DS Project1

Word book

이름	정무영(팀장)	오재원
학과	컴퓨터공학과	컴퓨터공학과
학번	2013722021	2013722034

Introduction

이번 프로젝트는 단어장 프로그램을 만드는 것이다. 이 프로그램은 단어와 뜻이 저장되어 있는 txt파일을 불러와 거기 있는 단어들을 외우고 암기 여부를 테스트까지 하는 프로그램이다. 프로젝트 구성을 크게 3개로 나누면 Queue, Binary search tree(BST), Circular linked list(CLL) 이렇게 세 가지 class로 나눌 수 있으며 이들을 전부 컨트롤 하는 Manager class가 있다.

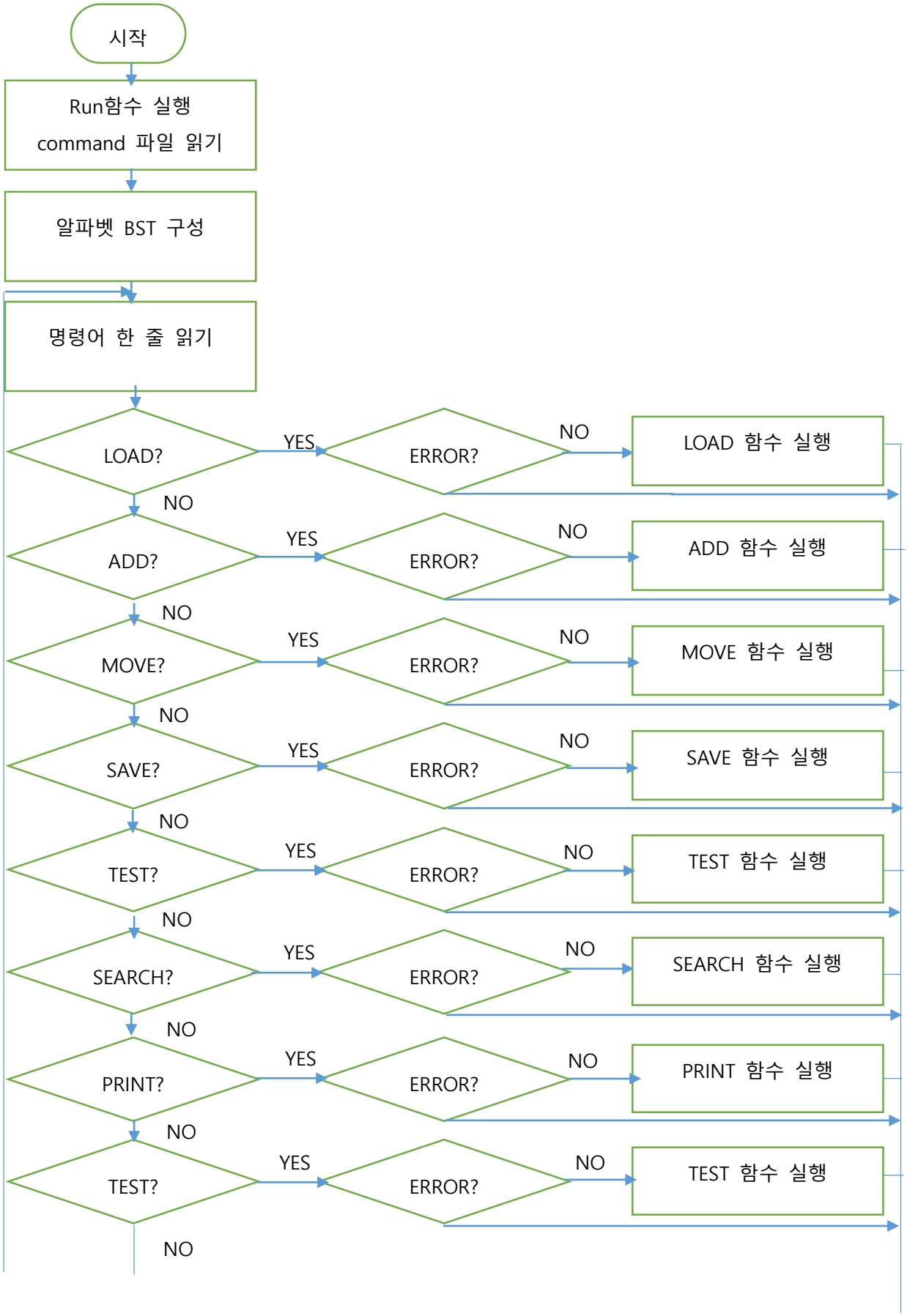
Queue에서는 ADD, LOAD 명령어에 사용 될 push, MOVE 명령어에 사용 될 pop기능이 필요하며 PRINT명령어에 따라 단어 목록을 출력 해 줄 출력 기능도 필요하다.

