

Data Structure Lab. Project #1

제출일자: 2016년 10월 07일 (금)

학 과: 컴퓨터공학과

담당교수: 이기훈 교수님

학 번: 2013722020

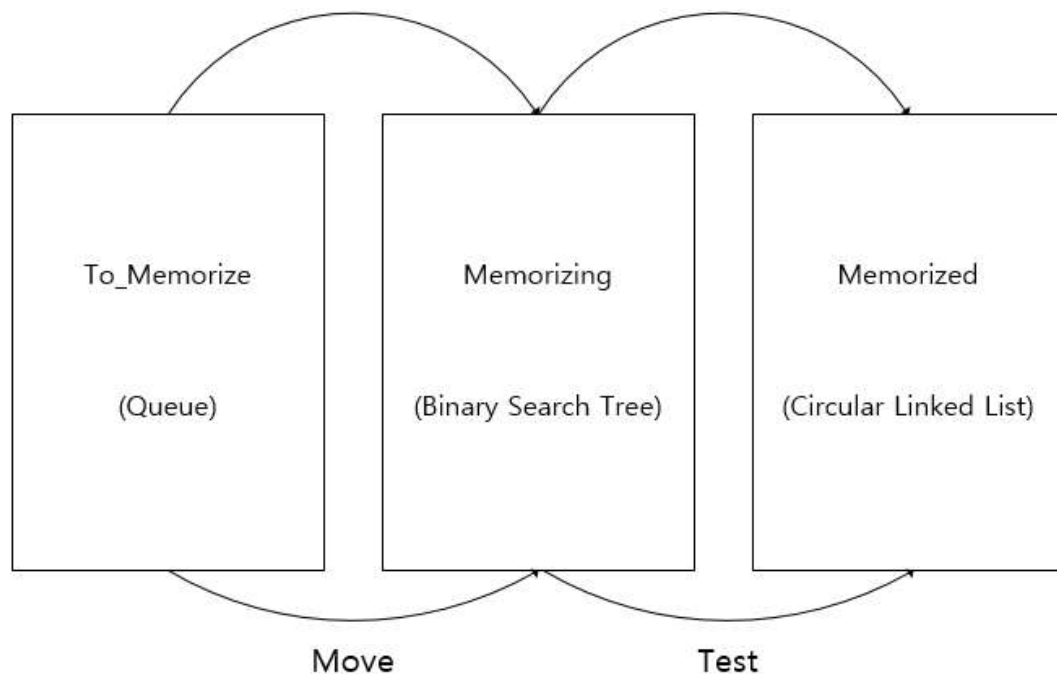
조 장: 이전제

팀 원: 2013722002이민형

2013722004 최민기

Introduction

영어 단어장 프로그램



영어 단어장 프로그램은 앞으로 외워야 할 단어장(To_Memorize), 현재 외우고 있는 단어장(Memorizing), 이전에 외운 단어장(Memorized)으로 구분되어 구성되어 있고 각각 Queue, Binary Search Tree, Circular Linked List로 구현했다. 사용자가 사용할 모든 동작들은 명령어 파일을 통해 동작에 대한 명령을 실행하며 모든 출력은 로그 파일에 저장한다.

1. 외워야 할 단어장(To_Memorize)의 Queue

Queue는 처음에 넣은 데이터가 먼저 나오는 FIFO구조이고 Push와 Pop으로 동작한다.

따라서 Queue에서 새로운 단어가 들어오면 Push 동작을 해서 단어장에 저장해주고,

외워야 할 단어장(To_Memorize)에서 현재 외우고 있는 단어장(Memorizing)으로 옮겨 줄 때에는 Move 명령어를 사용하고 이 때 외워야 할 단어장(To_Memorize)의 Queue는 Pop 동작을 수행한다.

2. 현재 외우고 있는 단어장(Memorizing)의 Binary Search Tree

현재 외우고 있는 단어장(Memorizing)은 Alphabet BST와 Alphabet BST내의 Alphabet

Node들은 앞에서 외워야 할 단어장(To_Memorize)로부터 n개의 단어를 입력 받아 구축된

Word BST를 가지고 있다. 현재 외우고 있는 단어장(Memorizing)의 BST는 최대 100개의 Word Node를 가질 수 있다. Test명령어를 통해서 단어를 외웠으면 현재 외우고 있는 단어장(Memorizing)의 단어를 이전에 외운 단어장(Memorized)으로 옮길 수 있다.

3. 이전에 외운 단어장(Memorized)의 Circular Linked List

이전에 외운 단어장(Memorized)의 Word는 현재 외우고 있는 단어장(Memorizing)에서 입력한 단어를 찾아 뜻이 맞는지 확인하고, 맞을 경우 해당 Word를 저장하는 단어장이다.

여기서 Word는 Circular Linked List로 구성되어 있고, 처음 들어올 단어가 Head pointer를 가리키고 들어온 단어는 Head point를 가리키는 Circular Linked List 자료구조로 저장된다.

