Data Structure Lab. Project #1

과목명	데이터구조실습
담당 교수	공진흥
학과	컴퓨터정보공학부
학번	2021202078
이름	최경정

1. Introduction

본 프로젝트는 binary search tree와 linked-list, 그리고 queue를 이용하여 간단하게 사진 파일 편집 프로그램을 구현하는 것을 목표로 한다. 이 프로그램은 특정 디렉토리에 저장된 사진 파일에 대한 정보가 들어 있는 csv 파일의 정보를 linked list로 저장한다. 그리고 검색의 용이함을 위해 linked list를 binary search tree의 형식으로 변환한 뒤, 사용자가 입력한 이미지 파일을 찾아내어 해당 파일을 읽는다. 이후에 queue를 활용하여 이미지를 변형하는 기능을 명령어에 따라 추가하여 이미지 편집 프로그램을 구현한다. 각 명령어에 대한 설명은 다음과 같다.

1) LOAD

CSV 파일 내에 들어있는 디렉토리명과 사진의 이름, 고유번호에 대한 정보를 읽어오는 명령어이다. 이 3가지의 정보를 읽어온 후 이를 linked list 자료구조에 저장하고, linked list 내에 저장되어있는 사진 순으로 파일이름과 고유 번호를 출력한다. 만약 linked list의 node 개수가 100개가 넘는다면, 먼저 들어왔던 node 순으로 제거 후 새로운 node를 추가하도록 한다. 만약 CSV 파일이 존재하지 않는다면, 에러 코드를 출력한다.

2) ADD

LOAD와 기능이 흡사한 명령어로, 똑같이 CSV 파일의 데이터 정보를 읽어온다. CSV 파일 내에 들어있는 데이터 정보는 디렉토리명과 사진의 이름, 고유번호이며, LOAD와 동일하게 이를 읽어와 linked list 자료구조에 저장한다. 이때, CSV 파일이 들어있는 디렉토리는 새롭게 탐색하도록 하며, 기존 linked list에 노드를 2차원 형식으로 추가하는 식으로 구현한다. 만약 linked list의 node 개수가 100개가 넘는다면, 먼저 들어왔던 node 순으로 제거 후 새로운 node를 추가하도록 한다. 만약 CSV 파일이 존재하지 않거나, 인자가 한 개 이상 존재하지 않거나, 기존 Loaded_LIST가 존재하지 않는 경우에는 에러 코드를 출력한다.

3) MODIFY

기존에 만들었던 Loaded_LIST에 존재하는 노드 데이터 중 고유 번호를 수정하기 위한 명령어이다. 이때, 단순히 고유 번호를 초기화하는 것이 아니라 Loaded_LIST에 존재하고 있는 특정 고유번호 노드를 찾아내어 이를 삭제하고, 새로운 고유번호의 노드를 추가하는 방식으로 수정하도록 한다. 파일 고유번호는 모든 데이터 노드에 대해서 중복이 존재하지 않는다. 만약 인자가 한 개 이상 존재하지 않거나, 파일 이름에 대한 노드가 존재하지 않거나, 고유 번호가 중복되어 할당될 경우에는 에러 코드를 출력한다.

4) MOVE

Loaded_LIST의 데이터들을 Database_BST로 옮기는 명령어이다. 여기서 Loaded_LIST가 2차원 linked list로 되어 있다면, Datavase_BST는 binary search tree의 형태를 하고 있다. BST의 노드는 사진 이름과 고유 번호 데이터로 구성되어 있으며, 고유번호에 따라 binay search tree의 형식을 맞춰 정렬한다. 만약 BST의 노드 개수가 300개를 넘어갈 경우, 트리 내에서 고유번호가 낮은 순서대로 노드를 삭제하고 새로 추가한다. 만약 Loaded_LISt에 노드가 존재하지 않을 경우에는 에러 코드를 출력한다.

5) PRINT

Database_BST에 저장되어 있는 사진 이름과 고유 번호를 출력해주는 명령어이다. Binary search tree를 중위 순회 방식(In-order)을 사용하여 노드를 순회하면서 출력을 진행한다. 만약 Database_BST가 존재하지 않는 경우에는 에러 코드를 출력한다.

