

데이터 구조 설계 및 실습

Project1 Report

제출일자: 2016년 10월 08일 (금)

학 과: 컴퓨터공학과

담당교수: 이기훈 교수님

실습분반: 화요일 3,4 교시

학번 이름: 2013722022옥준영

2013722073김태영

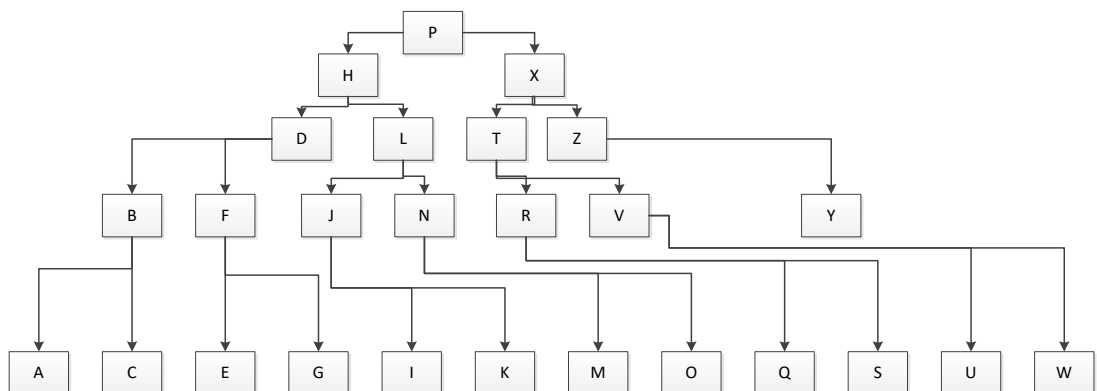
2013722083조형훈

1. Introduction

단어를 외우고자 하는 사용자에게 이진 탐색 트리와 환형 연결리스트와 큐를 이용하여 단어 암기에 도움이 되는 프로그램을 만든다. 최대 100개의 영어 단어와 한글 뜻을 입력 받아 외우고자 하는 단어(To memorize), 외우고 있는 단어(Memorizing), 외운 단어(Memorized)로 구분하여 구현한다. 이 때 순서대로 Queue, BST, Circular linked list로 구성한다.

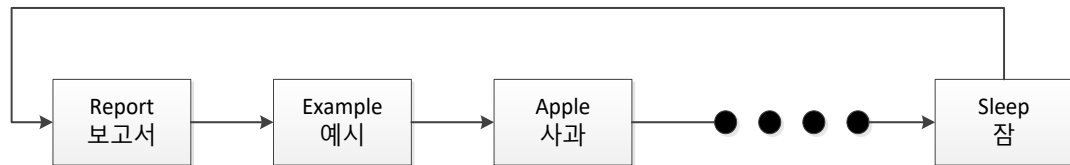
To memorize는 Queue로 구성되며, 새로운 단어가 들어오면 Push를 하고 memorizing으로 단어를 옮길 때 Pop을 한다. 처음 입력되는 단어들은 다른 클래스로 들어가지 못하고 오직 To memorize로만 들어갈 수 있다. To memorize->Memorizing->Memorized 순서대로 단어의 학습 진행 과정에 따라서 단어가 이동하게 된다. Memorizing에는 최대 100개의 단어만 존재할 수 있다.

Memorizing에는 alphabet BST와 Word BST가 존재한다. Alphabet BST는 반드시 아래와 같이 P, H, X, D, L, T, Z, B, F, J, N, R, V, Y, A, C, E, G, I, K, M, O, Q, S, U, W순으로 추가되어야 하며 프로그램 시작과 동시에 구축된다.