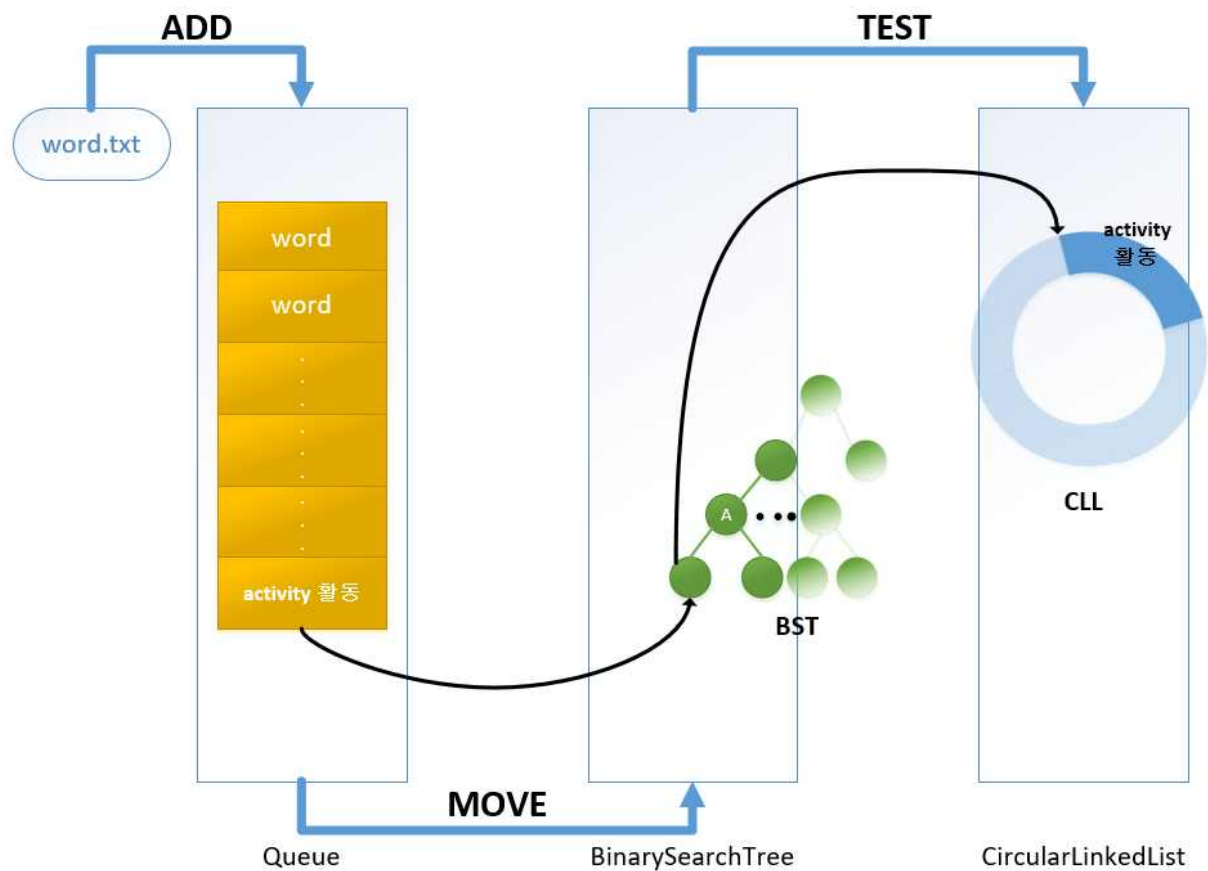
Program Command and function

Command	Function
LOAD	기존의 단어장 정보를 불러오는 명령어로 각각 지정된 텍스트 파일에 단어장 정보가 존재 할 경우 텍스트 파일을 읽어 저장한다. 3개의 텍스트 파일이 존재 하지 않거나 이미 데이터가 들어가 있을 경우에는 ERROR코드를 출력한다.
ADD	Word.txt 파일에 들어있는 단어장 정보를 외워야 할 단어장(To_memorize)에 저장하는 기능을 수행한다. 텍스트 파일에 단어가 존재하지 않거나 텍스트 파일이 존재 하지 않을 경우에 ERROR코드를 출력
MOVE	외워야 할 단어장(To_Memorize)의 단어들을 현재 외우고 있는 단어장(Memorizing)으로 옮기는 명령어 1~100 사이의 정수를 입력 받고, 100이 넘어가면 ERROR코드를 출력한다.
SAVE	현재 단어장 정보를 저장하는 명령어 각각 저장된 자료들을 지정된 텍스트 파일에 저장한다. 단어장 정보가 없을 경우 ERROR 출력

TEST	<p>단어를 외웠는지 확인하는 명령어</p> <p>입력한 단어가 뜻이 맞는지 Search하고 맞을 경우 현재 외우고 있는 단어장(Memorizing) 에서 이전에 외운 단어장(Memorized)로 이동 시켜준다.</p> <p>입력한 단어가 현재 외우고 있는 단어장(Memorizing)에 존재하지 않을 경우와 뜻이 다를 경우에 ERROR코드를 출력</p>
SEARCH	<p>각각 단어장에 입력된 단어가 존재할 경우 단어와 뜻을 찾아 출력하는 명령어이다. 입력한 단어가 존재하지 않는 경우 ERROR코드 출력</p>
PRINT	<p>현재 외우고 있는 단어장(Memorizing)은 BST구조로 이루어져 있다. 따라서 단어를 출력 할 때 출력 방식을 선택 할 수 있다.</p> <p>현재 외우고 있는 단어장(Memorizing)에서 PRINT 명령어 사용시 PRINT 단어장종류 출력방식 입력 나머지 단어장은 단어가 들어가 있는 순서대로 출력된다. 나머지 단어장에서 명령어 사용시 PRINT 단어장종류 입력 단어장이 비어 있으면 ERROR코드 출력</p>
UPDATE	<p>단어의 뜻을 변경하고 싶은 단어를 입력 했을 때 단어장에 존재할 경우 변경하고 싶은 단어의 뜻을 입력해주면 변경한 뒤 출력해준다.</p> <p>단어장에 단어가 존재하지 않거나 단어장의 정보가 존재하지 않을 경우 ERROR코드 출력</p>
EXIT	<p>프로그램 메모리 해제 및 종료</p>

