 노드는 오른쪽 서브 트리에 위치시킨다
2. 노드를 제거할 때, 양쪽 자식 노드가 모두 존재할 경우엔 왼쪽 자식 노드 중 가장 큰 노드를 제거되는 노드 위치로 이동시킨다.



PRINT

PRINT명령어는 Database_BST에 있는 정보를 출력하는 명령어이다. Database_BST가 존재하지 않는다면 에러 코드를 출력한다.

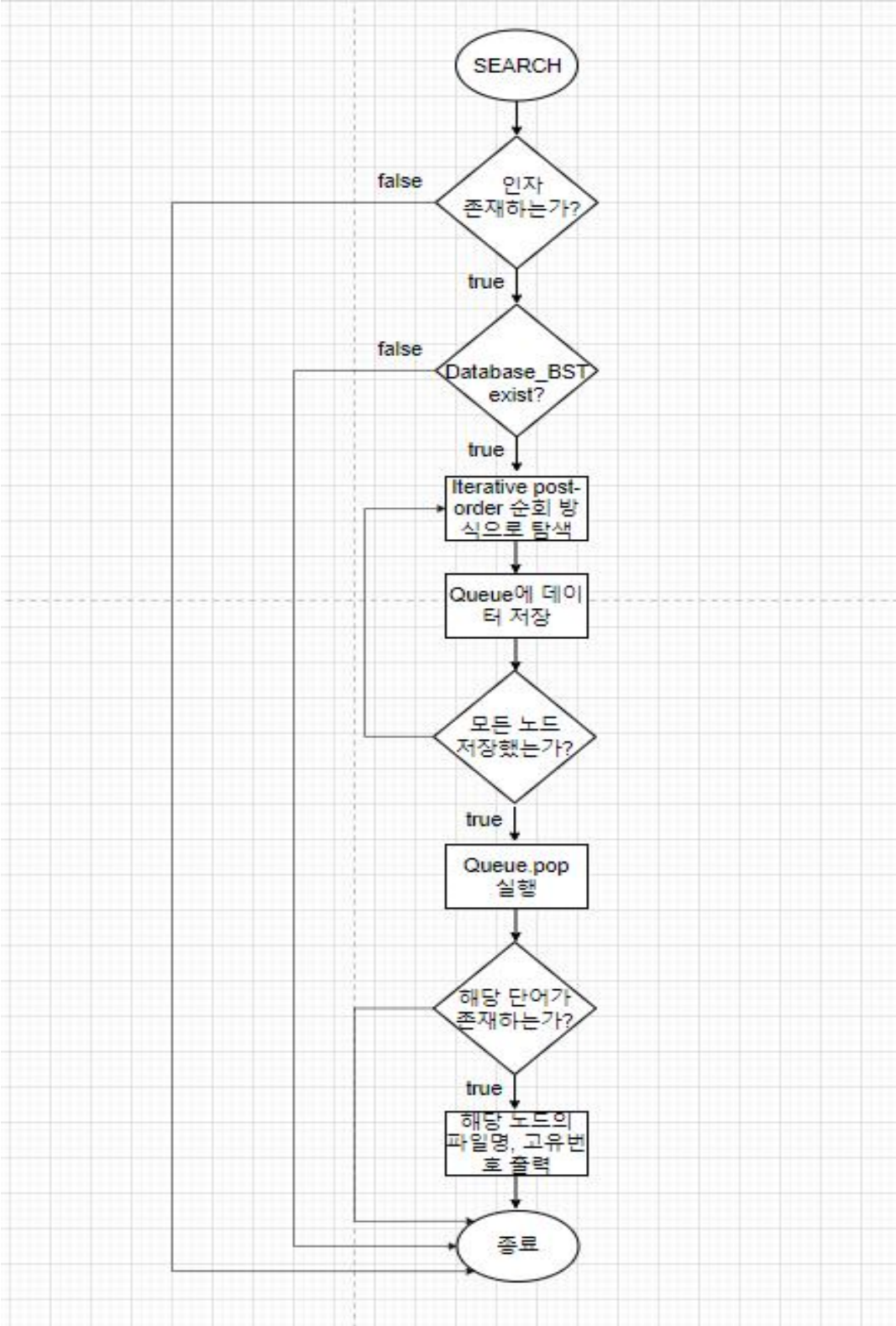
또한 출력할 때 중위 순회 방식(Inorder)으로 트리를 순회하며 해당 정보를 출력한다.



SEARCH

SEARCH명령어는 특정 단어가 포함된 파일의 이름을 탐색하여 출력하기 위한 명령어이다. 인자로 하나 이상의 단어를 받으며 보이어 무어 알고리즘을 이용해 파일을 탐색하여 출력한다.

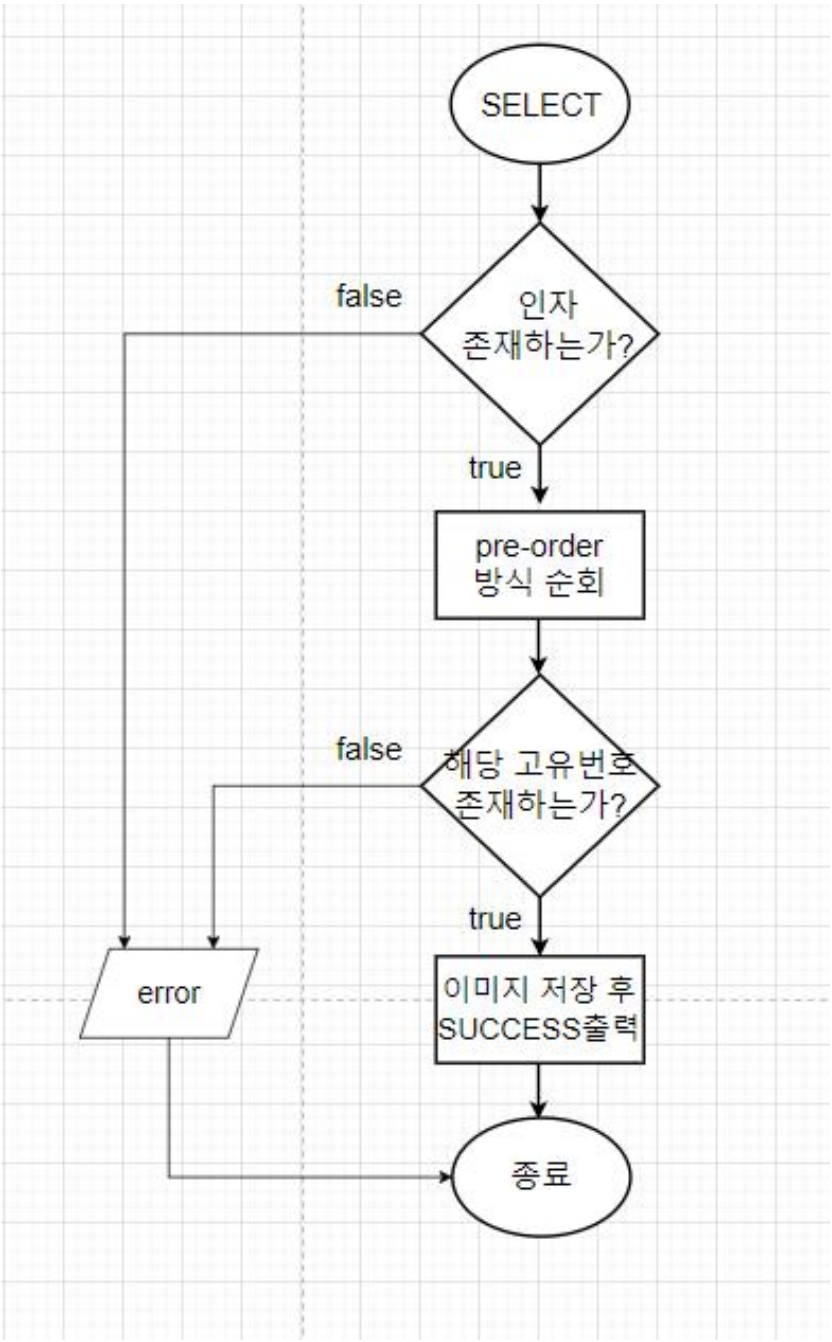
만약 인자가 없거나 Database_BST가 비어있는 경우 에러코드를 출력한다.