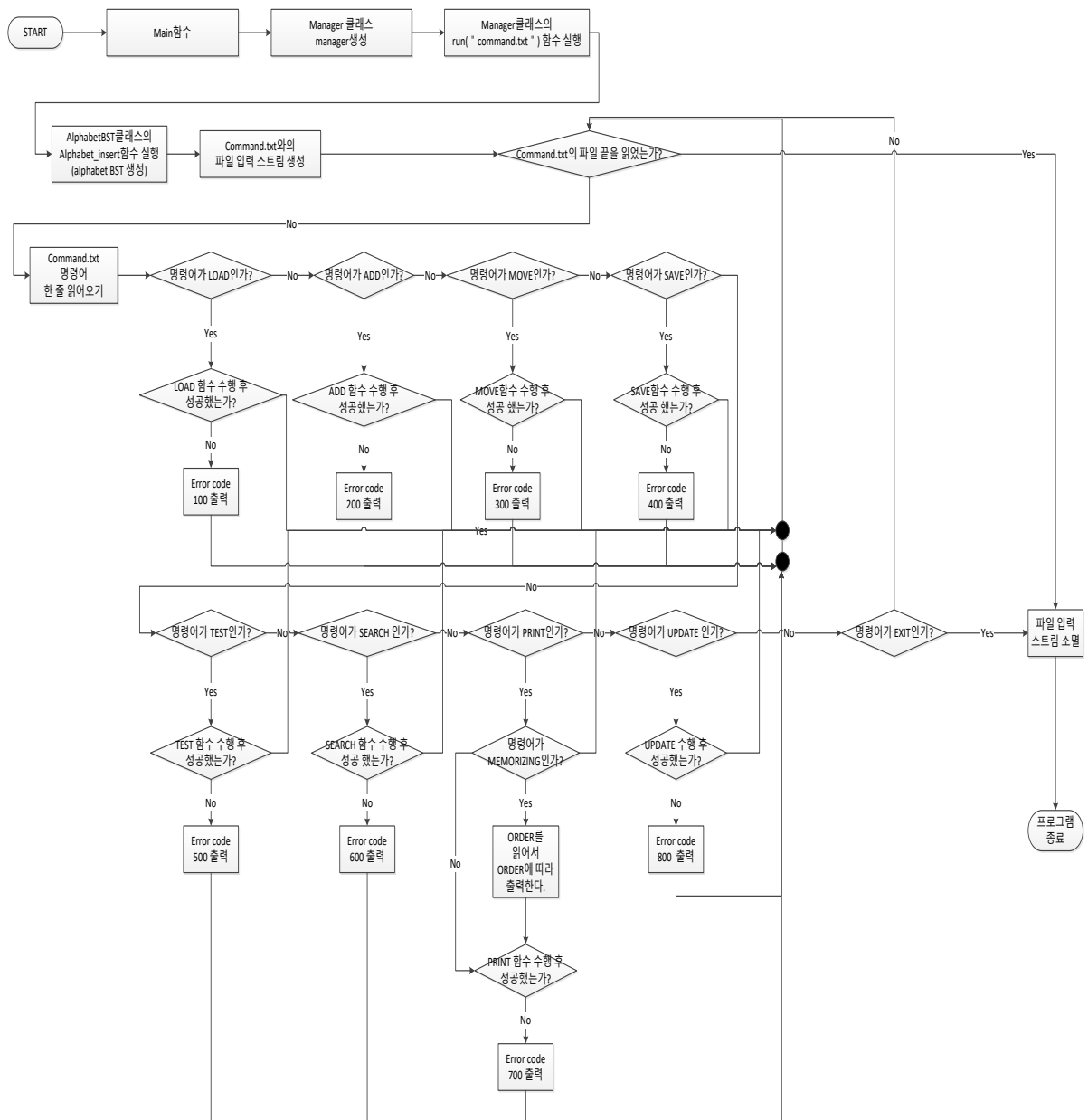


이 때, 각각의 알파벳 노드는 각각의 단어 BST를 갖고 있으며 MOVE를 통해 To memorize로부터 단어를 입력 받을 수 있다. Memorizing에 존재하는 단어는 Test명령어를 통해 테스트를 할 수 있으며 단어를 외운 것이 확인 된다면 Memorized로 단어를 옮긴다. 이 때, BST의 연결규칙이 있는데, 부모 노드보다 단어의 사전적 순서가 작은 노드는 왼쪽, 큰 노드는 오른쪽 서브 트리에 위치해야 하며, 노드를 제거 할 때에는 양쪽 자식 노드가 모두 존재할 경우에는 왼쪽 자식 노드 중 가장 큰 자식 노드를 제거되는 노드 위치로 이동시켜야 한다. 왼쪽 자식 노드가 존재하지 않을 경우에는 오른쪽 자식 노드 중 가장 작은 노드를 제거되는 노드 위치로 이동시킨다.

Memorized의 circular linked list는 아래와 같이 구성되며, 처음 단어가 들어올 때는 Head pointer가 들어온 단어를 가리키고 들어온 단어는 head pointer를 가리키게 된다. List에 단어가 이미 존재할 경우에는 새로 들어온 단어를 head pointer가 가리키는 노드와 바로 뒤의 노드 사이에 연결한다.



2. Flowchart



위 순서도는 이번 프로젝트의 순서이다. 시작과 동시에 Main 함수를 실행하며 제일 먼저 Manager 클래스로 manager를 생성한다. 그 다음, manager 클래스의 run 함수를 실행하면서 "command.txt"를 인자로 전달한다. 이 때, run의 실행과 동시에 alphabetBST 클래스의 Alphabet_insert 함수가 실행되면서 AlphabetBST를 생성한다. 그 후, 인자로 전달된 command.txt 파일과의 입력 스트림을 생성한다. 그 후, 반복문을 진행하는데 반복 조건으로는 파일의 끝을 읽었는지 eof() 함수를 통해 확인하며, 파일의 끝을 읽었다는 것이 확인될 경우 반복문을 탈출하여 프로그램을 종료한다.

