

# Data Structure (영어단어 암기 프로그램)

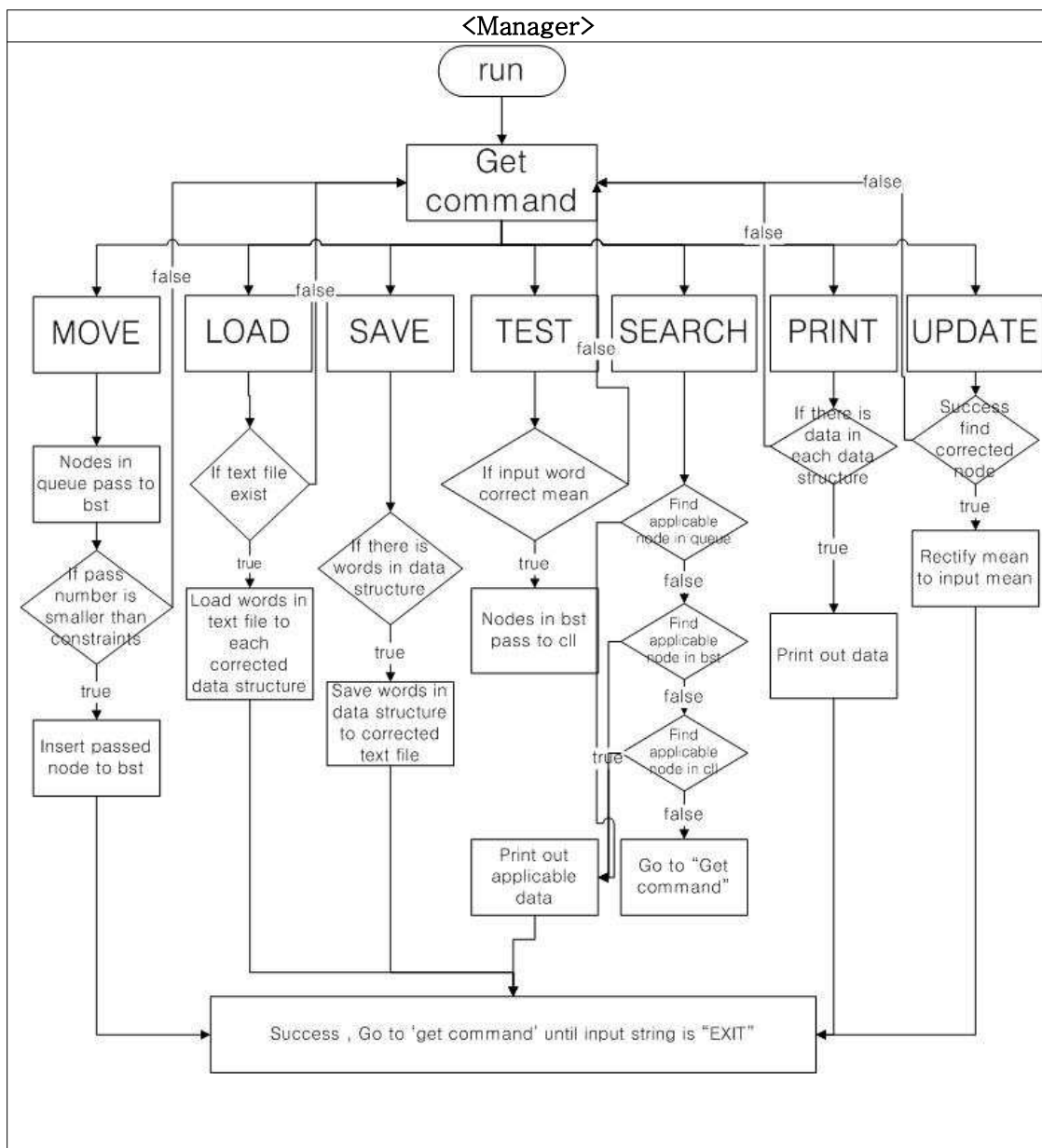
## Project #1

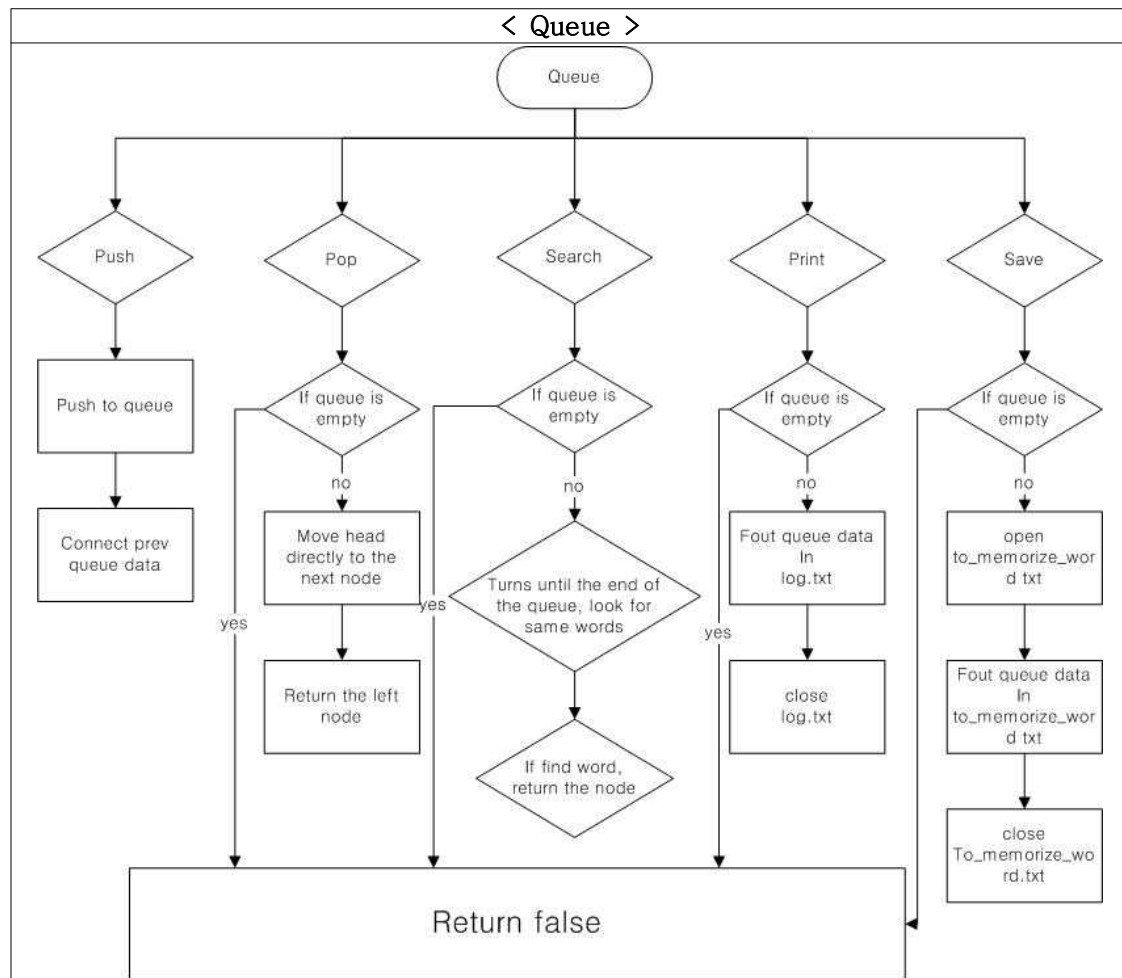
Professor	이 기훈
Department	Computer engineering
Student ID	2012722001
	2012722009
	2013722061
Name	김 기선
	김 지웅
	이 호영
Date	2016. 10 . 07