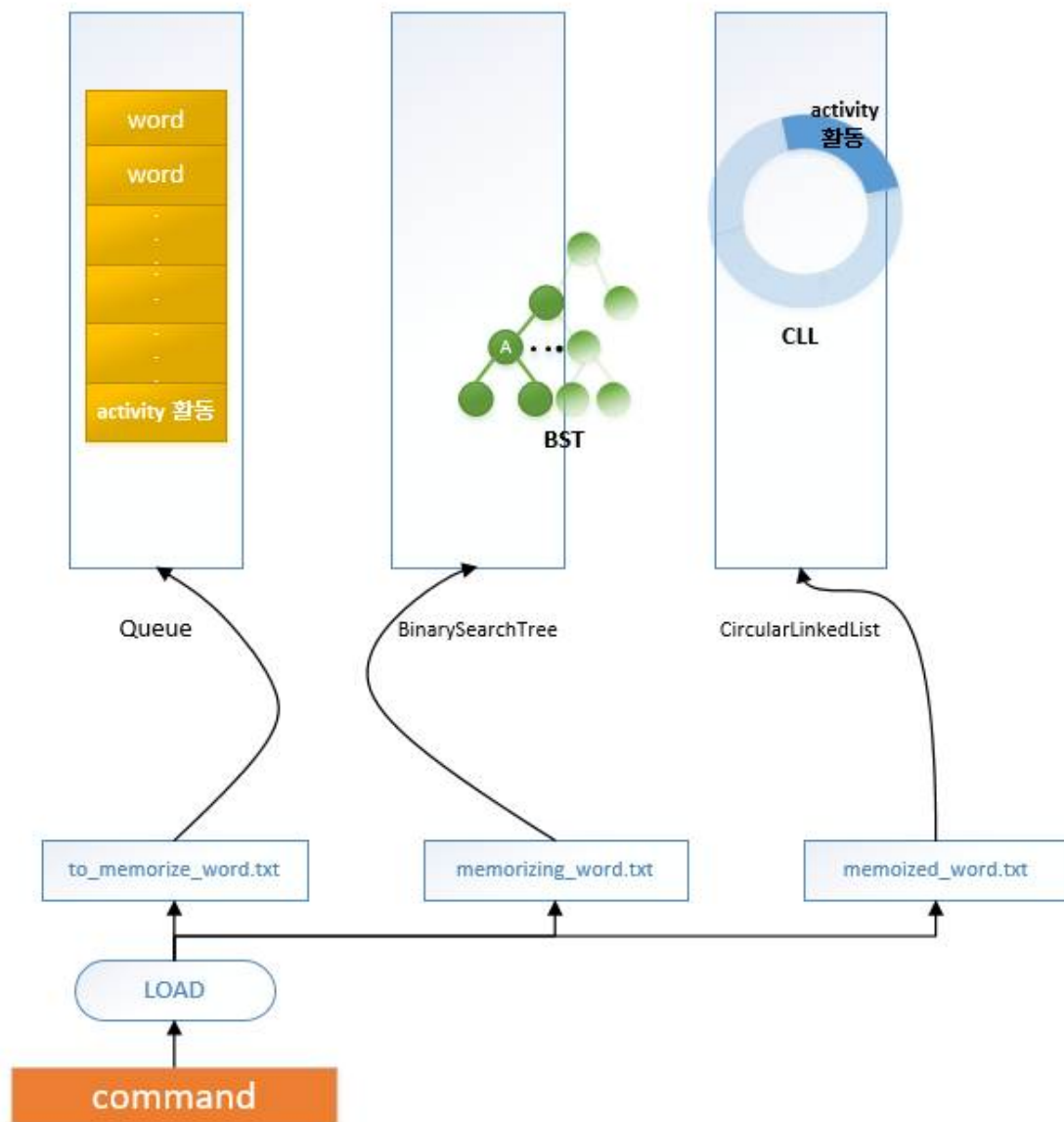
Flow Chart

Overall Flow Chart



Word.txt파일에서 ADD 명령어를 사용해서 외워야 할 단어장(To_Memorize)에 단어를 추가시킨다. 그 다음 MOVE 명령어를 사용해서 외워야 할 단어장(To_Memorize)에서 현재 외우고 있는 단어장(Memorizing)으로 옮기고 TEST 명령어를 통해 단어를 외웠는지 확인하고 이전에 외운 단어장 (Memorized)에 단어를 옮긴다.

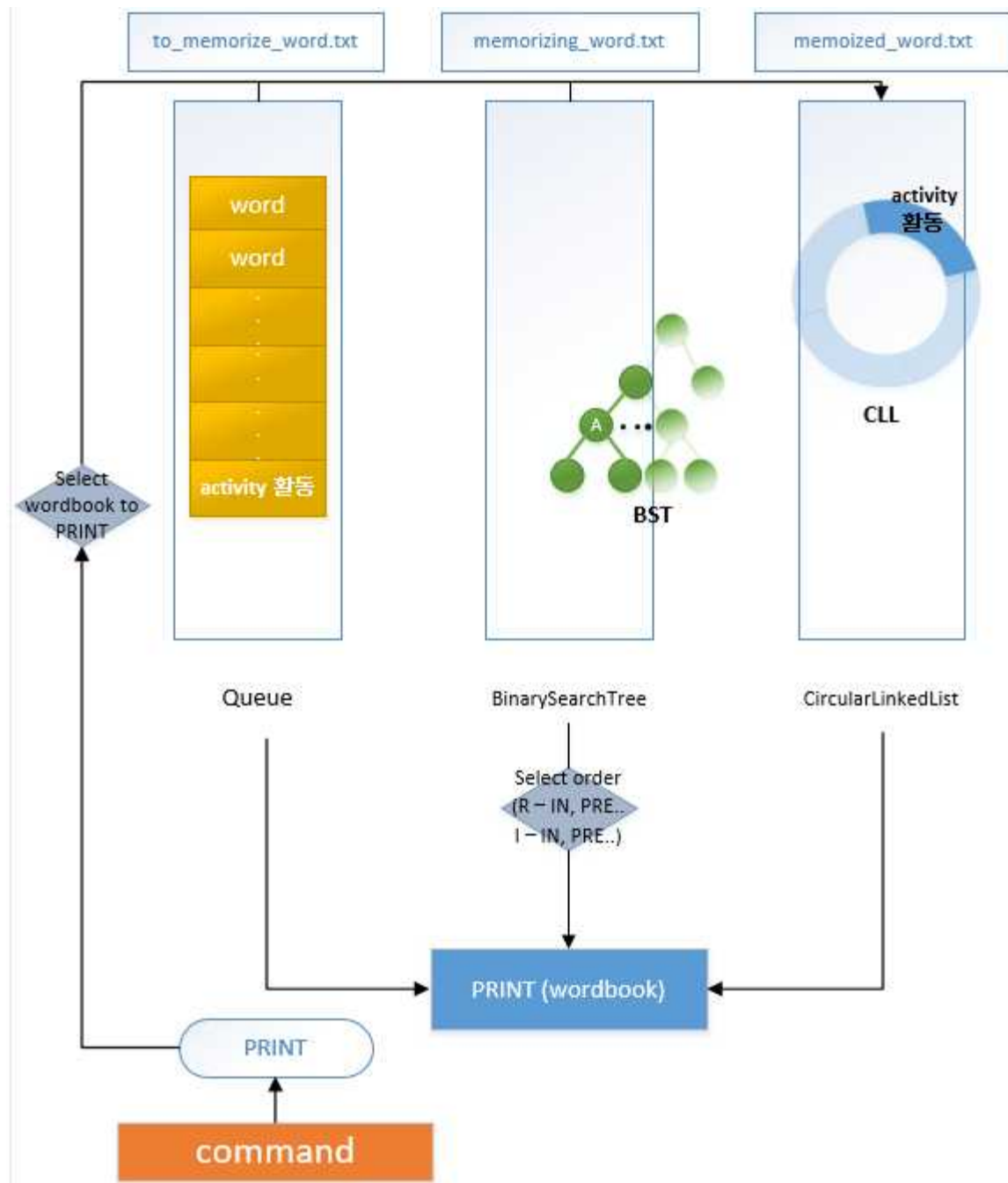
Load Flow Chart



각각의 텍스트 파일 to_memorize_word.txt, memorizing_word.txt, memorized_word.txt에 단어장 정보가 존재할 경우 텍스트 파일을 읽어 외워야 할 단어장(to_memorize), 현재 외우고 있는 단어장(memorizing), 이전에 외운 단어장(memorized)의 Queue, BST, Circular