SELECT

SELECT 명령어는 EDIT 명령어를 위해 Database_BST 에서 파일 고유번호를 기반으로 전위 순회를 통해 이미지의 경로를 찾아서 프로그램 내에 저장하는 명령어이다.

만약 인자가 없거나 파일 고유번호가 존재하지 않을 경우 에러 코드를 출력한다.

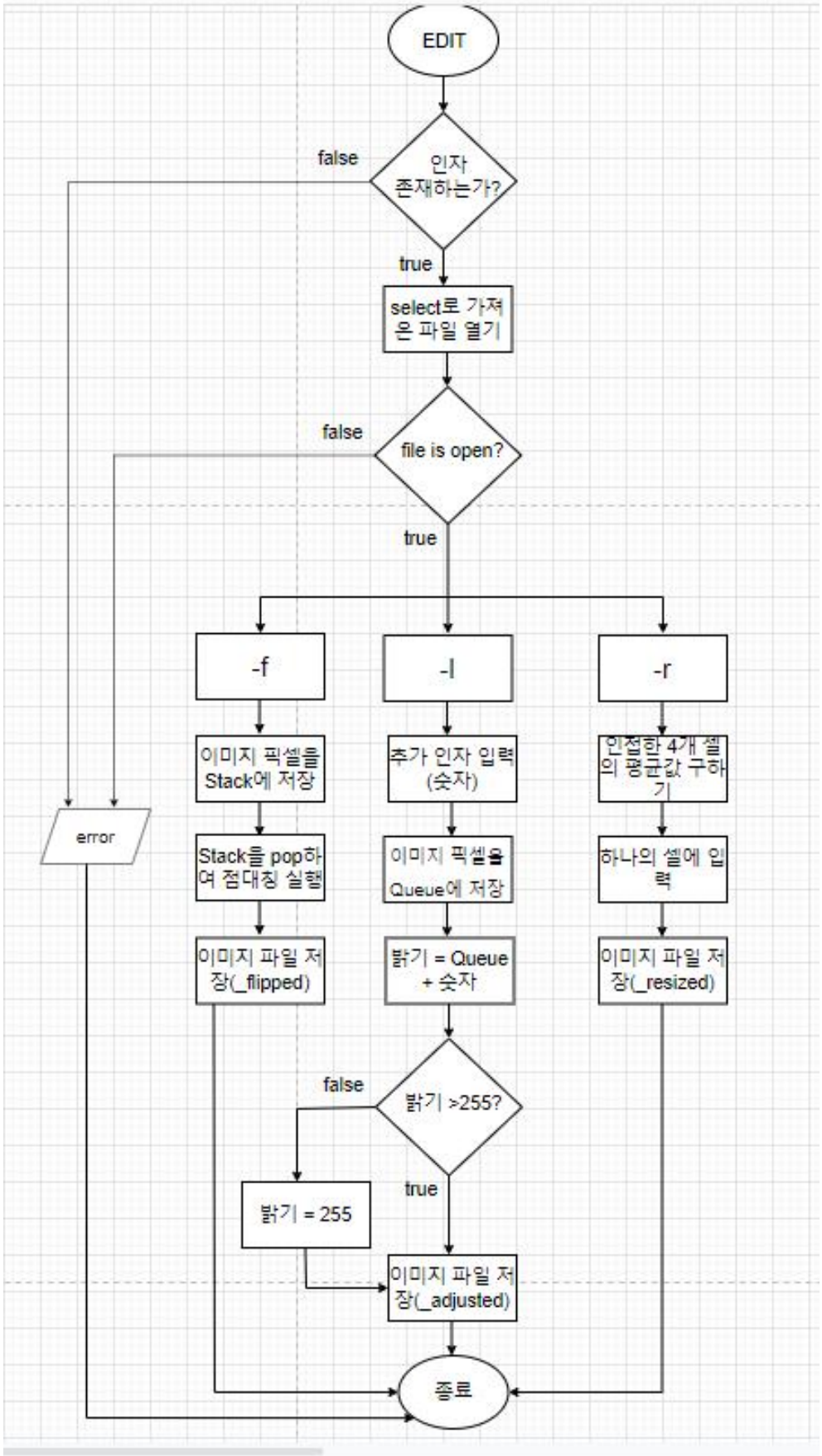


EDIT

EDIT 명령어는 SELECT 명령어를 통해 가져온 이미지를 편집하는 명령어로 주어진 코드를 이용하여 이미지 파일을 읽어와서 수정하고 저장한다.

-f, -l, -r의 세가지의 옵션을 가지며 각각의 옵션들은 점대칭, 밝기 조정, 크기 조정하는 역할을 한다. 또한 -l옵션 같은 경우 하나의 인자를 추가적으로 가진다. 각각의 옵션으로 편집된 이미지는 "Result" 디렉토리에 저장된다.

인자가 존재하지 않는 경우 에러코드를 출력한다.


EDIT
인자 존재하는가?
false
true
select로 가져온 파일 열기
file is open?
false
true
-f
-l
-r
error
이미지 픽셀을 Stack에 저장
추가 인자 입력 (숫자)
인접한 4개 셀의 평균값 구하기
Stack을 pop하여 점대칭 실행
이미지 픽셀을 Queue에 저장
하나의 셀에 입력
이미지 파일 저장(_flipped)
밝기 = Queue + 숫자
이미지 파일 저장(_resized)
밝기 >255?
false
true
밝기 = 255
이미지 파일 저장(_adjusted)
종료

Algorithm

본 프로젝트에서 사용한 알고리즘은 Linked List, Binary Search tree, Queue, Stack, Boyer-Moore Algorithm 총 4 개이다.

