Data Structure Report

Assignment 1

Professor	이기훈 교수님
Department	Computer engineering
Student ID	2012722070
Name	임연수
Class	설계 (o) / 실습 (화)
Date	2016. 10. 07

Signature _____

- **Introduction**

이번 데이터 구조 실습에서 구현할 프로젝트는 단어장에 대한 프로그램이다. 내부적으로 Queue와, Binary Search Tree, Circular Linked-List에 대한 자료 구조를 이용하여 작성하는 데, 단어장 목적에 맞게 외워야 할 단어들은 FIFO(First-In, First-Out)구조를 가지는 Queue를, 외우는 중인 단어들은 영단어의 Word를 Primary Key로 하는 Binary Search Tree를, 암기가 끝난 단어들은 처음과 끝이 순환하는 Circular Linked-List를 각각 이용한다.

이러한 자료구조를 작성하는 데에 앞서 고려해야할 사항들이 존재하는데, 먼저 각 자료구조에 대한 이해이다.

Queue는 앞서 설명한 것과 같이 FIFO구조를 가진다. 따라서 Front부터 Rear까지 순차적으로 링크를 연결하면서 디스크에 상주하는 word.txt의 정보들을 Heap 메모리 영역에 불러온다. 이러한 과정을 Insertion이라고 한다. Queue 자료 구조 내에 특정 데이터를 찾기 위해서 Search 함수를 필요로 하는 데, 이는 Front부터 Rear까지 순차적으로 Node를 거치면서 해당 단어를 찾는 함수이다. Queue 구조에 저장된 데이터를 빼내는 과정도 필요한 데, 이는 Delete(POP) 함수를 이용하여 구현할 수 있다. Delete 함수는 Front, 즉 Head의 값을 return한다. 주의해야할 점은 Head를 return 하기 전에 재설정해야 한다는 점이다.

Binary Search Tree는 root에서 시작하여 특정 Key값을 기준으로 root의 Key보다 작을 경우 Left SubTree에, 클 경우 Right SubTree에 저장이 되는 이진 트리이다.

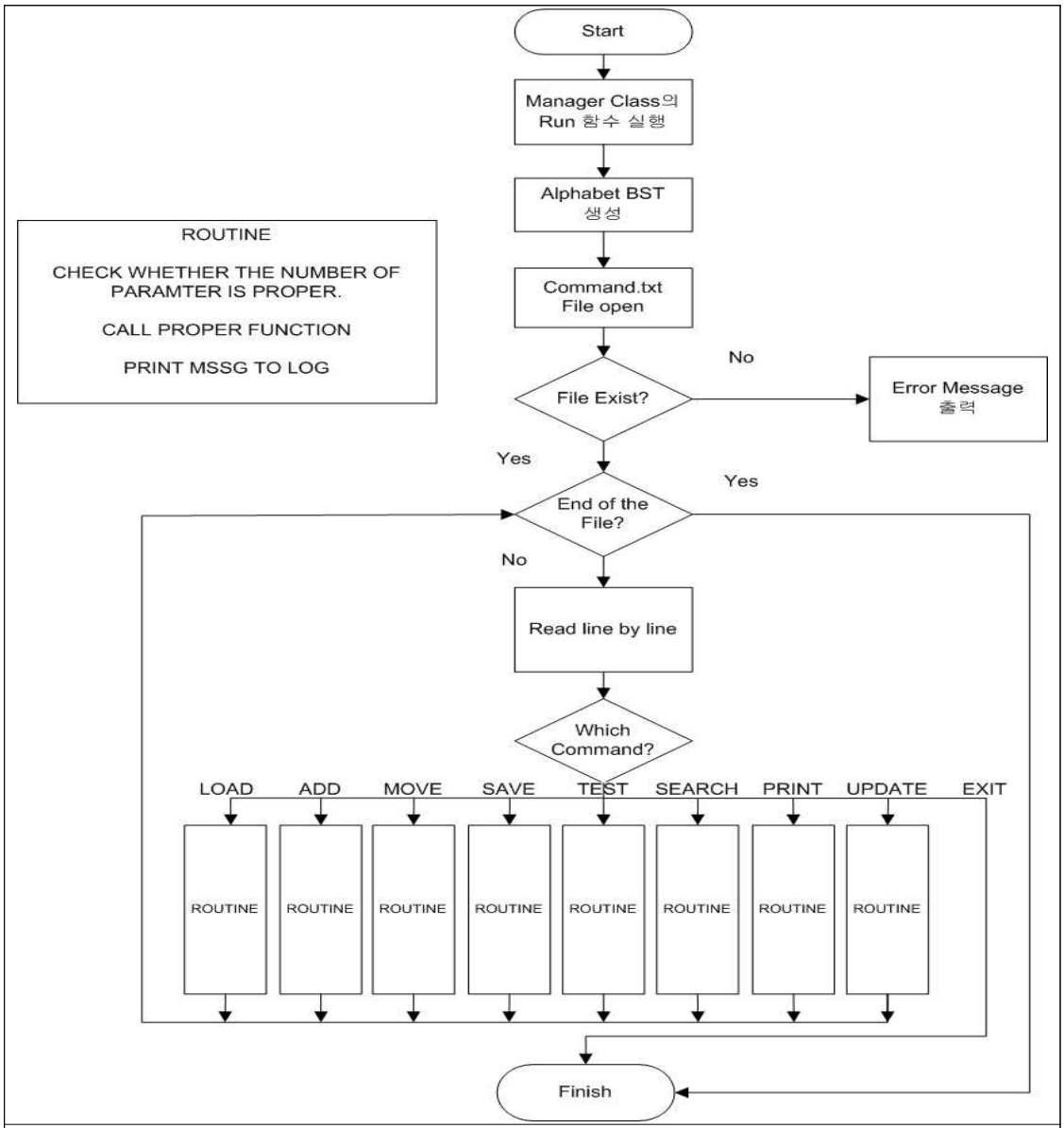
Queue와 마찬가지로 값을 삽입,삭제,검색 기능을 필요로 하며, 추가적으로 순회하는 함수를 필요로 한다. 삽입을 하는 함수, Insert함수의 경우는 Key값을 기준으로 판단하여, 기존에 BST에 저장되어있는 Key값들의 적절한 범위 내에 Child로 Node를 연결한다. Delete함수의 경우는 Delete할 Node가 자식이 없는 경우, 자식이 있는 경우로 나누어지는데, 자식이 없는 경우는 해당 노드의 데이터를 해당 부모의 링크에서 링크를 끊는다. 자식이 있는 경우는 또 자식이 하나 있는 경우와 자식이 둘이 있는 경우가 있는데, 자식이 만약 Left Child라면, Left Child를 root로 하는 Subtree내에 Key가 가장 큰 값을 삭제해야할 노드에 대체한다. Right Child경우는 이와 반대로 Right Child를 root로 하는 Subtree내에 Key가 가장 작은 값을 삭제해야할 노드에 대체한다. 자식이 둘 다 있는 경우는 Left Child만 있는 경우와 동일한 로직을 가진다. 순환하는 기능을 하는 함수, Print함수의 경우는 방문(VISIT)을 언제 하냐에 따라 Pre-Order, In-Order, Post-Order로 나누어지며, 구현 방법에 따라 Recursive인지, Iterative인지로 나누어진다.

Circular Linked-List는 Queue와 흡사하나, Queue에서 rear부분의 Node, Tail의 링크를 Head에 연결하여 순환하는 구조로 만든 모델이다. 기본적으로 구현해야할 핵심 함수는 Queue와 동일하나, 다른 점은 Tail의 링크를 Head로 연결이 되도록, 혹은 검색 시 무한정으로 돌지 않도록 고려해야 한다는 점을 필요로 한다.

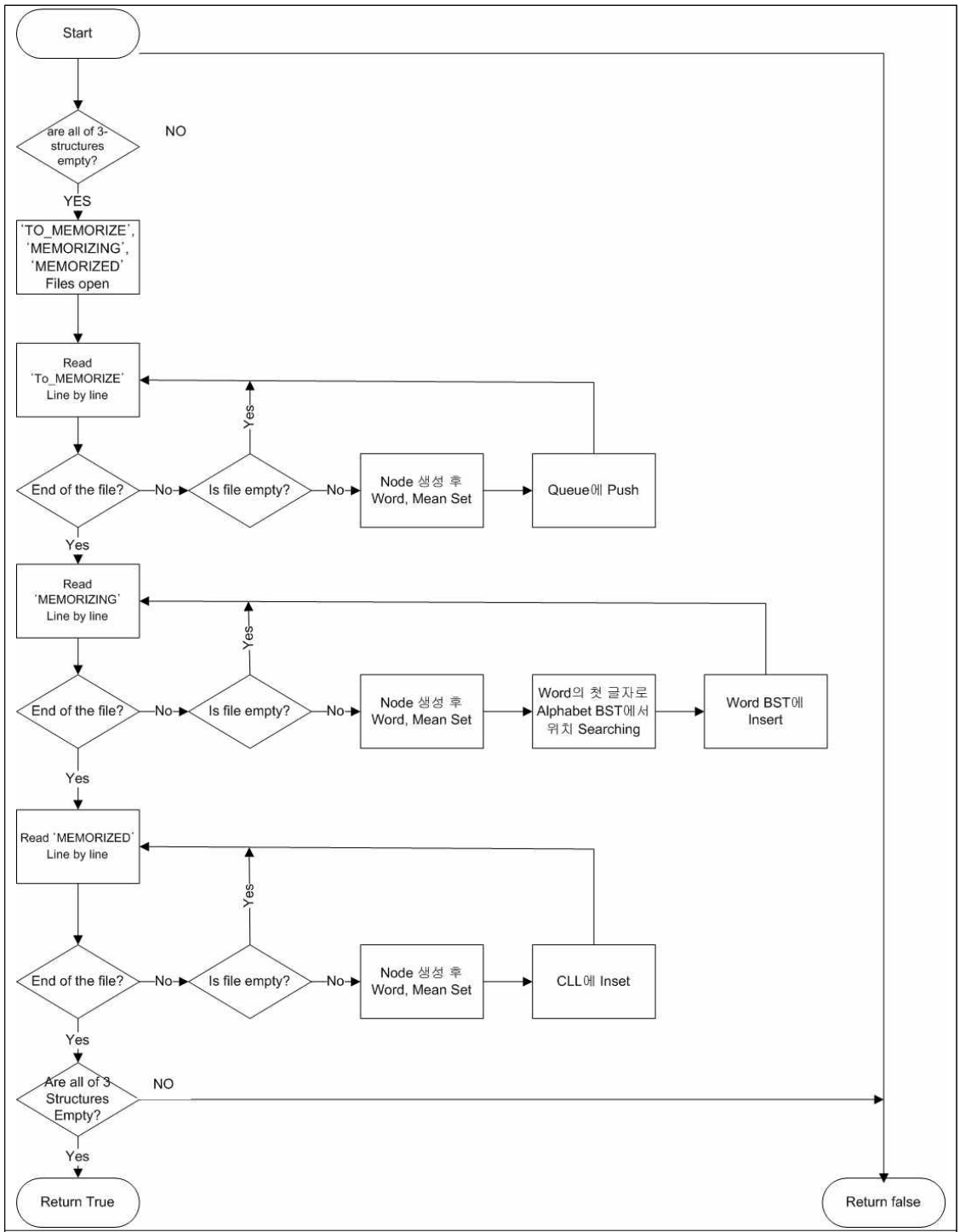
자료구조 외에도 text 파일과의 통신을 필요로 하는데, 이를 위해 파일 클래스를 이용할 줄 알아야 하며, 쓰기 옵션을 목적에 맞게 쓰는 것을 필요로 한다.

이러한 모듈들의 동작은 Manager 클래스에서 관리하며, 각 목적에 맞게 LOAD, ADD, PRINT, SAVE등으로 각 기능별로 함수로 구현하여 동작을 제어한다.

- **Flow Chart**



Program에 대한 High-Level Abstraction.
process가 생성되면 main에서 Manager의 run 함수를 호출하는 데, 매개변수로 주어진 command 디스크 파일과의 통신을 시작한다. 파일에 저장되어 있는 명령에 맞게 주어진 모듈들을 수행하는데(Which Command? 부분), 파일 통신에 있어 기본적으로 지켜야 할 예외처리 - 파일과의 통신이 성공하여 파일 객체가 값을 가지고 있는지(즉, 파일이 존재하는지), 파일 내용이 있는지 - 등을 검사한다. 파일 객체의 커서가 파일의 끝으로 가면 해당 프로그램은 종료되고 소멸자가 호출되어 process에 할당 되어 있던 모든 메모리를 반납한다.