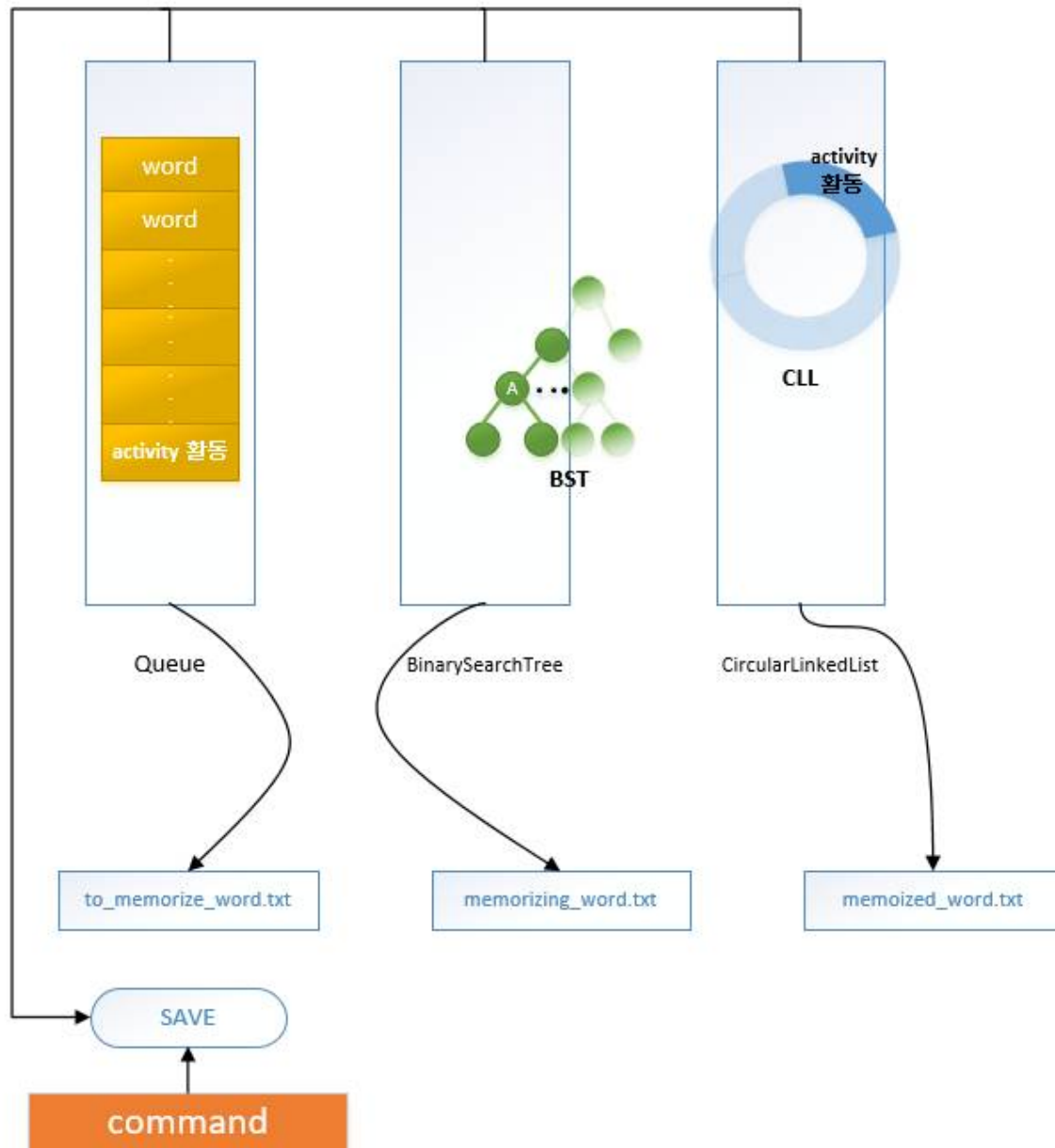
Linked List 자료구조에 이전과 동일한 연결순서를 가지도록 저장한다.

PRINT Flow Chart



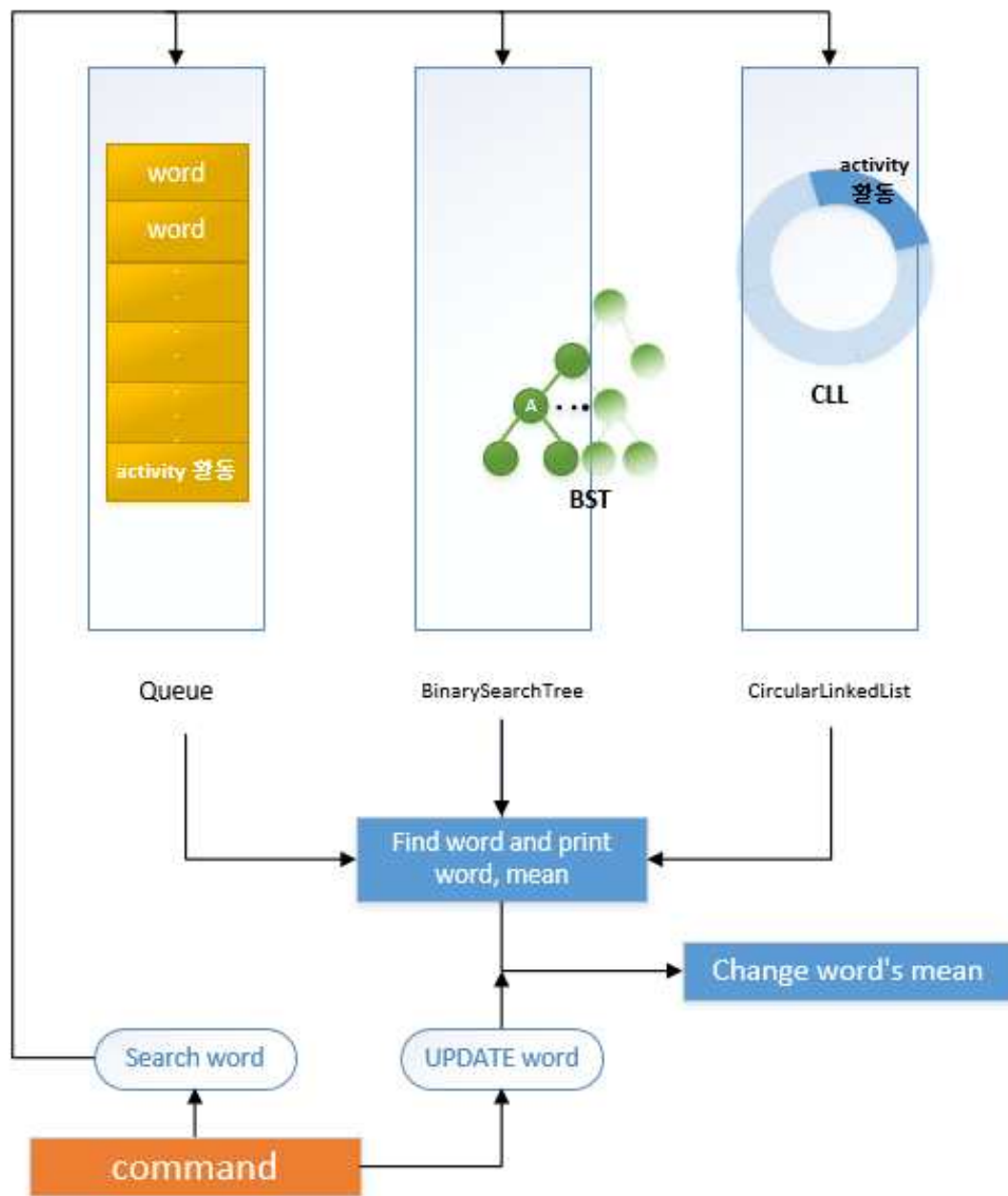
PRINT 명령어는 입력한 단어장에 있는 단어들을 출력하는 명령어로, 현재 외우고 있는 단어장 (Memorizing)에서는 출력방식을 선택해서 출력하고 나머지 단어장들은 저장된 단어장들을 그대로 출력한다.