## -Introduction

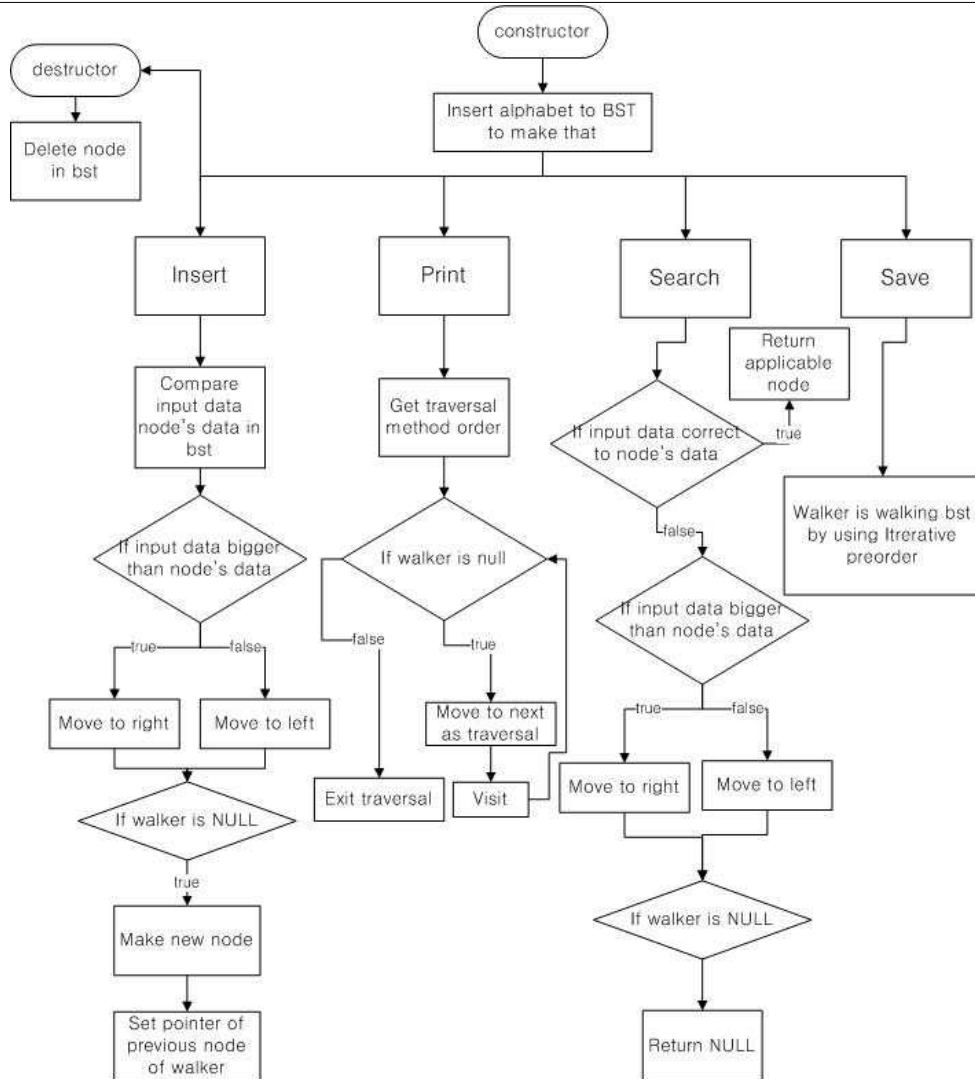
이번에 하는 프로젝트는 단어장 암기 프로그램으로 외워야할 단어장( TO\_MEMORIZE )에 있는 단어들을 현재 외우고 있는 단어장( MEMORIZING )으로 넘기고 TEST를 통하여 암기한 단어를 이전에 외운 단어장 ( MEMORIZED )로 넘기는 프로그램입니다. TO\_MEMORIZE는 Queue로 구현하고, MEMORIZING은 BST로, MEMORIZED 는 Circular Linked List로 구현한다.