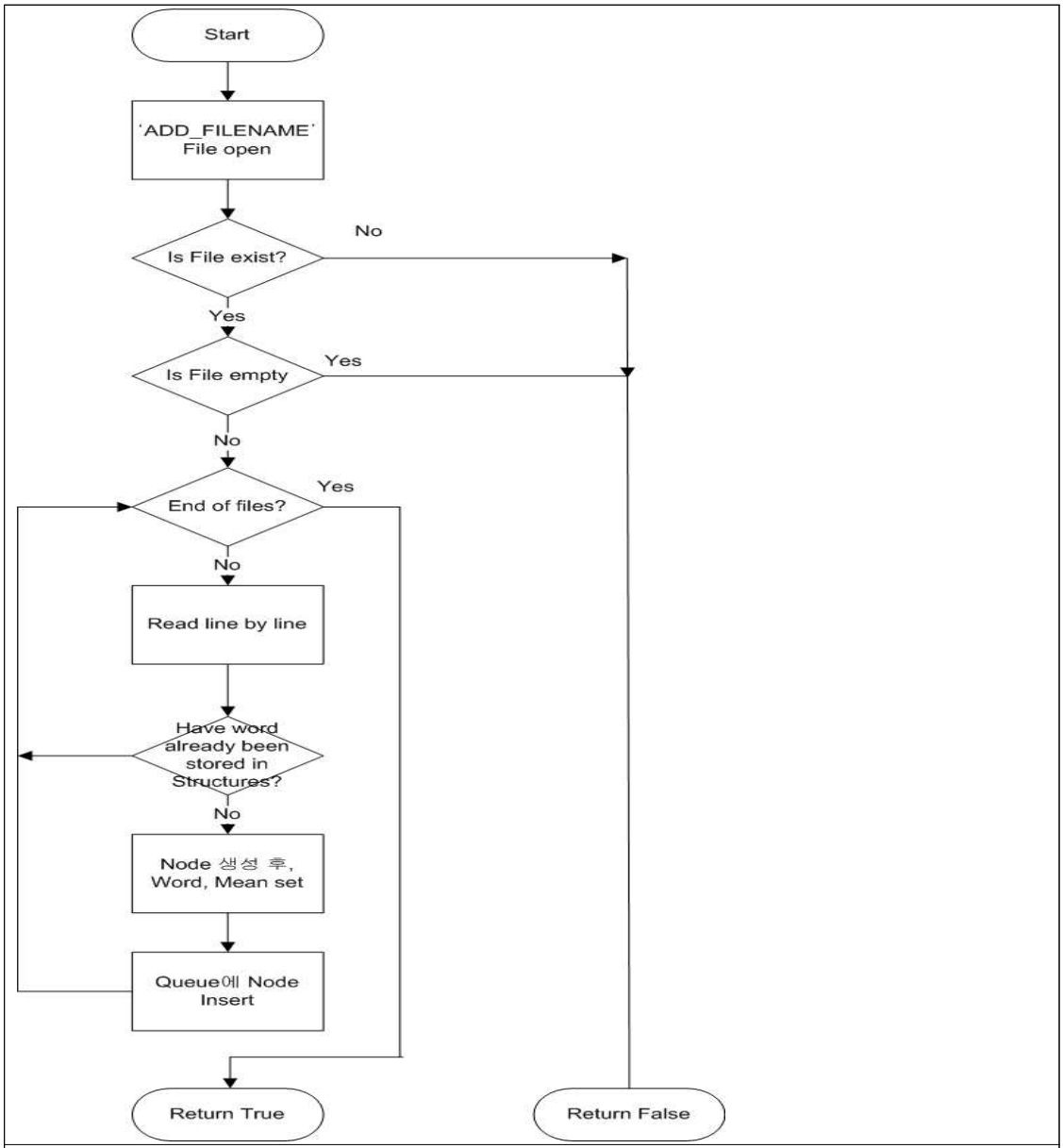
LOAD 함수 부분.

LOAD는 직접적으로 메모리에 있는 각 자료 구조 모델과 디스크에 있는 텍스트 파일과의 통신을 실시한다. 이에 따라 제안서를 기준, 고려해야 할 사항들이 있는데, 이는 다음과 같다.

1. 자료구조 형 내에 데이터가 이미 존재하는지?
2. 파일이 존재하는지?
3. 파일의 내용이 비어있는지?

핵심은 위의 과정중 1번이며, 3개의 자료구조 형 중에서 하나라도 데이터를 가지고 있다면, LOAD 함수는 에러 코드를 출력한다.

즉, 자료구조 형 전부 데이터가 없을 때(empty) LOAD를 수행하며, LOAD가 수행이 된 이후에도 자료구조 형 전부 데이터가 없다면, 에러 코드를 출력한다.

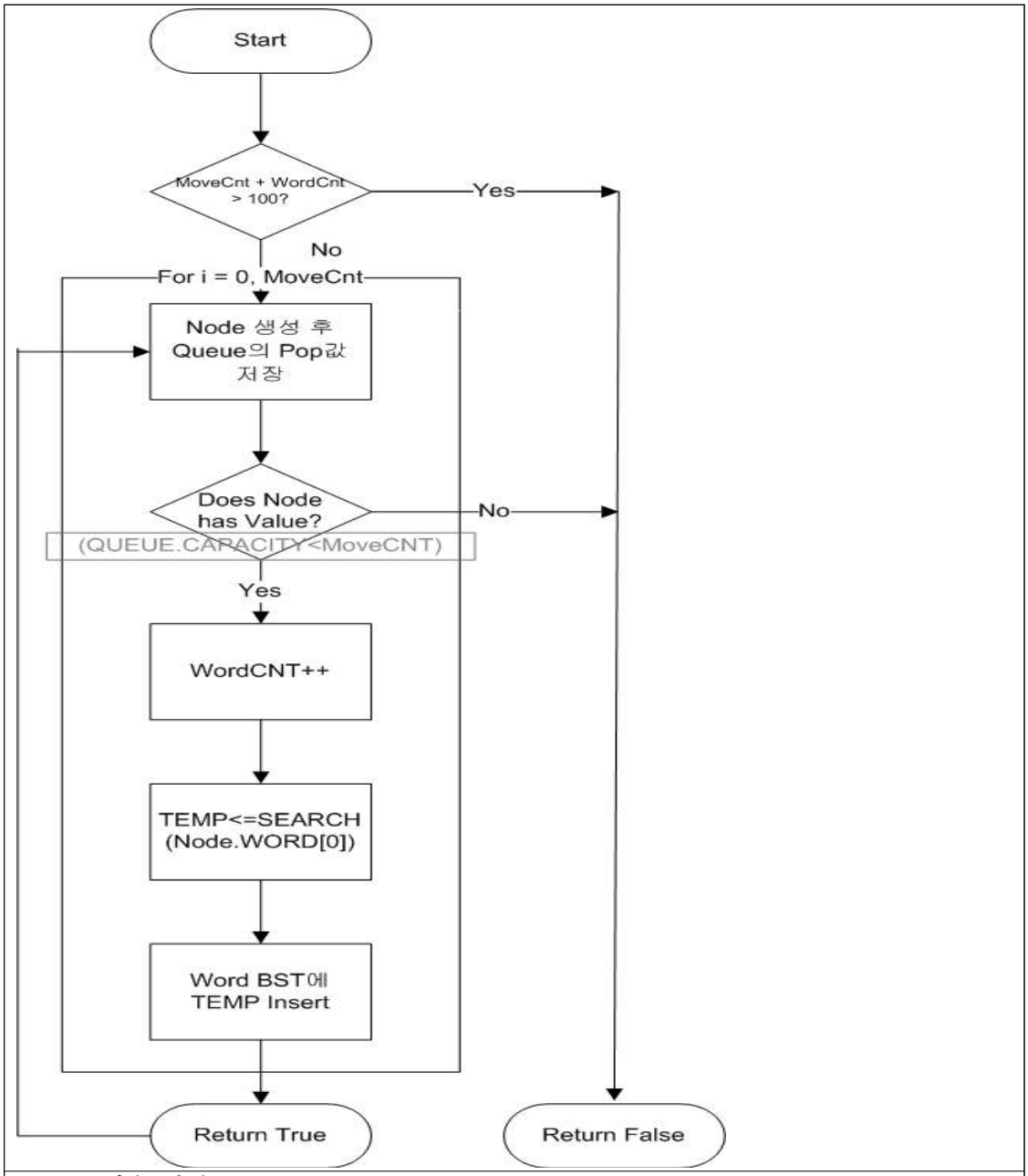


ADD 함수 부분.

ADD는 텍스트 파일에 저장 되어 있는 단어 정보를 불러와 Queue 자료 구조 모델에 저장한다. 이에 따라 제안서를 기준, 고려해야 할 사항들이 있는데, 이는 다음과 같다.

1. 자료구조 형 내에 데이터가 이미 존재하는지?
2. 파일이 존재하는지?
3. 파일의 내용이 비어있는지?

파일이 존재하지 않거나, 파일의 내용이 비어있으면, ADD함수는 에러 코드를 출력하고, 파일 객체의 커서가 파일의 끝에 도달하면 함수가 종료 된다.



MOVE 함수 부분

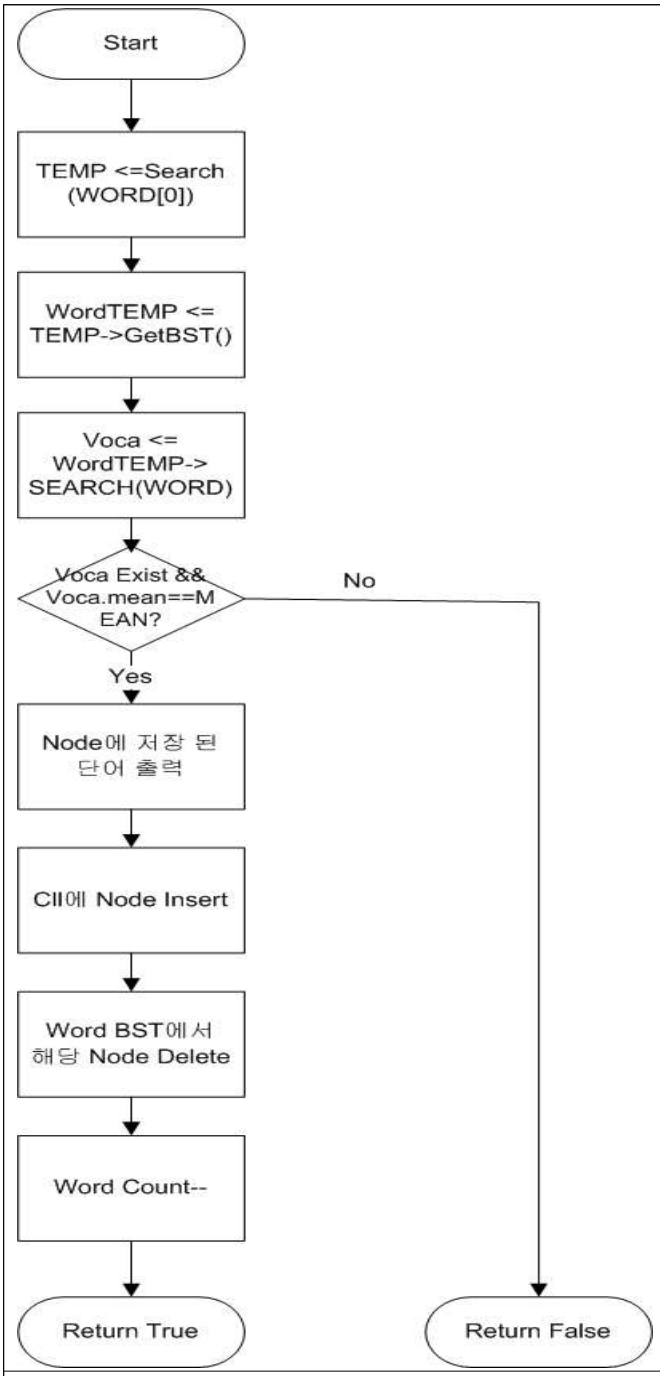
MOVE함수는 Queue 자료 구조 모델의 데이터들을 입력된 수만큼 Binary Search Tree 자료 구조 모델로 전달하는 기능을 한다. 데이터를 전달함에 있어서 Queue 자료 구조 모델은 First-In-First-Out 방식으로 데이터를 전달하고, 전달 후에는 Queue 자료 구조 모델 내부에서는 해당 데이터를 삭제한다. 데이터를 수신함에 있어서 Binary Search Tree 자료 구조 모델은 root에서 시작하여 특정 Key값을 기준으로 root의 Key보다 작을 경우 Left SubTree에, 클 경우 Right SubTree에 저장을 한다.