Save Flow Chart



Save 명령어는 현재 단어장 정보를 저장하는 명령어로 외워야 할 단어장(To_Memorize), 현재 외우고 있는 단어장(Memorizing), 이전에 외운 단어장(Memorized)을 각각의 txt 파일에 저장한다.

SEARCH & UPDATE Flow Chart



Search 명령어는 단어의 뜻을 찾아 출력하는 명령어로 외워야 할 단어장(To_Memorize), 외우고 있는 단어장(Memorizing), 이전에 외운 단어장(Memorized)에 입력한 단어가 존재할 경우 영어 단어와 한글 뜻을 출력한다. Update 명령어는 단어의 뜻을 변경하는

명령어로 SEARCH명령어를 수행 후 단어의 한글 뜻을 입력한 뜻으로 변경하여 출력한다.

Algorithm

Queue

먼저 집어 넣은 데이터가 먼저나오는 FIFO 구조로 저장하는 형식

IsEmpty() Queue가 비어있는지 확인 해 준다.

PUSH PUSH는 front, rear형식인 HeadNode와 TailNode라는 방향을 지정해주고

TailNode방향으로 단어를 추가해준다.

POP MOVE명령어를 사용하면 To_Memorize에서 Memorizing으로 이동시키는 역할을 수행

Queue에서 POP은 임시 저장소를 만들어 두고, 삭제할 Node를 임시 저장소에 저장하고
current node를 delete해준다.

SEARCH strcmp함수를 사용해서 반복문을 통해 입력한 단어를 찾아

단어가 존재하면 영어 단어와 한글 뜻을 출력해준다.

PRINT "log.txt"파일을 current가 NULL이면 false반환한다.

그렇지 않으면 반복문을 통해서 "log.txt"파일의 단어와 단어의 의미를 출력하고

true 출력

SAVE 만약 Queue가 비어있다면 false를 반환하고, 데이터가 있을 경우 ofstream을 선언해

to_memorize_word.txt를 열어 반복문을 통해서 단어와 단어의 의미를 저장한다.

파일을 종료한다.

ccl의 주요 멤버 함수로 크게

Insert, Search, Save가 있다.

Insert는 LOAD, TEST 명령어에 사용되고

Search는 SEARCH, UPDATE 명령어,

Save는 SAVE 명령어에 사용된다.

Insert는 우선 넣고자 하는 단어를

search 함수를 통해 중복된게 있나 검사를 한 후,

중복이 없을 경우 원형 링크드리스트의 마지막부분을 while문을 통해

접근하여 마지막 node에 넣고자 하는 단어를 넣어주고, 그 단어가

NextNode로 pHead를 가리키게 한다.

Search는 문자열 비교함수 strcmp를 통해 노드의 word를 확인하고,

while문의 반복을 통하여 다음node로 넘어가는 과정을 반복하여

Next Node가 pHead일 때 까지 반복, 찾는 값이 있으면 current 반환하고

없으면 NULL을 반환

Save는 CLL이 비어있을경우 false를 반환하고

데이터가 있을 경우 ofstream을 선언, memorized_word.txt 파일에

while문을 통한 반복을 통하여 단어와 뜻을 line by line으로 nextnode가

pHead일때까지 반복하여 저장한다.

BST Algorithm

a. Insert

Root가 NULL일 경우는 입력 받은 node가 Root가 됩니다.

NULL이 아닐 경우 새로운 WordNode current을 생성해서 Root의 값을 처음에 대입 한 후 node를 탐색 하면서 입력 받은 node와 strcmp 크기비교를 합니다. 입력한 node의 값이 크고 오른쪽 자식이 없으면 오른쪽 자식에 대입을 하고 작고 왼쪽 자식이 없으면 왼쪽 자식에 넣습니다. 자식이 있다면 탐색을 계속 한 후 추가합니다.

b. Delete

WordNode p와 p의 부모를 담은 q을 선언합니다. P가 NULL이 아니고 p가 삭제할 node가 아닐 때 동안 q에 p를 담고 p와 삭제할 node의 단어를 비교해서 단어가 크다면 오른쪽 작다면 왼쪽으로 p를 이동시킵니다. p가 NULL이라면 return NULL을 해줍니다.

NULL이 아니라면 node의 자식이 2개있는 경우 오른쪽만 있는 경우 왼쪽만 있는 경우 하나도 없는 경우로 나눠 Delete를 실시합니다.

자식이 없는 경우 : 삭제할 node의 부모가 NULL이라면 root는 null이 됩니다.

아니라면 삭제할 node가 부모의 왼쪽 일 경우 부모의 왼쪽을 삭제할 node의 오른쪽과 연결합니다. 오른쪽 이라면 삭제할 node의 왼쪽과 연결합니다.

