Data Structure Report

Project #1

Professor	이기훈 교수님
Department	Computer Engineering
Team Leader	나승학(2012722020)
Member1	주정인(2013722006)
Member2	송민규(2013722042)
Date	2016. 10. 07(금)

Contents

1. Introduction

- Program Explanation

1차 project는 영어 단어장 프로그램을 구현해야 한다. 영어 단어장에는 앞으로 외워야 할 단어장(TO_MEMORIZE), 현재 외우고 있는 단어장(MEMORIZING), 단어를 암기한 단어장(MEMORIZED)를 포함한다. 각 단어장은 각각 Queue, Binary Search Tree, Circular Linked List의 자료구조로 설계한다.
프로그램은 명령어가 저장된 텍스트 파일로 동작을 수행하며 에러 발생 시 에러 코드를 출력하고 출력 결과에 대해서는 로그 파일에 저장된다. 각 단어장의 동작은 다음과 같다.

TO_MEMORIZE는 MOVE 명령어를 사용하여 N개의 단어를 MEMORING으로 옮길 수 있으며 새로운 단어가 들어오면 Push를 수행하여 노드에 단어를 저장하고 Pop을 통해 MEMORIZING으로 단어를 이동시킨다. TO_MEMORIZE에서 단어를 전달 하여 MEMORIZING 단어장에 저장된 단어들이 모여 BST를 이룬다. BST에는 단어들이 저장되어 있는 단어 노드와 첫 글자가 같은 노드들의 집합인 단어 BST가 있고, 알파벳의 첫 글자를 저장하는 알파벳 노드와 알파벳 노드의 집합인 알파벳 BST가 있다.

MEMORINZG의 특징으로는 최대 100개의 단어 이하가 존재 해야하며 이를 어길 경우 MOVE 명령어에 대한 에러 코드를 출력한다. 단어 BST가 형성되는 규칙으로는 부모 노드보다 알파벳 순서가 낮은 노드는 왼쪽 서브 노드에 위치해야 하고, 알파벳 순서가 큰 노드는 오른쪽 서브 트리로 형성 되어야 한다. 단어를 외워 MEMORING의 노드가 제거 될 때에는 제거할 노드의 왼쪽 자식 노드가 존재할 경우, 왼쪽의 자식 노드 중 알파벳 순서에서 가장 뒤에 존재하는 노드가 제거되는 노드 위치로 이동하며, 왼쪽 노드에 자식 노드가 존재 하지 않을 경우 오른쪽 자식 노드 중 알파벳 순서에서 가장 앞에 존재하는 노드가 제거되는 노드의 위치로 이동한다.

MEMORIZED 단어장은 MEMORIZING의 단어장에서 TEST명령어를 사용하여 단어를 외웠다고 판단할 시에 MEMORIZED로 옮겨지고, 옮겨진 첫 단어는 Head Pointer가 가리키며 그 단어가 다시 Head Pointer를 가리키는 환형 연결 리스트구조이다. 만약 단어가 이미 존재 할 경우, Head Pointer가 가리키는 노드의 바로 뒤에 있는 노드 사이에 연결하여 단어의 중복을 피한다.

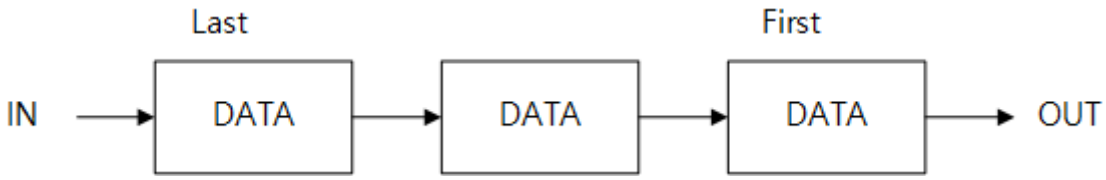
이러한 동작을 하는 단어장은 앞서 설명하였던 것처럼 명령어에 의해 동작을 하며, 그 명령어에는 LOAD, ADD, MOVE, SAVE, TEST, SEARCH, PRINT, UPDATE가 있고 만약 명령어의 오류 및 잘못된 인자값에 의한 에러 코드를 출력 한다. 그에 해당하는 에러 코드와 명령어의 세부 동작 및 프로그램 설계에 대한 요구사항과 이번 프로젝트에서 사용한 추상 자료형에 대한 설명은 아래에서 확인이 가능하다.