6) SEARCH

특정한 단어가 사진 이름에 포함된 노드를 출력하기 위한 명령어로, 인자에 하나 이상의 단어가 포함되어야 한다. 탐색은 보이어-무어 알고리즘을 이용하여 다음 규칙에 따라 파일 탐색 후 출력을 진행한다.

1. 사진 이름을 기반으로 한 검색을 위해 binary search tree를 Iterative post-order 방식으로 순회하며 queue 자료구조에 각 노드에 존재하는 사진 이름과 고유 번호에 대한 정보를 담는다.
1.1 Iterative post-order 방식 구현 시 반복문이나 재귀함수의 활용을 금지한다.
2. 모든 노드의 정보가 큐에 저장되었다면, 순차적으로 pop을 진행하여 사용자가 검색한 단어가 사진 이름 중 포함되어 있는지를 확인한다.
3. 만약 포함되어 있다면, 해당 노드의 사진 이름 전체와 고유번호를 출력한다.
4. 한 번 검색이 완료된 이후의 큐는 반드시 비게 된다.

7) SELECT

EDIT 명령어 구현을 위해 Database_BST에서 파일의 고유번호를 기반으로 하여 전위 순회 방식(pre-order)으로 사진의 경로를 찾아내어 사진을 불러오는 명령어이다. 만약, 인자가 한 개 이상 부족하거나 파일에 입력한 고유 번호가 존재하는 노드가 없다면 에러 코드를 출력한다.

8) EDIT

SELECT 명령어를 이용하여 불러온 이미지를 편집하는 명령어이다. 주어진 코드를 통해 사진 파일을 노드에서 읽어온 뒤 수정하고 저장한다. 인자로는 세 가지의 옵션 '-f', '-l', '-r'이 존재한다. '-f' 옵션은 이미지를 점대칭으로 전환하고, '-l' 옵션은 이미지의 밝기를 조절하며, '-r'은 이미지의 크기를 조절한다. 이때 '-l' 옵션은 조정할 밝기에 대한 추가 정보를 인자로 받아야 한다. 이를 통해 편집된 이미지를 "Result" 디렉토리에 저장한다. 만약 인자가 한 개 이상 부족할 경우, 에러 코드를 출력한다.

1. 이미지의 점대칭(f)

각 이미지의 픽셀을 stack 자료구조에 순서대로 push(입력)하고, stack 자료구조 특성에 따라 pop하며 이미지를 점대칭으로 뒤집는다. 여기서 각 픽셀에 대한 데이터 할당은 반드시 처음 push한 픽셀부터 재할당하도록 한다. 작업을 마치면, 이미지 파일을 "파일이름_filipped"로 저장한다.

2. 이미지의 밝기 조정

각 이미지의 픽셀을 stack 자료구조에 순서대로 push(입력)하고, stack 자료구조 특성에 따라 pop하며 각 픽셀에 특정 값을 더해 이미지의 밝기를 조절한다. 단, 이미 밝기가 최대인 픽셀에 대해서는 최대 값인 255를 할당하도록 한다. 만약 특정 값을 다 더했을 때 최대 밝기가 넘어갈 경우, 더할 수 있는 만큼만 더한다. 작업을 마치면, 이미지 파일을 “파일이름_adjusted”로

저장한다.

3. 이미지의 크기 조정

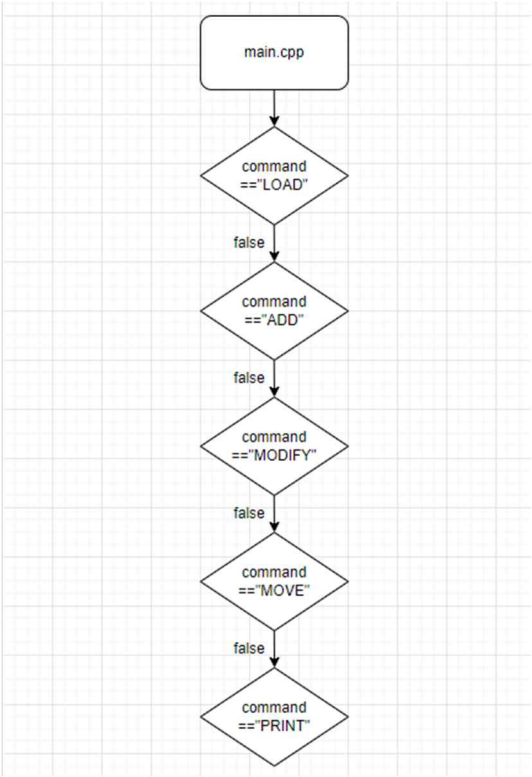
입력된 이미지의 크기를 4분의 1로 조정하는 동작을 진행하도록 한다. 인접해 있는 4개 셀의 평균값을 구해내어 셀에 다시 입력하는 방식으로 이를 구현한다. 이미지 파일을 "파일이름_resized"으로 저장한다.