파일의 끝을 읽기 전까지는 반복문이 계속 반복된다. Command.txt에는 명령어가 한 줄씩 저장되어 있기 때문에 1회 반복 시마다 명령어를 한 줄씩 읽어 들인다. 읽은 명령어를 해석하는데 해석하는 조건은 총 9개로 나뉘어져 있다. ERROR, LOAD, ADD, MOVE, SAVE, TEST, SEARCH, PRINT, UPDATE, EXIT가 있는데, 각 조건이 성립하여 명령의 실행이 완료되거나 명령 처리 과정에서 에러가 발생한 이후에는 다시 반복문의 처음으로 돌아가서 다음 명령어를 읽어 들이고 명령어를 처리하는 과정을 계속 반복한다. 만약, 명령어가 EXIT이거나 파일의 끝을 읽은 경우 파일 스트림을 소멸하고 프로그램을 종료한다.

3. Algorithm

프로그램에서 사용한 클래스와 함수는 여러가지 종류가 있다. 이번에 구현한 프로그램은 입력된 명령어 대로 실행되는데 각 명령어에는 각기 다른 클래스와 함수들이 사용된다.

먼저 명령어들을 실행하는 RUN 함수는 Manager 클래스에 있는 함수이다.파일 입력 스트림으로 Command.txt파일을 불러와 명령어를 한 줄씩 읽어 실행한다.명령어는 대문자여야만 하고 크게 LOAD, ADD, MOVE, SAVE, TEST, SEARCH, PRINT, UPDATE, EXIT 등이 있다.입력된 명령어에 따라 해당 명령어 함수를 실행하고,실행 과정에서 오류가 발생한 경우 각 명령어에 맞는 Error 코드를 출력한다.

LOAD는 이전에 SAVE명령어에 의해 저장된 단어 데이터가 있을 경우,저장된 단어 데이터를 불러와서 TO MEMORIZE, MEMORIZING, MEMORIZED의 각 클래스로 데이터를 저장한다.데이터의 저장을 위해 각 클래스의 SAVE함수가 사용된다. 만약 SAVE 되어 있던 파일이 한 개라도 없거나,이미 클래스에 저장되어 있던 단어와 겹치는 단어가 있을 경우 에러 코드 100을 반환한다. Introduction 설명 대로, 각 단어장은 queue, BST, Circular Linked List의 구조로 구현된다.

ADD 함수는 word.txt 파일로부터 단어를 TO_MEMORIZE 클래스로 불러오는 함수이다. TO_MEMORIZE가 Queue로 구성되어 있기 때문에 Queue의 push 함수가 사용된다.Word.txt에 단어가 없거나 이미 존재하는 단어라면 불러오지 않는다.만약 불러온 단어가 하나도 없을 경우 에러 코드 200을 반환한다. Queue는 WordNode로 구성되며 WordNode는Pair STL로 구성되어 영어 단어와 뜻에 대해서 pair로 저장한다.

MOVE명령어는 사용자가 입력한 숫자 만큼 TO_MEMORIZE 클래스에서 MEMORIZING 클래스로 단어를 옮긴다. TO_MEMORIZE Queue의 Pop 함수와 MEMORIZING클래스의 BST의 insert 함수가 동시에 사용된다. Queue에서 Pop한 데이터를 BST의 insert로 받아서 저장한다.만약 TO_MEMORIZE queue에 저장된 단어가 없을 경우 에러 코드 300을 반환한다.

SAVE는 현재 프로그램에 존재하는 TO_MEMORIZE, MEMORIZING, MEMORIZED 클래스의 모든 단어를 텍스트 파일로 저장하는 함수이다.각 클래스 별로 따로SAVE 함수가 사용되며 각 SAVE 함수 실행 과정에서 에러가 발생하면 에러카운트를 증가시키고에러 카운트가 하나라도 발생한 경우 에러코드 400을 반환한다.

TEST는 MEMORIZING 클래스에 있는 단어를 외웠는지 테스트 하는 함수이다. BST의 Search 함수를 통해 단어를 검색하고,단어와 뜻이 일치 할 경우, MEMORIZING 클래스의 Delete함수로 인해 삭제되는 데이터를 Memorized 클래스의 insert 함수로 받아서 저장한다.단어와 뜻이 틀릴 경우 에러코드 500을 반환한다.

SEARCH는 TO_MEMORIZE, MEMORIZING, MEMORIZED 클래스에 단어가 존재하는지 검색 하는 함수이다.입력된 단어를 인자로 받아서 검색하고,단어가 존재 할 경우 단어를 출력 하고 단어가 존재하지 않을 경우 에러코드 600을 출력한다.

PRINT는 첫번째 인자로 클래스 명을 입력 받는다.특히 MEMORIZING의 경우 두번째 인자로 Order를 추가로 입력 받아서 입력된 Order에 맞추어 현재 입력된 클래스가 가지고 있는 단어를 전부 출력한다.만약 클래스에 단어가 존재하지 않을 경우 에러코드 700을 출력한다.