- ADTs

■ Queue

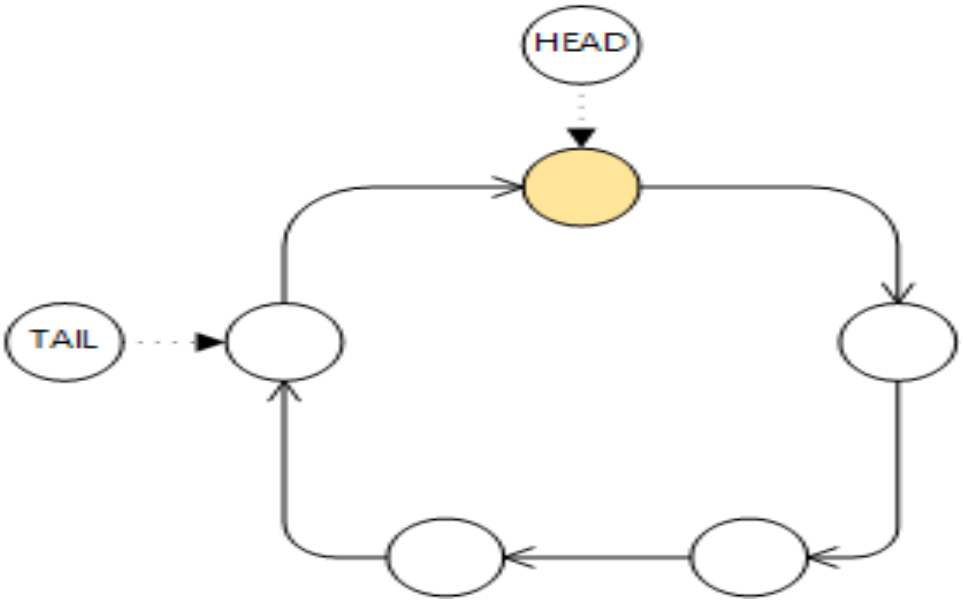
Queue란 '줄', '줄을 서서 기다리다.'라는 뜻을 가지고 있다. 일상 생활에서 이러한 큐 구조를 많이 접할 수 있는데, 예를 들면, 은행 창구에서 번호표를 뽑아 대기하거나, 신호를 기다리는 자동차 등은 일종의 큐 구조에 해당한다고 볼 수 있다. 이러한 큐 구조에 공통적으로 적용되는 특징이 있는데 FIFO (First in, First out), 또는 LILO (Last in Last out)구조라는 것이다. 즉 먼저 들어간 것이 가장 먼저 나오고, 가장 마지막에 들어간 것이 가장 마지막에 나온다. 이러한 큐 구조는 컴퓨터 과학 전반에 자주 쓰이는 자료구조 인데 가장 대표적인 예로 '버퍼(Buffer)'를 들 수 있다. 다량의 데이터가 입력이 되면, 이를 처리하지 못할 수가 있다. 이를 해결하기 위해, 순서대로 입력된 데이터를 보관해두는 곳을 바로 버퍼라고 한다. 은행에 가면 바로 은행원을 만날 수 없어서 번호표를 뽑고 의자에서 기다리는 것은 버퍼와 비슷한 개념에 해당한다.