<linked List> (Loaded_LIST)

Linked List란 추상적 자료형인 리스트를 구현한 자료구조로, 말 그대로 어떤 데이터 덩어리(node)를 저장할 때 그 다음 순서의 자료가 있는 위치를 데이터에 포함시키는 방식으로 자료를 저장한다. 예를 들어 해당 프로젝트에선 파일을 열어 데이터를 입력 받는데 이때 한 node에 데이터를 filename, file directory, file unique num, next node를 저장시키는 것을 예로 들 수 있다. 이는 즉, 데이터를 저장하는 것뿐만 아니라 다음에 데이터가 들어온다면 전에 만든 node의 다음 node로 이동시켜서 데이터를 할당해 줄 수 있다는 말과 같다.

일반 배열과는 달리 데이터의 추가/삽입 및 삭제가 용이하지만 순차적으로 탐색하지 않으면 특정 위치의 요소에 접근할 수 없기 때문에 일반적으로 탐색 속도가 떨어진다. 따라서 탐색, 정렬을 자주 한다면 배열을 사용하는 것이 유리하며 추가/삭제가 많다면 Linked List를 사용하는 것이 유리하다.

해당 프로젝트에선 파일의 데이터를 읽어와 추가하는 행동이 많기 때문에 Linked List를 사용하는 것이 유리하다고 볼 수 있다.

-해당 Linked list의 implement

1.리스트의 이름은 Loaded_LIST로 정의한다.

2. 각 노드는 파일의 이름, 디렉토리 명, 파일의 고유번호에 대한 정보를 갖는다.

3. 노드가 입력되는 순서대로 링크드 리스트를 구현한다.

4. 만약 Linked List의 node개수가 100개 넘는다면 먼저 들어온 node를 삭제하고 새로운 노드 생성한다.

-다음은 Linked List(Loaded_LIST)에서 구현한 함수들이다.

반환 타입	함수	설명
Loaded_List_Node*	Load(string num,string name)	List 구조 생성 후 삽입
Loaded_List_Node*	Add(string num, string name, const char* dir)	List 2차원으로 생성
Loaded_List_Node*	Modify(const char* Dir, const char* filename, string num)	해당 노드 삭제 후 재생성
Loaded_List_Node*	GetEnd()	List의 가장 끝 노드 반환
Loaded_List_Node*	FindDel(Loaded_List_Node* CurNode)	해당 노드의 전 노드 반환

이외에 List의 데이터 출력 및, head를 반환하는 함수는 표에 삽입하지 않았다.

Load

Load함수는 manager class에서 LOAD 명령어가 입력되었을 때에 해당 파일 내의 데이터들을 읽어와 Linked List를 만드는 함수이다.

이 때, 파일의 위치는 ./images/filesnumbers.csv로 고정적이며 해당 파일에서 한줄씩 읽어와 Linked list를 만든다.

Filesnumbers.csv 내에

1,there are lots of people in the park.RAW

2,the man is gorgeous.RAW

3,the woman is smiling.RAW

와 같이 저장되어 있다고 한다면, Load 명령어를 통해 1,there are lots of people in the park.RAW 를 읽어와 1은 num으로 there are lots of people in the park는 Name으로 받아와 이를 하나의 노드로 저장한다.

Add

Add함수는 manager class에서 ADD 명령어가 입력되었을 때에 기존에 존재하던 Linked List에 2차원으로 노드를 연결해주는 역할을 하는 함수이다.

만약 기존에 Linked List가 존재하지 않는다면 에러 코드를 출력한다.

Add함수가 Linked List를 구현하는 방법은 다음과 같다.

1) ADD 명령어를 통해 입력 받은 새로운 파일위치를 열고 만약 해당 파일이 비어 있다면 에러코드를 출력하고 비어 있지 않다면 해당 파일을 연다.
2) Linked List가 존재한다면 해당 명령어를 통해 받은 directory정보를 담은 노드를 new를 사용해 생성한다.(일종의 헤드노드 역할을 하며 해당 directory를 빠르게 찾기 위하여 이러한 방식을 채택하였다.)
3) Directory의 정보만을 받은 노드가 생성되었다면 그 다음 노드부터는 입력 받은 고유번호와 파일 이름을 가진 노드를 순차적으로 생성해 다음 노드를 가리키도록 한다.
4) 만약 ADD 명령어를 통해 연 새로운 파일의 정보를 다 저장했다면 함수를 종료한다.
5) 추가로 ADD 명령어가 여러 번 입력되었다면 위 행동을 반복하면 GetDown함수를 이용해 밑으로 계속 정보를 추가해 나간다.

Modify

Modify함수는 manager class에서 MODIFY 명령어가 입력되었을 때에 Linked List에 있는 정보와 비교해 입력 받은 정보와 해당 정보가 같다면 해당 노드를 삭제하고 새로운 노드를 생성해 해당 위치에 위치시키는 역할을 하는 함수이다. 만약 Linked List가 존재하지 않거나 해당 정보가 존재

하지 않으면 nullptr을 반환하고 에러코드를 출력한다.

GetEnd

GetEnd함수는 사용자가 MOVE 명령어를 입력하면 사용되는 함수이다. MOVE 명령어가 입력되면 Linked List의 정보를 모두 Binary Search Tree로 데이터를 넘겨주어야 하는데 이때 처음부터 값을 넘겨주는 것이 아닌 Linked List의 끝에서부터 정보를 넘겨주어야 하는 것이 이번 프로젝트의 목적이다. 따라서 GetEnd함수를 이용해 2차원으로 생성된 Linked List가 있다면 가장 오른쪽 밑에 있는 노드를 반환한다. 따라서 반환형은 node를 반환해야 하므로 Loaded_List_Node*이다.

FindDel

FindDel함수는 위의 GetEnd로 찾은 노드를 MOVE 명령어로 옮길 때 해당 노드의 이전 노드에서 next 포인터 값을 nullptr로 바꿔주어야 하기 때문에 사용하는 함수이다. Binary Search Tree로 옮기고 난 후 FindDel로 찾은 노드의 next를 nullptr로 바꿔준다. 따라서 MOVE명령어를 입력하면 위 2함수로 인해 Linked List가 비어 있게 된다.

<Binary Search Tree> (Database_BST)

Binary Search Tree 즉, 이진 탐색 트리란 이진 탐색(binary search)와 연결리스트(linked list)를 결합한 자료 구조의 일종입니다. 이진 탐색의 효율적인 탐색 능력을 유지하면서도 자료 입력과 삭제를 가능하게 하는 자료구조이다.
이진 탐색 트리는 다음과 같은 속성을 지닌다.

1) 각 노드의 왼쪽 서브트리에는 해당 노드의 값보다 작은 값을 지닌 노드들로 이루어져 있다.
2) 각 노드의 오른쪽 서브트리에는 해당 노드의 값보다 큰 값을 지닌 노드들로 이루어져 있다.
3) 중복되는 노드가 없어야 한다.
4) 왼쪽 서브트리, 오른쪽 서브트리 또한 이진 탐색트리이다.