자식이 1개 인 경우: 왼쪽이 없을 경우 을 보면 삭제할 node의 부모가 NULL이면 root을 지우는 말이기 때문에 삭제할 node의 오른쪽이 root가 됩니다. 삭제할 node가 부모의 왼쪽자식이면 부모의 왼쪽을 삭제할 node의 오른쪽과 연결합니다 오른쪽이라면 부모의 오른쪽을 삭제 할 node의 오른쪽과 연결합니다. (왼쪽이 없기 때문에)

오른쪽이 없을 경우도 비슷한 알고리즘을 가지고 있습니다.

자식이 2개 인 경우: 자식이 2개인 경우는 전전 node을 담은 Word Node와 전 node을 담은 WordNode와 현재 node을 담은 WordNode 총 세개가 필요합니다.

현재 node가 NULL이 아닐 때까지 prevpre node는 pre 가 되고 prev node는 현재node. 현재node는 현재의 오른쪽 node가 됩니다. currnode가 NULL이 되면 del 이라는 새로운 WordNode 추가하고 이 node에 지울 node의 정보를 담습니다. 지울 node에 prev에 node 정보를 담고 prevprev가 삭제할 node라면 prevprev에 왼쪽에 prev의 오른쪽을 담습니다. 아니라면 prevprevnode의 오른쪽에 prev의 오른쪽node와 연결하고 prev는 삭제시킵니다. 담아뒀던 del의 정보를 return 시킵니다.

c. Search

새로운 WordNode current에 root을 담아서 current가 NULL이 아닐 동안 검색할

word와 current의 node의 단어 크기를 비교해서 word가 크다면 current node의 오른쪽으로 가고 작다면 왼쪽으로 갑니다. 그게 아니라면 그때의 node를 반환시킵니다.

d. Print

Parameter로 어떤 print을 할 지 정보를 넘겨 받은후 strcmp로 비교를 해 그에 따른 print 함수를 실행합니다

Recursive order는 함수의 재귀적 호출과 단어를 출력하는 위치에 따라 달리 구현 할 수 있습니다.

재귀적이 아니라 반복적 함수를 구현할 때는 stack과 queue을 사용합니다.

I_PRE 함수는 1개의 stack을 사용해서 while문 안에서 가장 위에서는 node에 top의 정보를 확인, 출력하고 오른쪽 자식부터 있으면 stack에 넣고 없으면 지나 갑니다 이 과정을 stack이 비어 있을 때 까지 반복합니다.

I_IN 은 반복문안에서 왼쪽자식이 있을 때 까지 node을 stack에 넣으면서 이동합니다. 그 후 current에 stack의 top값을 넣고 정보를 출력합니다. 추가적으로 pop을 합니다

I_POST 함수는 2개의 stack을 사용하고 먼저 child stack을 이용해서 후위 순위를 하며 parent stack에 정보들을 쌓아놓습니다. 그 뒤 parent stack이 empty가 될 때까지 pop을 하여 정보를 출력합니다.

I_LEVEL 함수는 레벨에 따라 출력하는 order로 먼저 root을 출력 한 후 stack 에 왼쪽의 자식을 넣고 그 후 오른쪽의 자식을 넣습니다 이 과정을 stack이 empty가 될 때 까지 반복합니다..

e. Save

Pre_save 라는 함수를 사용하여 저장합니다. Pre_save함수는 Pre_order 순서의 재

귀적 print 에서 save로 바꿔 준 함수입니다.

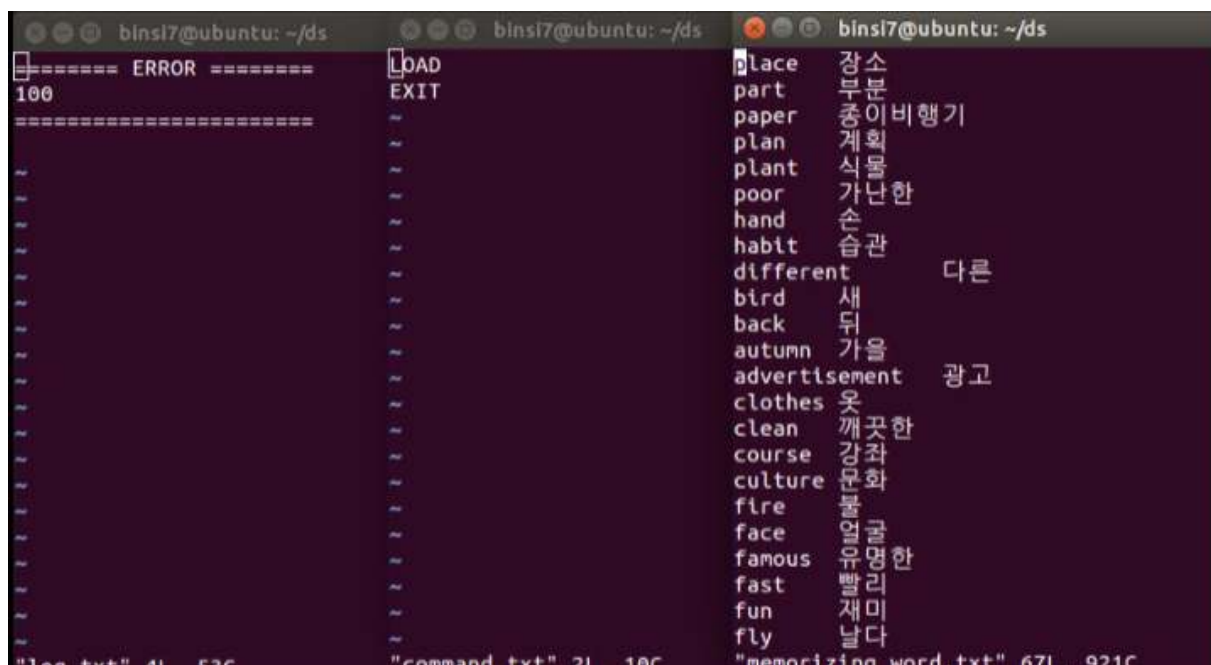
f. Stack

Iterative 에 사용할 stack을 추가적으로 구현합니다. 기본적인 동작은 같고 array 에서 linked list형식으로 바꿔주었으며 data로는 WordNode을 받습니다.

g. Queue

l_level order에 사용할 queue을 추가적으로 구현합니다. 기본적인 동작은 같고 array 에서 linked list형식으로 바꿔주었으며 data로는 WordNode을 받습니다