< Queue >

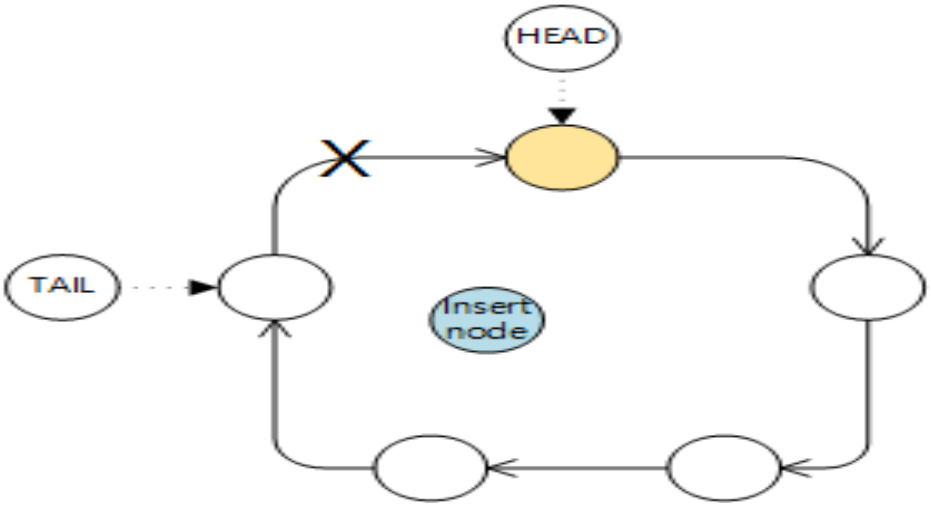


■ Circular Linked List

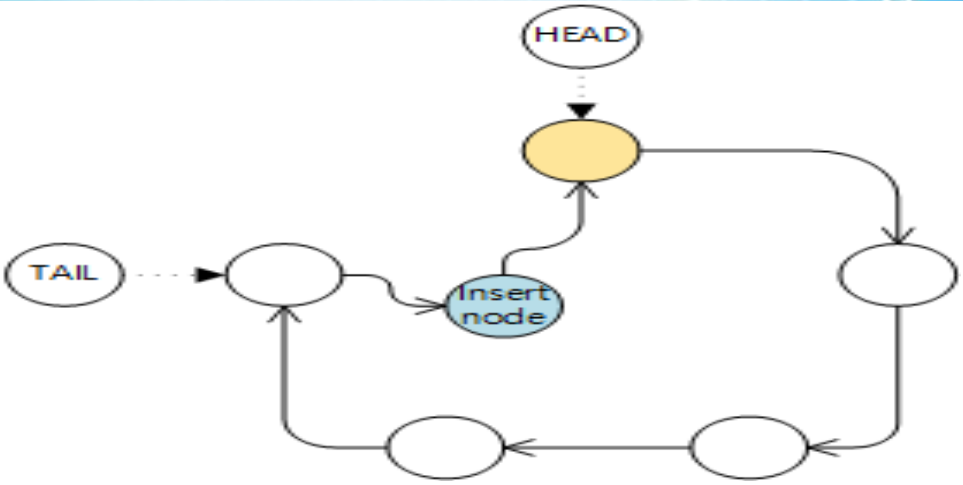
Circular Linked List는 Singly Linked List의 특정 노드 검색 시 대상 이전의 노드들에 대해서 검색이 불가능하다는 점과 검색 시 항상 첫 노드부터 검색해야 하는 단점을 보완하여 첫 번째 노드와 마지막 노드를 가리키도록 구성이 되어있고 첫 번째 노드와 마지막 노드의 구분이 없다. 또한 어떤 노드의 연결 필드에서도 NULL 값을 갖지 않고 현재 위치에서 계속해서 검색 작업을 할 수 있다는 장점이 있다.



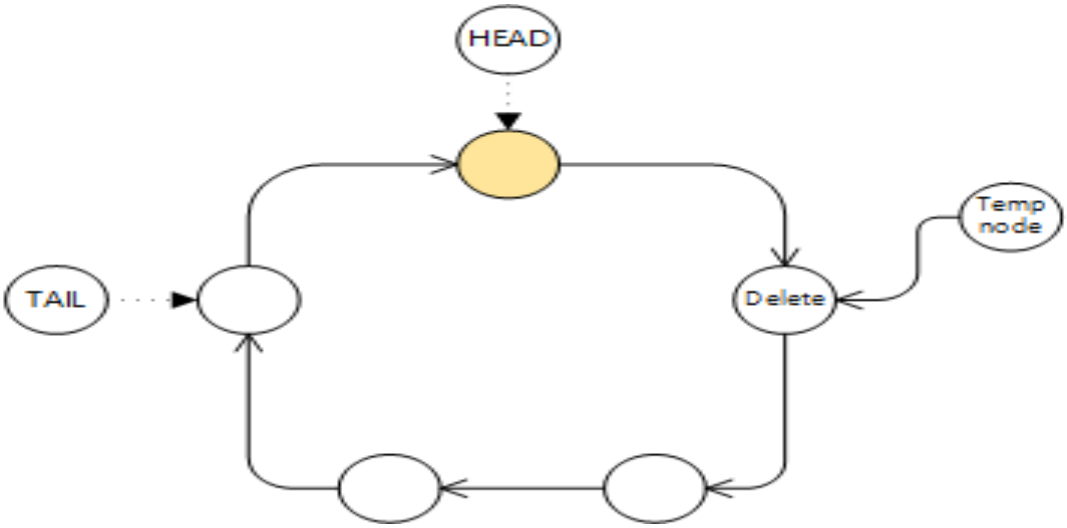
- 노드 추가시 기존의 연결을 끊고



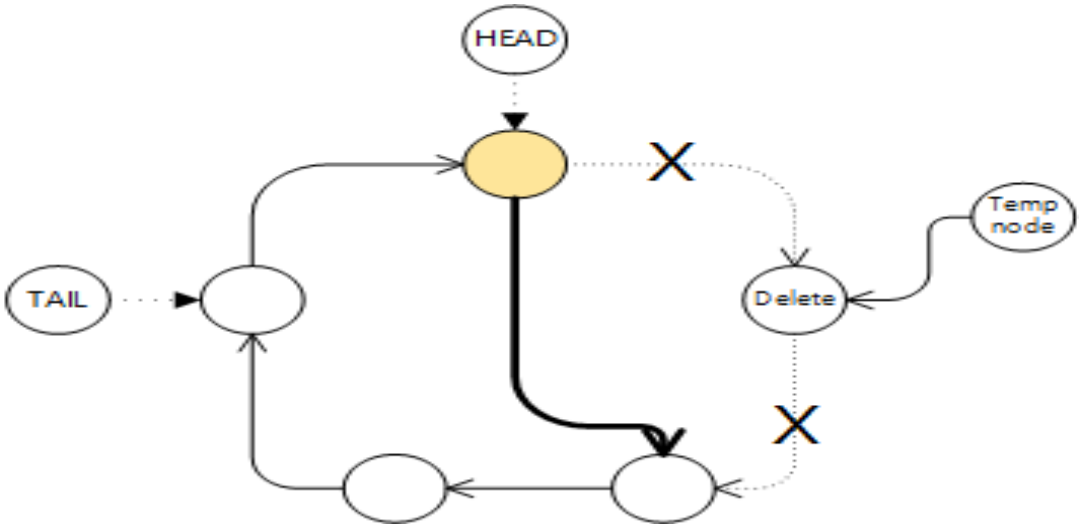
- 추가하고자 하는 노드에 연결을 시켜주면 된다.