-해당 Binary Search Tree의 implement

1) BST의 class 이름은 DataBase_BST로 정의한다.
2) MOVE 명령어를 통해 Loaded_LIST에서 데이터를 가져올 때는 Loaded_LIST의 가장 마지막 노드부터 헤드노드까지의 순서대로 데이터를 입력한다.
2) 만약 BST의 노드 개수가 300개가 넘는다면 가장 작은 고유번호를 가지는 노드를 삭제 후 새로운 데이터를 입력 받는다.

-다음은 BST(Database_BST)에서 구현한 함수들이다.

반환 타입	함수	설명
Void	Move(Loaded_List* list)	해당 Linked List가 비어 있는지 확인하고 Insert함수 호출
Void	Insert(Loaded_List_Node* CurNode, Loaded_List_Node* DelNode, Database_BST_Node* bstNode, Loaded_List* list)	Move함수로부터 받은 정보를 기반으로 BST 생성
Void	Delete(string value)	만약 BST의 개수가 300개가 넘는다면 가장 작은 고유번호를 가진 노드 삭제
Void	InsertQueue	Iterative postorder방식을 이용해 queue에 데이터 삽입
Int	Seacrh(string ptr2, Database_BST_Node* node)	BoyerMoore함수 실행하는 함수
Void	BoyerMooreSearch(char* ch, char* name, string num)	BoyerMoore 알고리즘을 이용해 해당 데이터가 있다면 문자열 출력하는 함수
String	Inorder(Database_BST_Node* root, string key)	InorderSelect함수 호출 및 해당 노드 파일 이름 반환
Void	InorderSelect(Database_BST_Node* root, string key)	입력 받은 고유 번호와 맞는 노드의 파일 이름을 찾아내는 함수

위에 적힌 함수 이외에 함수들은 해당 함수 내에서 호출해 실행되기 때문에 표에 삽입하지 않았다.

MOVE

Move 함수는 MOVE 명령어가 입력된다면 Loaded_List에 있는 GetEnd함수와 FindDel함수를 이용해 링크드 리스트 내의 데이터들을 가져와 해당 노드들의 데이터들을 Insert 함수를 호출해 Insert 함수에 데이터를 전달하는 역할을 한다. 만약 Linked List에 Directory 정보만 가지고 있는 노드만 남게 된다면 해당 헤드 노드를 삭제하고 함수를 종료한다.

Insert

Insert 함수는 Move함수에서 호출된다. 이 함수는 Move함수로부터 받은 정보를 기반으로 해당 데이터를 입력하는 역할을 한다. 만약 BST가 존재하지 않는다면 새로운 BST node를 생성해 삽입한다. 이외에 헤드 노드가 존재한다면 노드 안의 고유번호를 비교해 삽입할 노드의 고유번호가

기존에 존재하던 노드보다 작다면 왼쪽 노드로 크다면 오른쪽 노드로 이동하면서 데이터를 삽입한다. 이러한 방식으로 삽입을 진행하다가 노드 개수가 300개가 넘는다면 고유번호가 가장 작은 노드를 삭제하고 입력 받은 데이터를 가진 노드를 생성해 해당 노드를 삽입한다.

InsertQueue

InsertQueue 함수는 SEARCH 명령어가 입력되었을 때에 호출되는 함수이다. 이 함수는 BST에 저장되어 있는 데이터들을 Queue로 옮기는 작업을 한다.
Queue란 선입선출의 개념을 가지는 자료구조의 한가지이다. 한마디로 먼저 집어넣은 데이터가 먼저 나오는 FIFO(First in First Out)의 개념을 가진다.(스택과는 반대된다)
Queue로 옮길 때 Iterative한 Post-Order 방식으로 순회하며 Queue에 저장해야 한다. Post-Order는 후위 순회로 탐색을 할 때 사용되는 알고리즘으로, left->right->root순으로 탐색을 실시하는 것이 특징이다.
대표적인 postorder의 방법으로는 2가지가 있다.

1. 재귀적인 postorder
2. 반복적인 postorder

재귀적인 postorder은 루트노드를 방문하기 전에 left child가 있다면 먼저 방문하고 right child가 있다면 방문하고 마지막으로 해당 루트 노드를 순회하는 방식을 함수를 재호출하는 방식을 채택하여 사용한다. 예를 들면 루트 노드가 함수의 인자 기본값이라면 해당 루트 노드가 왼쪽 자식 노드들 중 왼쪽 자식 노드가 없는 노드로 이동하게끔 root->leftchild 이런식으로 함수의 인자를 넘겨주어서 재호출하는 방식이다.

반복적인 postorder는 위의 재귀적인 postoder와 기본 개념은 같지만 재귀 함수를 사용하지 않는 것을 말한다. 이 때 사용한 방법이 stack 2개를 이용하여 이를 실행하는 것이다.
다음은 그에 관한 설명이다.
-Post order 방식, 즉 왼쪽, 오른쪽, 루트 노드 순서대로 탐색하는 것을 반대로 해서 스택에 삽입한 후 나중에 해당 스택에 저장된 값을 꺼내서 탐색 결과를 뒤집어 반환하는 방식이다.

따라서 InsertQueue는 반복적인 postorder 방식을 사용해 queue에 데이터를 queue에 넣는 함수이다.

Search

Search함수는 Queue가 비어 있을 때까지 Queue에 저장되어 있는 데이터를 큐의 front에서 뽑아와서 BoyerMoore 알고리즘을 사용하고 큐에 저장되어 있는 정보를 삭제하는 역할을 하는 함수이다.
다음은 BoyerMoore함수의 설명이다.

BoyerMoore

BoyerMoore함수는 Search 함수에서 받아온 데이터를 기반으로 해당 문자열에 SEARCH 명령어를 통해 입력 받은 문자가 있는지 확인하고 만약 같은 문자가 있다면 해당 문자를 출력하는 역할을 하는 함수이다.
보이어 무어 알고리즘은 전체 문자열과 문자 2가지를 가지고 실행하는데 전체 문자열에서 입력 받은 문자의 끝 문자가 있는지 검사한다. 만약 있다면 끝 문자가 검출된 문자열의 위치에서 앞에 배열을 검사하고 맞다면 위의 방식을 계속 반복하여 끝까지 맞을 경우 true를 반환하는 알고리즘이다.

Delete

Delete함수는 해당 BST의 가장 작은 고유번호값을 가지는 노드를 재귀 함수를 통해 찾는다.
다음으로 Successor 함수를 이용해 해당 삭제해야 할 노드의 대체 노드(왼쪽 자식 노드중 가장 큰 값 혹은 오른쪽 자식 노드중 가장 작은 값)를 찾은 후 해당 노드를 삭제 후 Successor 함수를 이용해 찾은 노드를 삭제시킨 노드가 있던 자리에 위치시킨다.