9) EXIT

프로그램 상의 모든 메모리를 할당 해제하고, 프로그램을 종료한다.

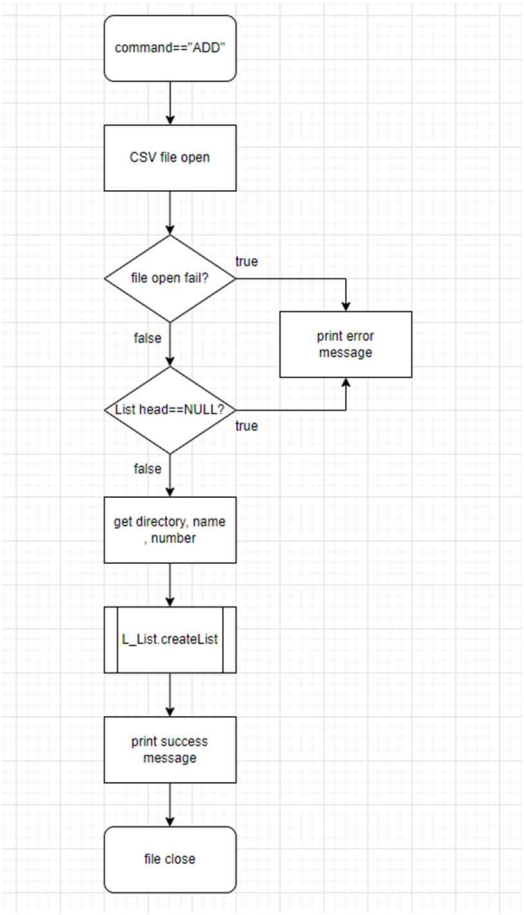
2. Flowchart

▼ main.cpp

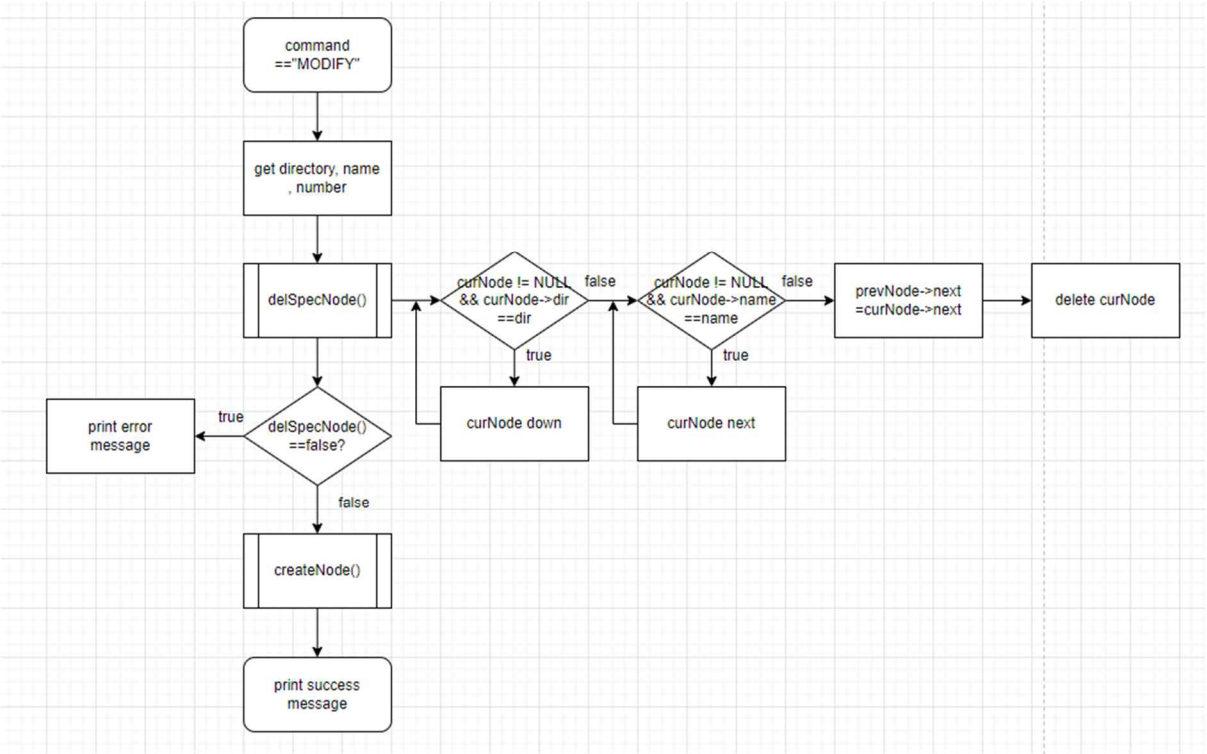


▼ LOAD

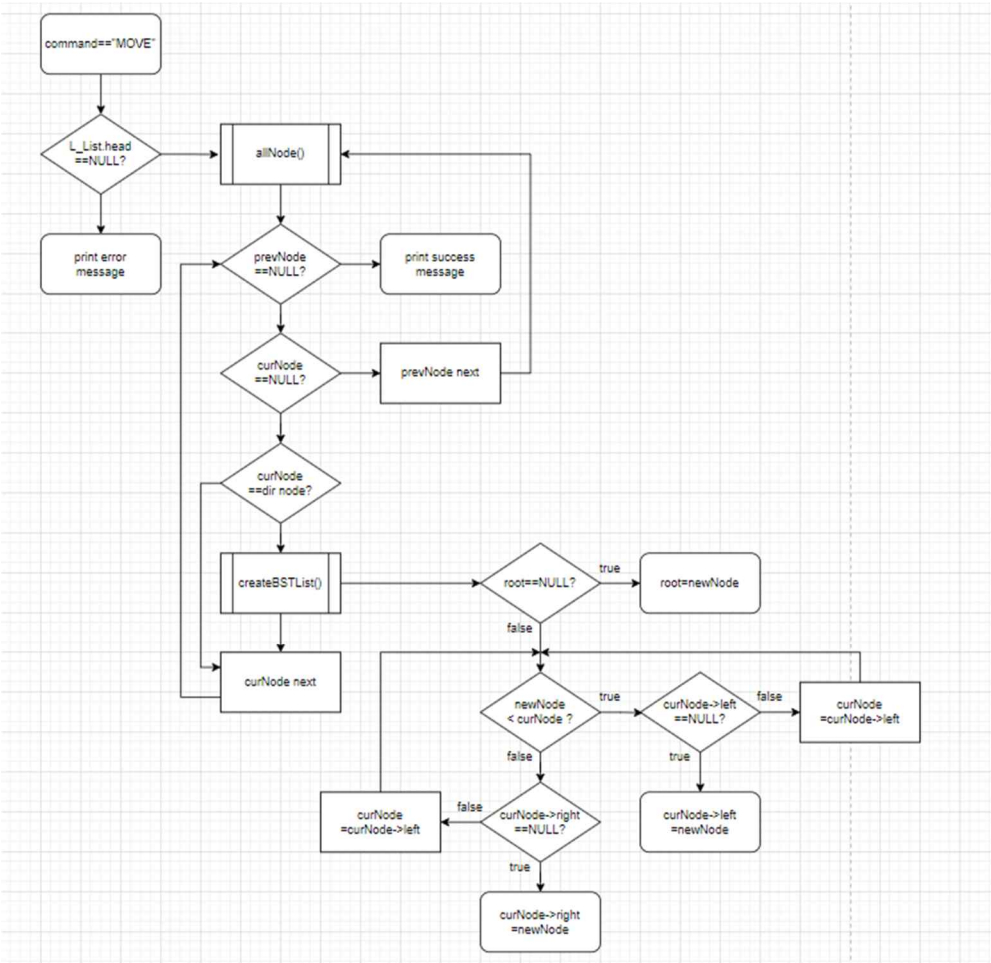
▼ ADD


command=="ADD"
CSV file open
file open fail?
true
print error message
false
List head==NULL?
true
false
get directory, name , number
L_List.createList
print success message
file close