- 노드 삭제 시에도 임의의 Temp node를 설정 후 삭제 하고자 하는 노드를 가리킨 다음

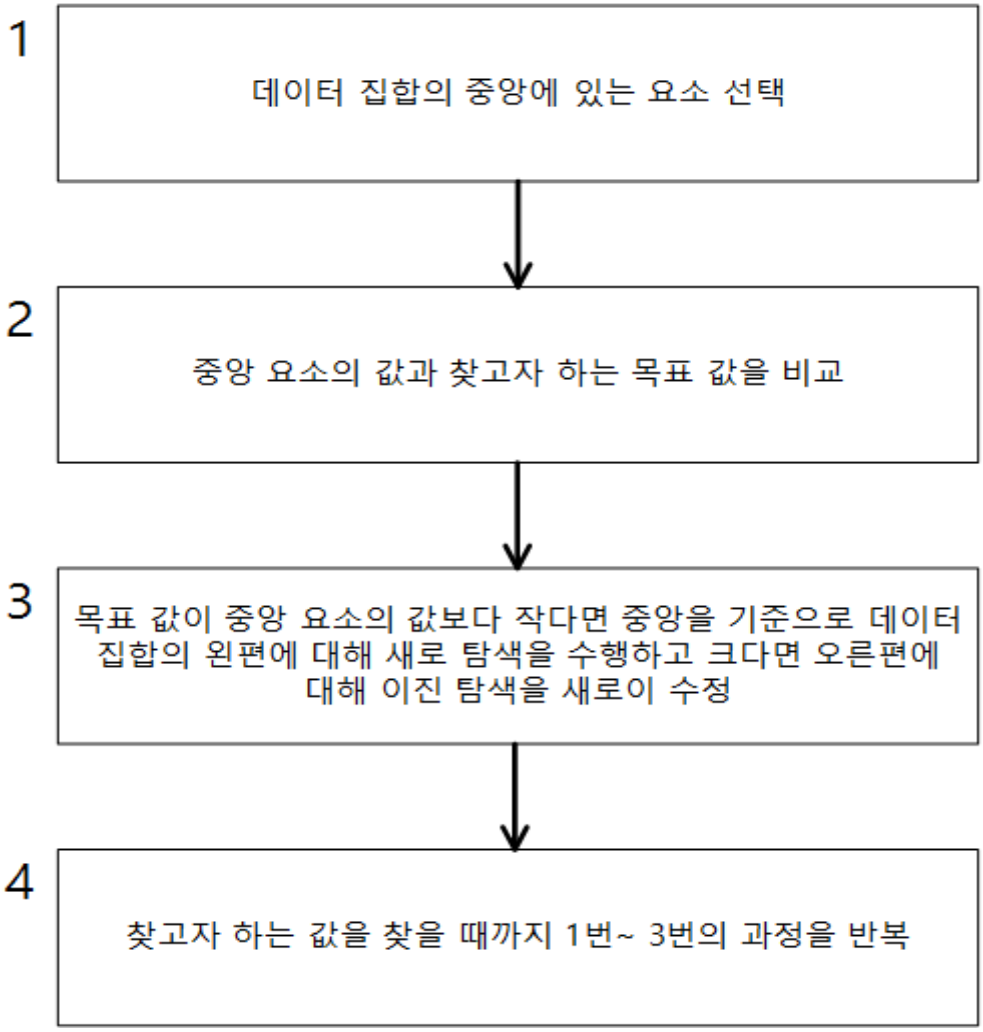


- Delete 노드에 연결을 없애주고 각 노드를 연결시키면 된다.



■ Binary Search Tree

Binary Search Tree는 정렬된 데이터의 집합에서 사용할 수 있는 고속 탐색 알고리즘이다.



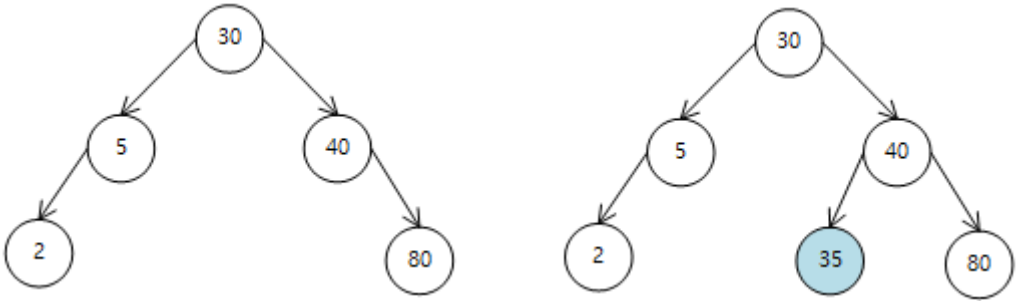
Binary Search Tree는 탐색 대상의 범위가 1/2 -> 1/4 -> 1/16 -> ...
으로 계속 감소하기 때문에 탐색의 시간이 적게 걸린다.
N을 데이터 집합의 크기, X를 탐색 반복의 횟수라고 하면