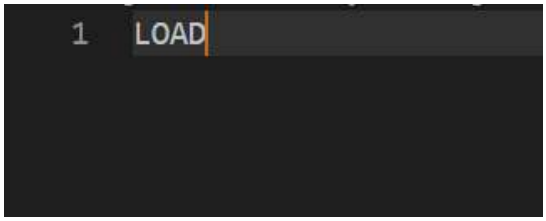
Inorder, InorderSelect

Inorder 함수는 SELECT 명령어가 들어왔을 경우에 실행되는 함수이다. 해당 함수는 InorderSelect 함수를 호출하며 Selectch라는 전역 변수를 하나 설정해 입력 받은 고유 번호와 같은 고유번호를 가지는 노드를 찾고 해당 노드의 파일 이름을 Selectch에 저장하는 역할을 한다.

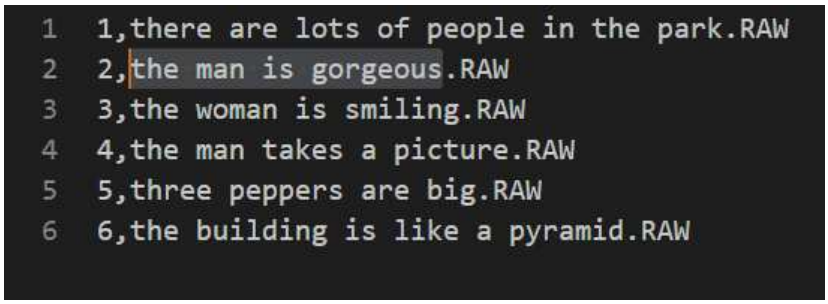
Result screen

다음은 검증을 위해 사용한 Command.txt와 log.txt의 사진이다.

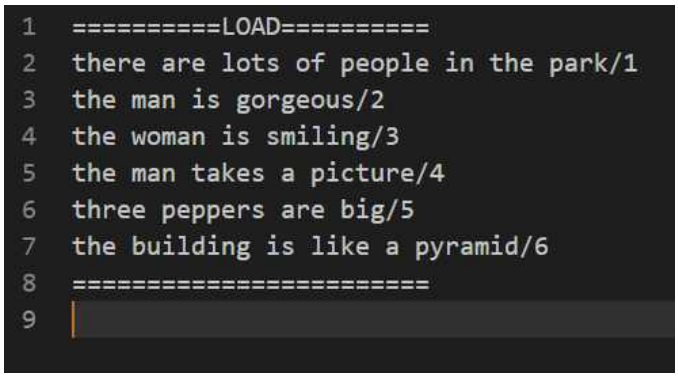
아래는 Command.txt의 사진이고 LOAD 명령어가 있는 것을 확인할 수 있다.



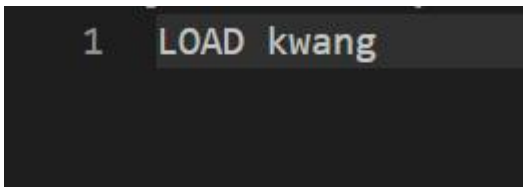
아래는 LOAD 명령어가 입력되면 읽어올 filesnumber.csv파일이다. 파일 내용은 다음과 같다.

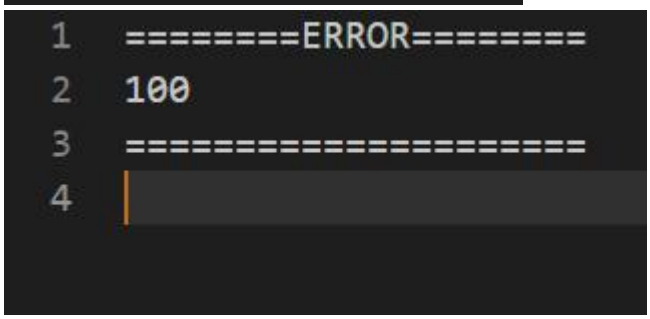


아래는 log.txt의 사진이다. Filesnumber.csv 파일 안의 정보들이 LOAD 된 것을 확인할 수 있다.



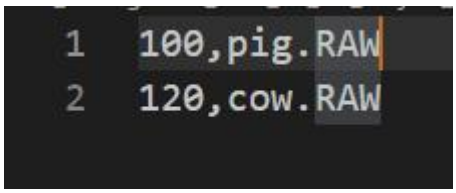
LOAD 명령어는 LOAD외에 입력이 있다면 에러 코드를 출력하게 하였다. 아래는 그에 대한 사진이다.



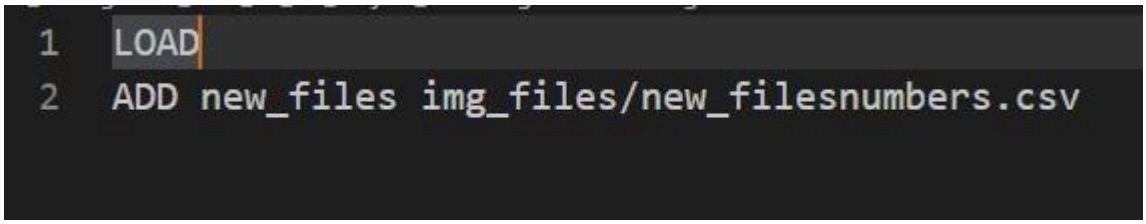


위와 같이 에러코드를 출력하는 것을 확인할 수 있다.

다음은 ADD 명령어에 관한 결과 사진이다.

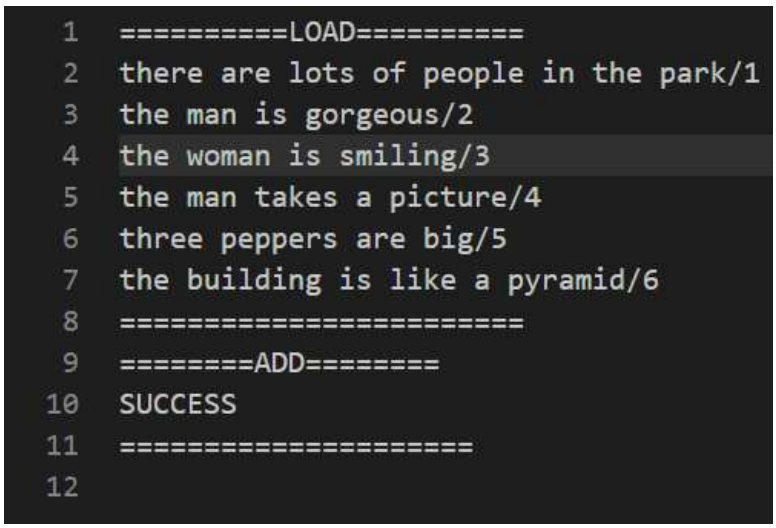


ADD 명령어를 위한 newfilesnumbers.csv 안의 데이터들은 위와 같다.



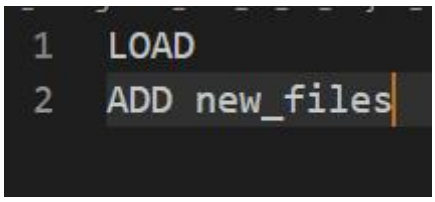
ADD 명령어는 LOAD가 무조건 선행되지 않으면 에러 코드를 출력하므로 LOAD 명령어를 먼저 입력한 것을 확인할 수 있다.