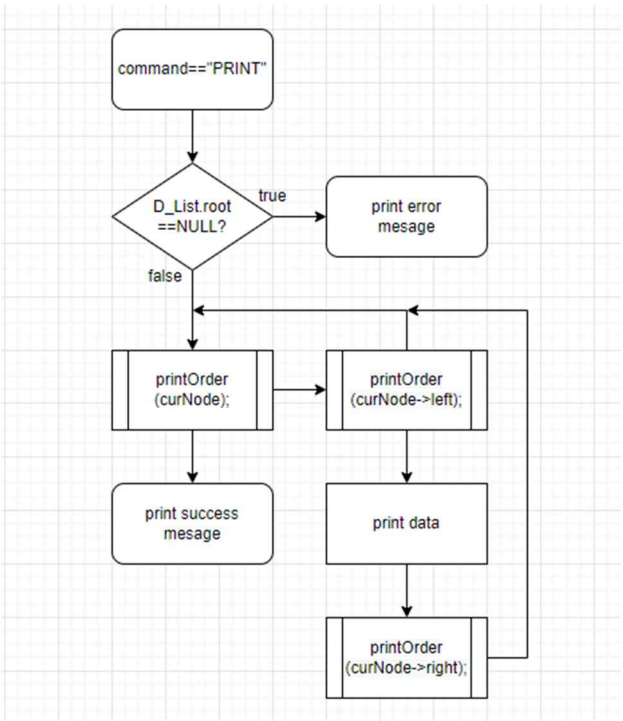
▼ MODIFY


command =="MODIFY"
get directory, name , number
delSpecNode()
curNode != NULL && curNode->dir ==dir
false
curNode != NULL && curNode->name ==name
false
prevNode->next =curNode->next
delete curNode
true
true
curNode down
curNode next
print error message
true
delSpecNode() ==false?
false
createNode()
print success message

▼ MOVE


command=="MOVE"
L_List.head ==NULL?
allNode()
print error message
prevNode ==NULL?
print success message
curNode ==NULL?
prevNode next
curNode ==dir node?
createBSTList()
root==NULL?
true
root=newNode
false
curNode next
newNode < curNode ?
true
curNode->left ==NULL?
false
curNode =curNode->left
false
true
curNode =curNode->left
false
curNode->right ==NULL?
curNode->left =newNode
true
curNode->right =newNode

▼ PRINT


command=="PRINT"
D_List.root ==NULL?
true
print error mesage
false
printOrder (curNode);
printOrder (curNode->left);
print success mesage
print data
printOrder (curNode->right);

3. Algorithm

먼저 Loaded_LIST와 Database_BST를 인스턴스화 해서 만들어준다. 그리고 "command.txt"파일과 "log.txt"파일을 open해준 뒤 command에 명령어를 입력받는다.

1) LOAD

"img_files/filesnumbers.csv" 파일을 open한 뒤, is_open 함수를 통해 파일이 제대로 열렸는지를 확인한다. 그리고 만약 제대로 열렸지 않았다면 error 메세지를 출력한다. 그리고 string 변수 str에 한 줄씩 csv 파일을 읽어온다. 그리고 먼저 str에서 ',' 단위로 문자열을 파싱한 뒤 num에 넣는데, 이때 글자의 깨짐 방지를 위해 제일 앞의 세 글자는 지워지도록 설정한다. 그리고 string형으로 받은 숫자를 c_str 함수를 통해 char형으로 바꿔주고, char형으로 바뀐 문자열을 atoi 함수를 통해 int형으로 변환해준다. 그리고 그 다음 남은 문자열을 name으로 받아 주고, 잘라진 변수들을 createList 함수로 보내준다. 그리고 변수들을 출력하고 파일을 닫으면서 명령어를 종료한다.