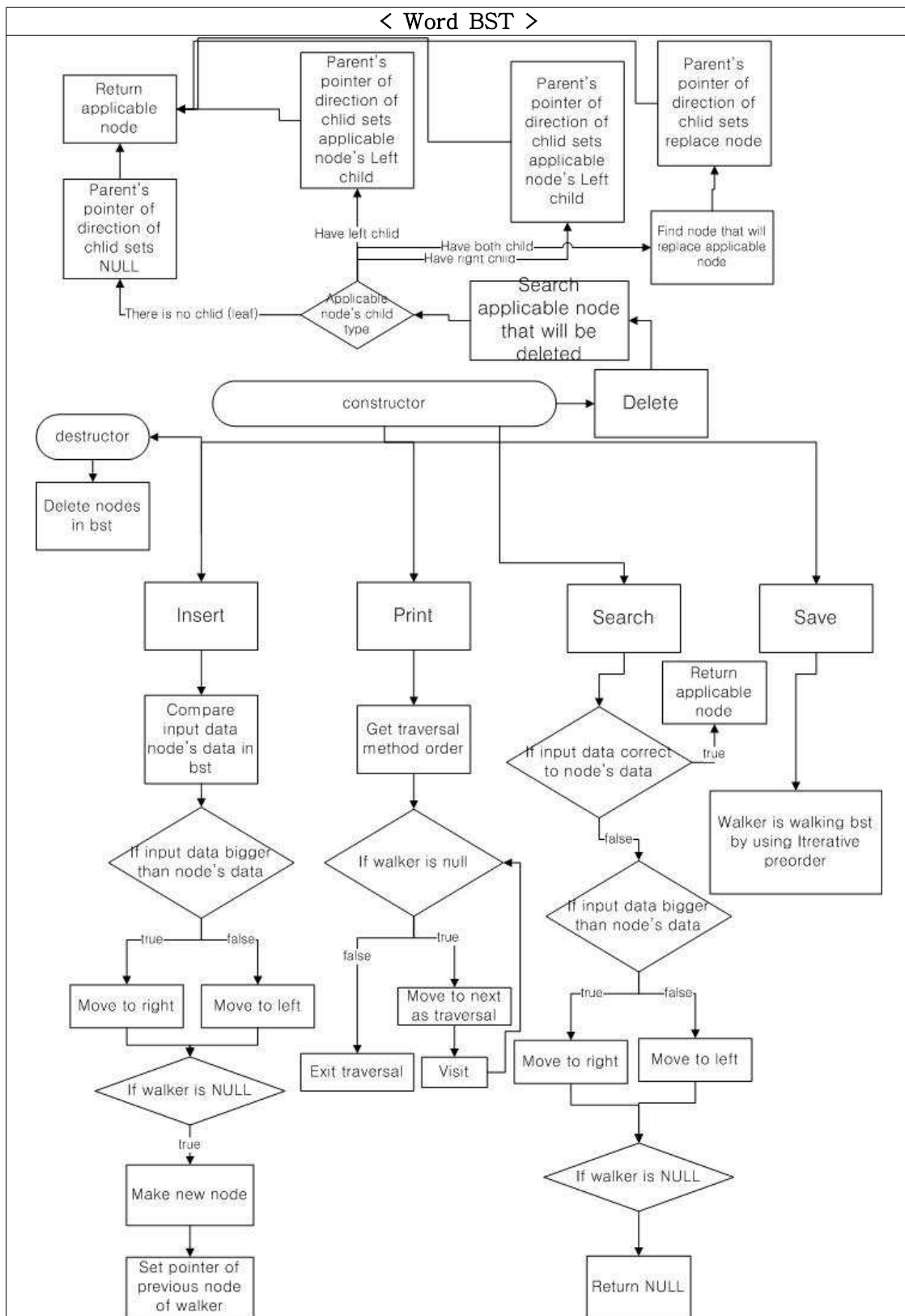
## -Flowchart



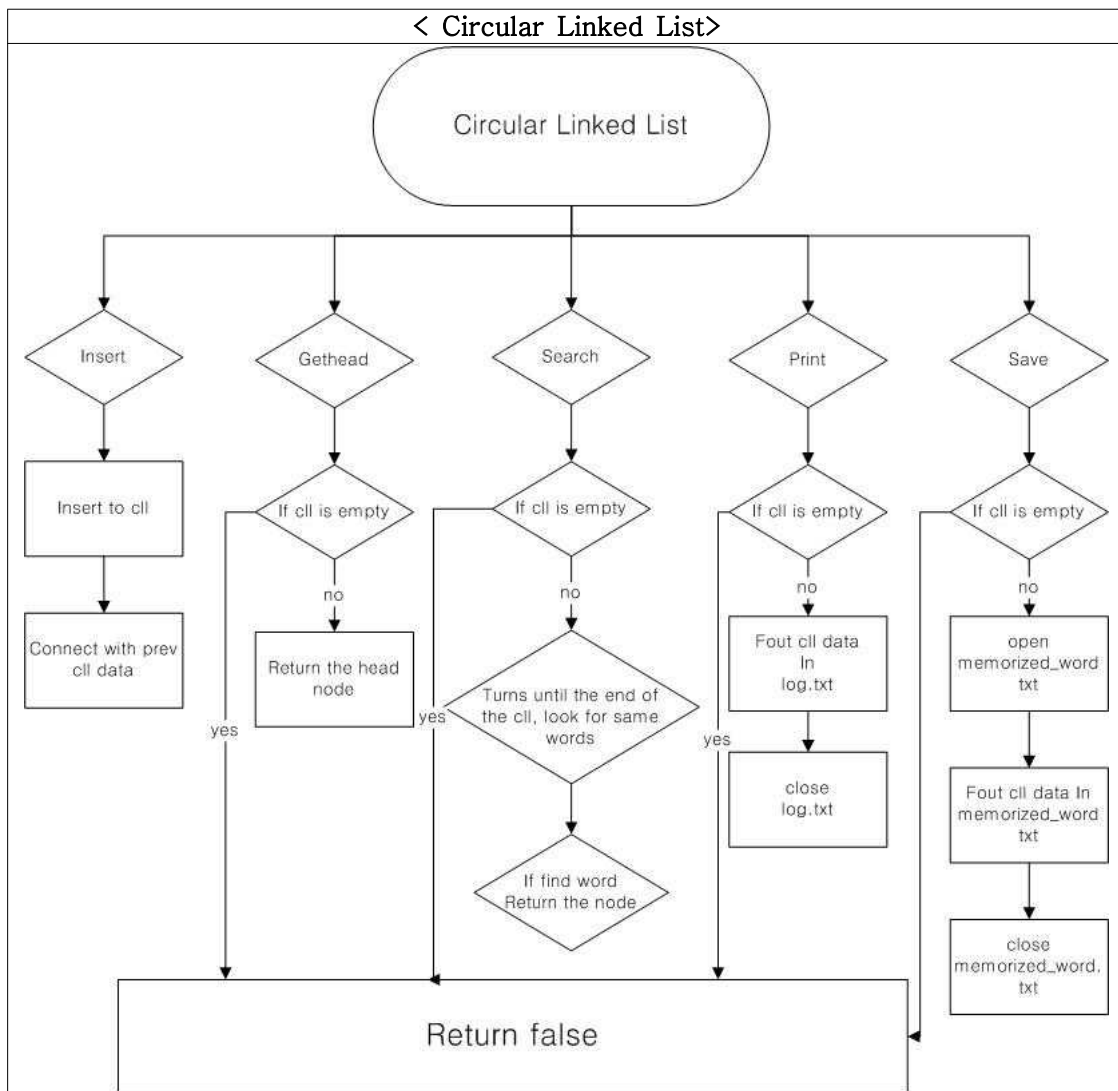


# < Alphabet BST >









## -Algorithm

### ●MANAGER

#### run

메인함수에서 호출하는 함수로 command를 입력받아 다른 함수들을 관리하는 함수이다.

if 문을 사용하여 command에 따라 함수를 호출하고 그에 따른 함수의 반환 값을 반환 받아 에러인지 아닌지 구분하고 그에 따른 메시지를 출력한다.

또한 command 자체에 오류가 있는지 확인하고 그에 따른 에러코드를 출력한다.

## LOAD

세 개의 자료형중 하나라도 이미 자료가 있으면 load를 진행하지 않고 에러 코드를 출력하고, load후에 하나라도 자료형이 비어있으면 에러코드를 출력한다.

일단 세 개의 텍스트 파일을 모두 open하고 3개의 자료형에 자료가 있는지 확인하여 하나라도 자료가 있다면 load를 진행하지 않고 false를 반환한다.

3개의 자료형에 자료가 모두 없다면 queue, BST, CLL의 load를 호출하여 load를 진행한다.

모두 load한 뒤에 3개의 자료형이 비어있는지 확인하고 하나라도 자료형이 비어있다면 false를 반환한다.

위에서 false가 반환되지 않으면 true를 반환하고 함수를 종료한다.

## ADD

word.txt의 단어들을 queue에 저장하는 함수

word.txt를 열고 getline을 이용하여 한 줄씩 읽는다. 한 줄씩 읽을 때마다 tab을 기준으로 token하여 단어와 뜻을 따로 구분한다.

구분한 단어와 뜻을 pair를 사용하여 queue에 push한다.

처음 파일을 열 때 파일이 없다면 false를 반환하고 아니라면 위의 과정을 수행한다.

위의 모든 과정을 수행하면 true를 반환하고 함수를 종료한다.

## MOVE

queue에 있는 단어들을 BST로 넘기는 함수로 100개 이상은 넘길 수 없고, BST에 100개 이상의 단어가 저장되면 안된다.

move의 조건에 따라 if를 사용하여 예외처리를 하고

## SAVE

3개의 자료형중 하나라도 자료형이 있으면 성공하게 된다.

그것을 확인하기 위해서 3개의 자료형 모두 SAVE를 하고 3개의 자료형이 모두 비어있으면 false를 반환하여 오류메시지를 출력한다.

그 외의 경우는 모두 true를 반환한다.

## TEST

BST 자료형을 SEARCH하여 단어가 있는지 찾고 텍스트 파일의 뜻과 일치하는지 확인하여 일치하면 CLL로 넘기고 그렇지 않다면 error메세지를 출력한다.

## **SEARCH**

단어를 입력받고 3개의 자료형을 모두 돌며 일치하는 단어가 있는지 확인하고 뜻과 함께 출력하는 함수.

queue부터 search함수를 호출하여 일치하는 단어가 있는지 보고 있다면 그 단어의 뜻과 함께 출력하고 없으면 BST로 넘어간다.

BST에서 search함수를 호출하여 일치하는 단어가 있다면 그 단어의 뜻과 함께 출력하고 없으면 CLL로 넘어간다.

CLL에서 search함수를 호출하여 일치하는 단어가 있다면 그 단어의 뜻과 함께 출력하고 없으면 false를 반환한다.