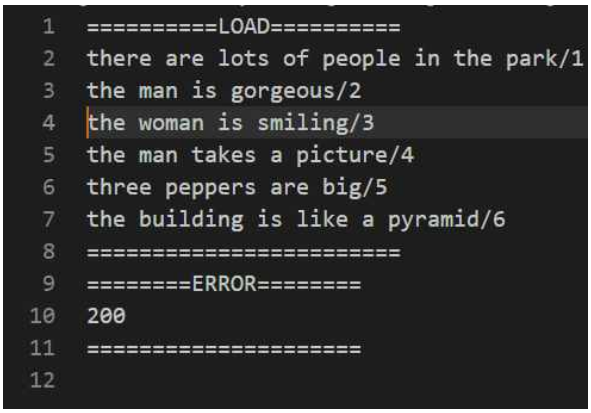
ADD 명령어는 인자를 두개 가지며 첫번째 인자는 directory이름 세번째 인자는 경로를 뜻한다.



위의 사진과 같이 ADD가 성공한 것을 확인할 수 있다.

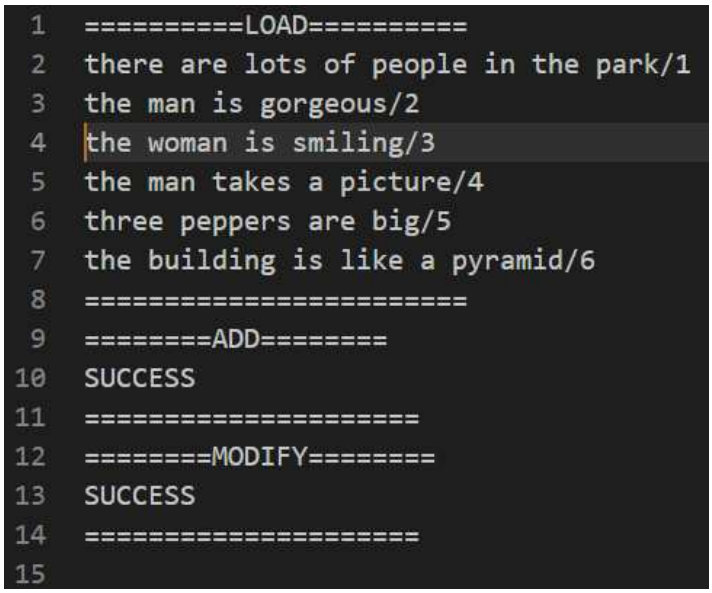
ADD 명령어도 LOAD와 마찬가지로 추가 인자를 받게 되거나 인자가 없으면 에러코드를 출력하게 하였다.



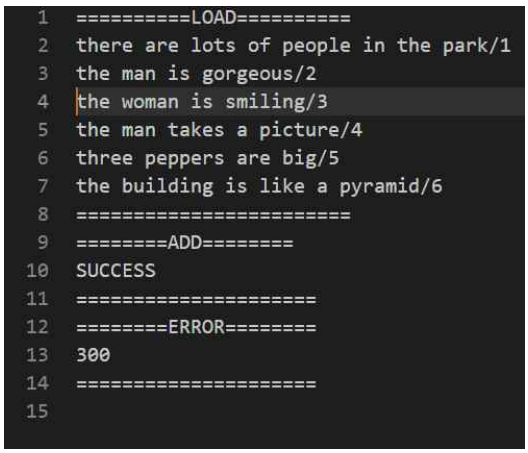


다음은 MODIFY 명령어에 관한 결과 사진이다.

위 사진에서 확인할 수 있듯이 MODIFY는 3개의 인자를 가지며 첫번째 인자는 바꿔야 할 노드의 Directory명, 2번째 인자는 해당 노드의 파일 이름, 3번째 인자는 고유번호를 입력 받는다.
위 사진에서는 cow라는 파일 이름을 가진 노드의 고유번호를 300으로 바꾸라는 명령어로 해석할 수 있다.

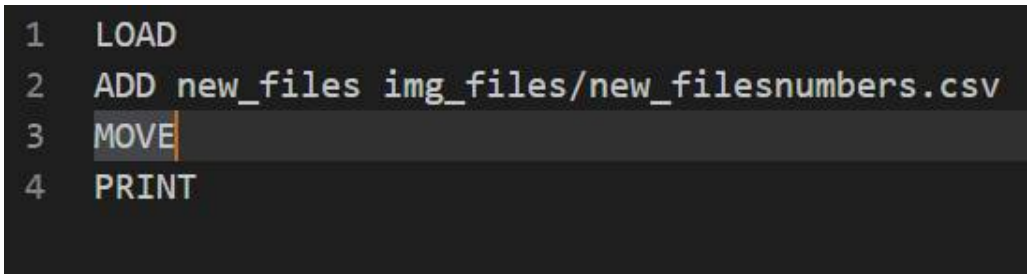


위 사진에서 MODIFY가 성공했음을 확인할 수 있다.
또한 MODIFY명령어는 다른 노드에서 입력 받은 고유번호를 가지고 있다면 에러 코드를 출력하게 한다. 고유번호를 1로 바꾸라고 한다면 이미 filesnumber.csv에 고유번호가 1인 노드가 존재하므로 에러코드가 떠야 한다.

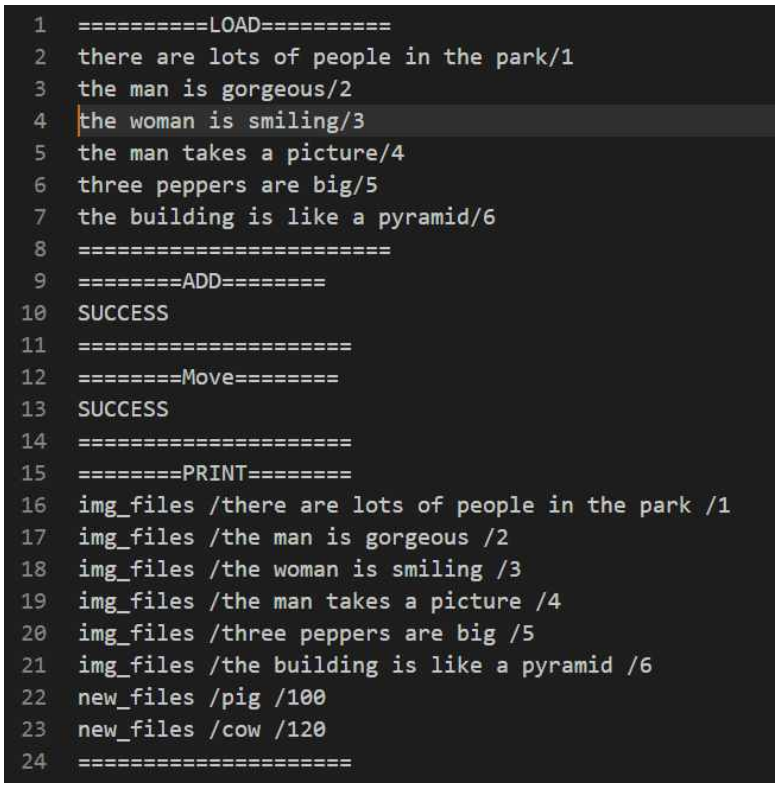


정상적으로 뜨는 것을 확인 할 수 있다.