이에 따라 제안서를 기준, 고려해야 할 사항들이 있는데, 이는 다음과 같다.

1. Queue 자료 구조 모델의 데이터가 존재하는지?
2. Binary Search Tree내에 저장 된 데이터의 개수가 100개를 초과하는지?

Queue 자료 구조 모델에 데이터가 존재하지 않거나, BST내에 저장 된 데이터의 개수가 100개를 초과할 경우, MOVE함수는 에러 코드를 출력한다.

MOVE 함수 부분



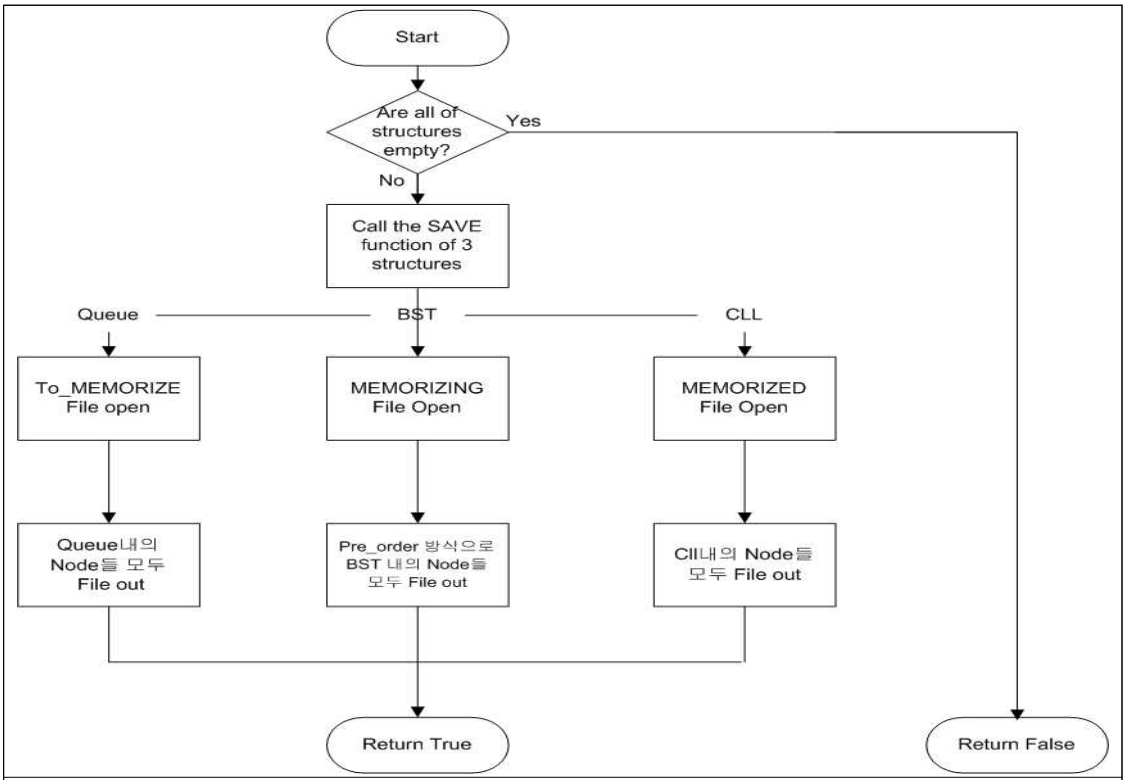
TEST 함수 부분

TEST는 입력 된 데이터를 Binary Search Tree 자료 구조 모델에서 찾아, 해당 데이터를 Circular Linked-List 자료 구조 모델로 전달하는 기능을 한다.

이에 따라 제안서를 기준, 고려해야 할 사항들이 있는데, 이는 다음과 같다.

1. 단어가 Binary Search Tree에 존재 하는가?
2. BST에 저장 된 단어의 뜻과 입력된 단어의 뜻이 같은가?

단어가 Binary Search Tree내에 존재 하지 않거나, 입력된 단어의 뜻이 BST에 저장된 단어의 뜻과 다르면 TEST함수는 에러 코드를 출력한다.

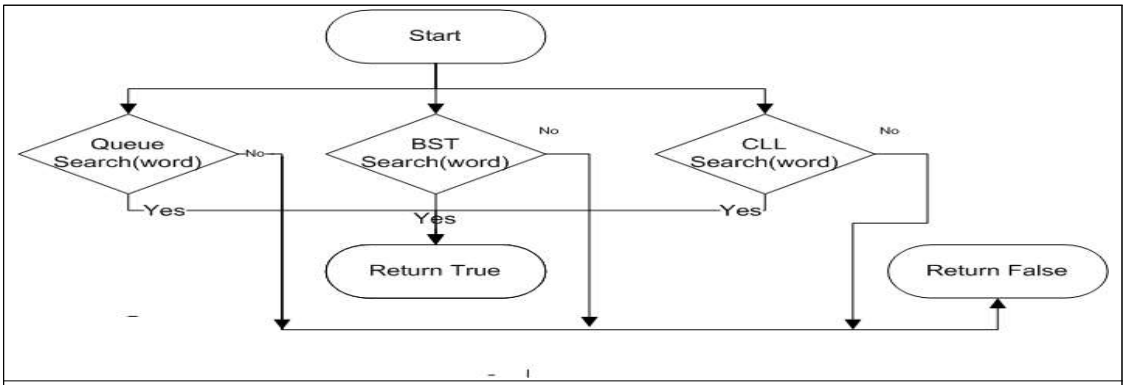


SAVE 함수 부분.

SAVE는 3개의 자료 구조 내에 있는 데이터를 디스크에 있는 각각의 텍스트 파일에 출력하는 기능을 한다. 이에 따라 제안서를 기준, 고려해야 할 사항들이 있는데, 이는 다음과 같다.

1. Queue 자료 구조에 출력할 데이터가 있는가?
2. Binary Search Tree 자료 구조에 출력할 데이터가 있는가?
3. Circular Linked List 자료 구조에 출력할 데이터가 있는가?

3개의 자료 구조 모델에 출력할 데이터가 없을 경우, 즉, 비어있을 경우 SAVE 함수는 에러 코드를 출력한다.

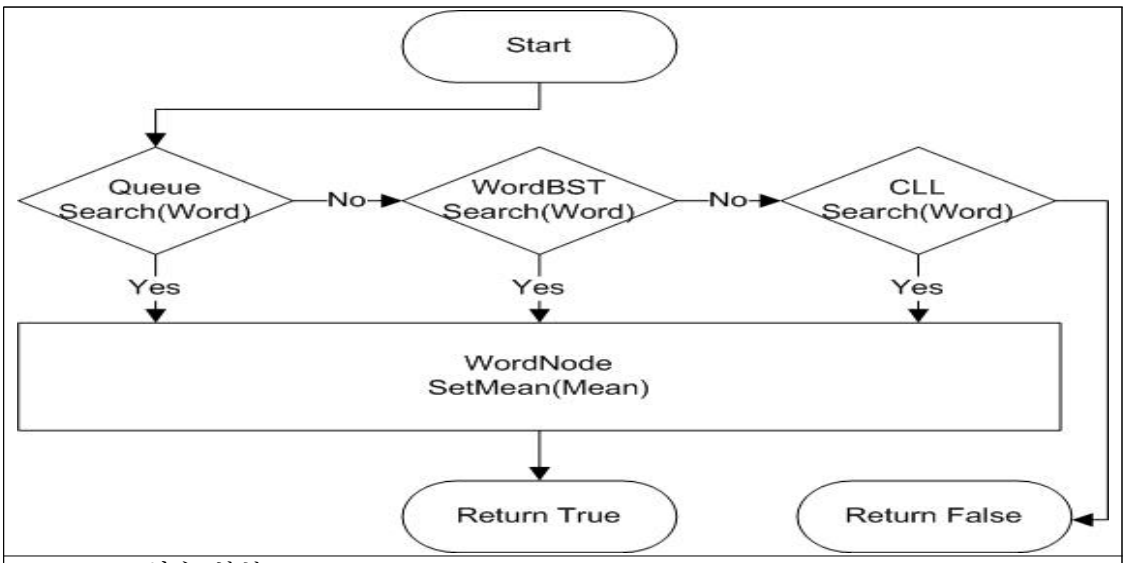


SEARCH 함수 부분

SEARCH는 3개의 자료 구조 내에 텍스트에서 읽어 온 데이터가 존재하는지 확인하는 기능을 한다. 이에 따라 제안서를 기준, 고려해야 할 사항들이 있는데, 이는 다음과 같다.

1. Queue 자료 구조 내에 입력 된 단어가 존재 하는가?
2. Binary Search Tree 자료 구조 내에 입력 된 단어가 존재 하는가?
3. Circular Linked List 자료 구조 내에 입력 된 단어가 존재 하는가?

3개의 자료 구조 중 어느 곳에도 입력 된 단어가 존재 하지 않는 경우, SEARCH 함수는 에러 코드를 출력한다.



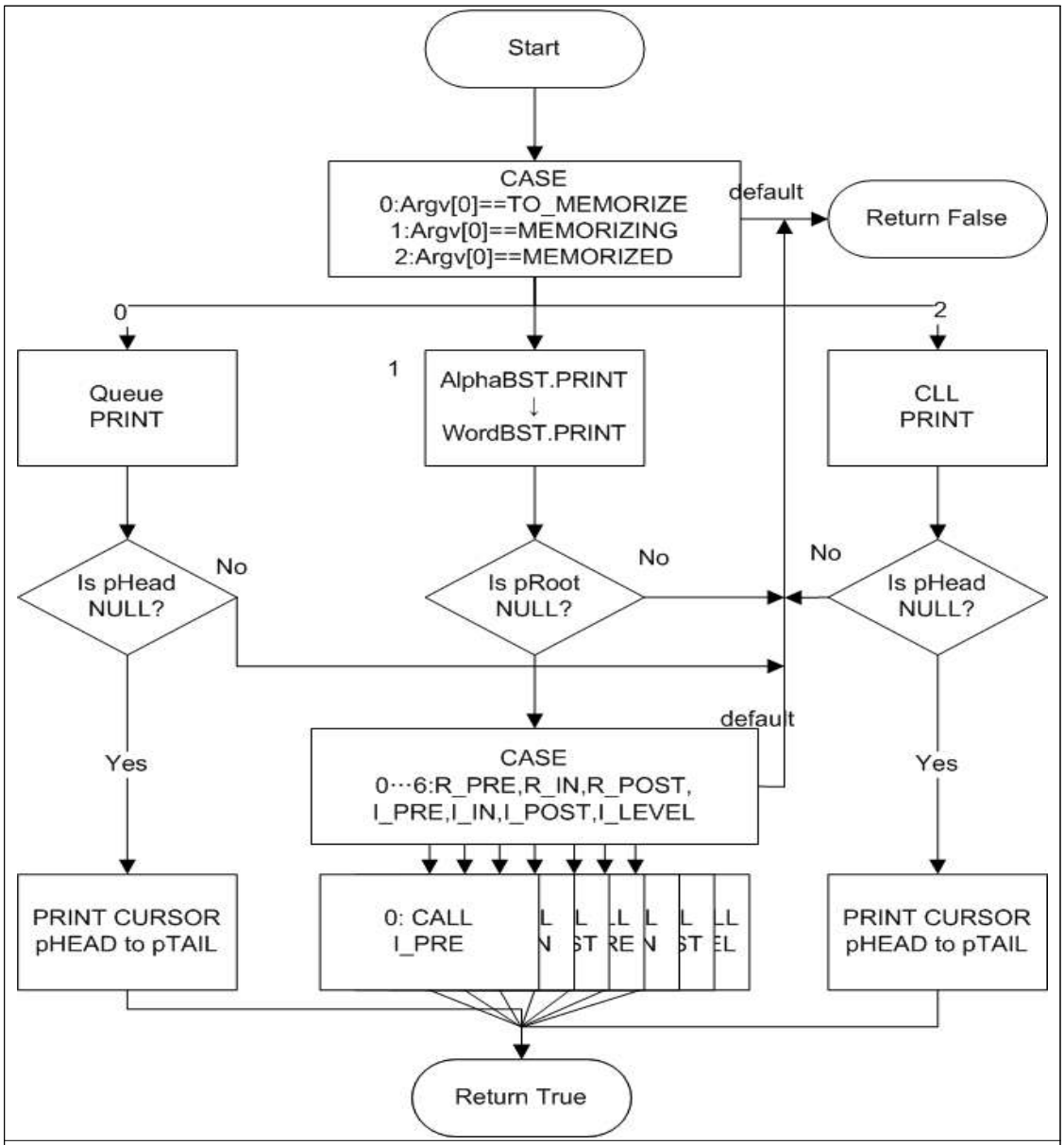
UPDATE 함수 부분

UPDATE 함수는 3개의 자료 구조 내에 텍스트에서 읽어 온 단어가 존재하는지 확인 한 후, 저장 되어 있는 단어의 뜻을 입력 된 단어의 뜻으로 변경하는 기능을 한다.