## **PRINT**

command에서 입력받은 명령어를 stack을 이용하여 저장하고 그것에 따른 print 형태에 따라 출력한다.

MEMORIZE = queue에 있는 단어들을 출력한다.

MEMORIZING = BST의 print함수로 들어가서 명령에 입력에 따른 order방식으로 출력한다.

MEMORIZED = CLL에 있는 모든 단어들을 출력한다.

## **UPDATE**

command에서 입력받은 명령어를 stack을 이용하여 저장하고 그것을 search하여 일치하는 단어가 있는지 확인하고 일치하는 단어가 있다면 뜻을 입력 명령어로 바꾸어준다.

만약 일치하는 단어가 없다면 false를 반환한다.

## ●queue

### **-constructor**

pHead와 wordcnt를 생성과동시에 초기화하였다.

### **-destructor**



만약 queue에 따로 데이터가 없으면 소멸을 따로 시켜주지 않았고, 자료구조 queue를 순회하기위해서 pCur이라는 주소값을 가지는 변수를 선언하고, 순회함과 동시에 데이터를 소멸시키기위해 주소값을 가지는 변수 pTemp를 선언하였다. pCur에는 pHead의 주소값을 넣어주었고 pTemp에는 pCur의 주소값을 넣어주었다. while문을 이용하여 pCur을 처음부터 끝까지 돌면서 pCur이 해당 자료구조에 들어갈때마다 그 주소값을 pTemp에 넣어주고 그 pTemp를 delete시키는 식으로 queue자료구조를 소멸하였다.

큐(TO\_MEMORIZE)에서는 5가지의 함수를 사용한다.  
Push, Pop, Search, Print, Save이다.

### **Push**

word.txt 즉 처음 주어진 단어장이다. 이단어장에 있는 정보들을 노드의 형태로 받아 큐에 집어넣는다. 이때 노드를 처음 받을때는 큐의 가장 앞에있는 노드 즉 헤드를 처음받은 노드로 지정한다. 계속 노드를 push 하면 가장 마지막에 추가했던 노드를 찾기위해 while문으로 가장마지막인 노드를 찾았고 그 노드의 바로 다음에 연결하여 push한다. 이때, 이전에 있었던 단어정보를 가지고있는 노드를 받으면 에러코드를 띄우지않고 추가하지 않는다. 이때 이전에 받았던 단어인지 확인하기 위해서 strcmp를 사용하였다.

### **pop**

pop은 큐의 성질에 의해 헤드를 반환하는 함수이다. 일단, 큐자체가 존재하지 않으면 false를 리턴하였고, 헤드를 반환하기위해 pCur이라는 주소값을 가지는 변수를 선언하였고, 여기에 헤드의 주소값을 넣었다. 헤드를 연결되어 있는 바로 옆 노드로 주소값을 바꾸었고 아까 만들었던 pCur의 다음노드와의 연결고리를 NULL로 초기화해준뒤 pCur을 반환한다.

### **Search**

Search는 queue의 자료구조안에 찾는 단어가 있으면 그노드를 반환하는 함수이다. 일단 큐자체가 존재하지 않으면 false를 리턴하였고, pCur이라는 큐를 도는 변수하나를 선언한뒤 while문을 돌려서 queue 자료구조를 탐색하였다. 이때 strcmp를 사용하여서 찾는단어와 현재 pCur이 있는 노드의 단어가 일치하면 그 노드를 반환하고 아닐시에는 pCur이 그다음노드로 이동하는식

으로 구현하였다.

### **Print**

Print는 queue의 자료구조안에 있는 단어와 그 뜻을 log.txt 에출력하는 함수이다. 먼저 log.txt를 쓰기전용으로 ofstream하였다. queue를 돌기위해서 pCur이라는 주소값을 가지는 변수를 선언하였고 큐에저장되어있는 노드들을 차례대로 순회하면서 그 노드 안에있는 단어와 뜻을 log.txt에 출력하였다. 끝까지 출력한뒤 파일을 닫았다. 이때 queue안에 아무런데이터가 없다면 false를 return하였다.

### **Save**

Save는 queue안에있는 노드들의 데이터를 to\_memorize\_word.txt에 출력하는 함수이다. 먼저 queue안에 아무런데이터가 없다면 false를 return하였다. save를하기위하여 먼저 to\_memorize\_word.txt를 쓰기전용으로 ofstream하였다. queue를 돌기위해서 pCur이라는 주소값을 가지는 변수를 선언하였고 queue의 처음부터 끝까지 while문을 사용하면서 순회하였다. 이때 다음노드로 움직일때마다 그 노드안에 있는 뜻과 단어들을 getword와 getmean을 사용하여 to\_memorize\_word.txt에 출력하였다.

## ● Alphabet BST

### **-constructor**

생성자에서는 알파벳을 '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', 'W' 순으로 배열에 저장한 뒤, 반복문을 사용해 차례대로 member함수인 Insert 함수를 호출하며 최초 생성시 Alphabet으로 구성된 binary tree를 만듭니다.

### **-destructor**

소멸자에서는 Iterative Postorder를 사용하여 노드의 동적 할당을 해제합니다. 이 때, Iterative Postorder를 사용하는 이유는 뒤의 node부터 차례로 접근하며 그 노드가 동적 할당이 해제 되었을 때, 연결이 중간에서 끊기면 안되기 때문입니다. 접근하여 동적 할당을 해제할 때, 노드에 포함되어있는 word bst를 방문하여 그 안의 bst를 해제하며, 진행합니다.

### **-Insert**

pCur노드는 bst를 걷는 walker역할을 합니다. 이는 처음 노드를 넣을 때, 넣을 위치를 찾는 역할을 합니다. pCur은 root부터 시작하며 값을 비교했을 때, pCur의 data가 새로 만든 노드의 데이터 값보다 크면 오른쪽으로 이동하고 작다면 왼쪽으로 이동합니다. 만약, 값이 같다면 이는 이미 중복되는 데이터가 존재하는 것이기 때문에 insert를 실패했다고 볼 수 있습니다. 따라서 null(false)값을 반환합니다. pCur이 NULL에 도착했다면 그 곳이 새로운 데이터가 저장되어질 위치입니다. 따라서 pCur의 이전 노드인 pPrev의 다음 노드를 새노드로 설정해 줍니다.

### **-Print**

print 함수는 총 7개의 순회order를 가지고 있습니다. 종류로는 preorder, inorder, postorder, level order가 있으며 preorder와 inorder, postorder는 각각 재귀함수를 사용한 Recursive 형식과 반복문을 사용한 Iterative형식을 가지고 있고, level order는 iteration한 방법으로 구현하여 총 7개의 순회방법을 가지고 있는 것입니다. 이 때 만약 해당노드의 visit은 해당 노드가 가지고있는 bst의 print함수를 방문하며, 명령어 또한 전달합니다. 아래는 각 order의 알고리즘을 설명합니다.

### **-traversal**

#### **-Recursive preorder**

재귀를 사용한 preorder는 간단합니다. Preorder는 먼저 root를 방문하고 왼쪽 subtree를 방문하고 오른쪽 subtree를 방문합니다. 때문에 먼저 walker node를 visit하고 함수에 walker node의 left child를 넣어주고 이를 다 방문이 마친다면 다음으로 right child를 재귀함수에 넣어줍니다.