$$1 = \mathrm{N} \times \left(\frac{1}{2}\right)^{x}$$

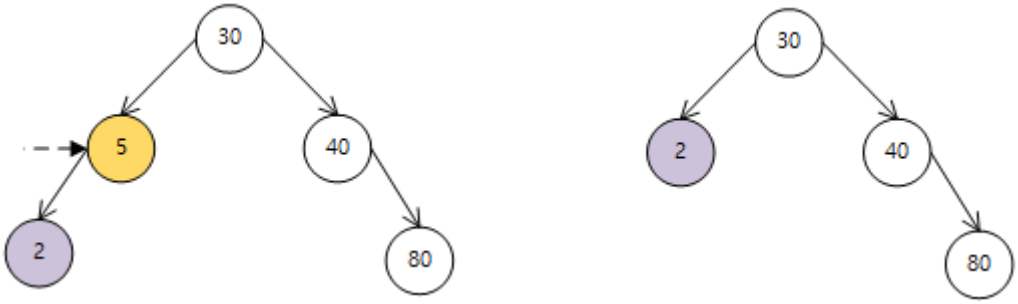
$$x = log_2 \mathrm{N}$$

위와 같은 식으로 나타낼 수 있다.

Binary Search Tree 삽입을 하기 위해서 기존의 원소들이 가지고 있는 키 값과 다른 가를 확인하여야 한다. 이를 위해 탐색이 수행된다. 만약 탐색이 실패하면 탐색이 종료된 그 지점에 쌍을 삽입한다. 예를 들어 키 값 80을 가진 쌍을 트리에 삽입하기 위해서는 먼저 트리에서 80을 탐색해야 한다. 이 탐색은 실패하고 마지막으로 검사한 노드의 키 값이 40이다. 새로운 쌍은 이 노드의 오른쪽 자식으로 삽입하면 된다. 이미 이 키를 가진 쌍을 포함하고 있으면 단순히 이 키에 연관된 원소를 변경하면 된다.



삭제하고자 하는 노드의 왼쪽 자식이 있으면 왼쪽 subtree 중 가장 큰 노드를 선택해서 삭제 할 노드의 위치로 이동시키고 왼쪽 자식 노드가 없을 경우, 오른쪽 subtree중 가장 작은 노드를 삭제 할 위치로 이동시킨다.



- Instruction Function & Example

instruction	Instruction Function & Command Example	Error Code
LOAD	LOAD 명령어는 단어장의 정보를 불러오는 명령어다. 단어장의 정보가 텍스트 파일에 존재 할 경우 각각의 단어장에 있는 자료구조에 이전과 동일한 형식을 띄도록 데이터를 저장하고, 자료구조에 데이터가 들어가 있거나 파일이 존재하지 않을 경우 에러 코드를 출력한다. ex) LOAD	100
ADD	ADD 명령어는 단어장의 단어들을 읽어온다. 읽어온 단어들을 TO_MEMORIZE(외워야 할 단어장)에 저장한다. 만약 단어장에 이미가 존재하는 경우, 에러를 발생하지 않고 중복을 제외한 단어를 저 TO_MEMORIZE 단어장에 저장한다. 만약 단어 텍스트 파일이 존재하지 않거나 파일에 단어 정보가 존재하지 않을 경우 에러 코드를 출력한다. ex) ADD	200
MOVE	MOVE 명령어는 입력한 수만큼 TO_MEMORIZE(외워야 할 단어장)의 단어를 MEMORIZING(현재 외우고 있는 단어장)으로 이동 시키는 명령어로, 1부터 100사이의 숫자만 입력해야 하며 입력한 수와 MEMORIZING에 저장되어 있는 단어의 수를 더한 수가 100을 초과할 수 없다. 만약 100을 초과하거나, 입력한 수만큼의 단어가 TO_MEMORIZE에 존재하지 않을 경우 에러 코드를 출력한다. ex) MOVE 50	300
SAVE	SAVE 명령어는 단어장의 정보를 저장한다. 각각의 자료구조에 저장된 단어들을 단어장의 텍스트 파일에 저장하는 기능을 가진다. 만약 MEMORIZING 단어장에 단어를 알파벳 순서로 저장할 경우 노드가 한쪽으로 쏠리는 Skewed Binary Tree가 되므로 주의해야 하며, 단어장의 정보가 존재하지 않을 경우 에러 코드를 출력한다. ex) SAVE	400
TEST	TEST 명령어는 단어를 외웠는지 확인하는 명령어다. 영어 단어와 한글 뜻을 입력하여MEMORIZING(현재 외우고 있는 단어장)에 존재하면서 뜻이 맞으면, MEMORIZING에서 MEMORIZED(외운 단어장)으로 이동시킨다. 만약 입력한 단어가 MEMORING에 존재하지 않거나 한글 뜻이 다를 경우 에러 코드를 출력한다. ex) TEST data 자료	500