이에 따라 제안서를 기준, 고려해야 할 사항들이 있는데, 이는 다음과 같다.

1. Queue 자료 구조 내에 입력 된 단어가 존재 하는가?
2. Binary Search Tree 내에 입력 된 단어가 존재 하는가?
3. Circular Linked List 내에 입력 된 단어가 존재 하는가?

3개의 자료 구조 중 어디에도 입력 된 단어가 존재 하지 않는 경우, UPDATE 함수는 에러 코드를 출력한다.



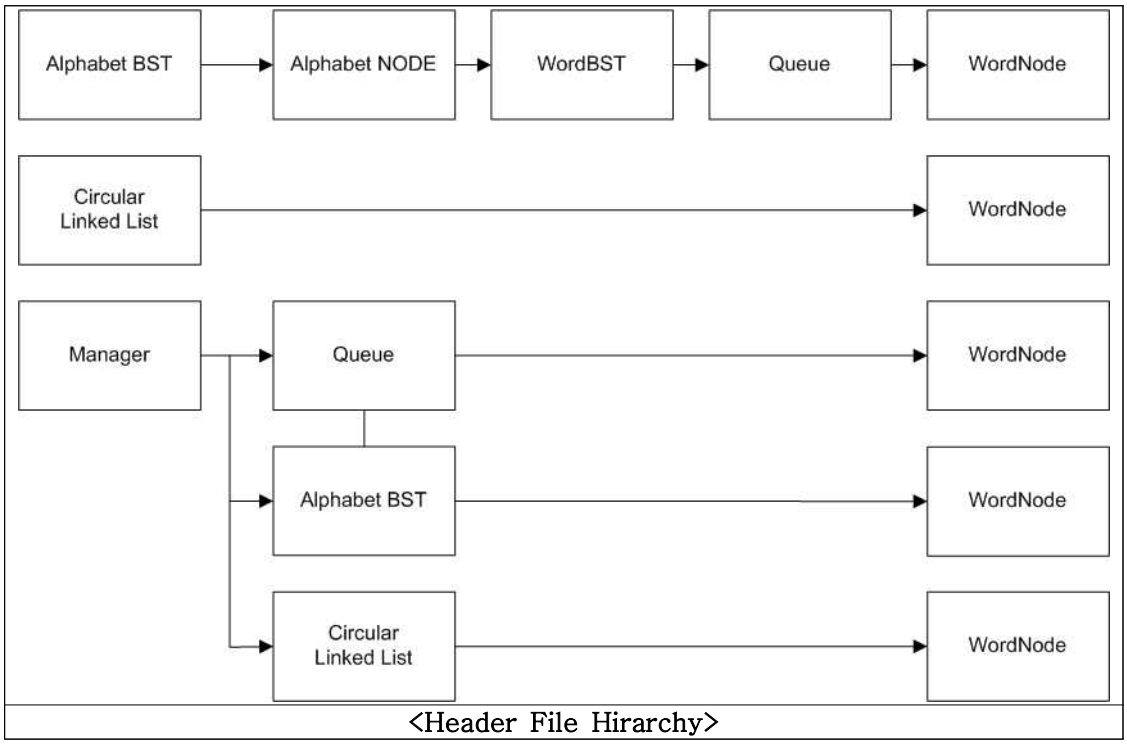
PRINT 함수 부분

PRINT 함수는 3개의 자료 구조 내에 저장 되어 있는 데이터들을 출력 하는 기능을 한다. Queue 자료 구조와 Circular Linked List 자료 구조 내의 데이터들을 출력함에 있어서는 pHead부터 pTail까지 출력을 하고, Binary Search Tree 자료 구조 내의 데이터들을 출력함에 있어서는 방문을 언제 하냐에 따라 Pre-order, In-order, Post-order로 나눠지며, 구현하는 방법에 따라서 Recursive와 Iterative로 나눠진다. 이에 따라 제안서를 기준, 고려해야 할 사항들이 있는데, 이는 다음과 같다.

1. 자료 구조 내에 출력 할 Data가 존재하는가?

특정한 Key값에 따라 3개의 자료 구조 중 출력할 자료 구조가 선택되어지는데, 이 때 해당 되는 자료 구조에 출력 할 Data가 존재하지 않는 경우, 즉 비어있을 경우 PRINT 함수는 에러 코드를 출력한다.

- **Algorithm**



<Header File Hirarchy>

**** WordNode ****

WordBST, Queue, CircularLinkedList 자료 구조 형의 데이터와 연결 구조를 위해 사용되는 클래스이다.

따라서 멤버 변수로 다음을 가진다.

1. pair type의 wordmean. 내부적으로 word와 mean의 주소를 가리키는 char type 포인터를 가지고 있다.
2. BST에서 LeftChild를 위한 자기 참조 포인터 pLeft
3. BST에서 RightCHild를 위한 자기 참조 포인터 pRight
4. Queue, CircularLinkedList를 위한 잠기 참조 포인터 pNext

멤버 함수로는 다음을 가진다.
멤버 변수의 값을 초기 값으로 설정하는 '생성자'와 동적 메모리 할당 부분을 해제하는 '소멸자'를 가진다.

각 멤버 변수의 값을 리턴하는 GetWord(), GetMean(), GetLeft(), GetRight(), GetNext()를 가지고 있다.
각 멤버 변수의 값을 설정하는 SetWord(), SetMean(), SetLeft(), SetRight(), SetNext()

이에 pair의 word와 mean을 같이 수행하는 SetVoca()함수를 추가적으로 만들어 구현하였다.

WordNode 멤버 함수에서 구현한 핵심 부분은 소멸자 부분과 SetWord, SetMean 함수이다.

SetWord, SetMean 함수는 매개변수로 char 포인터 형을 받는데, const char *가 아님으로, 내부적 함수 내에서 해당 char 포인터가 가리키는 문자열의 글자만큼을 길이로 동적 할당하고 문자열을 복사하여, 각각 pair의 first, second가 이를 가리키도록 설정하였다. char 포인터가 널을 가리키고 있을 수 있으므로, 먼저 이에 대한 예외처리를 한다.

SetMean 함수의 경우 UPDATE시 값이 바뀌므로, 이에 대한 고려를 해야하는데, 이미 pair의 second 변수가 값이 존재한다면 UPDATE의 경우이므로, 이 경우에 한하여 기존의 동적 메모리를 해제하고, 다시 재할당 받아 이전 SetMean 함수의 로직과 동일하게 작동하도록 한다.

소멸자의 경우 pair의 word,mean을 동적할당 하였으므로, 이를 해제하는 함수이다.
허나, 위에서 앞서 말했듯이, WordNode는 3개의 자료형에서 전부 쓰이는 노드이므로, Binary Search Tree에서 pLeft, pRight로 비선형으로 연결된 노드들을 제거하는 작업을 필요로 하게 되는데, 이에 따라 기존 Recursive Post-order 방식으로 구현하였으며, VISIT부분은 앞서 말한 pair내의 word,mean 동적 메모리를 해제하는 부분이 된다.
이에 대한 수도 코드는 다음과 같다.

```
if(!this) RETURN
this->pLeft->~WordNode
this->pRight->~WordNode
DELETE <wordmean.first, wordmean.second>
```

**** Queue ****

TO_MEMORIZE 자료구조형의 구현과 이러한 모델을 바탕으로 WordBST에서 Traversal ordering을 위해 부가적으로 해당 클래스를 사용한다.

따라서 멤버 변수로 다음을 가진다.
WordNode 포인터 타입의 pHead.
이에 BST의 Traversal을 위해 pTail을 추가적으로 구현하였다.

멤버 함수로는 다음을 가진다.
멤버 변수의 값을 초기 값으로 설정하는 '생성자'와 동적 메모리 할당 부분을 해제하는 '소멸자'를 가진다.

데이터들을 Linking하고 Un-Linking하는 Push, Pop함수를 가진다.
queue에서 Last-out 기능이 있다면 Stack 구조처럼 사용할 수 있으므로, BST의 Traversal을 위해 마지막 노드를 Un-Linking하는 Pop_back함수와 pTail의 노드를 리턴하는 Pop_back를

추가적으로 구현하였다.

데이터들을 검색하거나 출력하는 함수 Search, Print 함수를 가진다.

text파일과의 통신을 위해 save함수를 가진다.
isEmpty()함수를 추가적으로 구현하였다.

++각 함수를 구현하는 방법은 기본적인 연결리스트(Linked-List)를 구현하는 방법과 매우 비슷하다. 고려해야 할점은. pHead와 pTail의 값이 계속해서 바뀌므로 적절하게 재설정이 필요하다는 점이다.

++또한 Push&Pop시, pHead가 NULL인경우(자료가 없는 경우)에 따른 예외처리 고려를 필요로 한다.
Push와 Pop에 대한 수도 코드는 다음과 같다.

```
Push
WordNode *pCur <= pHead
if(!pHead) pHead <= pTail <= node
else {
    while(pCur->pNext) pCur <= (pCUr->pNext)
    (pCur->pNext) <= node
    pTail <= node
}

POP
WordNode *pCur <= pHead
if(!pHead) return NULL
pHead <= (pHead->pNext)
(pCur->pNext) <= NULL
return pCur
```

**** CircularLinkedList ****
MEMORIZED 자료구조형의 구현을 위해 사용되는 클래스이다.

따라서 멤버 변수로 다음을 가진다.
WordNode 포인터 타입의 pHead.
이에 순환의 편의성을 위하여 WordNode 포인터 타입의 pTail.

멤버 함수로는 다음을 가진다.
멤버 변수의 값을 초기 값으로 설정하는 '생성자'와 동적 메모리 할당 부분을 해제하는 '소멸자'를 가진다.

데이터들을 Linking하는 Insert함수와
검색, 출력하는 함수 Search, Print 함수를 가진다.

text파일과의 통신을 위해 save함수를 가진다.
isEmpty()함수를 추가적으로 구현하였다.

++각 함수를 구현하는 방법은 이전에서 설명한 큐를 구현하는 방법과 매우 비슷하다. 고려해야 할점은. pHead와 pTail의 값이 계속해서 바뀌므로 적절하게 재설정이 필요하다는 점이다.

++마찬가지로 Insert시, pHead가 NULL인경우(자료가 없는 경우)에 따른 예외처리 고려를 필요로 한다.

Insert에 대한 수도코드는 다음과 같다.

```
Insert
if(!pHead) {
    pHead<=node
    (pHead->pNext) <=pHead
}
else    {
    (node->pNext) <= pHead
    (pTail->pNext) <= node
}
pTail <= node
```

****WordBST****
MEMORINZG 구조를 구현하기 위해 WordBST 클래스를 필요로 한다.
Binary Search Tree이며, Key값은 Word의 값이 된다.

멤버 변수로는 root를 가진다.

멤버 함수로는 생성자와 소멸자를 가지며,
삽입,삭제, 검색을 하는 Insert, Delete, Search를 가진다.
Traversal을 하는 Print 함수를 가진다.
text파일과의 통신을 위해 save함수를 가진다.

Print함수의 경우 VISIT을 언제 하느냐에 따라와 재귀적으로, 혹은 반복자를 이용하냐에 따라 구현하는 모듈들이 나누어지므로
R_PRE, R_IN, R_POST, I_PRE, I_IN, I_POST, I_LEVEL, 그리고 VISIT함수를 추가로 구현하였다.

Recursive에 관한 R_PRE, R_IN, R_POST 함수의 수도코드는 다음과 같다.

```
if(!node) return false
VISIT(node)
CALL RECURSIVELY with node->pNext
CALL RECURSIVELY with node->pLeft
```

여기서 VISIT이 먼저 나온 다음과 같은 형태가 R_PRE이다.

```
if(!node) return false
CALL RECURSIVELY with node->pNext
VISIT(node)
CALL RECURSIVELY with node->pLeft
```

VISIT이 중간에 있으므로 R_IN 이다.

```
if(!node) return false
CALL RECURSIVELY with node->pNext
CALL RECURSIVELY with node->pLeft
VISIT(node)
```

VISIT이 마지막에 있으므로 R_POST이다.