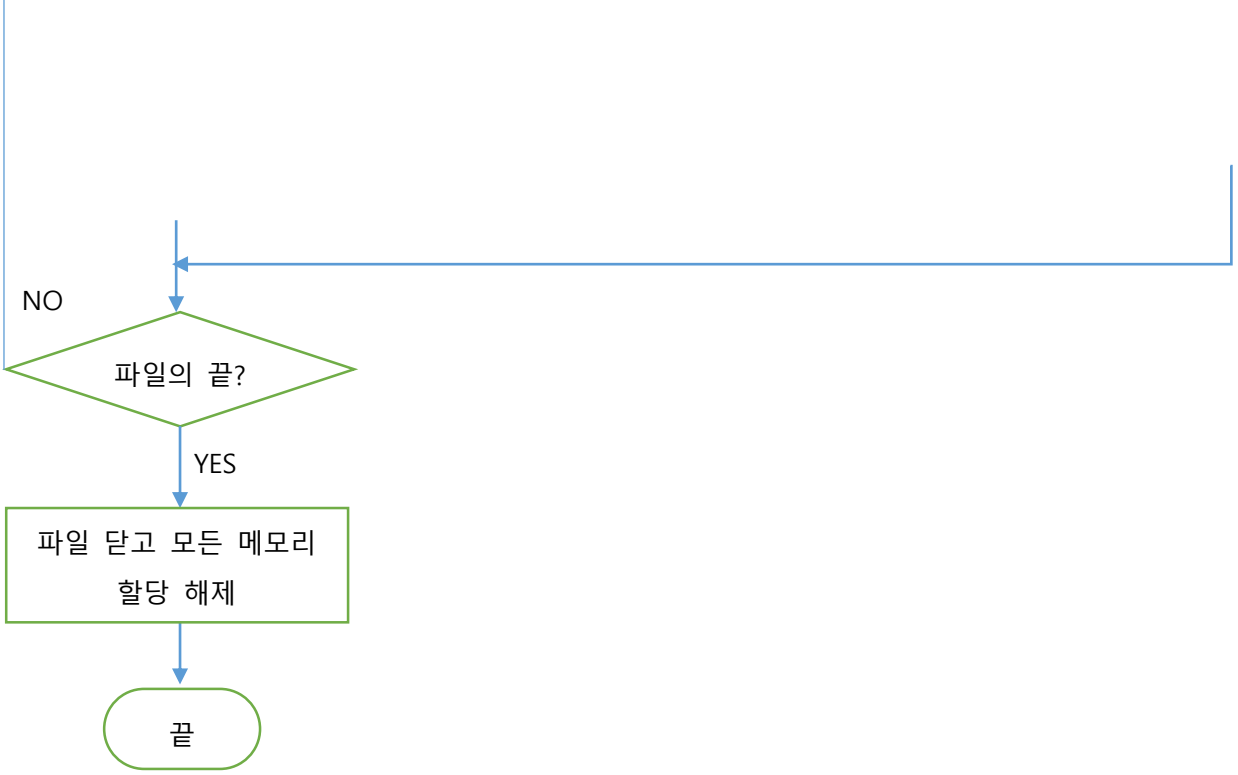
BST에서는 MOVE, LOAD 명령어에 사용 될 insert, TEST 명령어에 사용 될 delete기능이 필요하며 이때 insert는 BST구성에 맞도록 삽입 해야 하고 delete는 삭제 할 노드의 연결 상태에 따라 분리해서 delete기능을 수행 해야 한다. Delete 기능이라고 메모리를 delete해서 할당 해제하는 것이 아닌 CLL로 이동 되도록 해야 한다. 사용자가 원하는 순회 방식대로 출력해주는 PRINT명령어에 따라 단어들을 후위, 중위, 전위, 레벨순회 방식으로 출력해 줄 수 있는 출력 기능도 필요하다.