#### **-Iterative preorder**

반복문을 사용한 preorder는 stack이 필요합니다. 먼저 루트를 방문해야되기 때문에 walker를 반복문에 들어가기 전에 pCur를 stack에 push합니다. pCur은 stack의 top을 받습니다. 그리고 visit을 한 뒤 pop을 동작합니다. 그러면 root가 먼저 visit이 됩니다. 그리고 만약 pCur의 right child가 존재한다면 stack에 먼저 push를 한 뒤, left child의 존재 여부를 따집니다. 만약 left child가 존재한다면 다음으로 stack에 삽입을 합니다. 먼저 right child를 삽입했기 때문에 stack자료구조 안에서는 left가 먼저 visit이 되게 됩니다.

#### -Recursive inorder

Inorder는 왼쪽 subtree를 방문하고 root를 거쳐 오른쪽을 방문합니다. 때문에 재귀를 사용할 때, left child를 먼저 재귀함수에 넣고, visit을 한 뒤 right child를 재귀함수에 넣습니다.

#### -Iterative inorder

반복문을 사용한 inorder는 stack이 필요합니다. 먼저 왼쪽으로 이동하며 stack에 push를 합니다. 만약 walker가 null이 나오면 이중 반복문 속 반복문은 break됩니다. 그리고 pCur은 stack의 top을 받고 먼저 pop을 한 뒤, visit을 진행하게 됩니다. 그리고 right child로 이동하며 그안에서 또 왼쪽으로 끝까지 이동하며 push를 진행합니다. 만약 stack이 비었다면, 모든 노드를 visit한 것이기 때문에, 함수를 종료합니다.

#### -Recursive postorder

Postorder는 왼쪽 subtree를 가장 낮은 level부터 방문하여 오른쪽 subtree를 방문하고 root를 방문합니다. 때문에 먼저 left child를 재귀함수에 넣어 호출한 뒤, right child를 재귀함수에 넣어 호출하고, 마지막에 visit을 수행합니다.

#### -Iterative postorder

반복문을 사용한 post order는 먼저 push를 진행한다. pCur은 right child로 이동을 하며 끝까지 push를 한다. 먼저 오른쪽으로 이동하는 이유는 stack이 first in last out자료 구조이기 때문이다. 그리고 left child를 받는다. 끝까지 이동시 이중 반복문은 깨진다. 그리고 pCur은 stack의 top을 받고 pop을 진행한다. 그리고 만약 pCur의 오른쪽 node가 NULL이 아니고 right child와 stack의 top이 일치한다면, stack은 겹치기 때문에 pop을 먼저 진행한뒤 오른쪽으로 이동하며 pCur을 push하게 된다. 만약 그것이 아니라면 pCur은 visit을 수행하고 반복을 계속 진행하게 된다. 만약 stack이 empty상태라면 모든 node에 방문한 것이기 때문에 반복문을 깨고 함수를 종료하게 된다.

#### -Iterative Level order

Level order는 상위 Level인 root부터 가장 하위 Level의 leaf까지 level순으로 visit을 하는 순회 방법이다. iterative level order는 다른 순회와 다르게

queue자료구조를 사용한다. 반복은 walker인 pCur이 NULL에 다다를 때 까지 반복되며, 먼저 visit을 진행한 뒤, 먼저 만약 left child가 존재하면 left child를 queue에 push하고 만약, right child가 존재한다면 right child를 push합니다 그리고 이 조건을 통과하면, pCur은 queue의 front를 받아오고 pop을 진행하게 됩니다. 만약 queue가 empty상태면 순회는 종료됩니다.

### **-Search**

입력받은 data를 tree를 돌며 일치하는 data를 가진 node가 있는 지 찾습니다. 만약 input data가 현재 node의 data보다 값이 크다면 오른쪽으로, 작다면, 왼쪽으로 이동합니다. 만약 data를 비교했을 때, 같다면, search는 성공적으로 이루어지게 됩니다. 만약 pCur이 null까지 같다면 일치하는 data가 존재하지 않는다고 볼 수 있기 때문에 null값을 반환합니다.

### **-Save**

Iterative preorder의 순회방법으로 node에 visit합니다. 왜냐하면 preorder를 사용하지 않으면, 다음에 load를 할 때, data가 한 쪽으로 쏠려 skewed tree가 될 수 있기 때문입니다. 위에서 설명한 preorder의 방법으로 돌지만 visit 함수만 함수 오버로딩을 사용하여 data를 print할 때 스트림을 file out스트림으로 출력하게 합니다.

## ● Word BST

alphabet bst의 기본 알고리즘과 동일하기 때문에 생략하고 차이인 delete 함수만 설명합니다.

### **-constructor**

가지고 있는 member를 초기화 해줍니다.

### **-destructor**

소멸자에서는 Iterative Postorder를 사용하여 노드의 동적 할당을 해제합니다. 이 때, Iterative Postorder를 사용하는 이유는 뒤의 node부터 차례로 접근하며 그 노드가 동적 할당이 해제 되었을 때, 연결이 중간에서 끊기면 안 되기 때문입니다.

### **-Delete**

delete함수는 먼저 지울 노드의 data를 입력받고 해당 노드가 있는지를



search 합니다. 반복문을 통해 진행하며 prev walker를 두어 parent node를 기억합니다. 그리고 네 가지의 경우를 따져 replace를 진행합니다. 경우는 지을 node가 자식이 없는 경우(leaf node인 경우), left child를 가진 경우, right child를 가진 경우, 양쪽 child를 모두 가진 경우가 있다. 먼저 leaf노드인 경우 만약 해당노드가 head라면 root pointer만 NULL로 설정 해주면 된다. 아니라면, 만약 해당노드의 부모의 해당노드가 왼쪽자식인지, 오른쪽 자식인지를 따져 NULL을 설정해줍니다. 한 가지 노드를 자식으로 갖는 경우는 그 자식인 노드를 replace하면 됩니다. 왼쪽 node를 가졌다면 부모 node의 해당 자식노드의 방향 포인터를 왼쪽 node로 설정하고 오른쪽이라면 오른쪽 node를 설정해주면 됩니다. 만약 양쪽 자식을 모두 갖는 경우라면, 둘 중 한 가지 방법을 사용하면 됩니다. 두 가지 방법에는 먼저 대체 노드는 왼쪽 해당 노드의 왼쪽 subtree의 노드중 가장 큰 data를 갖는 노드를 대체하는 것이고 다른 방법은 오른쪽 subtree의 노드중 가장 작은 data를 갖는 노드로 대체하는 것입니다. 저는 후자의 방법을 택했습니다. 먼저 세 개의 포인터를 선언합니다. 모두 walker이며 현재 walker인 pCur과 이를 뒤따라오는 prev 그리고 이 prev를 뒤따라 오는 prevprev가 선언됩니다. pCur은 먼저 오른쪽 노드로 이동한 후, 왼쪽으로 이동하며 null이 나올 때 까지 이동을 합니다. 만약 NULL이 나왔다면 prev 포인터가 바로 대체되어질 node입니다. 대체는 data만 해당노드에 넣어 주고, prev노드를 delete해주게 되면 됩니다. 이 때, 만약 prev노드가 오른쪽 자식을 가지고 있다면 그것을 prevprev와 연결 시켜주면 됩니다.

## ●Circular Linked List

### -constructor

pHead와 생성과 동시에 초기화하였다.

### -destructor

만약 cl에 따로 데이터가 없으면 소멸을 따로 시켜주지 않았고, 자료구조 cl을 순회하기위해서 pCur이라는 주소값을 가지는 변수를 선언하고, 순회함과 동시에 데이터를 소멸시키기위해 주소값을 가지는 변수 pTemp를 선언하였다. pCur에는 pHead의 주소값을 넣어주었고 pTemp에는 pCur의

주소값을 넣어주었다. while문을 이용하여 pCur을 처음부터 끝까지 돌면서 pCur이 해당 자료구조에 들어갈때마다 그 주소값을 pTemp에 넣어주고 그 pTemp를 delete시키는 식으로 cll자료구조를 소멸하였다.