Result Screen



```
binsl7@ubuntu: ~/ds
===== ERROR =====
100
=====

binsl7@ubuntu: ~/ds
LOAD
EXIT

binsl7@ubuntu: ~/ds
place 장소
part 부분
paper 종이비행기
plan 계획
plant 식물
poor 가난한
hand 손
habit 습관
different 다른
bird 새
back 뒤
autumn 가을
advertisement 광고
clothes 옷
clean 깨끗한
course 강좌
culture 문화
fire 불
face 얼굴
famous 유명한
fast 빨리
fun 재미
fly 날다
"log.txt" 41 53C
"command.txt" 21 10C
"memorizing_word.txt" 671 921C
```

LOAD 명령어 수행 결과, ERROR code 100 출력

memorizing_word.txt 파일에 이미 단어가 존재하여 LOAD 명령어 수행시 ERROR 발생

```
binsi7@ubuntu: ~/ds
===== ERROR =====
100
=====
===== ADD =====
Success
=====

"log.txt" 8L, 107C

binsi7@ubuntu: ~/ds
LOAD
ADD
EXIT

"command.txt" 3L, 14C
```

ADD 명령어 수행 결과, Success 출력

word.txt 파일에 있는 단어들을 외워야 할 단어장(to_memorize)에 모두 저장

```
binsi7@ubuntu: ~/ds
===== ERROR =====
100
=====
===== ADD =====
Success
=====
===== MOVE =====
Success
=====

"log.txt" 12L, 163C

binsi7@ubuntu: ~/ds
LOAD
ADD
MOVE 50
EXIT

"command.txt" 4L, 22C
```

MOVE 50 명령어 수행 결과, Success 출력

입력한 숫자 50만큼 to_memorize의 단어들을 memorizing으로 옮기는 동작 수행

```

===== ERROR =====
100
=====
===== ADD =====
Success
=====
===== MOVE =====
Success
=====
===== ERROR =====
300
=====

"log.txt" 16L, 216C

LOAD
ADD
MOVE 50
MOVE 51
EXIT
"command.txt" 5L, 30C

```

MOVE 50명령어 수행 후 MOVE 51 명령어 수행 결과, ERROR code 300 출력

Memorizing 단어장에 입력 받은 수의 합이 100을 넘어가게 되어서 오류가 발생

```

===== PRINT =====
activity 활동
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
letter 편지
life 인생
light 빛
listen 듣다
lot 많이
make 만들다
mind 마음
movie 영화
name 이름
newspaper 신문
paper 종이
part 부분
person 사람
place 장소
plan 계획
plant 식물
problem 문제
same 같음
space 공간
spring 봄
start 시작
story 이야기
street 거리
summer 여름
trip 여행
vacation 휴가
visit 방문
volunteer 자원봉사자
watch 시계
win 이기다
winter 겨울
work 일

LOAD
ADD
MOVE 50
PRINT MEMORIZING R_IN
EXIT
"command.txt" 5L, 44C
1,1 All

```

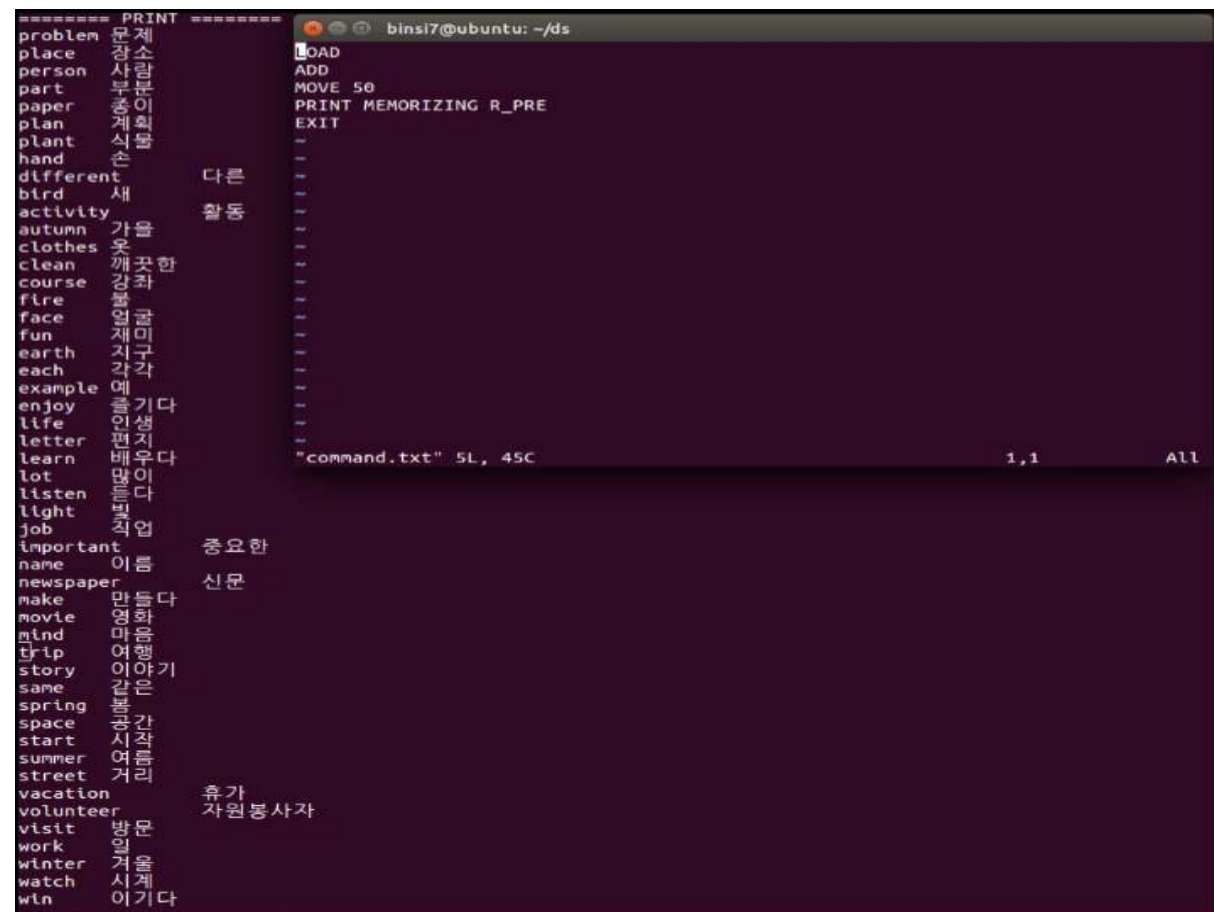

PRINT MEMORIZING R_IN 명령어 수행 결과 화면

현재 외우고 있는 단어장(Memorizing)의 단어들을 재귀함수를 이용한 in-order방식으로 출력



```
summer 여름
street 거리
vacation 휴가
volunteer 자원봉사자
visit 방문
work 일
winter 겨울
watch 시계
win 이기다
=====
===== TEST =====
Pass
=====
===== TEST =====
Pass
=====
===== TEST =====
Pass
=====
```