CLL에서는 TEST, LOAD 명령어에 사용 될 insert기능이 필요하며 따로 단어를 삭제할 일이 없기 때문에 그에 관련된 기능은 없어도 된다. CLL도 PRINT명령어에 따라 단어 목록을 출력 해 줄 출력 기능이 필요하다.

프로젝트가 시작되자 마자 알파벳 BST는 먼저 구성되어야 하며 전체적으로 단어들이 하나도 중복돼선 안 된다. SEARCH 명령어는 입력한 단어를 찾아서 뜻까지 출력 해주는 기능을 하고 UPDATE 명령어는 단어의 뜻을 수정 해 주는 기능을 한다. TEST, SEARCH, UPDATE 명령어에 단어를 입력 받을 땐 단어의 대소문자를 구분 하지 않고 입력 받도록 해야 한다. SEARCH, UPDATE 명령어는 프로젝트 전체를 대상으로 하는 명령어이므로 프로젝트 전체를 모두 확인 해야 한다. SAVE 명령어로 단어를 SAVE 할 땐 단어 순서 그대로 저장되어야 하며 LOAD할 때 이전 단어 순서 그대로 불러와져야 한다. EXIT명령어로 프로그램을 종료 할 땐 파일 입출력을 위해 열었던 파일 입출력 stream을 모두 닫아 주어야 하며 할당된 메모리들을 전부 할당 해제 해줘야 한다. 이렇게 여러 명령어들을 입력하는 과정에서 발생한 출력들은 모두 log.txt에 계속 저장 되어야 한다.

Flow chart


시작
Run함수 실행
command 파일 읽기
알파벳 BST 구성
명령어 한 줄 읽기
LOAD?
YES
ERROR?
NO
LOAD 함수 실행
NO
ADD?
YES
ERROR?
NO
ADD 함수 실행
NO
MOVE?
YES
ERROR?
NO
MOVE 함수 실행
NO
SAVE?
YES
ERROR?
NO
SAVE 함수 실행
NO
TEST?
YES
ERROR?
NO
TEST 함수 실행
NO
SEARCH?
YES
ERROR?
NO
SEARCH 함수 실행
NO
PRINT?
YES
ERROR?
NO
PRINT 함수 실행
NO
TEST?
YES
ERROR?
NO
TEST 함수 실행
NO



Algorithm

1. Manager

   - Run : 매개변수로 문자열을 받은 후 그 문자열의 이름을 가진 텍스트 파일을 열어 속에 있는 명령어들을 space를 기준으로 잘라서 명령어들에 맞는 행동을 하게 한다. 일단 프로젝트가 실행 되자 마자 알파벳 BST가 생성 되어야 하므로 함수가 실행 되자 마자 생성 해 준다. 명령어를 가진 파일이 열리지 않았으면 바로 프로그램을 종료한다. 명령어를 한 줄 씩 읽으면서 형식에 맞지 않으면 에러 코드를 출력해 주거나 무시해 버린다. 계속 명령어를 읽다가 더 이상 읽을 명령어가 없으면 함수를 종료한다.