CLL ( MEMORIZED ) 에서는 5가지의 함수를 사용한다.  
Push, Pop, Search, Print, Save이다.

### **Insert**

Insert는 BST에서 TEST 한 단어들을 Circular Linked List자료구조에 insert 하는 함수 이다. 이때 test한 단어들을 노드의 형태로 받아 cll에 집어넣는다. 이때 노드를 처음 받을때는 cll의 가장 앞에있는 노드 즉 pHead를 처음들어온 노드로 지정한다. 계속 노드를 insert 하면 가장 마지막에 추가했던 노드를 찾기위해 while문으로 가장마지막인 노드를 찾는데 이때 두가지 조건이 있다. 첫 번째는 pCur의 다음노드는 NULL이 아니어야되고 또한 phead도 아닐때까지 반복한다. 마지막 노드를 찾았으면 그 노드의 바로 다음에 연결하여 insert한다. 이때 마지막으로 추가한 노드를 phead와 연결하여 circular linked list의 형태를 갖춘다.

### **Gethead**

Gethead는 간단한 함수이다. Circular Linked List 의 헤드를 반환하는 함수이다.

### **Search**

Search는 cll의 자료구조안에 찾는 단어가 있으면 그노드를 반환하는 함수이다. 일단 cll자체가 존재하지 않으면 false를 리턴하였고, pCur이라는 큐를 도는 변수하나를 선언한뒤 while문을 돌려서 cll 자료구조를 탐색하였다. 이때 strcmp를 사용하여서 찾는단어와 현재 pCur이 있는 노드의 단어가 일치하면 그 노드를 반환하고 아닐시에는 pCur이 그다음노드로 이동하는식으로 구현하였다. 이때 순회하는 pCur이 다시 phead에 오면 종료하는 식으로 구현하였다.

### **Print**

Print는 cll의 자료구조안에 있는 단어와 그 뜻을 log.txt 에출력하는 함수이다.

먼저 log.txt를 쓰기전용으로 ofstream하였다. cll을 돌기위해서 pCur이라는 주소값을 가지는 변수를 선언하였고 cll에 저장되어있는 노드들을 차례대로 순회하면서 그 노드 안에있는 단어와 뜻을 log.txt에 출력하였다. 끝까지 출력한뒤 파일을 닫았다.이때 마지막노드를 찾는 방법은 pCur이 head까지 순회했거나 pCur이 NULL값일때까지 순회하였다. cll안에 아무런데이터가 없다면 false를 return하였다.

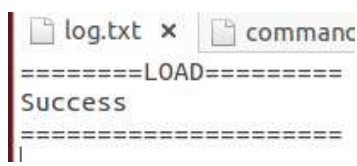

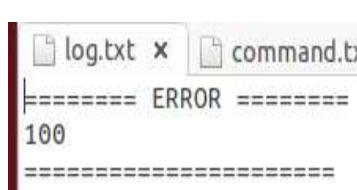

## Save

Save는 cll 안에있는 노드들의 데이터를 memorized\_word.txt에 출력하는 함수이다. 먼저 cll안에 아무런데이터가 없다면 false를 return하였다.

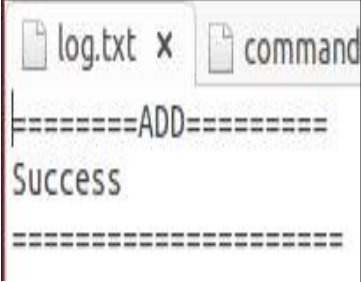
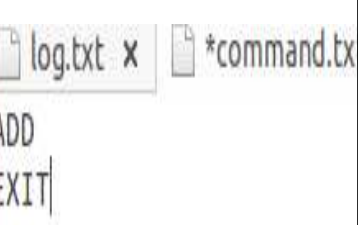
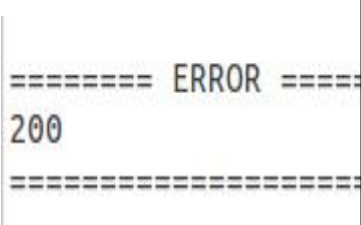

save 를하기위하여 먼저 memorized\_word.txt를 쓰기전용으로 ofstream하였다. cll을 돌기위해서 pCur이라는 주소값을 가지는 변수를 선언하였고 cll의 처음부터 끝까지 while문을 사용하면서 순회하였다. 마지막 노드를 찾기위해서는 pCur이 pHead가 될 때까지 순회한다. 이때 다음노드로 움직일때마다 그 노드안에 있는 뜻과 단어들을 getword와 getmean을 사용하여 memorized\_word.txt에 출력하였다.

## -Result Screen

### LOAD

		LOAD가 성공하는 경우 LOAD를 할 때에 3개의 자료형이 모두 비어있고, LOAD한 후에 자료형이 모두 차있을 때 성공한다.
		LOAD가 실패하는 경우. 자료형에 이미 자료가 있을 때는 3개의 자료형 모두 LOAD가 되지 않고 LOAD를 한 후에 자료형이 하나라도 비어있으면 실패

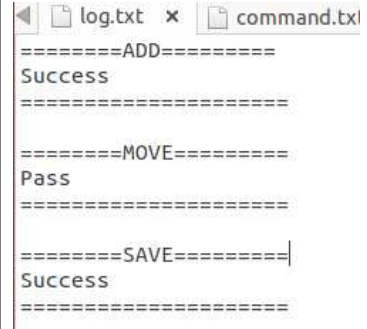
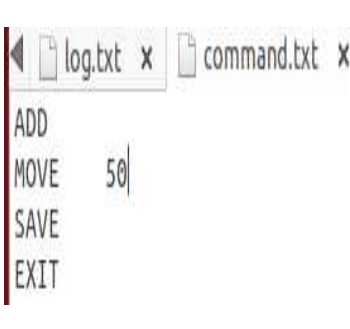

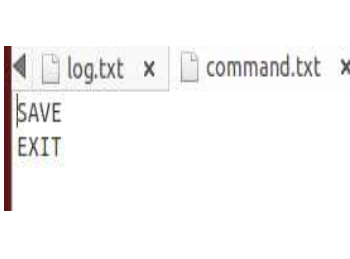
## ADD

 <pre> =====ADD===== Success ===== </pre>	 <pre> ADD EXIT </pre>	ADD가 성공하는 경우
 <pre> ===== ERROR ===== 200 ===== </pre>	 <pre> ADD EXIT </pre>	ADD가 실패하는 경우 word.txt가 없을 때 실패 하게 된다.