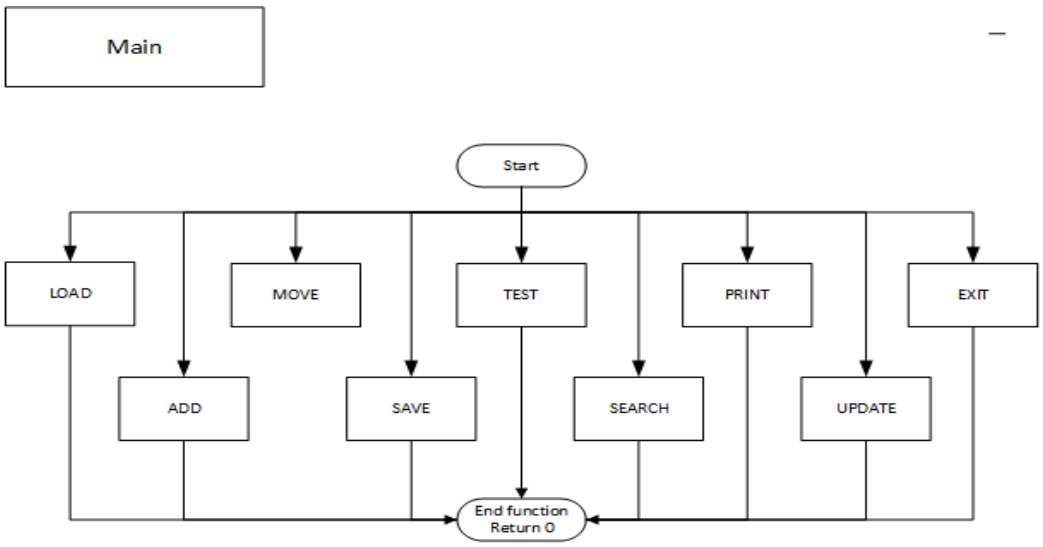
SEARCH	SEARCH 명령어는 단어를 찾아 뜻을 출력한다. 3개의 단어장에 입력한 단어를 찾아 단어가 존재하면, 영어 단어와 한글 뜻을 출력하고, 단어가 없다면 에러 코드를 출력한다. ex) SEARCH data	600
PRINT	PRINT 명령어는 출력할 단어장의 이름을 입력하여 입력한 단어장의 단어들을 출력하는 명령어다. TO_MEMORIZE를 입력하면, TO_MEMORIZE에 저장된 단어들이 Header에서부터 순서대로 출력된다. MEMORIZING은 Binary Search Tree구조로써, 뒤에 R(재귀함수) 또는I(반복문) 에 PRE(전위순회), IN(중위순회), POST(후위순회), LEVEL(레벨순회)를 단어장 이름 뒤에 추가로 입력하여 단어 출력 형식을 정하여 그에 맞게 단어를 출력한다. MEMORIZED는 단어를 외운 순서대로 출력한다. 만약, 입력한 단어장의 정보가 존재 하지 않을 경우 에러 코드를 출력한다. ex) PRINT TO_MEMORIZE ex) PRINT MEMORIZING R_PRE(재귀함수를 이용하여 pre order로 출력한다) ex) PRINT MEMORIZED	700
UPDATE	UPDATE 명령어는 단어의 뜻을 변경한다. 단어를 입력하여 3개의 단어장에 입력한 단어가 존재하면, 단어의 뜻을 업데이트 하여 바뀐 결과에 대한 정보를 출력한다. 만약, 단어가 존재하지 않거나 단어장의 정보가 존재하지 않을 경우 에러 코드를 출력한다. ex) UPDATE data 자료	800
EXIT	프로그램에 할당된 메모리를 해제하며, 프로그램을 종료한다. ex) EXIT	-