UPDATE는To memorize, Memorizing, Memorized 전부에 사용된다. 첫번째 인자로 단어,두 번째 인자로 뜻을 입력 받아서 입력된 단어가 존재할 경우에 작동한다.먼저 입력된 의미를 삭제한 후, 새로 입력된 의미로 변환시킨다.만약 단어가 존재하지 않을 경우 에러코드 800을 출력한다.

Error와 Success는 log.txt와 display로 확인이 가능하며, run 함수 내에서 Error 또는 Success 되는지 쉽게 확인 할 수 있다.

EXIT가 입력됐다면 command.txt와의 파일 입력 스트림을 소멸하고 프로그램을 종료한다.

A. To memorize

Queue는 To memorize에 사용되며 queue 구조를 기본으로 한다. node를 push와 pop하는 기능이 사용되며, search에서는 queue 안에 있는 word를 찾아낸다. print에서는 queue 안에 있는 node를 log와 컴파일 창에서 display한다. save는 to_memoerize.txt에 저장한다.

B. Memorizing

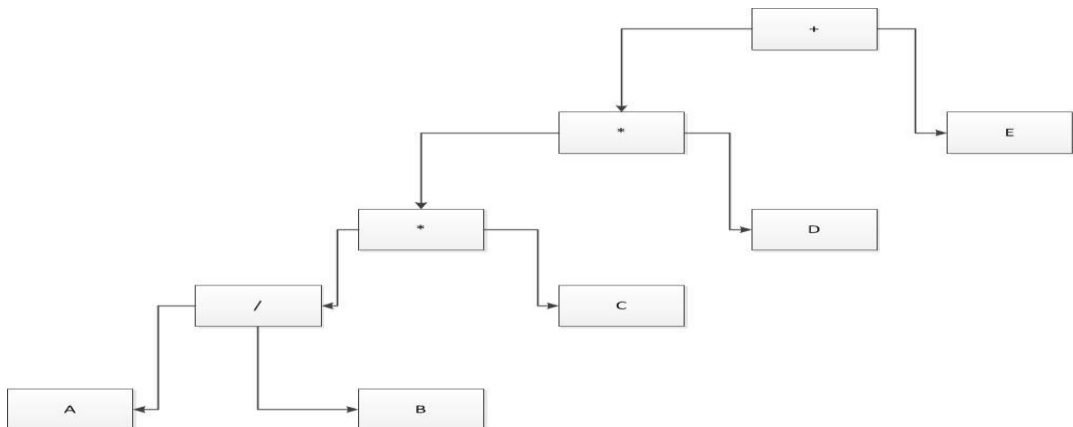
AlphabetBST는 Memorizing에 사용되며 BST구조를 기반으로 한다. 또한 프로그램이 시작되면 alhapbet_insert 함수를 사용하여 바로 구축된다. Insert는 move, load에서 사용되며, search는 load, move, test, search, update등에 사용된다. Save는 command에서 입력 받았을 때에만 preorder 방식을 사용해 node들을 memorizng.txt에 저장한다. print는 아래에 서술한 여러 정렬 방식 중 order를 입력 받아 node와 log를 display 한다.정렬방식은 Recursive inorder,Recursive preorder, Recursive postorder, Iterative Inorder, Iterative preorder, Iterative postorder, Iterative levelorder 의 방식으로 구현되어 있다.

AlphabetNode는 AlphabetBST에 사용되는 node가 어떻게 작동하는지 구현되어있으며 WordBST를 구성요소로 갖는다.

WordBST는 Memorizing에 사용되며 동일한 알파벳 첫 글자를 가진 단어들을 모아 놓은 트리이다. 또한 AlpahbetBST와 동일하게 BST 구조를 기반으로 한다. insert는 load와 move에 사용되며, Delete는 test하는데 사용되며 search는 load, move, test, search, update등에 사용된다. WordBST는 AlphabetNode의 구성 요소가 된다.

WordNode는 WordBST를 구성하는 노드이다. 같은 WordBST에 존재하는 모든 WordNode들은 똑같은 알파벳 첫 글자를 가진다. Alphabetnode와 동일한 방식으로 구성 되어 있다.

BST의 order방식은 아래 설명된다.



먼저 위에 그림은 기본적인 트리 구조이다.

Recursive inorder : $A/B * C * D + E$

Recursive preorder : $+ ** / ABCDE$

Recursive postorder : $AB / C * D * E +$

Iterative Inorder : $A/B * C * D + E$

Iterative preorder : $+ ** / ABCDE$

Iterative postorder : $AB / C * D * E +$

Iterative levelorder : $+ * E * D / CAB$

각 order별로 출력 결과는 위와 같이 나타나는데, Iterative와 Recursive의 출력 결과가 동일함을 알 수 있다. 둘의 차이점은 Recursive는 재귀함수로서 자기 자신을 다시 호출하여 재귀적으로 출력하며 Iterative는 반복함수로서, 반복문을 통해 Order를 진행한다는 점이 다르다.