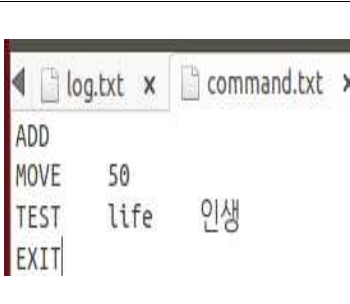
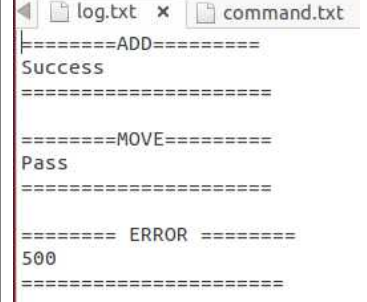
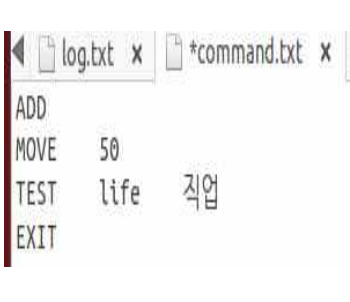
## MOVE

 <pre> =====ADD===== Success =====  =====MOVE===== Pass ===== </pre>	 <pre> ADD MOVE 50 EXIT </pre>	MOVE가 성공하는 경우 MOVE를 하고 난 후에 BST에 100개 이하로 있을 것으로 예상되면 성공이다. 이 경우에는 BST에 자료형 이 없고, ADD를 300개 한 뒤 MOVE를 50개 하였다.
 <pre> =====ADD===== Success =====  ===== ERROR ===== 300 ===== </pre>	 <pre> ADD MOVE 101 EXIT </pre>	MOVE가 실패하는 경우 MOVE를 하고 난 후에 BST에 100개 이상이 있을 것으로 예상되면 실패이다. 이 경우에는 MOVE가 100 개 이상이므로 실패이다.

## SAVE

 <pre> =====ADD===== Success ===== =====MOVE===== Pass ===== =====SAVE===== Success ===== </pre>	 <pre> ADD MOVE 50 SAVE EXIT </pre>	<p>SAVE가 성공하는 경우 3개의 자료형중 하나라도 자료형이 있으면 성공. 이 경우에는 ADD를 300개 하고 MOVE를 50개 해서 BST로 넘겨 주었기 때문에 2개의 자료형이 있고 하나 의 자료형은 없는 상태이다.</p>
 <pre> =====ERROR===== 400 ===== </pre>	 <pre> SAVE EXIT </pre>	<p>SAVE가 실패하는 경우 자료형이 하나도 없는 상태 에서 SAVE를 하면 실패한 다. 이 경우에는 아무 것도 없 는 상태에서 SAVE를 하였 기 때문에 실패이다.</p>

## TEST

 <pre> =====ADD===== Success ===== =====MOVE===== Pass ===== =====TEST===== Success ===== </pre>	 <pre> ADD MOVE 50 TEST life 인생 EXIT </pre>	<p>TEST가 성공하는 경우 BST에 있는 단어중에 word와 mean이 일치하는 경우에 TEST가 성공한다. 이 경우에는 MOVE한 단어 중에 life 인생이 있기 때문 에 성공이다.</p>
 <pre> =====ADD===== Success ===== =====MOVE===== Pass ===== ===== ERROR ===== 500 ===== </pre>	 <pre> ADD MOVE 50 TEST life 직업 EXIT </pre>	<p>TEST가 실패하는 경우 BST에 일치하는 단어가 없 을 경우에 실패이다. 이 경우는 life는 있지만 mean이 다르기 때문에 실패 이다.</p>



## SEARCH

<pre> =====ADD===== Success =====  ===== SEARCH ===== life 인생 ===== </pre>	<pre> log.txt x command.tx ADD SEARCH life EXIT </pre>	<p>SEARCH가 성공하는 경우 세 개의 자료형 중에 일치하는 단어가 있다면 뜻과 함께 출력한다.</p>
<pre> =====ADD===== Success =====  ===== ERROR ===== 600 ===== </pre>	<pre> log.txt x comm ADD SEARCH lifestyle EXIT </pre>	<p>SEARCH가 실패하는 경우 세 개의 자료형 중에 일치하는 단어가 하나도 없다면 실패한다.</p>

## PRINT

<pre> log.txt x command.txt x Ma =====ADD===== Success =====  =====MOVE===== Pass =====  =====TEST===== Success =====  ===== PRINT ===== problem 문제 story 이야기 lot 많이 =====  ===== PRINT ===== earth 지구 job 직업 =====  ===== PRINT ===== life 인생 ===== </pre>	<pre> log.txt x command.txt x Ma ADD MOVE 3 TEST life 인생 PRINT TO_MEMORIZE PRINT MEMORIZING I_PRE PRINT MEMORIZED EXIT </pre>	<p>PRINT가 성공하는 경우 명령어에 따른 자료형에 단어가 있다면 모두 출력한다. TO_MEMORIZE 명령어는 QUEUE에 있는 단어들을 출력하고, MEMORIZING I_PRE 명령어는 BST에 있는 단어들을 Preorder형태로 출력하고, MEMORIZED는 CLL에 있는 단어들을 출력한다. 이 경우는 6개 단어를 ADD하고, 3개 단어를 MOVE하고, TEST를 하여 1개의 단어가 CLL로 넘어가기 때문에 모든 자료형에 단어가 있고 3개의 명령어를 모두 수행한다.</p>
<pre> log.txt x command.txt x Mana =====ADD===== Success =====  =====MOVE===== Pass =====  ===== PRINT ===== problem 문제 story 이야기 lot 많이 =====  ===== PRINT ===== earth 지구 life 인생 job 직업 =====  ===== ERROR ===== 700 ===== </pre>	<pre> log.txt x command.txt x Manage ADD MOVE 3 PRINT TO_MEMORIZE PRINT MEMORIZING I_PRE PRINT MEMORIZED EXIT </pre>	<p>PRINT가 실패하는 경우 명령어에 따른 자료형에 단어가 없을 경우에 error가 출력된다. 이 경우는 다른 자료형에는 모두 단어가 있지만 CLL에 단어가 없기 때문에 CLL에서 error가 나온다.</p>

## UPDATE

<pre> log.txt x command.txt x =====ADD===== Success ===== =====MOVE===== Pass ===== ===== PRINT ===== earth  지구 life   인생 job    직업 ===== =====UPDATE===== life  인생 -&gt; 직업 ===== ===== PRINT ===== earth  지구 life   직업 job    직업 =====   </pre>	<pre> log.txt x command.txt x ADD MOVE 3 PRINT MEMORIZING I_PRE UPDATE life 직업 PRINT MEMORIZING I_PRE EXIT </pre>	<p>UPDATE가 성공하는 경우 세 개의 자료형 중에 명령어와 일치하는 단어를 찾고 뜻을 바꾸어 준다.</p> <p>자료형 안에서 단어를 찾으면 성공이다.</p> <p>이 경우는 BST에 life가 있고 그것을 찾아서 뜻을 바꾼 뒤 뒤에 바뀐 뜻이 PRINT되는 것을 볼 수 있다.</p>
<pre> log.txt x command.txt x =====ADD===== Success ===== =====MOVE===== Pass ===== ===== PRINT ===== earth  지구 life   인생 job    직업 ===== ===== ERROR ===== 800 ===== ===== PRINT ===== earth  지구 life   인생 job    직업 ===== </pre>	<pre> log.txt x command.txt x ADD MOVE 3 PRINT MEMORIZING I_PRE UPDATE lifestyle 직업 PRINT MEMORIZING I_PRE EXIT </pre>	<p>UPDATE가 실패하는 경우 세 개의 자료형에 명령어와 일치하는 단어가 없다면 실패한다.</p> <p>이 경우는 3개의 자료형에 lifestyle이 없기 때문에 실패한다.</p>