Iterative에 관한 I_PRE, I_IN, I_POST, I_LEVEL 함수의 수도 코드는 다음과 같다.

```
WordNode *currentNode <= root
Queue s

if(!root) return false
ofstream f(MEMORIZING_FILENAME, ios::app)
if(!f) return false

while(true) {
    while(currentNode) {
        if(save) FILEOUT WORD, MEAN
        else VISIT(currentNode)
        s.Push(currentNode)
        currentNode<= (currentNode->pNext)
    }
    if(s.isEmpty()) {
```

```
        f.close
        return true;
    }
    else currentNode <= s.top()
    s.Pop_back()
    currentNode <= (currentNode->pNext)
}
```

위의 수도 코드는 I_PRE에 관한 코드이다.
로직은 다음과 같다.
VISIT-> LEFT -> RIGHT 순이므로,
currentNode가 순회하면서 먼저 VISIT한 후 스택에 푸쉬한다.
이 과정을 LEFT CHILD가 없을때까지 반복해서 한다.
다음 조건은 스택이 비어있을 때이다. 스택이 비어있다는 말은 오더링을 다하였다는 말이므로, 함수를 종료한다.
그렇지 않다면, 스택에서 데이터를 하나 꺼내고, 오른쪽으로 이동한다.

I_IN 함수의 경우,
VISIT이 Push 다음 라인이 아닌,

```
    else currentNode <= s.top()
    VISIT(currentNode)
    s.Pop_back()
```

사이에 삽입함으로써, LEFT -> VISIT -> RIGHT 순으로 구현할 수 있다.

I_POST 함수의 경우
LEFT -> RIGHT -> VISIT 순이므로, 벡터의 방향성이 하나가 아닌 두개가 존재한다. 따라서, 기존의 LEFT가 NULL이 나올때까지 Push하는 과정까지는 동일하나, 바로 RIGHT로 넘어가지 않고 RIGHT가 널인지, 널이 아닌지로 분기하여 다음 수도 코드와 같이 조건 검사를 추가해야 한다.

```
if(!currentNode->pRight) {
VISIT(currentNode)
while(true) {
    temp <= currentNode
    s.pop_back()
    if(s.isEmpty()) return true
    currentNode <= s.top
    if(currentNode->pRight == temp) VISIT (currentNode)
    else    {
        currentNode <= NULL
        break
```

```
    }
}
}
```

핵심부분은 currentNode->pRight와 temp가 같냐이다.
이 의미는, 이전 스택에 있는 노드의 right child가 temp와 같냐는 말인데, 이 경우 이미 RIGHT를 하였다는 의미이므로, VISIT을 한다.

I_LEVEL의 수도 코드는 다음과 같다.

```
WordNode *currentNode <= root
Queue wordQ
WordNode *temp
if(!root) return false
wordQ.push(root)
while(true) {
    temp <= wordQ.pop()
    if(!temp) BREAK
    VISIT(temp)
    if(temp->pLeft) wordQ.Push(temp->pLeft)
    if(temp->pRight) wordQ.Push(temp->pRight)
}
```

즉, VISIT하고 왼쪽자식,오른쪽 자식 순서대로 큐에 넣어 넘버링 순서대로 값이 큐에 저장되도록 설정해두고, 큐에 자료가 없을때까지 반복한다.

INSERT의 경우 root의 key값을 기준으로 인자로 들어온 노드의 key값과 비교한다. 이때 비교는 strcmp를 이용하여 구현하였고,
만약 argv.KEY < root.KEY면 root의 Leftsubtree에서 다시 비교를 수행하고, argv.KEY > root.KEY면 root의 Rightsubtree에서 다시 비교를 수행한다. 이 과정을 반복하다 보면 subtree가 NULL인 경우가 생기는데, 해당 위치에 인자로 들어온 노드를 링킹한다.

Search의 경우또한 INSERT와 매우 흡사한데, subtree가 NULL인 경우 해당 BST에 인자로 들어온 노드와 일치하는 데이터가 없다는 의미이며, argv.KEY == root.KEY가 되는 경우가 생긴다면 해당 값을 출력한다.

Delete
Delete 의 경우 degree에 따라 조건 분기를 하여야 하는데,
조건 분기의 수도 코드는 다음과 같다.
(먼저, ppCur는 지우려는 노드의 부모, pCur는 지우려는 노드이다.)

```
CASE : Degre 1 or 2
if(pCur->pLeft || pCur->pRight) {
```

```
    temp<= (pCur->pLeft) ? (pCur->pLeft) : (pCUr->pRight)
    char *tWord<=NULL, *tMean<=NULL
    if(!pCur->pLeft) while(temp->pLeft) temp <= (temp->pLeft)
    else while(temp->pRight) temp <= (temp->pRight)

    tWord <= temp ->GETWORD()
    tMean <= temp ->GETMEAN()
    Delete (temp->GETWORD())
    pCur->SetVoca(tWord,tMean)
}
case : No Degree
if(!ppCur) root<=NULL
else ppCur->pLeft == pCur? (ppCur->pLeft)<=NULL : (ppCur->pRight)<=NULL
(pCur->pLeft)<=NULL
(pCur->pRight)<=NULL
return pCur
```

**** AlphabetNode ****

AlphabetBST를 구성하기 위해 필요한 Node 클래스이다.
AlphabetNode는 내부에 변수로 WordBST를 가진다.
따라서, 이러한 AlphabetNode를 이용하여 AlphabetBST를 만든다면,
영단어 첫글자를 색인으로 하는 26개의 WordBST가 생성될 것이다.

멤버변수로는 다음을 가진다.
char alphabet,
wordBST *bst,
AlphabetNode *pLeft, *pRight

멤버 함수로는 다음을 가진다.
생성자와 소멸자를 가지고 있으며,

멤버 변수의 값을 리턴하는 GetAlphabet,GetBST,GetLeft,GetRight
멤버 변수의 값을 설정하는 SetAlphabet,SetLeft, SetRight

input에 대한 output의 분기 조건이 거의 없어, 간결하게 구현이 가능하다.

소멸자의 경우, WordNode와 마찬가지로, recursive post-order 방식으로 AlphabetBST형태로 저장된 동적메모리를 해제한다.
다음은 소멸자에 대한 수도코드이다.

```
if(!this) return
```

this->pLeft->~AlphabetNode()
this->pRight->~AlphabetNode()

delete bst

AlphabetBST
MEMORIZING 구조를 위해 Top-module로 구성되는 클래스이다.
멤버 변수와 멤버 함수가 WordBST와 매우 흡사하다.

멤버변수로는 다음을 가진다.
AlphabetNode * root
int WordCnt

WordCnt의경우 최대 100개가 넘으면 안되는 조건에 따라 멤버 변수로 이러한 카운터를 세기 위함이다.

멤버함수로는 다음을 가진다.
생성자와 소멸자가 있으며,

삽입,검색,출력,저장을 위해 Insert,Search,Print,Save를 가진다.

text파일간의 통신을 위해 isEmpty,isFull을 추가로 구현했으며,

print ordering 조건을 위해 WordBST와 동일하게 R_PRE, R_IN, R_POST, I_PRE, I_IN, I_POST, I_LEVEL을 추가로 구현하였다.

WordCnt는 WordNode의 Insert, Delete시에 증/감 되야하므로, 이러한 연결을 위해 incCnt, decCnt, getCnt를 추가로 구현하였다.

구현에 있어 WordBST와 대부분이 동일하며, 특이사항은 WordCnt가 WordBST의 Insert시와 동기화가 안된다는 점이다.
따라서, WordBST Insert와 Delete를 호출할때마다 성공여부에 따라 incCnt, decCnt를 수행시킨다.

Manager
해당 프로그램의 모든 모듈간의 데이터 통신을 담당하는 부분이다.

멤버 변수로는 다음을 가진다.
CircularLinkedList *cll
AlphabetBST *bst

Queue *queue
char *argv[2]

argv[2]는 Manager의 주요 기능을하는 함수들에게 인자를 전달하기 위하여 추가로 구현하였다.

멤버함수로는 다음을 가진다.
MANAGER(), ~MANAGER() 생성자와 소멸자를 가진다.

run(const char * command)
시스템을 구축하고 커맨드라인을 읽어 각 함수들을 call하고 각 기능등을 수행하며, log.txt에 결과를 저장시키는 함수

위에서 부터 순차적으로 LOAD부터 UPDATE까지 함수는 오직 run에 의해서만 호출이 된다.

따라서, command 파일에서 한줄씩 읽어 들여 토큰하여, 적절하게 이를 분배해야한다.

이를 위해 문자열 상수배열로 LOAD부터 UPDATE까지 값을 가지고 있고,
command 파일에서 읽어 들여 토큰한 첫번째 문자열과 이를 strcmp를 이용하여 비교한 다음,

이를 switch문을 통하여 해당 case의 함수를 수행하는 방식으로 구현하였다.

간략하게 수도코드로 이를 표현하면 다음과 같다.

code[9] <= "LOAD","ADD",..."UPDATE","EXIT"

buf<= command.getline
fname <= buf.token
argv[0] <= buf.token
argv[1] <= buf.token
if(buf.token) RETURN

for i ranged[0,9)
strcmp(code[i],fname)

SWITCH(OP)
case 0: CALL LOAD
case 1: CALL ADD

...
case 8: CALL EXIT
default: RETURN

case 내부에서는 함수를 호출하기전에 매개변수가 적절한지 AND와 OR 연산을 이용하여 체크한다. 예를들어, 매개변수가 1개만 필요로 하는 LOAD함수의 경우 다음처럼 구현이 가능하다.
case 0:
if(argv[0] || argv[1] || !LOAD()) PRINT ERROR
else PRINT SUCCESS
BREAK

즉, 매개변수 2개중 하나라도 존재한다면 LOAD는 실행되지 않고 에러메세지를 출력할 것이다.
둘 다 없을 때에만 LOAD함수를 수행하여, LOAD함수의 반환 값이 true라면 3항에 대한 2개의 OR연산 최종 결과가 FALSE가 되어 ELSE가 실행되고 SUCCESS 메세지를 log.txt에 출력하게 된다.

TEST,SEARCH,PRINT,UPDATE에서는 대,소문자가 어떻게 들어오더라도 이를 적절하게 포맷팅해주어야 하는데, 이는 argv[0]을 toupper, tolower하여 구현할 수 있다.

LOAD
LOAD 기능의 핵심은 다음 두가지의 경우이다.
1.자료구조형에 데이터가 전부 들어가 있으면 안된다.
2.LOAD이후 자료구조형에 적어도 하나라도 데이터가 들어가 있어야한다.

조건문 1번을 수도코드로 구현하면 다음처럼 구현할 수 있다.

if(! (Queue.ISEMPTY && BST.ISEMPTY() && CLL.ISEMPTY())) return false
셋 다 비어있을때만 2개의 AND 연산의 결과가 TRUE가 되고, 앞에 NOT이 붙었으므로 해당 조건문은 거짓이 된다.
만약, 셋 중 하나라도 존재한다면 AND 연산의 결과가 FALSE가 되고, 앞에 NOT에 의해 값이 반전되어 해당 조건문이 실행되고, LOAD함수는 실행되지 않는다.

LOAD하는 과정에 대한 Data Flow는 위의 FlowChart로 설명할 수 있다.

이후 조건문 2번을 검사해야하는데, 수도코드로 구현하면 다음과 같다.
return Queue.ISEMPTY && BST.ISEMPTY() && CLL.ISEMPTY() ? false : true
셋다 비어있으면, AND 연산의 결과가 참이 될것이며, 이는 들어온 데이터가 없다는 의미이므로 false를 반환하고, 아닐시 참을 반환하게 된다.

ADD
word.txt에 저장되어있는 파일들을 queue 자료구조형에 불러오는 함수
위의 FlowChart와 Flow가 매우 흡사하며, 핵심은 중복된 단어가 존재한다면, 이를 건너뛰는 부분인데, 이는 다음처럼 구현이 가능하다.
Queue->Search(WORD) || BST->Search(Word[0])->GetBST()->Search(WORD) || CLL->Search(WORD) RETURN CONTINUE

MOVE
Queue에서 BST로 정해진 횟수만큼 자료를 옮기는 함수.

MOVE의 카운터를 저장한 argv[0]은 문자열이므로
이를 정수형으로 바꾸어야한다. 이를 atoi함수를 이용하여 구현하였다.

MOVE에서 예외처리할 조건은 다음과 같다.
1.MOVE의 카운터 값이 1과 100사이인가?
2.MOVE의 카운터 값 + BST내 Word 카운터의 값이 100을 넘지 않는가?

이 두 조건은 간단한 수식을 세워 검사할 수 있다.
ex if(atoi(argv[0]) + bst->getCnt() > 100) return FALSE

SAVE
자료구조형에 저장되어있는 데이터들을 text파일에 저장하는 함수.