Inorder는 왼쪽 끝 자식 노드부터 시작하여 부모 노드->부모의 우측 자식노드->부모의 부모노드 순서대로 방문한다.

Preorder는 Root부터 순서대로 왼쪽 자식노드를 쪽 방문한 다음 끝까지 방문 했다면 우측 자식노드 순서로 방문한다.

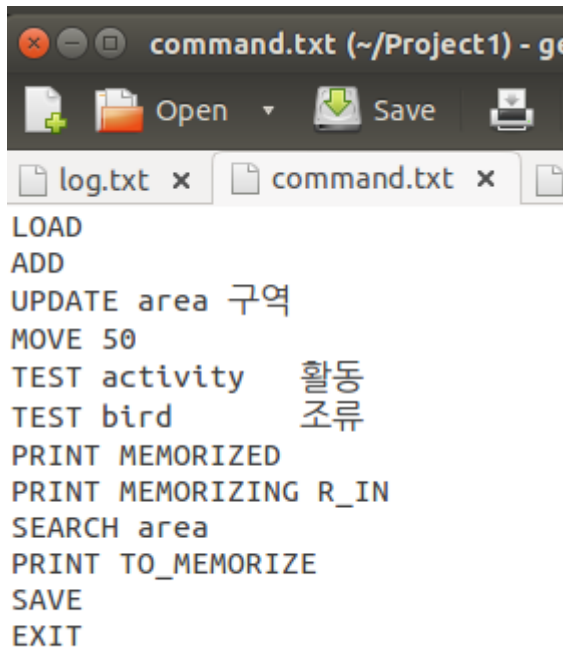
Postorder는 왼쪽 끝 자식 노드부터 시작하여, 왼쪽 자식노드, 형제 노드, 부모노드, 부모의 형제노드 순서로 방문한다.

Level order는 루트부터 시작하여 왼쪽부터 오른쪽으로 노드에 번호를 부여하고, 부여된 번호 순대로 출력하는 방식이다.

C. Memorized

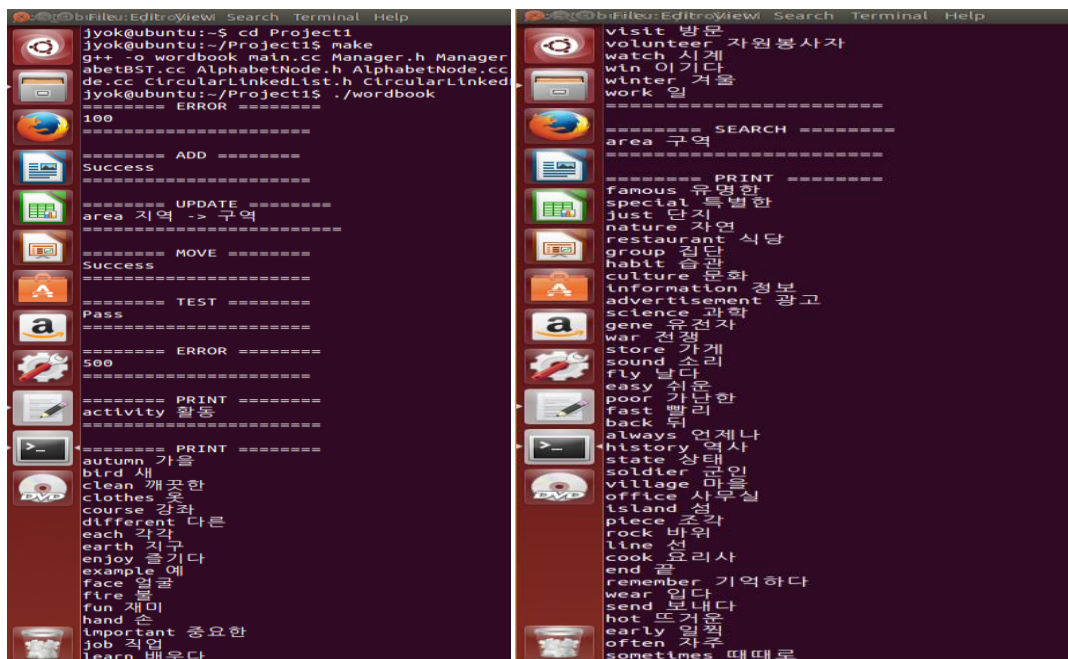
CircularLinkedList는 Memorized에 사용되며 원형 linked list로 구현되어있다. insert는 load, test에 사용되며, search는 update와 search 하는데 사용된다. save는 memoerized_word에 저장된다.