   - LOAD : 불러올 파일 정보가 없거나 이미 프로그램상에 단어들이 존재하면 에러를 출력 한다. 세가지 파일을 읽어서 각각 단어 정보가 들어갈 WordNode의 메모리를 할당 한 후 정보를 넣고 Queue의 push, BST의 insert, CLL의 insert함수를 통해서 알맞게 넣어 준다.

   - ADD : 불러올 파일이 없으면 에러 코드를 출력 한다. "word.txt"파일을 열고 한 줄씩 읽어서 단어의 정보를 받는다. 받은 정보를 메모리를 새로 할당 한 WordNode에 저장 한 뒤 Queue의 push함수를 이용해서 Queue에 전부 넣어 준다.

   - MOVE : BST에 들어있는 단어 수와 넘겨줄 단어 수를 합쳐서 100이 넘어가거나 입력한 숫자 만큼 Queue에 단어들이 없을 때 에러 코드를 출력 해 준다. Queue의 pop함수를 이용해서 받은 단어를 BST에 insert함수로 넣어 준다.

   - SAVE : 함수의 시작 부분에서 "memorizing.txt"를 초기화 시켜준다. 이는 BST에서 단어를 저장 할 때 이어 쓰기 방식을 사용 하기 때문에 하는 것이다. 그 후 Queue, BST, CLL의

SAVE함수를 이용해서 저장 해 준다. 이 때 모든 SAVE함수들이 false를 반환하면 저장할 단어 정보가 아무것도 없다는 의미 이므로 에러 코드를 반환 해 준다.

- TEST

  BST에서 Search를 이용해서 같은 단어를 찾은 후 해당 단어를 받아서 뜻과 단어가 맞으면 Delete와 CLL의 insert를 이용해서 BST에서 CLL로 이동 시켜준다. 단어가 존재하지 않거나 틀리면 에러 코드를 출력 한다.

- SEARCH

  모든 연결 리스트의 Search함수를 이용해 단어를 찾아서 단어의 정보를 출력 해 준다. 단어가 존재하지 않으면 에러 코드를 출력 해 준다.

- PRINT

  들어오는 입력 옵션에 따라 알맞은 리스트와 출력 방식을 선택해서 해당 리스트의 print 함수를 호출 해 준다.

- UPDATE

  해당 단어를 찾고 모든 리스트에서 Search함수를 이용해서 같은 단어를 찾은 후 그 단어의 뜻을 수정 해 준다. 해당 단어가 없으면 에러 코드를 출력 해 준다.

2. Queue

- Push : 밖에서 이미 단어와 뜻까지 채워진 상태의 Wordnode의 주소 값을 받아서 그것을 Queue에 넣어준다. 넣어 줄 때 Queue가 비어 있는 상태라면 Queue의 Head와 Tail을 모두 node로 설정해준다. 비어 있지 않을 때는 Tail의 Next에 node를 설정해 주고 그 node를 Tail로 설정한다. Node를 한 개 push할 때 마다 Queue의 단어 개수를 증가 시켜 준다.

- Pop : Queue에 들어있는 node를 내보내는 역할을 한다. Head가 비어 있다면 아무것도 Pop하지 않고 하나라도 들어 있으면 Head의 다음 노드를 Head로 설정하고 head였던 node를 반환 해준다. Node를 한 개 Pop하 때 마다 Queue의 단어 개수를 감소 시켜 준다.

- Search : Queue의 head부터 Tail까지 탐색해서 입력한 문자열과 같은 단어가 있으면 출력 해준다.

- Print : Queue의 head부터 Tail까지 돌면서 단어와 뜻을 출력 해 준다.

- Save : Queue의 head부터 Tail까지 돌면서 순서대로 “to_memorize.txt”에 저장 해준다.