핵심은 세 자료형중 적어도 하나에 자료구조형에 데이터가 존재한다면 SAVE 기능을 수행한다는 것이다.
이에 대한 수도코드는 다음과 같다.

```
if(! (Queue.ISEMPTY && BST.ISEMPTY() && CLL.ISEMPTY()) )
    CALL Queue.SAVE
    CALL BST.SAVE
    CALL CLL.SAVE
    return TRUE
else return FALSE
```

TEST
단어를 암기하였는지 확인하고, 일치한다면 BST에서 CLL로 데이터를 옮기는 함수
먼저, 단어가 존재하는지를 검사한다. AlphabetBst의 Search와 WordBST의 Search를 통하여 이를 검사할 수 있다.
이후 단어가 존재한다면, 매개변수의 의미가 해당 단어의 의미와 일치하는지를 검사한다. 이는 strcmp를 이용하여 검사할 수 있다.

만약 둘다 통과한다면, bst에서 delete를 하며, cll에 insert를 한다.

둘중 하나라도 통과하지 못한다면, FALSE를 반환한다.

SEARCH
인자로 지정한 데이터가 자료구조형 안에 있는지 검사하는 함수

모든 자료구조형을 돌면서 해당 단어가 있는지를 체크한다.
이에 대한 수도코드는 다음과 같다.

searchList[0] <= (queue->Search(argv[0])
searchList[1] <= (bst->Search(argv[0][0])->GetBST()->Search(argv[0]))
searchList[2] <= (cll->Search(argv[0])

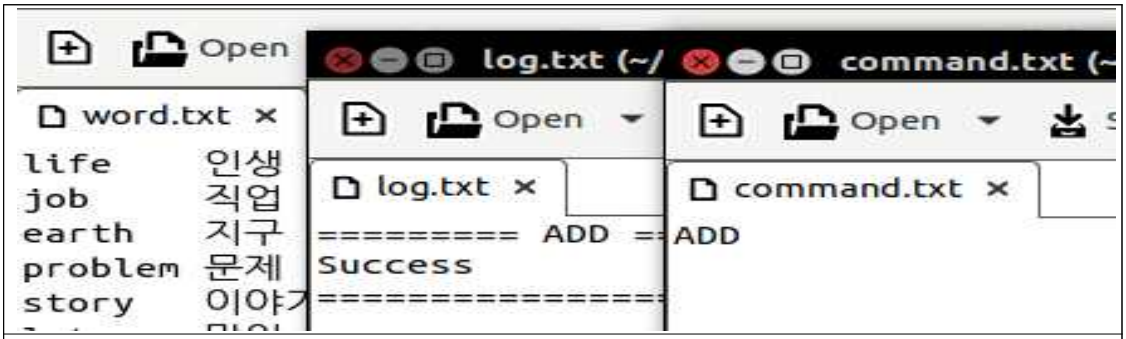
만약 셋 다 NULL이라면 이는 해당 단어가 존재하지 않는다는 의미가 된다.

셋 중 하나에 해당 단어가 존재한다면, 셋 중 하나는 NULL이 아닌 값을 가지게 될것이다.
이 경우 두 가지를 고려하게된다.
하나는 argv[1]이 존재하는 경우이다.
이미 run에서 search를 호출할 때 매개변수가 적합한지를 체크하였다.
즉, 이경우는 UPDATE에 의하여 search가 호출된 경우이며
따라서 UPDATE에 맞게 SetMean()을 해주어야한다.
argv[1]이 없다면 SEARCH에 맞게 word와 mean을 출력한다.

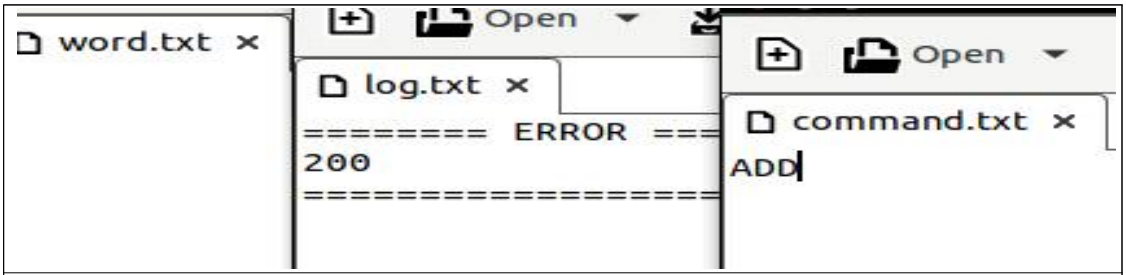
PRINT
인자로 지정한 자료형에 대한 출력을 하는 함수
argv[0]과 TO_MEMORIZE, MEMORIZING, MEMORIZED를 비교한다.
MEMORIZING의 경우 argv[1]이 존재하는지도 체크한다.
이후 각 문자열 상수에 맞게 각 자료구조형의 Print를 출력해준다.

UPDATE
인자로 지정한 단어의 뜻을 변경하는 함수
SEARCH를 호출하고 해당 리턴값을 리턴한다.

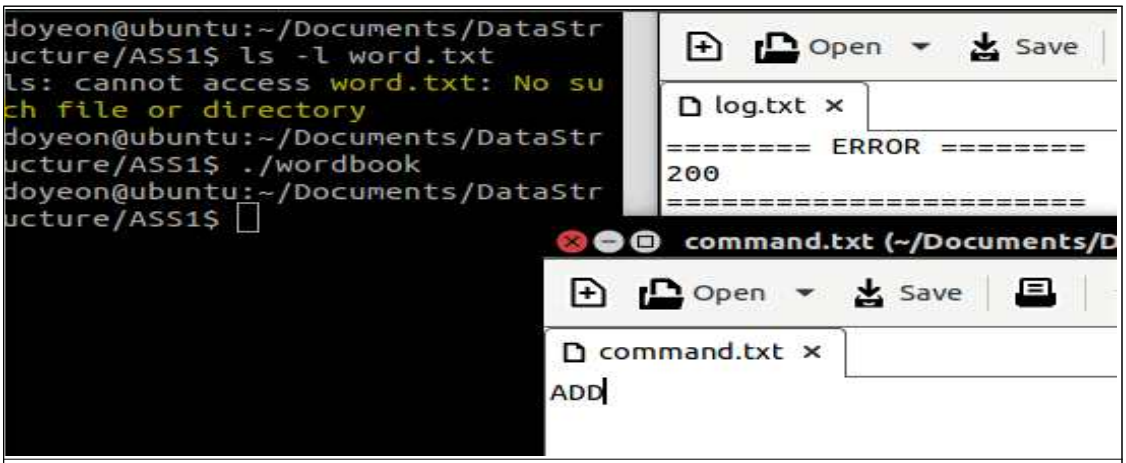
- **Result Screen**



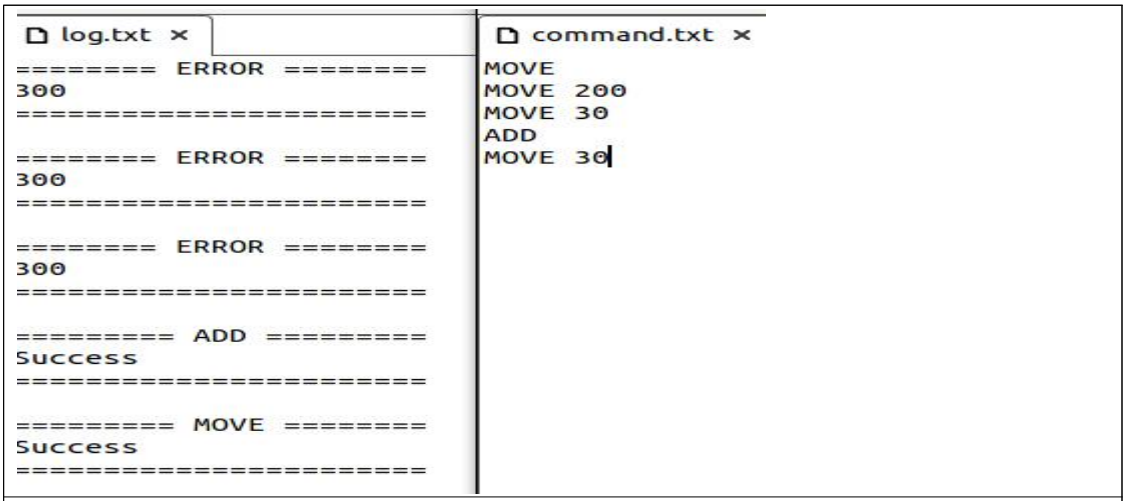
ADD 명령어 실행 화면



Text파일이 빈파일 일 때, ADD 실행 화면이다.
ADD의 ERROR 코드(=200)가 출력 된 것을 확인 할 수 있다.

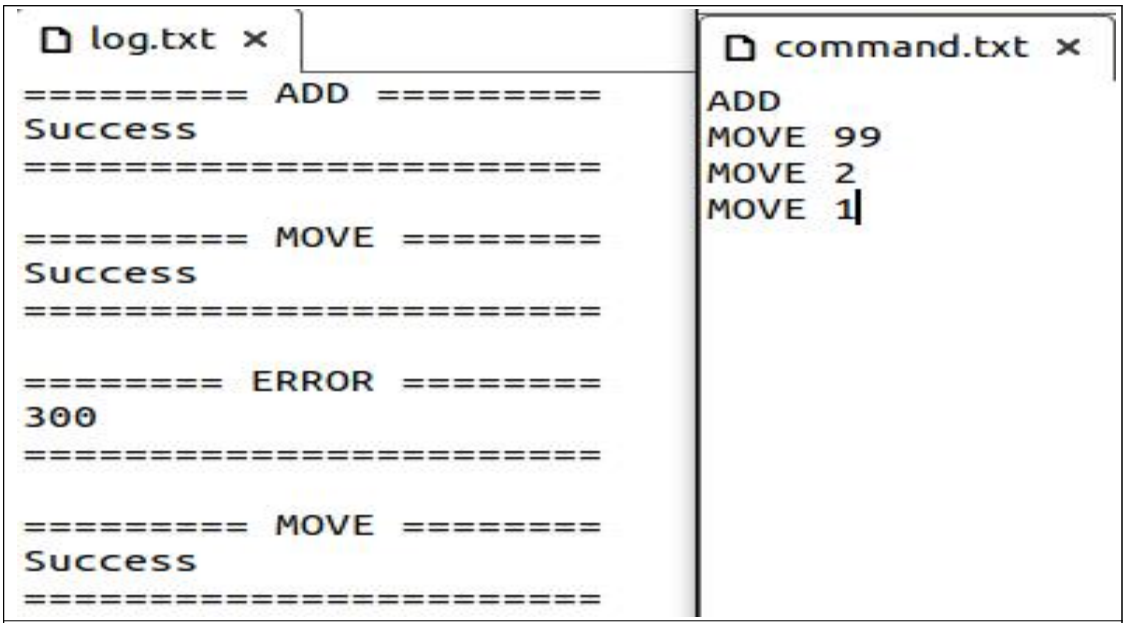


Text파일이 존재 하지 않을 때, ADD 명령어 실행 화면이다.
ADD의 ERROR 코드(=200)가 출력 된 것을 확인 할 수 있고,
Terminal 창에 file or directory를 찾을 수 없다고 출력 된 것을 확인 할 수 있다.