다음은 MOVE 명령어에 관한 결과 사진이다. MOVE가 제대로 실행되었는지 확인하기 위해서 PRINT 명령어도 같이 사용하였다.



위의 사진은 Command.txt에 있는 명령어창이다. LOAD ADD를 통해 Linked List를 만들었고 이를 MOVE 명령어를 통해 BST로 이동시킨 후 PRINT하였다.



MOVE를 통해 LOAD된 데이터와 ADD를 통한 데이터가 BST로 이동된 것을 확인할 수 있다.

MOVE 명령어는 Linked List가 존재하지 않는다면 에러 코드를 출력한다.

```
1  MOVE
2  PRINT
```

위의 사진은 Command.txt에서 명령어를 입력한 사진이다.

```
1  ========ERROR========
2  400
3  ======================
4  ========ERROR========
5  500
6  ======================
7
```

Linked List가 존재하지 않으므로 에러코드를 각각 출력하는 모습을 확인할 수 있다.

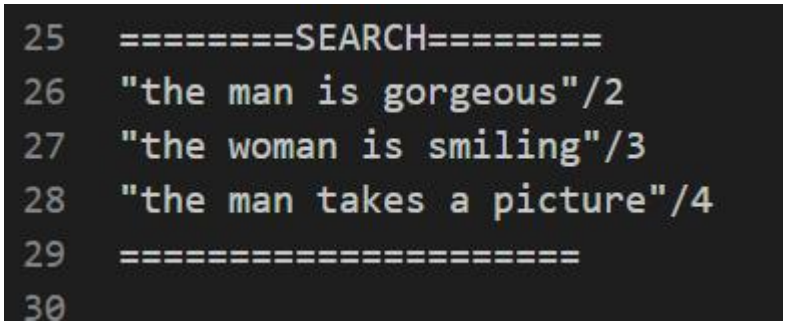
다음은 SEARCH 명령어에 관한 출력 결과이다.

```
1  LOAD
2  ADD new_files img_files/new_filesnumbers.csv
3  MOVE
4  PRINT
5  SEARCH "man"
```

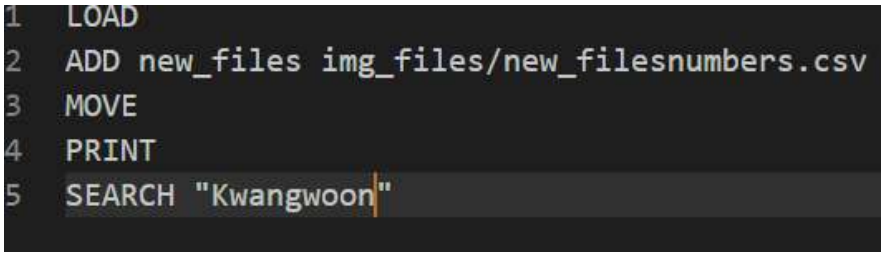
위의 사진은 Command.txt에 있는 명령어들의 사진이다.

```
1  1,there are lots of people in the park.RAW
2  2,the man is gorgeous.RAW
3  3,the woman is smiling.RAW
4  4,the man takes a picture.RAW
5  5,three peppers are big.RAW
6  6,the building is like a pyramid.RAW
```

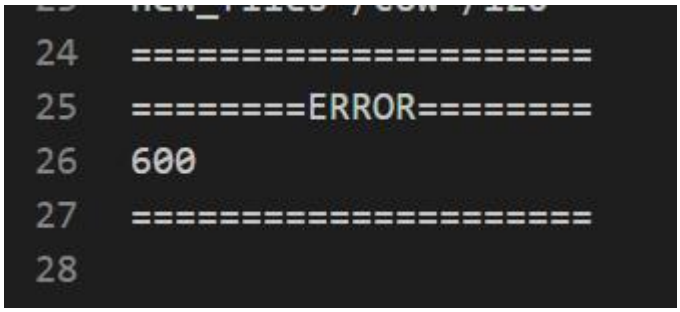
Filesnumbers.csv 파일에는 위와 같은 단어들이 존재한다.



위의 사진에서 보면 man이 들어간 문자열의 데이터를 가지고 있는 노드들의 정보를 출력하는 것을 확인할 수 있다.

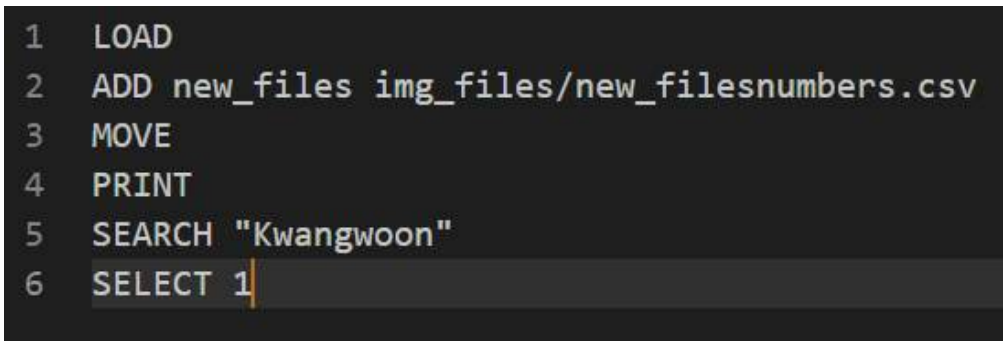


위의 사진과 같이 해당 단어가 존재하지 않을 경우

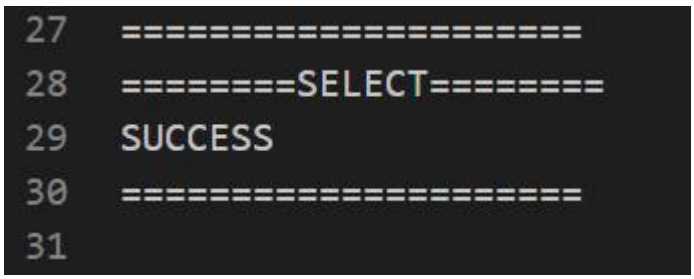


에러 코드를 출력하는 것을 확인할 수 있다.

다음은 SELECT에 관한 출력 결과이다.

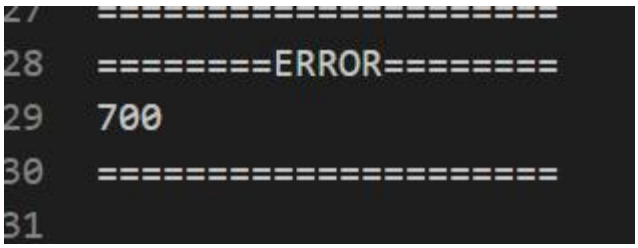


위의 사진과 같이 명령어를 입력하고 filesnumbers.csv에 있는 there are lots of people in the park 의 정보를 select한다.