3. Binary search tree

- Insert : 밖에서부터 이미 완성된 Wordnode의 주소 값을 받아서 그것을 BST에 넣어 준다. BST에는 Insert가 두가지 종류가 있는데, 알파벳을 넣기 위한 insert는 프로그램이 시작 하자 마자 중위 순회로 출력했을 때 순서대로 출력 되도록 알파벳을 넣어 준다. 두 번째 insert는 단어 들을 넣어 주기 위한 insert이다. 이 것은 먼저 들어갈 단어의 이니셜을 가져와서 같은 알파벳을 찾고 그 속의 단어 BST로 들어가서 BST가 비어 있으면 root로 설정한다. 그렇지 않을 땐 root로부터 하나씩 단어를 비교하면서 사전편찬 순서상으로 앞쪽이면 왼쪽자식 노드로 그렇지 않으면 오른쪽 자식 노드로 가서 단어가 존재하면 계속 비교하고 그렇지 않고 비어 있으면 그 자리에 위치 시키면 된다. 비교하다가 같은 단어를 발견 하면 넣으려던 단어의 메모리를 할당해제 하고 넣지 않는다.

- Print : 다른 연결 리스트와 다르게 BST에서는 재귀함수를 사용한 것과 그렇지 않은 전위, 중위, 후위순회와 재귀함수를 사용하지 않은 레벨순회까지 총 7가지 출력 방식이 있다. 모든 순회는 root가 NULL이라면 바로 종료 된다.

  >R_Preorder

  Root부터 전위 순회는 자기 자신을 가장 먼저 방문 해야 하므로 해당 노드의 출력 명령어를 가장 먼저 코딩 하고 이번 프로그램에서는 왼쪽 자식을 먼저 방문하게 돼 있으므로 왼쪽 자식 노드로 가서 R_Preorder함수를 호출해주고 오른쪽 자식도 마찬가지로 호출해 준다. 이런 식으로 자식을 계속 방문 하다 NULL이 나오면 실타래가 풀리듯이 재귀함수들이 차례로 종료된다.

  >I_Preorder

  재귀함수를 쓸 수 없으므로 Stack을 사용해서 구현한다. root부터 시작해서 자기 자신을 포함 해 왼쪽 자식 노드만 계속 Stack에 넣어 주면서 출력한다. 그러다 더 이상 방문할 왼쪽 자식이 없으면 Stack의 맨 위 항목을 Pop해주고 그 항목의 오른쪽 자식 노드를 방문하고 그 노드로 위의 순서를 반복한다. 계속 실행 하다 더 이상 방문할 노드가 없고 Stack이 전부 비워지면 함수를 종료한다.

  >R_Inorder

  재귀함수를 사용한 전위 순회에서 출력코드의 순서만 바꿔주면 된다. 왼쪽 자식의 재귀함수 호출문장 다음에 출력문장을 넣고 그 뒤에 오른쪽 자식의 재귀함수 호출 문장을 넣어주면 된다. root부터 시작한다. 그러면 해당 노드가 더 이상 방문 할 왼쪽 자식 노드가 없을 때 출력을 하고 그 후 오른쪽 자식 노드를 방문 하게 된다.

  >I_Inorder

  Root부터 시작한다. 1) Driver에 root부터 넣어준 후 스택에 driver를 push하고 driver는 왼

쪽 자식을 가리킨다. '1)'을 driver가 NULL을 가리킬 때까지 반복한다. 그 후 Stack을 살펴봐서 Stack이 비어 있으면 출력을 종료 한다. 그렇지 않으면 2) driver에 스택의 맨 위쪽을 가리키도록 한다. 그리고 스택의 맨 위 항목은 pop을 한다. 그리고 driver가 가리키는 값을 출력 하고 driver는 오른쪽 자식을 가리킨다. 이후 다시 1)로 돌아간다. 위 순서를 계속 반복하다 보면 더 이상 방문할 노드가 없어지고 Stack이 비게 되는데 이때 출력이 끝이 난다.