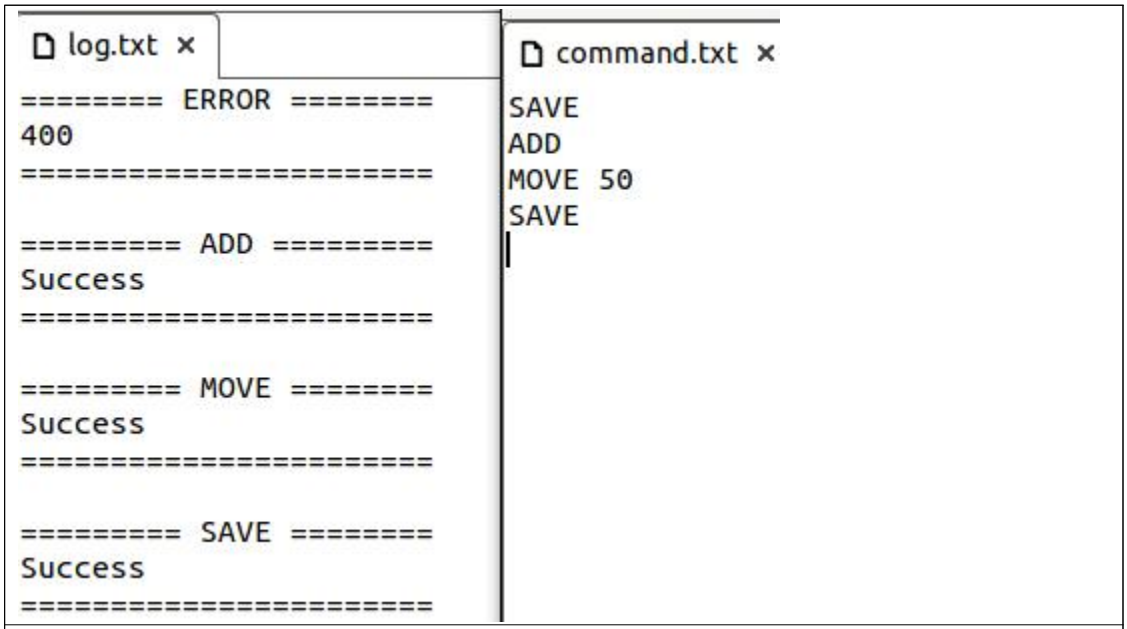
MOVE의 결과화면

1. 인자가 없으므로 Error
2. 범위가 초과했으므로 Error
3. MOVE 30 명령어 정상적으로 실행 확인
4. ADD 명령어 정상적으로 실행 확인
5. MOVE 30 명령어 정상적으로 실행 확인

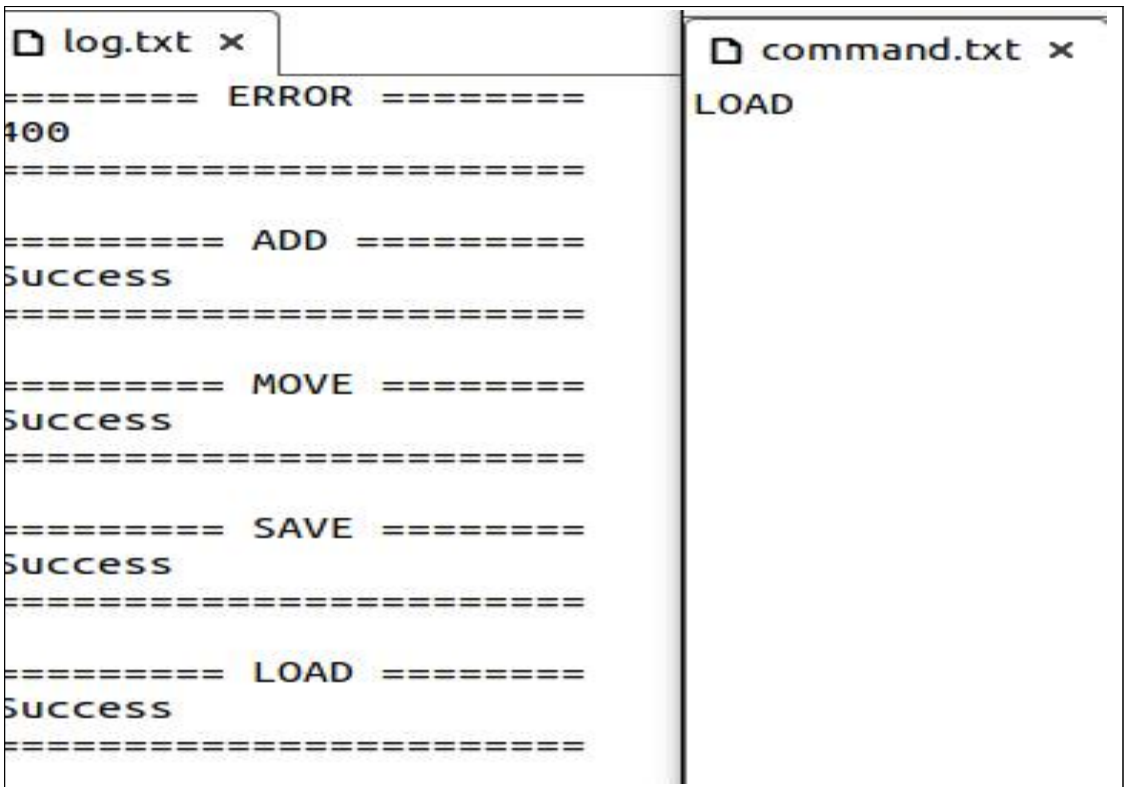


MOVE 99 이후, MOVE 2는 WordCnt 100을 초과하게 되므로, Error 출력

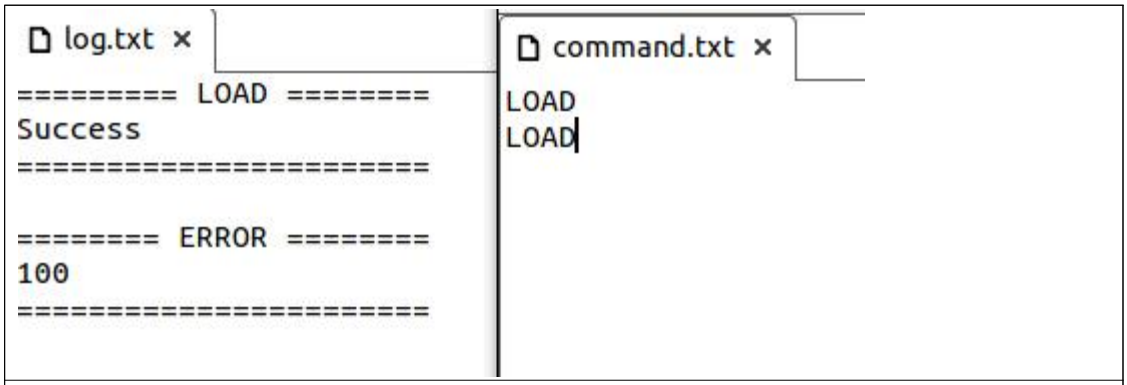
MOVE 1은 초과하지 않으므로 (=100) 정상적 출력



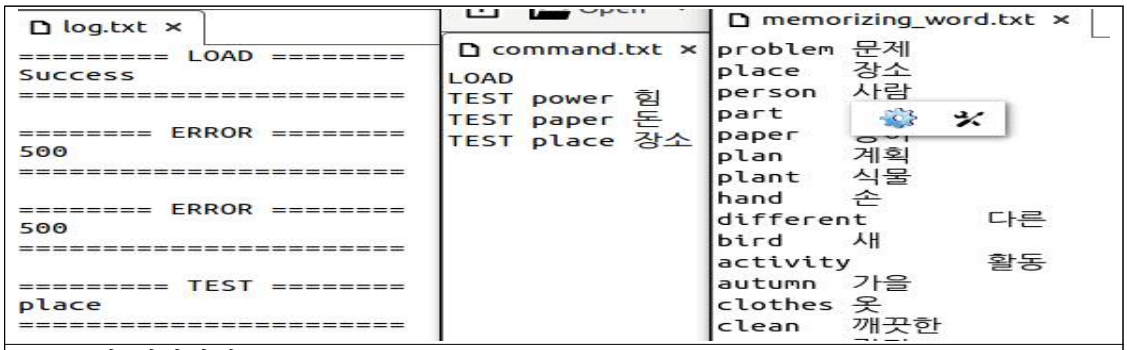
SAVE의 결과 화면.
3개의 자료 구조 전부 데이터가 없으므로 SAVE Error 이후, 자료 구조의 데이터가 있으므로 SAVE 성공



이 후, Text file에 단어장 정보가 있으므로 Load 성공



LOAD가 성공 한 상황에서 한번 더 LOAD를 하였으므로, Error(이미 자료구조 형 안에 데이터가 존재하므로)



TEST의 결과화면

Power 힘은 존재 하지 않는 단어 이므로 Error 출력

Paper 돈은 존재 하는 단어나, 의미가 달라서 Error 출력

Place 장소는 존재 하는 단어이고, 의미도 일치하므로 Success

log.txt ×

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

======== SEARCH =======
course 강좌
=======================

======== UPDATE =======
course 강좌 -> 과정
=======================

======== ERROR ========
600
=======================

======== ERROR ========
800
=======================
```

command.txt ×

```
LOAD
SEARCH course
UPDATE course 과정
SEARCH money
UPDATE money 지폐
```

SEARCH와 UPDATE의 결과화면.

1.Course는 단어장에 존재하기 때문에 SEARCH 성공

2.Course의 뜻을 강좌 -> 과정으로 업데이트 성공

3. Money는 단어장에 존재하지 않기 때문에 Error 출력 (=SEARCH)

4. Money는 단어장에 존재하지 않기 때문에 Error 출력 (= UPDATE)

Open Save

log.txt ×

```
======== PRINT ========
problem 문제
place   장소
hand    손
earth   지구
life    인생
lot     많이
job     직업
name    이름
story   이야기
work    일
=======================

======== PRINT ========
problem 문제
place   장소
hand    손
earth   지구
life    인생
lot     많이
job     직업
name    이름
story   이야기
```

command.txt (~/Docu

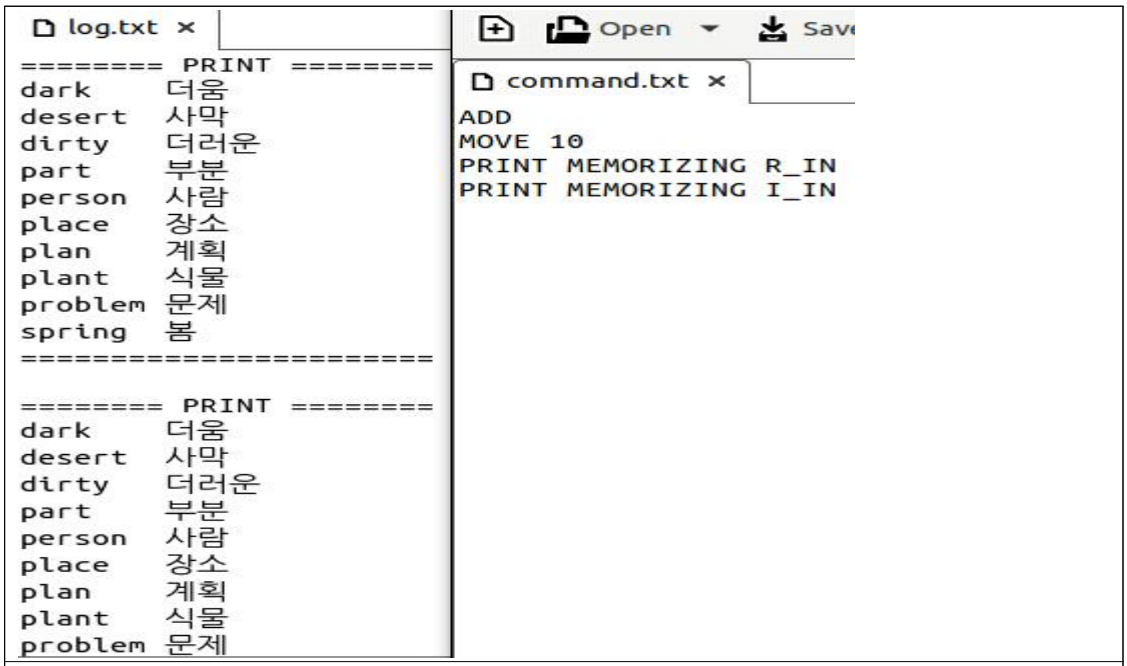
Open Save

command.txt ×