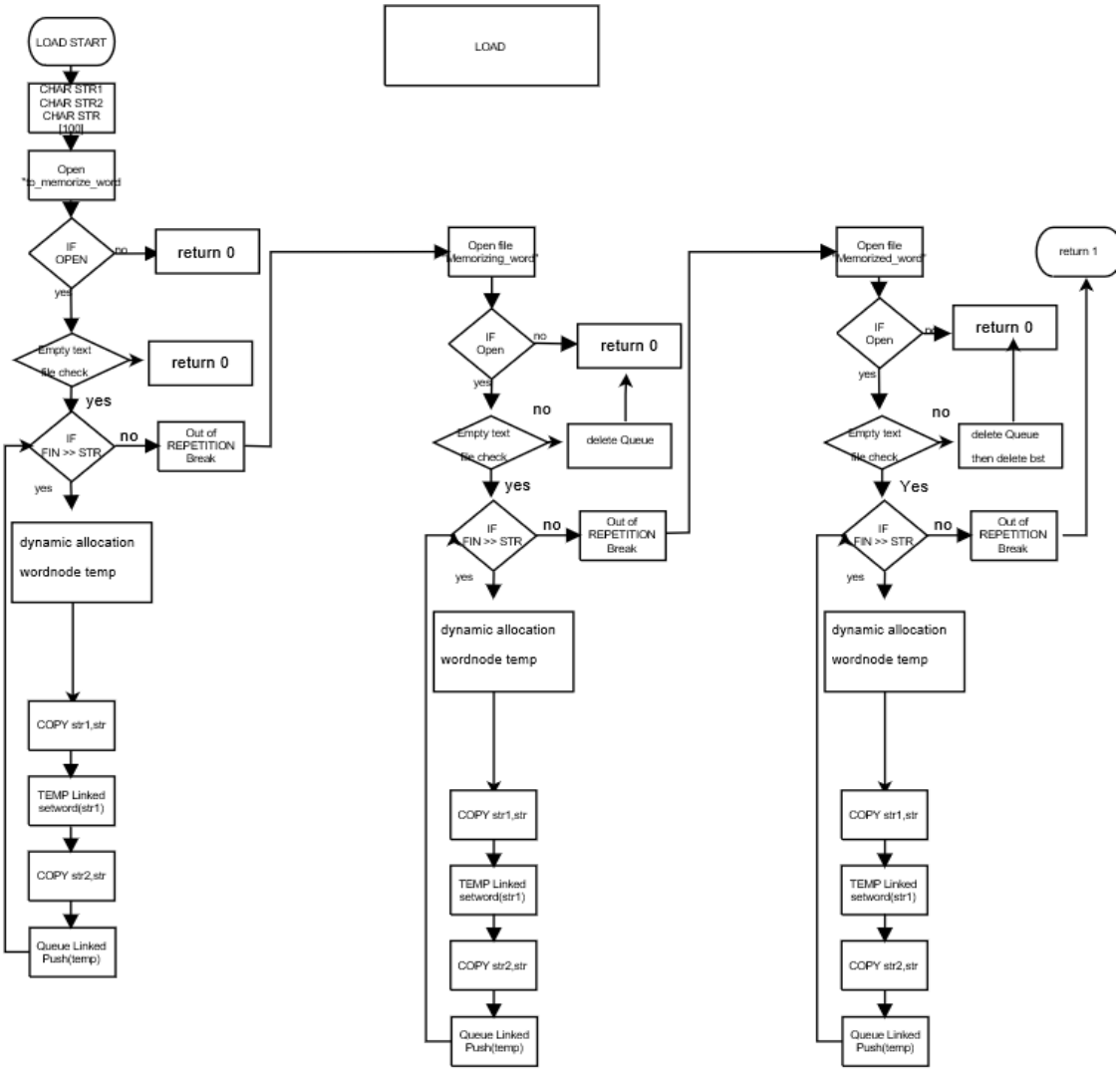
- Instruction Output Format & description

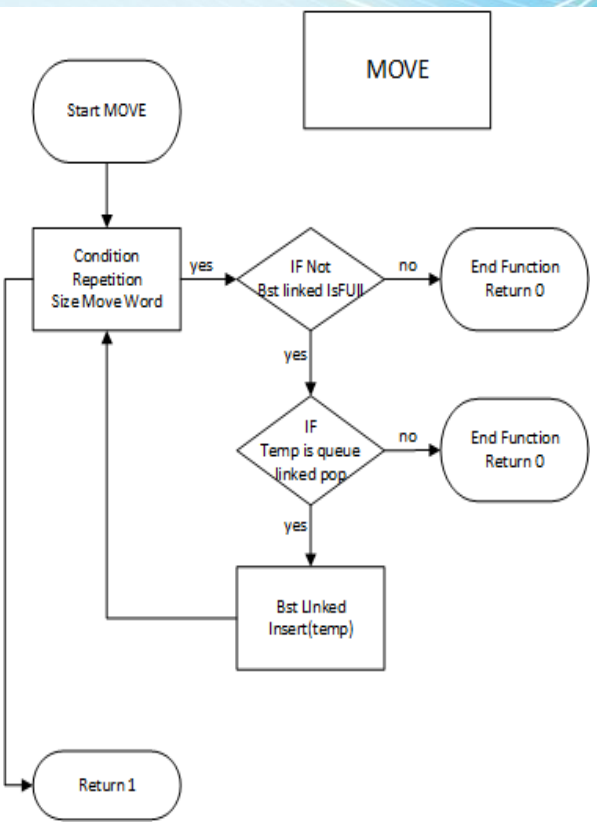
Instruction	Output format	description
에러(ERROR)	========ERROR======== 명령어에 해당하는 에러코드 ====================	에러가 발생하면 에러 출력 포맷에 맞는 에러 코드를 출력한다
LOAD	========LOAD======== Success ================== =======ERROR======= 100 ====================	단어장의 정보를 읽어오는 데 성공하면 'Success'를 출력하고, 실패 하면 에러 코드를 출력한다
ADD	========ADD======== Success ================== ========ERROR======== 200 ====================	단어장에 단어를 추가하는 것을 성공하면 'Success'를 출력하고, 실패하면 에러 코드를 출력한다
MOVE	========MOVE======== Success ================== ======== ERROR======== 300 ====================	단어장으로 단어를 이동시키면 'Success'를 출력하고, 실패하면 에러 코드를 출력한다
SAVE	========SAVE======== Success ================== ======== ERROR======== 400 ====================	단어장에 단어에 대한 정보를 저장하면 'Success'를 출력하고, 실패하면 에러 코드를 출력한다
TEST	========TEST======== PASS ================== ======== ERROR======== 500 ====================	TEST를 통과하면 'PASS'를 출력하고, 실패하면 에러 코드를 출력한다
SEARCH	========SEARCH======== art 미술 ====================	단어를 탐색하여 찾았으면 "영어단어 한글 뜻"순으로