## command

<pre> log.txt x  command.txt =====ADD===== Success ===== ===== PRINT ===== life  인생 job   직업 earth 지구 problem 문제 story 이야기 lot   많이 ===== ===== ERROR ===== 700 ===== ERROR ===== 100 =====MOVE===== Pass ===== ===== PRINT ===== problem 문제 story  이야기 lot    많이 ===== ===== PRINT ===== earth  지구 life   인생 job    직업 ===== =====TEST===== Success ===== ===== ERROR ===== 500 =====UPDATE===== job  직업 -&gt; 직장 ===== ===== PRINT ===== earth  지구 job    직장 ===== ===== PRINT ===== life   인생 ===== ===== SEARCH ===== job  직장 ===== </pre>	<pre> log.txt x  command.txt x  Mar ADD PRINT  TO_MEMORIZE PRINT  MEMORIZED LOAD MOVE   3 PRINT  TO_MEMORIZE PRINT  MEMORIZING      I_PRE TEST   life  인생 TEST   job   직장 UPDATE job   직장 PRINT  MEMORIZING      I_PRE PRINT  MEMORIZED SEARCH job EXIT </pre>	<p>ADD로 6개의 단어를 queue에 넣고 PRINT를 하였을 때 queue에 있는 단어는 출력되지만 cll에는 단어가 없기 때문에 에러가 출력된다.</p> <p>그 후에 LOAD를 하면 이미 queue에 자료형이 있기 때문에 오류가 난다.</p> <p>MOVE3을 하면 6개의 단어 중 순서대로 3개의 단어가 BST로 넘어가게 되고 queue를 PRINT하면 남은 3개의 단어가 출력된다. BST를 출력하면 이동한 3개의 단어가 출력된다.</p> <p>그 후에 TEST를 통해서 BST의 단어를 CLL로 넘겨주는 작업을 하는데 처음 TEST는 life가 존재하고 그 뜻을 맞췄기 때문에 성공하지만 두 번째 TEST는 job의 뜻이 틀렸기 때문에 에러가 난다. 즉, life만 CLL로 넘어가게 된다.</p> <p>다음으로 UPDATE를 통해 job의 뜻을 바꾸어 주는데 job이 있기 때문에 UPDATE가 성공하게 된다. 이제 BST를 PRINT하면 BST에 남아있는 단어들이 출력되고, CLL을 PRINT하면 CLL로 옮겨온 단어가 출력된다.</p> <p>마지막으로 job을 SEARCH하면 바뀐 뜻으로 찾게된다.</p>
--	--	---

## -Consideration

이름	프로젝트에서 맡은 역할	본인 스스로 생각하는 점수(10)
김기선	QUEUE,MANAGER함수,예외처리	10
고찰	<p>이번 프로젝트를 통해서 영어단어프로그램을 구현하였다. 본인은 팀프로젝트를 처음해봐서 되게 재밌게 프로젝트를 만들었다. 본인은 QUEUE와 MANAGER함수를 중점적으로 코딩을 하였는데 제 부분을 혼자서 코딩하는데에는 크게 어려움이 없었습니다. 하지만 우리가 택한 방식은 서로 맡은부분을 먼저 구현을하고 그다음에 합친다음 서로의 코드를보며 수정할부분은 수정하고 보완할부분들은 보완하는식으로 프로젝트를 진행하였습니다. 하지만 코드내용중 다른사람의 코드내용을 알지못하면 내코드를 작성하는데에도 까다롭고 예외처리를하는데에 있어서도 어려운부분이 많았습니다. 하지만 내가 생각하는부분이아닌 다른사람의 코딩을 보면서 ‘아 이렇게도 생각할수 있구나’ 하는 여러 가지 관점들에 대해서 확인할 수 있었습니다. 리눅스에대해서는 완전 처음해보는 프로그램이기 때문에 작동하는데에 있어서 어려움이 많았습니다. 아직 본인은 왜 이부분을 리눅스에서 돌려야하는지 잘 이해가 되지않습니다. 하지만 작동법이 강의자료에 자세히 나와있어서 많은도움이 되었습니다. 나중에 리눅스를 배울때에 좀더 편하게 접근가능할 것 같습니다. 또한 팀프로젝트를통해 팀워크에대해 배우면서 내스스로 화합하는방법도 많이 배운것같고 나중에 회사에 취직해서도 이런 팀프로젝트들이 많을것같은데 이러한 경험을통해 잘 적응해나갈수 있을 것 같습니다.</p>	

이름	프로젝트에서 맡은 역할	본인 스스로 생각하는 점수(10)
김지웅	CLL,리눅스컴파일,예외처리	10
고찰	<p>이번 프로젝트를 통해서 queue, BST, CLL에 대해서 공부하였습니다. 파일 입출력을 통하여 명령어를 입력하였고 word파일을 읽어서 자료형에 저장하였습니다. 또 그 자료형을 file에 출력하였습니다. 이번 프로젝트를 통하여 팀워크와 리더쉽을 가지게 되었고, 따로따로 설계를 했을때 합치는것에 어려움이 있다는 것도 알았습니다. 우리 팀은 각자 맡은 부분을 짰 뒤에 한번에 합쳐서 돌리기로 계획을 잡았는데 말로는 쉬워보였지만 막상해보니 예상못한 변수들이 많았습니다. 우리가 변수명도 다 다르고 예외처리 방식도 다 달라서 서로서로 이해하는데 어려움이 있었고 다음부터는 미리미리 얘기하여</p>	



	<p>맞춘다음 같이 해야 될 것 같습니다. 나중에 회사를 가면 이렇게 따로따로 작업하는 일이 많을 텐데 이런 프로젝트를 통해서 미리 경험해 보아서 뜻깊은 프로젝트였습니다.</p> <p>또한 처음으로 리눅스를 사용해 보았는데 사용법이 미숙하여 많은 구글링을 통하여 리눅스의 사용법을 숙지 하였습니다.</p>
--	--

이름	프로젝트에서 맡은 역할	본인 스스로 생각하는 점수(10)
이호영	Alphabet BST, Word BST, 예외처리	10
고찰	<p>BST부분을 맡으면서 먼저 전체적인 manager함수와 알고리즘을 찾기가 까다롭게 다가왔습니다. 정해진 form이 있어서 출제자의 의도를 파악하는 것이 꽤 어려웠던 것 같습니다. 하지만 그것을 알아내게 되고 그 형식으로 맞추어 코딩을 하면서 제가 기존에 사용하지 않던 방법을 접할 수 있었고, 안목이 더 길러진 것 같습니다. 또한 다른 부분보다 순회와 Delete함수를 짜는데에 어려움이 많았습니다. 물론 자료가 많이 나와 있지만, 이를 이해하고 제 방식으로 iteration한 code를 짜는데 어려움이 많았습니다. 하지만 각종 순회를 이를 통해 더 깊게 공부 할 수 있었고, 제가 미처 접근하지 못했던 여러 방식 또한 이해 할 수 있었습니다. 또한 bst를 delete하는 과정은 생각 보다 많은 경우의 수에 복잡하다고 느꼈지만 공부를 하고 좀 더 생각하고 직접 코딩을 하다 보니 겉으로는 마냥 어려워 보여도 일정한 규칙이 있고 이러한 알고리즘을 통해 delete를 진행하는 것을 새삼 느꼈습니다. 비교적 insert와 print, search, save함수는 알고리즘도 간단하고, 그 원리가 비슷해서 쉽게 짤 수 있었습니다. 코드를 짜는 것 보다 이번 프로젝트는 팀으로 진행되어서 코드를 서로 합쳤을 때, 예외가 많이 발생하게 되는 것을 확인 할 수 있었습니다. 하지만 실무에 가면 팀으로 진행하기 때문에 굉장히 좋은 경험이었습니다. 아직은 처음으로 팀 프로젝트를 진행해서 뼈격거리는 부분이 분명히 있었지만 이를 통해 많은 점을 배웠고, 팀플레이 중요하다는 것을 느꼈습니다.</p>	