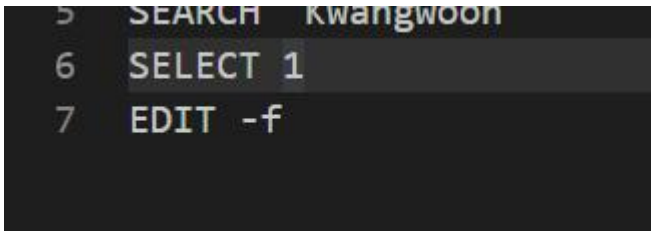
정보가 있다면 위의 사진과 같이 성공했다는 출력이 뜨게 된다.

만약 해당 고유번호를 가진 노드가 존재하지 않는다면



위의 사진과 같이 에러코드를 출력한다.

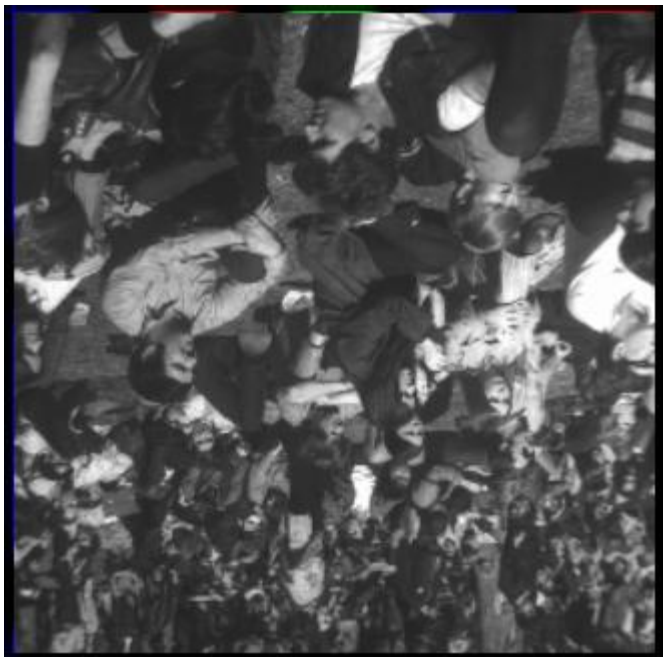
다음으로는 EDIT 명령어에 관한 출력결과이다.



Select한 노드를 EDIT한다고 하면 -f는 점대칭 동작을 수행하도록 명령하는 역할이다.

실행한다면

위의 사진과 같이 Result 폴더에 이미지를 생성하고, 해당 노드의 정보 중 입력 받은 고유번호를 가지는 파일 이름을 수정한다.



위의 사진과 같이 점대칭이 수행된 것을 확인할 수 있다.

다음으로는 -l 명령어이다.

```
6   SELECT 1
7   EDIT -l 100
```

위 명령어를 통해 발기를 100올려주는 역할을 수행하도록 한다.

위 사진에서 볼수 있듯이 밝기가 올라간 모습을 확인할 수 있다.

만약 밝기 최대치인 255를 넘어선다면 밝기를 255로 고정시킨다.

다음은 그에 관한 출력 결과이다.

```
6   SELECT 1
7   EDIT -l 255
```

위와 같은 명령어를 실행하게 되면

결과 사진이 전부 하얗게 변하는 것을 확인할 수 있다.

다음으로는 -r 명령어이다.

```
6  SELECT 1
7  EDIT -r
```

위와 같은 명령어를 실행하게 되면 사진의 크기를 4분의 1로 줄여주는 역할을 수행하게 한다.

위의 사진과 같이 원래 크기인 256X256에서 128X128만큼 사진크기가 작아진 것을 확인할 수 있다.

고찰

이 프로젝트는 Linked List와 BST를 이용해 사진 파일 편집 프로그램을 구현하는 것이다.

사진 파일 편집 프로그램을 구현하는 것의 핵심은 Linked List와 BST를 이용해 구현하는 것 같았다. 실제로 구현하는 중 문제가 제일 많은 부분은 Linked List에서 BST로 데이터를 옮겨줄 때가 가장 많았던 것 같다.

Linked List 객체를 선언할 때 각각의 class에서 객체를 선언해 주었는데 이때 이 객체에 있는 정보를 읽어와야 하는데 BST 내에 있는 함수에선 nullptr값만 읽어오는 것이 문제였다.

해당 문제는 한 객체를 다른 class 및 함수에서 공유하지 못하고 서로 다른 객체를 주고받아서 생긴 문제였다. 해당 문제는 manager class안에 있는 생성자에 list와 BST에 관한 객체를 선언하고 함수로 값을 넘길 때에 Linked LIst객체를 넘겨주는 식으로 진행하였더니 알맞은 객체에 대한 정보를 읽어오게 되었다.

또다른 문제는 파일 입출력 할 때 생겼다.

파일 입력은 문제없이 진행됐지만 출력을 할 때에 여러 번 make를 하고 출력을 하면 이전에 있던 출력 결과들이 새롭게 출력되는 결과 이전에 써져 있는 문제가 있었다.

또한 이것뿐만이 문제가 아니라 새로운 함수에서나 class에서 파일 출력을 하게 되면 이전에 있던 출력 결과들이 지워지는 현상도 있었다. 해당 문제는 처음 manager class 생성자에서 처음 파일을 열기 전에 remove함수를 사용해서 해당 파일을 비워주는 식으로 진행하였더니 이전에 써 있었던 출력결과들이 보이지 않게 되었고, 또한 manager class의 생성자에서 파일을 열 때 std::ios::app을 사용하였더니 새로운 함수나 class에서 해당 파일에 출력결과를 출력할 때 이전에 있던 값들이 사라지는 현상이 발생하지 않았다.

또한 맨 처음 프로젝트를 시작할 때에 실제로 이전까지는 모두 한 파일 안에서(main.cpp)에서 작업하였지만 이번 프로젝트를 기점으로 헤더파일 및 cpp파일을 분리하는 것을 보았다.

지금까지 못해본 작업 환경이었기에 알 수 없는 에러가 떠 힘들었던 점이 있었다.

여러 번 구글링 결과 그 이유가 여러 번 헤더파일을 참조해서 생긴 에러라는 것을 깨달았고 각각의 헤더파일에서 선언해야할 헤더파일 그리고 cpp파일에서는 무슨 헤더파일을 추가해야 하며 최종적으로 어떻게 나눠야 할지에 대해 많은 생각을 하였다.

이를 통해 코드를 관리하는 방법에 대해 한 발자국 더 나아갔다고 생각한다. 오히려 이전까지 작업하였던 방식, 즉 main.cpp에서 모든 함수나 class를 정의해 사용하는 방식이 미련하다고 생각이 들었다. 이번 프로젝트를 기점으로 코딩을 하는 능력을 키운 것에 감사하다고 생각한다.