>R_Postorder

재귀함수를 사용한 순회들의 출력코드 순서만 바꾸면 된다. 왼쪽자식 재귀 함수, 오른쪽 자식 재귀 함수, 출력 코드. 이 순서대로 작성하면 된다. 그러면 해당 노드는 더 이상 방문할 자식 노드가 없을 때 비로소 출력을 하게 돼서 후위 순회가 완성 된다.

>I_Postorder

여기서는 두개의 Stack을 사용한다. Driver에 root를 넣어준다. 1) Stack1, 2에 driver를 push 해주고 driver는 오른쪽 자식을 가리킨다. '1)'을 driver가 NULL을 가리킬 때까지 반복한다. 2) Stack1의 맨 위 항목을 Pop하고 그 항목의 왼쪽 자식을 driver가 가리키게 한다. 이 후 더 이상 방문 할 노드가 없거나 Stack1이 빌 때 까지 '1)반복 -> 2)'를 반복 해준다. 위 반복까지 빠져 나오면 Stack2에 맨 위 항목부터 순서 대로 출력 해 준다. 함수를 저렇게 짠 이유는 후위 순회의 방문 순서가 좌, 우, 본인 이므로 Stack2에는 본인, 우, 좌 순서로 들어 갈 수 있도록 하기 위해서다.

>I_Levelorder

레벨 순회는 Stack대신 Queue를 사용해서 구현한다. 처음엔 driver가 root를 가리킨다. 1) driver가 가리키고 있는 곳을 출력한다 그 후 왼쪽자식이 있으면 Queue에 왼쪽 자식을 push한다 그리고 오른쪽 자식이 있으면 오른쪽 자식을 push한다. 그 후 Queue가 비어 있으면 반복을 종료한다. 그렇지 않으면 driver에 Queue의 head를 넣고 Queue의 head는 비운다. '1)'을 계속 반복한다. 레벨 순회는 위쪽인 root부터 순서대로 출력 하기 때문에 Stack대신 Queue를 사용하여 구현했다.

- Search

매개변수로 전달받은 단어와 같은 단어가 있는지 찾아 보고 있다면 해당 단어를 담고있는 노드를 반환 해준다. BST에서 탐색 하는 것이므로 비교 후 사전 순서에서 앞 쪽이면 왼쪽 자식으로 가서 비교 하고 뒤 쪽이라면 오른쪽 자식으로 가서 비교한다. 같은 단어를 찾으면 반환 해 주고 NULL을 만나면 같은 단어가 없다는 의미 이므로 NULL을 반환 해 준다.

- Save

저장 후 불러 올 때 이전 트리와 같은 모양이여야 하므로 preorder를 이용해서 저장 해준다.

- Delete

일단 Search와 같은 알고리즘으로 매개변수로 온 단어와 같은 단어가 있는지 찾아 본 후 없다면 NULL을 반환하고 있다면 단어의 위치에 따라 조금씩 다르 알고리즘을 적용한다.

먼저 자식 노드가 없는 경우, 삭제될 노드가 root라면 root에 NULL을 넣고 해당 노드를 반환한다. Root가 아니고 부모의 왼쪽 자식 노드라면 부모의 왼쪽 자식 노드를 NULL로 설정하고 삭제될 노드를 반환 한다. 오른쪽 자식 노드라면 오른쪽 자식 노드를 NULL로 하고 삭제될 노드를 반환 한다.

삭제될 노드가 왼쪽이 없고 오른쪽 자식만 있을 경우이다. 해당 노드가 root면 자식 노드를 root으로 설정해 주고 해당 노드를 반환 한다. 해당 노드가 부모의 왼쪽 자식 노드 라면 부모의 왼쪽 자식을 노드의 자식으로 설정 하고 해당 노드를 반환 한다. 해당 노드가 부모의 오른쪽 자식 노드 라면 부모의 오른쪽 자식을 노드의 자식으로 설정 해 주고 해당 노드를 반환한다.

삭제될 노드가 오른쪽 자식이 없고 왼쪽 자식만 있을 경우는 위의 알고리즘과 같다.