4. Result Screen



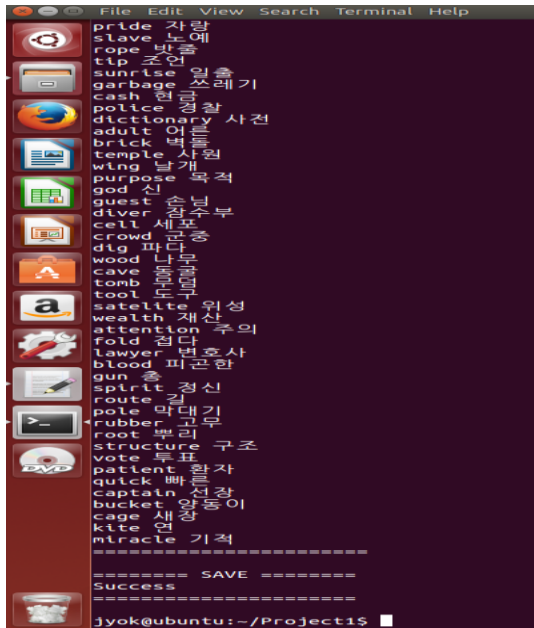
```
command.txt (~/.Project1) - ge
LOAD
ADD
UPDATE area 구역
MOVE 50
TEST activity 활동
TEST bird 조류
PRINT MEMORIZED
PRINT MEMORIZING R_IN
SEARCH area
PRINT TO_MEMORIZE
SAVE
EXIT
```

실행한 command..txt이다.

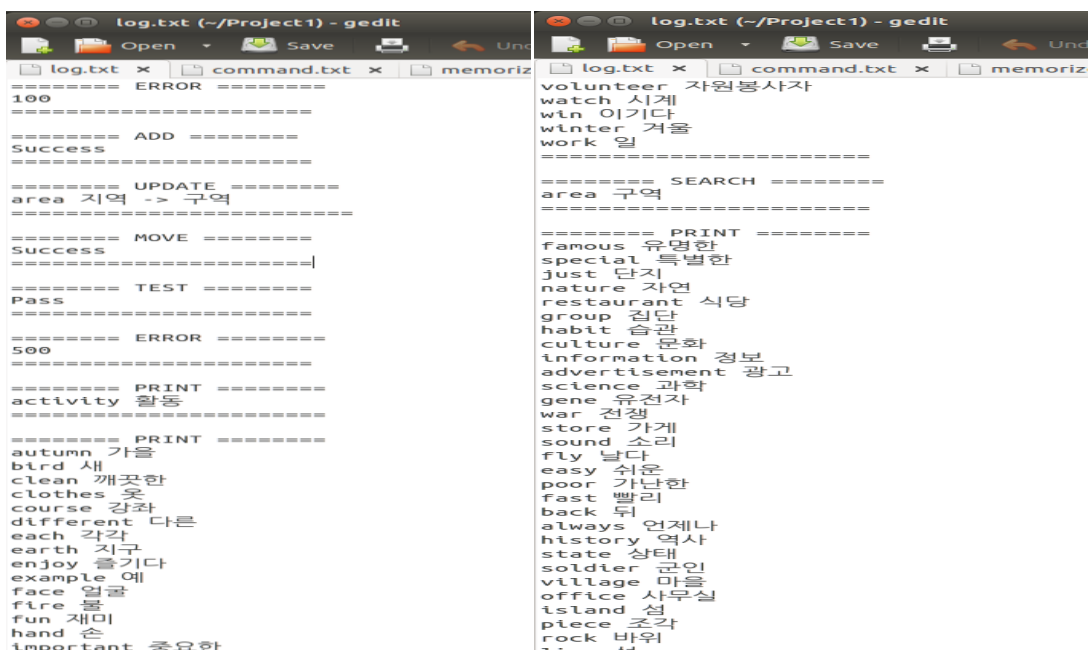


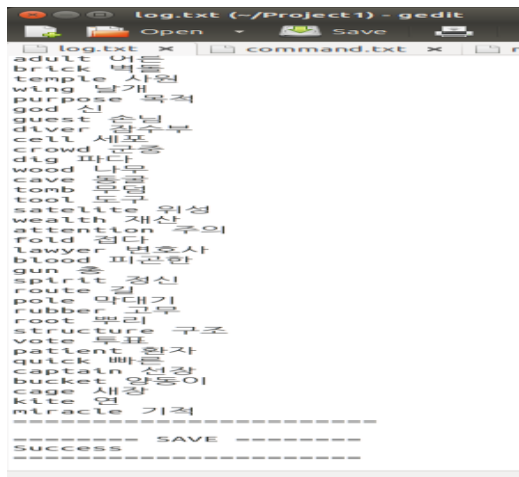
```
j yok@ubuntu:~$ cd Project1
j yok@ubuntu:~/Project1$ make
g++ -o wordbook main.cc Manager.h Manager
abetBST.cc AlphabetNode.h AlphabetNode.cc
de.cc CircularLinkedList.h CircularLinked
j yok@ubuntu:~/Project1$ ./wordbook
===== ERROR =====
===== ADD =====
Success
===== UPDATE =====
area 지역 -> 구역
===== MOVE =====
Success
===== TEST =====
Pass
===== ERROR =====
500
===== PRINT =====
activity 활동
===== PRINT =====
autumn 가을
bird 새
clean 깨끗한
clothes 옷
course 강좌
different 다른
each 각각
earth 지구
enjoy 즐기다
example 예
face 얼굴
fire 불
fun 재미
hand 손
important 중요한
job 직업
learn 배우다

visit 방문
volunteer 자원봉사자
watch 시계
win 이기다
winter 겨울
work 일
===== SEARCH =====
area 구역
===== PRINT =====
famous 유명한
special 특별한
just 단지
nature 자연
restaurant 식당
group 집단
habit 습관
culture 문화
information 정보
advertisement 광고
science 과학
gene 유전자
war 전쟁
store 가게
sound 소리
fly 날다
easy 쉬운
poor 가난한
fast 빨리
back 뒤
always 언제나
history 역사
state 상태
soldier 군인
village 마을
office 사무실
island 섬
piece 조각
rock 바위
line 선
cook 요리사
end 끝
remember 기억하다
wear 입다
send 보내다
hot 뜨거운
early 일찍
often 자주
sometimes 때때로
```