TEST명령어 수행 결과, Memorizing의 단어와 일치하여 결과 Pass를 출력



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

PRINT MEMORIZING R_PRE 명령어 수행 결과 화면

현재 외우고 있는 단어장(Memorizing)의 단어들을 재귀함수를 이용한 pre-order방식으로 출력

```
binsi7@ubuntu: ~/ds
visit 방문
work 일
winter 겨울
watch 시계
win 이기다
=====
===== TEST =====
Pass
=====
===== TEST =====
Pass
=====
===== TEST =====
Pass
=====
===== SEARCH =====
problem 문제
=====
"command.txt" 9L, 120C 1,1 A'
```

SEARCH 명령어 수행 결과, 외워야 할 단어장(To_memorize), 현재 외우고 있는 단어장 (Memorizing) , 이미 외운 단어장(Memorized)에 입력한 단어가 존재할 경우 영어 단어와 한글 뜻을 출력.

```
binsi7@ubuntu: ~/ds
job 직업
important 중요한
name 이름
newspaper 신문
make 만들다
movie 영화
mind 마음
trip 여행
story 이야기
same 같은
spring 봄
space 공간
start 시작
summer 여름
street 거리
vacation 휴가
volunteer 자원봉사자
visit 방문
work 일
winter 겨울
watch 시계
win 이기다
=====
===== TEST =====
Pass
=====
===== ERROR =====
500
=====
-- INSERT --
```

TEST명령어 수행 결과, Memorizing의 단어와 일치하지 않아 ERROR CODE 500 출력

```

===== TEST =====
Pass
=====
===== TEST =====
Pass
=====
===== TEST =====
Pass
=====
===== SEARCH =====
problem 문제
=====
===== PRINT =====
problem 문제
person 사람
activity 활동
=====
"command.txt" 10L, 136C
1,1 A

```

PRINT MEMORIZED명령어 수행 결과, TEST 명령어를 수행하여 Pass되어 이미 외운 단어장 (Memorized)에 저장된 단어들을 출력.

Consideration

프로젝트에 대한 고찰:

이름	프로젝트 에서 맡은 역할	본인 스스로 생각하는 자신의 점수(10)
이 전제	Memorizing의Alphabet BST, Word BST , Manager, 보고서의 algorithm	6점
고찰	<p>프로그램이 window에서 제대로 동작하는 것을 확인하고 Linux 에서 다시 make file로 실행을 했더니 많은 오류가 생겼습니다 strlwr 함수를 사용한 점 불완전하게 char * = "abcd" ; 을 사용한 점 등을 비롯해 make 파일이 오류를 많이 출력했습니다.</p> <p>또한 make파일의 오류를 고친 후 검증을 시작하는 과정에서 입력 값이 원하는 대로 출력하지 않아 디버깅하는데 많은 시간이 걸렸습니다.</p> <p>Window 에서는 문제없이 돌아가는 부분이 Linux에서는 아예 돌아가지 않았습니다 예를 들어 Manager 함수에서 명령어를 비교하고 가장 마지막으로 명령어가 해당되는 부분이 없으면 else 구문으로 들어가는 부분에서 오류가 생겨서 처음 디버깅할 때 명령어 실행조차 시작하지 못하였습니다. 후에 분명 Linux에서 바꿨다고 생각했던 메모장 파일들이 바뀌지 않아서 제대로 원하는 값이 도출이 안된 점과 window는 허용했지만 Linux에서는 허용하지 않는 BST의 Insert와 delete 코딩 때문에 test와 move search에서 지속적인 에러가 생겼고 도움과 정보를 찾아서 고쳤습니다.</p> <p>저의 코딩 설계가 심플하지 않은 점과 윈도우에서만 허용하고 Linux에서는 허용하지</p>	

	<p>많은 저의 코딩점수에 이번 프로젝트에서는 자신의 점수를 6점이라고 생각하지만 Linux사용방법등과 왜 디버그가 나가는지 어디서 어떻게 확인을 해야 하는지 경험을 해서 얻은 것도 많은 프로젝트였습니다.</p>	
이민형	MEMORIZED의 CircularLinkedList 구현 flow chart design	5점
고찰	<p>이번 프로젝트는 제가 해본 프로젝트중에 가장 큰 프로젝트였는데, 고급 프로그래밍을 미이수한것과 다른없는 저에게는 무척 넘기 힘든 산과 같았습니다..</p> <p>모르는것이 너무많고 내가 저 자료구조들을 다 구성해서 처리할 수 있을까 하였지만 팀프로젝트여서 좋았던점이 이런 제가 모르는부분들에 대해서 하나하나 물어보며 팀끼리 도울수 있고</p> <p>개인적으로는 팀에 많은기여를 하지는 못했을지라도 팀프로젝트여서 좋았습니다. 혼자했으면 우선 포기부터 했을것 같은데</p> <p>이렇게 어떻게 또 하게 되어서 조원들에게 감사합니다. MEMORIZED의 CircularLinkedList의 구현에 있어서 (이하 CLL), CLL 헤더파일과 cc파일의 구성만 하였는데 코드 확인에 있어 팀장의 도움을 많이 받고 Manager.cc 파일에서의 CLL이 활용되는 부분에 있어서는 제 힘이 닿지 못했습니다.</p> <p>같이 하면서 짜여진 프로그램을 보며 다양한 헤더파일과 클래스들이 서로 엮이고 엮여 하나의 프로그램을 이루는것에 프로그래밍이 쉽지 않음을 느꼈고</p> <p>많은 노력이 필요하다는, 잘하는사람도 많은 실패와 경험에서 잘하는것이라는걸 느낀 좋은 계기가 되었습니다.</p>	
최민기	TO_MEMORIZE의 Queue 구현 Introduction	4점
고찰	<p>명확히 자료구조에 대해서 개념이 잡혀 있지 않아서 이번 프로젝트에 대해서 감을 잡는데 많은 어려움이 있었습니다. 하지만 이번 프로젝트는 팀프로젝트로 진행되어서 마음이 맞는 팀원과 함께 해서 명확하지 않은 부분을 서로 도와주며 같이 진행 해 좋은 결과를 얻을 수 있었습니다.</p>	