삭제될 노드가 왼쪽, 오른쪽에 모두 자식이 있을 경우이다. 이 프로그램은 삭제될 노드를 해당 노드의 왼쪽 bst에서 가장 큰 노드로 대체한다. 그럼 bst의 규칙이 유지된다. 일단 자식 노드와의 관계를 먼저 비교 한다. 대체될 노드가 삭제될 노드의 자식 노드라면 삭제될 노드의 오른쪽 자식을 대체될 노드의 오른쪽 자식으로 설정 해 준다. 그 밖의 경우엔 대체할 노드의 부모 노드의 오른쪽 자식을 대체할 노드의 왼쪽 자식으로 설정 해 주고 삭제될 노드의 좌, 우 자식 노드를 대체할 노드의 좌, 우 자식 노드로 설정 해준다.

이제 삭제될 노드의 부모 관계를 따져본다. 노드가 root라면 root를 대체할 노드로 설정 해 준다. 삭제될 노드가 부모의 왼쪽 자식 이라면 부모의 왼쪽 자식 노드를 대체할 노드로 설정 해 주고 오른쪽이라면 오른쪽 자식으로 설정 해 준다.

부모 자식 관계를 전부 따져서 연결을 해준 후엔 해당 노드의 좌, 우 연결을 모두 NULL로 설정하고 해당 bst의 단어 개수를 1감소 시키고 노드를 반환 시킨다.

4. Circular linked list

- Insert

외부에서 완성된 WordNode를 매개변수로 받아서 CLL에 넣어준다. 이 때 이미 있는 단어라면 delete 해버리고 넣지 않는다. CLL이 비어 있을 땐 head, tail 두개가 해당 노드를 가리키게 한다. Tail의 next는 항상 head를 가리키게 한다. 이미 노드가 들어있을 땐 새로운 노드를 Tail의 next로 설정하고 해당 노드의 next를 head로 해당 노드를 tail로 설정 해 준

다.

- Search

  Head부터 Tail까지 순서대로 탐색 후 매개변수로 받은 단어와 같은 단어가 있으면 그 WordNode를 반환 해 주고 그렇지 않으면 NULL을 반환 해 준다.

- Print

  Head부터 Tail까지 순서대로 출력 해 준다.

- Save

  Head부터 Tail까지 "memorized.txt"에 저장 해 준다.

Result screen

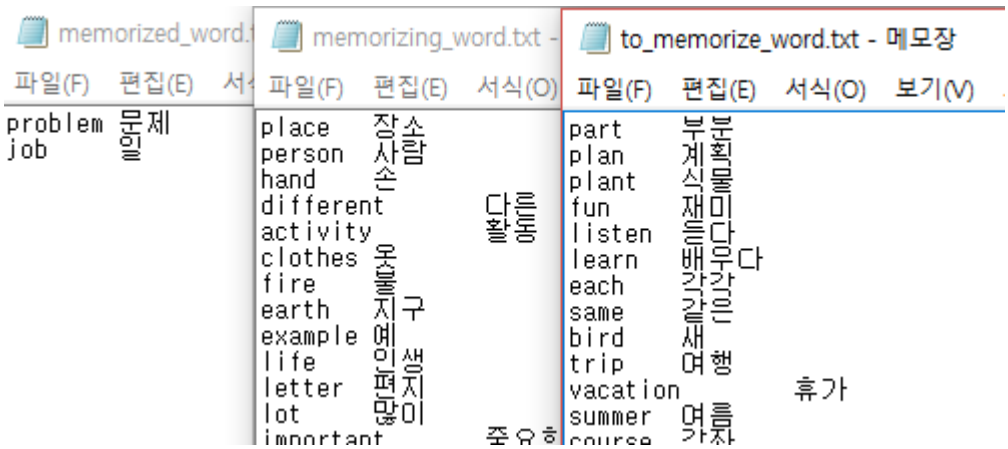
ADD, PRINT TO_MEMORIZE확인　MOVE확인(MOVE 20후 결과) PRINT MEMORIZING R_PRE

```
======== ADD ========   ======== ADD ========
Success                Success
====================   ====================

======== MOVE ========  ======== MOVE ========
Success                Success
======================  ======================

======== PRINT ======== ======== PRINT ========
part 부분               problem 문제
plan 계획               place 장소
plant 식물              person 사람
fun 재미                hand 손
listen 듣다             different 다른
learn 배우다            activity 활동
each 각각               clothes 옷
same 같은               fire 불
bird 새                 earth 지구
trip 여행               example 예
vacation 휴가           life 인생
summer 여름             letter 편지
course 강좌             lot 많이
spring 봄               job 직업
autumn 가을             important 중요한
winter 겨울             name 이름
space 공간              make 만들다
street 거리             movie 영화
paper 종이              story 이야기
newspaper 신문          work 일
face 얼굴               ======================
```