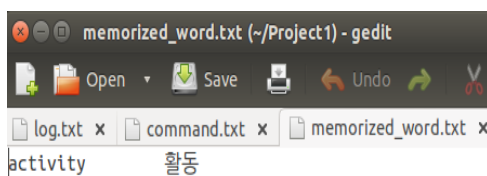


make file로 만든 후, object 파일을 실행했을 때의 결과이다. Load 명령어를 실행할 때, 프로그램을 처음 실행해서 각 단어장의 txt 파일이 저장되어 있지 않아서 error가 뜨는 것을 확인 할 수 있다. Add 명령어를 실행할 때는 word.txt에서 to memorize로 옮기는 것이므로 성공하는 것을 확인 할 수 있다. Update 명령어에서는 area의 뜻을 구역으로 바꾼다. Move 명령어는 to memorize에서 memorizing으로 50개의 단어를 옮긴다. 그 다음, Test에서는 bird는 틀린 답이며 error가 뜨는 것을 확인 할 수 있고 activity는 옳은 답이므로, memorized로 옮긴 것을 확인할 수 있다. 그 후, Memorized와 memorized r_in에 있는 것이 출력되는 것을 확인 할 수 있다. Search 명령어에서 area의 변경된 뜻을 확인 할 수 있으며, to_memorize가 잘 출력되는 것을 확인 할 수 있다.