1.1) createList

Linked list를 만들어 주는 함수이다. 여기서 우선 head가 NULL인 경우, 즉 아직 기존 링크드리스트가 존재하지 않는 경우 디렉토리 노드를 따로 하나 만들어 이를 head로 설정하고, 새로 만들어진 노드를 디렉토리 노드의 next로 설정한다.

curNode의 디렉토리명과 prevNode의 디렉토리명이 일치하지 않는 경우, curNode의 down이 비어 있는 상태가 아니라면 curNode를 하나 down 노드로 옮기고, prevNode를 curNode의 이전 상태 노드로 지정한다.. 그러나 curNode의 down이 비어 있다면, 반복문을 빠져나온다. 그리고 디렉토리를 하나 새로 만들어 가장 마지막 디렉토리 노드의 down에 연결하고, 새 노드를 디렉토리 노드의 next로 연결한다.

curNode을 그리고 Next로 계속 이동시켜 링크드리스트의 가장 끝에 오도록 지정한다. 그리고 curNode의 next에 새 노드를 연결한다.

2) ADD

전체적인 흐름은 LOAD와 매우 비슷하다. 그러나 이번에는 디렉토리명과 csv파일의 이름을 "command.txt"창에서 따로 string 형으로 받는다. 그리고 LOAD와 똑같은 작업을 통해 변수에 값을 저장하고, 새로운 디렉토리 명으로 createList함수를 연다. 그리고 성공 시 성공 메세지를 log.txt 파일에 띄운다.

3) MODIFY

우선 수정할 파일명을 string형으로 받은 뒤, 문자열을 파싱하여 변수에 저장한다. 이 작업은 LOAD, ADD와 거의 동일하다. 그리고 delSpecNode를 실행하여 입력받은 파일명 노드를 삭제한다. 여기서 delSpecNode는 특정 노드를 삭제하는 함수로, 링크드 리스트를 순회하며 전달받은 디렉토리명과 파일명이 모두 일치하는 노드를 발견할 경우 그 노드를 삭제한다. 이때 만약 동일한 파일명이 없을 경우, 에러 메세지를 띄운다. 그리고 함수 createNode를 실행하여 새로운 고유번호의 노드를 기존 링크드리스트의 가장 끝에 추가하고, MODIFY 성공 메세지를

출력한다.

4) MOVE
함수 allNode를 실행하여 함수를 순환하며 노드의 정보로 BST를 만든다.

5) PRINT
중위 순회의 방식으로 도는 printOrder 함수를 이용하여 BST의 정보들을 출력한다.

4. Result Screen

```
=======LOAD=======
img_files / there are lots of people in the park.RAW
 / 100
img_files / the man is gorgeous.RAW
 / 111
img_files / the woman is smiling.RAW
 / 200
img_files / the man takes a picture.RAW
 / 222
img_files / three peppers are big.RAW
 / 300
img_files / the building is like a pyramid.RAW
 / 333
img_files / river is under the bridge.RAW
 / 400
img_files / The house has windows and doors.RAW
 / 444
img_files / the boat sails a big river .RAW
 / 500
img_files / The celebrity posed for the picture.RAW
 / 555
img_files / there are lots of ariplane.RAW
 / 600
img_files / The woman is looking at the man .RAW
 / 666
img_files / lena is famous person.RAW
 / 700
img_files / the woman is wearing a hat.RAW
```

```
=======ADD=======
SUCCESS
====================
```

```
=======MOVE=======
SUCCESS
====================
```

5. Consideration

우선 첫 번째로 생긴 문제는 문자열을 정수형으로 바꿀 때 생겼었다. Csv 파일에서 문자열을 받아와 ',' 기준으로 파싱하면 name, dir, num은 모두 구할 수 있는데, 문제는 num도 문자열 형태로 저장된다는 것이었다. 처음에는 아스키코드므로 48을 뺄까 생각을 했는데, 여러 편의성을 위해 char 배열로 받는 것보다는 string으로 받는 것이 나을 것 같아서 모든 인자를 string으로 받았다. 따라서 string to int를 하기 위해 여러 함수를 찾아봤는데, 어떤 이유에서인지 stoi 함수가 실행되지 않았다. 따라서 atoi 함수를 실행하기 위해 우선 string의 주소값을 char의 포인터에 대입하여 string to char을 진행했고, 그 다음 atoi 함수를 실행해 char to int를 진행했다. 그러나 출력을 하고 보니 또 다른 문제가 생겼는데, 바로 제일 처음 세 글자가 깨져서 들어가는 것이었다. 따라서 처음에는 인코딩 방식을 UTF-8에서 ANSI로 바꾸는 형식으로 이를 해결하려고 했는데, 인코딩이 아니라 코드 자체에서 이 문제를 해결하면 좋겠다는 생각이 들었다. 그러던 중, erase 함수를 알게 되어 이를 사용하여 첫 세글자를 무시해 첫 세 글자가 깨지는 것을 방지할 수 있었다.