TEST확인, UPDATE확인 PRINT_MEMORIZED확인 SEARCH확인

```
======== TEST ========
Pass
======================

======== TEST ========
Pass
======================

======== UPDATE ========
job 직업 -> 일
========================

======== SEARCH ========
problem 문제
========================

======== PRINT ========
problem 문제
job 일
========================

======== SAVE ========
Success
======================
```

SAVE확인



LOAD확인 PRINT MEMORIZING I_LEVEL확인 PRINT MEMORIZING I_IN확인

PRINT MEMORIZING I_POST확인

```
======== LOAD ========
Success
======================

======== PRINT ========
place 장소
person 사람
hand 손
different 다른
life 인생
letter 편지
lot 많이
fire 불
name 이름
activity 활동
clothes 옷
earth 지구
example 예
important 중요한
make 만들다
movie 영화
story 이야기
work 일
======================
```

```
======== PRINT ========
activity 활동
clothes 옷
different 다른
earth 지구
example 예
fire 불
hand 손
important 중요한
letter 편지
life 인생
lot 많이
make 만들다
movie 영화
name 이름
person 사람
place 장소
story 이야기
work 일
======================

======== PRINT ========
activity 활동
clothes 옷
example 예
earth 지구
fire 불
different 다른
important 중요한
movie 영화
make 만들다
name 이름
letter 편지
lot 많이
life 인생
hand 손
story 이야기
work 일
person 사람
place 장소
======================
```

이름	프로젝트에서 맡은 역할	본인 스스로 생각하는 자신의 점수(10)
정무영	BST함수(전위, 중위 순회 제외), Delete함수/ Manager run, ADD, LOAD, SAVE, 전체적인 디버깅	9
고찰	프로젝트 규모가 아직 그렇게 크진 않아서 혼자 할 분량을 왜 팀으로 해야 하는지 처음에는 의문을 가졌지만 팀으로 진행하면서 역할을 분담하니 일이 절반으로 줄어서 팀프로젝트의 이점을 누릴 수 있었다. 내가 생각하지 못한 부분을 팀원이 발견 할 때도 있어서 같이 했다는 점에 안도한 적도 있었다. 처음 하는 팀 프로젝트였지만 수월하게 진행된 거 같아 기쁘다.	
오재원	Queue class 함수, CLL class 함수 BST 전위, 중위 순회 Delete함수 구현/ Manager TEST, PRINT, SEARCH, UPDATE, MOVE 리눅스에 맞게 수정	9
고찰	내가 프로그래밍 실력이 많이 부족해서 팀원 도움을 정말 많이 받았다. 구조체 개념을 이해하지 못해서 class 이후로 프로그래밍에 어려움이 많았는데 이번 프로젝트를 하면서 팀원이 정말 많이 고생했다. 혼자 했으면 못했을 프로젝트였는데 팀원이 있어서 할 수 있었다. 팀원에게 정말 미안하고 고맙다.	