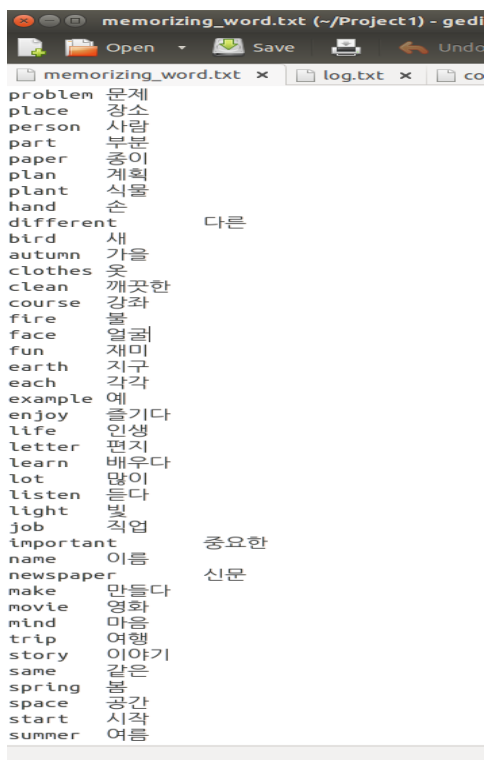




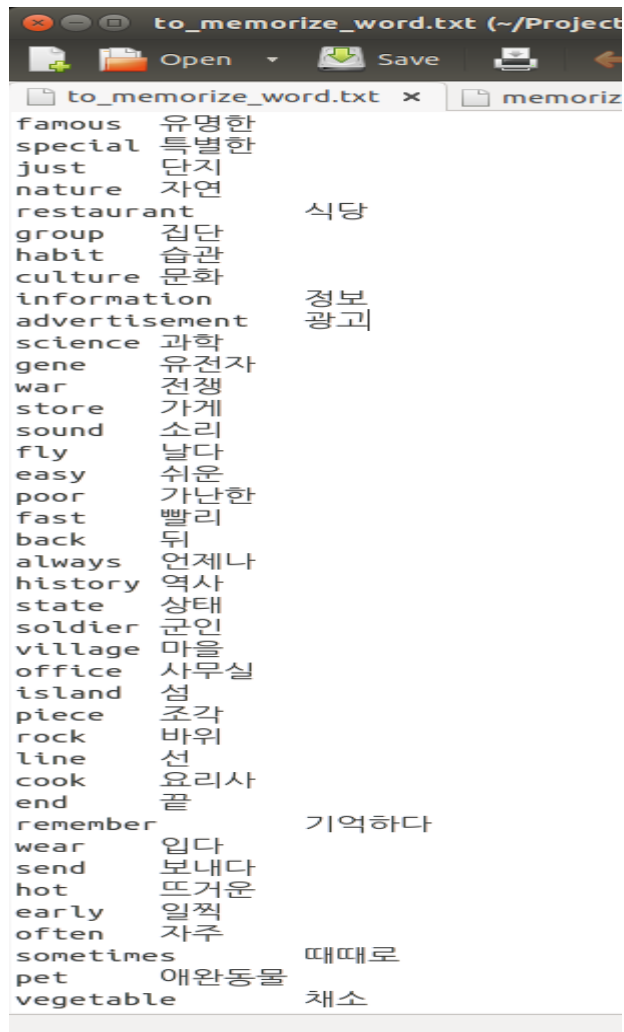
log에서 terminal창과 동일한 결과가 display 된 것을 확인 할 수 있다.



test를 통해 이동된 단어를 확인 할 수 있다.



memorizing된 단어를 확인 할 수 있다.



to memorize된 단어를 확인 할 수 있다.

5. Consideration

A. 옥준영

이름	프로젝트에서 맡은 역할	본인 스스로 생각하는 자신의 점수
옥준영	TO_MEMORIZE 클래스 코드 작성. 보고서 작성	5
고찰	1학년 C프로그래밍, 2학년 고급프로그래밍 수업을 들으면서 우리는 스택과 큐, 링크드리스트, 파일입출력에 대해서 충분히 배워왔다.내가 맡은 부분은 Queue였기 때문에 그다지 어렵지 않았다.하지만 BST는 이번 학기에 처음 배우는 내용이었다. 그래서 우리 팀이 코드를 구현하는데 있어서 다른 데이터 구조와 비교해서 가장 어려운 내용이 아니었나 생각한다. 다행히 김태영 학생이 코딩에 능숙하여 도움을 많이 받았다.조별과제라서 부담이 될 것 같았는데 김태영 학생의 실력이 우수하여 생각보다 문제 없이 프로젝트를 완성해 낼 수 있었다고 생각한다.	

B. 김태영

이름	프로젝트에서 맡은 역할	본인 스스로 생각하는 자신의 점수
김태영	MEMORIZED 코드 작성 및 프로젝트 전체 코드 수정 및 보완하여 프로젝트 완성	9
고찰	Queue, Circular linked list 의 전반적인 구성을 하는데 큰 어려움은 없었으나, BST를 구성하는데 약간의 어려움이 있었다.특히 AlphabetBST의 노드 안에 WordBST가 구성되어 있어 코드를 작성할 때 양쪽을 모두 생각하며 작성을 해야 했다.그리고 BST구조에서 PRINT부분의 경우 대부분의 방식을 데이터 구조 설계 시간에 배웠기 때문에 비교적 쉽게 작성할 수 있었으나, Iterative Postorder의 경우는 기존에 알고 있던 방식으로는 잘 구성되지 않아 인터넷을 참조한 뒤 직접 그림을 그려가며 작성하였다. 그 밖에 리눅스에서 실행 시 word.txt파일을 윈도우에서 복사하여 옮겼더니 한글이 잘 출력되지 않는 문제가 있었는데,조교님께 질문한 결과 리눅스에서 직접 txt 파일을 만들어 사용해야한다는 도움을 받아 문제를 해결할 수 있었다.	

C. 조형훈

이름	프로젝트에서 맡은 역할	본인 스스로 생각하는 자신의 점수
조형훈	MEMORIZING 클래스 코드 작성. 보고서 작성.	6
고찰	<p>지금까지 배워왔던 구조들과는 다르게 처음 배우는 BST를 구현한다는데 많은 어려움이 있었던 것이 사실이다. 특히 delete를 구현하는데, 어떤 방식으로 구현해야, 구조가 깨지지 않고 올바르게 작동하는지 많은 고민이 들었다. 또한, 여러 가지 방식을 사용하여 order를 하는데 같은 조원인 김태영 학생의 도움을 받았다. 또한 설계 시간에 제출된 과제가 BST를 구현 하는데 많은 도움이 되었다. 마지막으로, 리포트를 작성하는데 옥준영 학생과 쓰는 방식이 서로 달라 적절한 타협점을 찾을 수 있었다. 지금까지 해왔던 팀 프로젝트와는 다르게 코딩을 병행한 팀 프로젝트여서 많은 혼선이 있었지만 좋은 결과물을 만들 수 있어 만족한다.</p>	

참조) Iterative Postorder / <http://www.geeksforgeeks.org/iterative-postorder-traversal-using-stack/>