그리고 생긴 또 다른 문제는, segmentation fault에 관한 문제였다. Load를 진행할 때 segmentation fault라는 문구가 뜨면서 컴파일 자체가 아예 안되는 경우가 생겼었다. 조사를 해보니, 이는 잘못된 방식으로 메모리에 접근하려고 했을 때 뜨는 문구라는 것을 알게 되었으나 몇번째 줄에서 어떤 오류가 나는지조차 알 수 없어서 답답했었다. 코드를 한 줄 한 줄 씩 다 읽어가며 오류를 찾아낸 결과, if(head==NULL)가 아닌 if(head=NULL)이라고 적혀서 난 문제라는 것을 알게 되었다. 이번에는 코드가 길지 않아서 쉽게 찾아낼 수 있었으나, 2차, 3차 프로젝트를 진행할 때도 이와 같은 오류가 날 것을 대비하여 비주얼스튜디오 코드에 대해서 더 공부를 해봐야 할 것 같다.

또한 헤더 파일(.h)와 cpp 파일을 나눌 때 include를 어떻게 해야 하는지에 대한 문제가 있었다. 처음에는 Load_List에서 Load_List_Node를 참조할 수 없다는 식의 오류가 나서, 모든 헤더 파일과 cpp 파일에 자기 자신을 제외한 헤더 파일들을 모두 include했다. 그래도 문제가 해결이 나지 않자, 조사를 한 결과 우선 첫 번째 문제를 발견했다. 내가 따로 만든 헤더 파일을 include할 경우에는 "<>"가 아닌 " "" "으로 include시켜야 한다는 점이었다. 그러나 기호를 바꿨음에도 불구하고 오류는 계속 발견되었다. 또 고민하다가 알게 된 점은 헤더 파일에 무심코 첨부했던 "#pragma once"라는 문장이었다. 이를 포함하지 않으면 오류가 났는데, 헤더파일이 여러 번 include되지 않게 도와준다는 의미라는 것을 처음 알게 되었다. 또한 헤더파일은 상속과 그 기능이 비슷하기 때문에, 서로 헤더파일을 참조하는 것은 오류를 불러일으킨다는 점을 다시 한 번 알게 되었다. 예를 들자면 Loaded_List에서 Loaded_List_Node는 참조할 수 있지만(상속을 받는 것과 비슷한 느낌) 그 반대의 경우에는 쓸모도 없고, 오류만 발생시키는 것이었다.

마지막으로는 헤더 파일과 cpp 파일을 나누는 것에 대한 장단점을 생각해 보았다. 나는 맨 처음에는 main.cpp에 전부 모든 클래스와 함수를 넣고 디자인하다가, 중간에 클래스들을 나누고, 헤더 파일과 cpp 파일을 나눴었다. 나눈 이유는 , main 내에 코드가 너무 길어지다 보니까 원하는 코드를 찾기가 불편한 지경에 이르렀기 때문이었다. 그러나 코드들을 나누고 보니, 여러 군데에 코드들이 있어서 내가 원하는 코드가 어느 파일에 있었는지도 헷갈렸고, 다같이 한 cpp 파일 내에서 돌렸던 것과는 달리 파일이 분리되다 보니까 이쪽 파일에서 다른 파일로 넘어갈 때 인자 전달 등 신경써야할 요소가 더 많아졌던 것 같다. 그러한 부분에 대해서는 언제쯤 파일을

분리하면 좋을지 고민해보는 것도 나쁘지 않을 것 같다.