```
ADD
MOVE 10
PRINT MEMORIZING R_PRE
PRINT MEMORIZING I_PRE
```

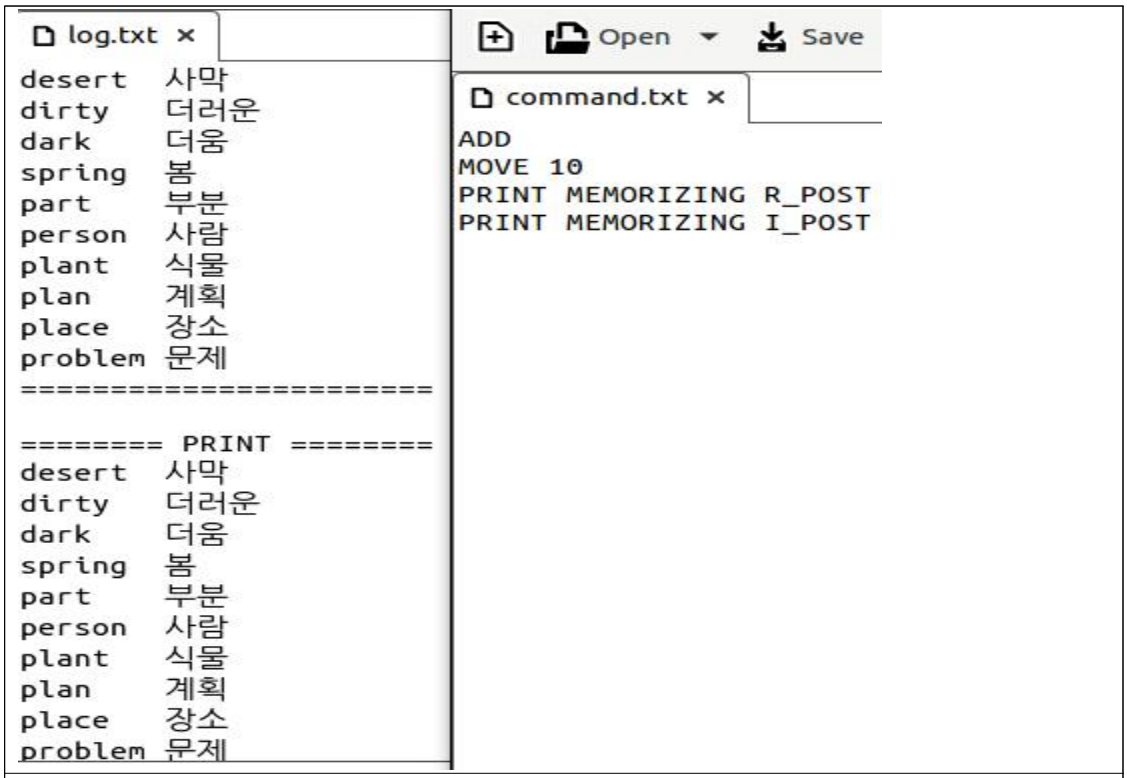
PRINT의 결과 화면

Memorizing 내의 단어들을 R_PRE(Reculsive 방식으로 구현 한 Preorder), I_PRE(Iterative 방식으로 구현 한 Preorder)로 PRINT한 결과.



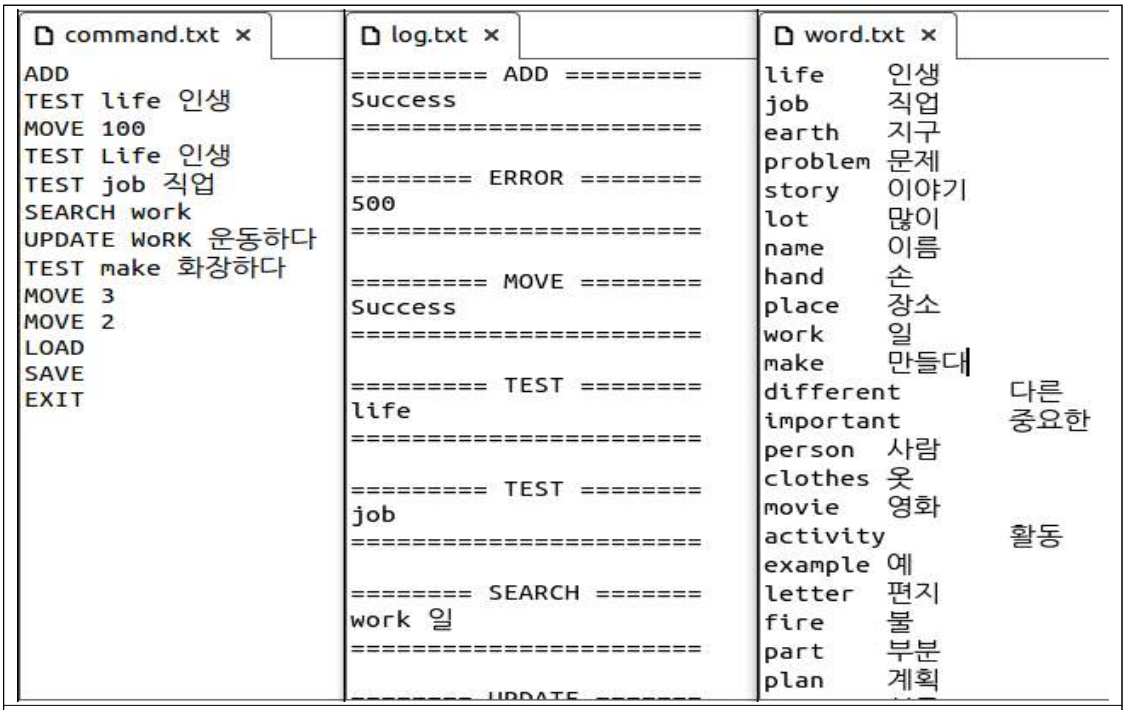
PRINT의 결과 화면

Memorizing 내의 단어들을 R_IN(Reculsive 방식으로 구현 한 Inorder), I_IN(Iterative 방식으로 구현 한 Inorder)로 PRINT한 결과.



PRINT의 결과 화면

Memorizing 내의 단어들을 R_POST(Reculsive 방식으로 구현 한 Postorder), I_POST(Iterative 방식으로 구현 한 Postorder)로 PRINT한 결과.

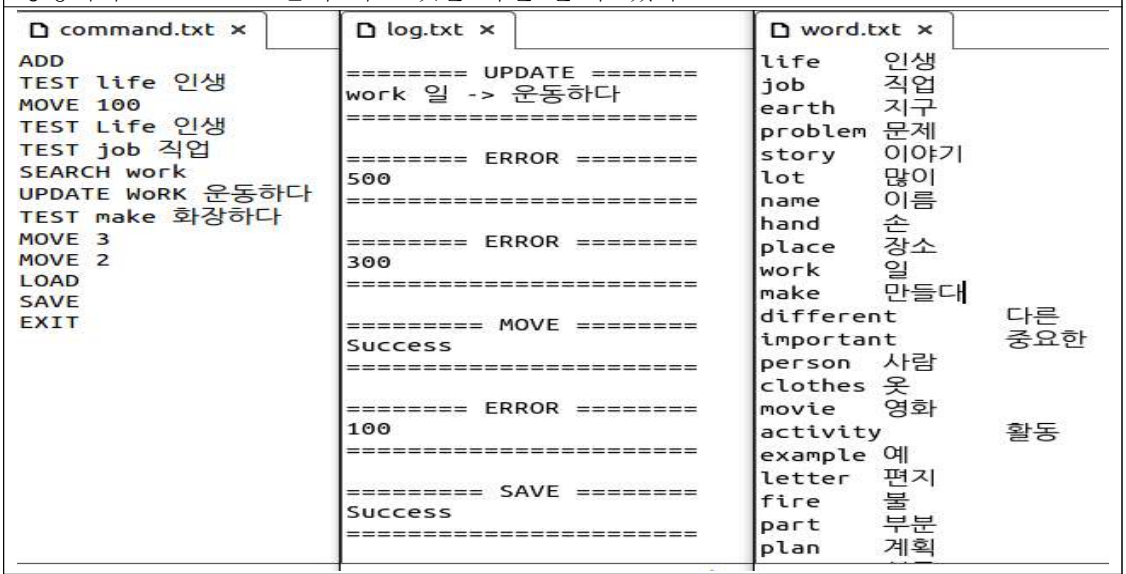


command.txt
```
ADD
TEST life 인생
MOVE 100
TEST Life 인생
TEST job 직업
SEARCH work
UPDATE WoRK 운동하다
TEST make 화장하다
MOVE 3
MOVE 2
LOAD
SAVE
EXIT
```

log.txt
```
========= ADD =========
Success
=======================

======== ERROR ========
500
=======================

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

========= TEST ========
life
=======================

========= TEST ========
job
=======================

======== SEARCH =======
work 일
=======================
```

word.txt
```
life      인생
job       직업
earth     지구
problem   문제
story     이야기
lot       많이
name      이름
hand      손
place     장소
work      일
make      만들다
different           다른
important           중요한
person    사람
clothes   옷
movie     영화
activity            활동
example   예
letter    편지
fire      불
part      부분
plan      계획
```

처음에 ADD함수가 실행되어 Success가 출력 되고,
BST가 비어있기 때문에 TEST 명령어에서 Error가 출력 된다.
그 다음, MOVE 명령어가 실행되어 Success가 출력 되고,
MOVE 명령어를 통해, BST에 Life와 Job이 존재하므로 TEST 명령어에 대해 Sucess가 출력 된다. 또한, work라는 단어가 기존의 텍스트 파일에 존재하였으므로 SEARCH 명령어가 Success로 출력 되는 것을 확인 할 수 있다.



command.txt
```
ADD
TEST life 인생
MOVE 100
TEST Life 인생
TEST job 직업
SEARCH work
UPDATE WoRK 운동하다
TEST make 화장하다
MOVE 3
MOVE 2
LOAD
SAVE
EXIT
```

log.txt
```
======== UPDATE =======
work 일 -> 운동하다
=======================

======== ERROR ========
500
=======================

======== ERROR ========
300
=======================

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

======== ERROR ========
100
=======================

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

word.txt
```
life      인생
job       직업
earth     지구
problem   문제
story     이야기
lot       많이
name      이름
hand      손
place     장소
work      일
make      만들다
different           다른
important           중요한
person    사람
clothes   옷
movie     영화
activity            활동
example   예
letter    편지
fire      불
part      부분
plan      계획
```

대•소문자에 관계없이 UPDATE 명령어가 정상적으로 실행되어 Success가 출력 되는 것을 확인 할 수 있고, make라는 단어의 뜻은 만들다이므로 의미가 일치 하지 않아 Error가 출력 되었다. BST에서 CLL로 넘어간 단어의 수는 총 2개이므로, WordCnt는 98개인 상태이다. 여기서, MOVE 3 명령어를 입력하면, WordCnt가 100개를 넘어버리므로, Error가 출력 되었고, MOVE 2 명령어를 입력하면, WordCnt가 총 100개가 되므로 Success가 출력 된다. 그리고, 현재 Queue, BST, CLL에 단어가 존재하므로, LOAD명령어에 대해 Error가 출력 되는 것을 확인할 수 있다. 마지막으로 텍스트 파일에 성공적으로 저장 되어 SAVE 명령어에 대해 Success가 출력 되는 것을 확인 할 수 있다.

• **Consideration**

이름	프로젝트에서 맡은 역할	본인 스스로 생각하는 자신의 점수
이도연	코드 전반에 걸쳐 구현(Manager,AlphabetBST,AlphabetNode, CircularLinkedList,Queue,WordBST,Word Node) 및 핵심 기술 구현(AlphabetBST-WordBST간의 WordCNT연동, Load,Save시의 텍스트 파일과의 통신,WordBST의 delete, Traversal Ordering 부분)	10
고찰	프로젝트 아이디어에 대해 상당히 참신하다는 느낌을 받았다. 외워야 할 단어는 알파벳을 색인으로 보는 방법이 좋으며, 외운 단어는 순환하는 방식이 암기 지속력에 좋다고 생각하기 때문이다. 학부에서 배우는 이론 과목이 이렇게 실생활에 접목될 수 있다는 부분에서 정말 좋았다. 이 프로젝트는 많은 모듈들이 서로 상호작용하여 통신하고 있다. 따라서, 각 통신간에 지켜야 할 프로토 타입을 명시해야 간결하고 목적에 맞게 송/수신 할 수 있다고 생각하는 데, 이러한 면에서 아쉬운 점은 제안서 내용이 애매모호하게 느낀 점이 많았다는 점이다. (예를 들어, BST 연결 규칙에서 Deletion시, 자식 노드가 1개일 경우에 대해 설계 수업에서 교수님께서 알려주신 방법과 프로젝트에서 구현하는 deletion 방법이 달랐다는 점에서와 다른 하나의 자식 노드를 가지는 경우가 symmetry한지에 대한 의문이 들었었다.) 팀 프로젝트에 대해 느낀 점은 제한되는 사항이 많았다는 점이다. 공모전 출전과 비교해서 생각해본다면 어떠한 알고리즘의 구현의 최적화를 찾는 일과 주제를 구현함에 다양성이 존중되는 점과의 차이를 먼저 생각하게 되었다. 따라서 다양성 보다는 특정 방향으로 많이 흘러가지 않았는지 다시금 생각해보게 된다. 만일, 잘 정의된 특정 모듈들을 이용하여 오픈 주제로 과제가 진행된다면 좀 더 흥미롭게 진행하였을 수도 있을 것 같단 생각이 든다.	
임연수	플로우 차트 작성 및 레포트	1
고찰	기본적으로 코딩 실력이 저조한 편인데, 마감을 얼마 안남은 상황에서 과제를 시작 하여 같이 하는 팀원에게 도움을 주지 못한 것 같다. 과제에 대한 이해도 적었고, 초반에 과제에 대한 노력 또한 적었 어서 결국 팀원 진도에 따라 맞추지 못하였다. 과제에 대한 고찰을 하기 이전에 팀원에게 미안한 생각이 많이 들었고, 더 노력 해야겠다는 생각이 들었다. 과제 난이도 또한 나에게는 어렵게 느껴져서 남들에 비해 많이 부족하다는 것도 깨달을 수 있었다.	