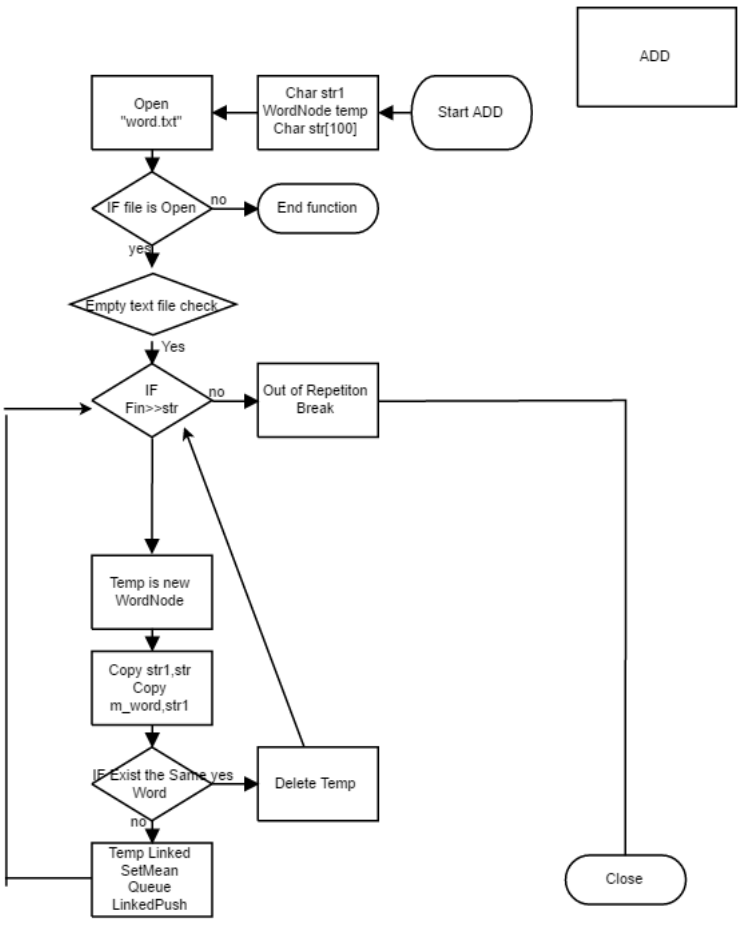
	======== ERROR======== 600 ========== 출력하고, 찾지 못하면 에러 코드를 출력한다	
PRINT	========PRINT======== kid 아이 Kite 연 ========== ======== ERROR======== 700 ========== 해당 단어장의 정보가 존재한다면 "영어단어 한글뜻"순으로 출력하고, 정보가 없으면 에러 코드를 출력한다	
UPDATE	========UPDATE======== art 미술 -> 예술 ========== ======== ERROR======== 800 ========== 변경할 단어가 존재 할 경우 출력 형식에 맞춰 결과를 출력하고, 존재하지 않다면 에러 코드를 출력한다	

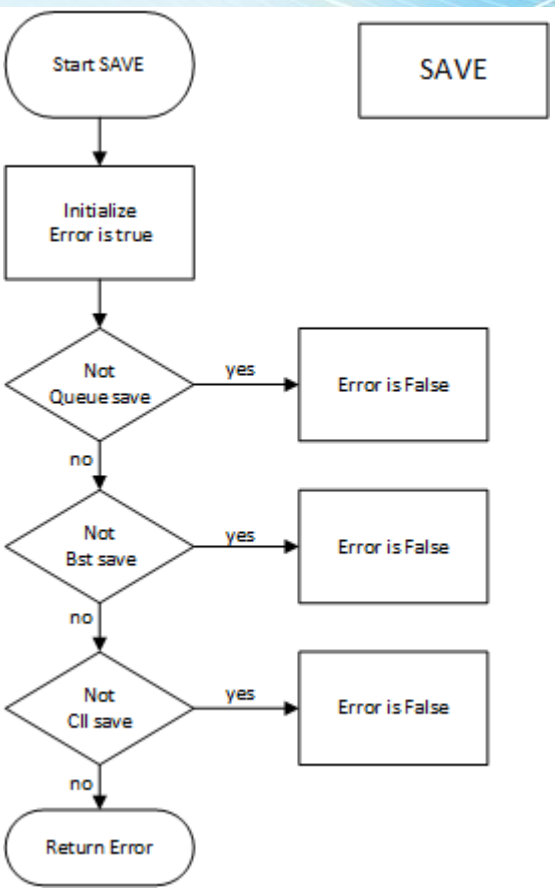
2. Flowchart

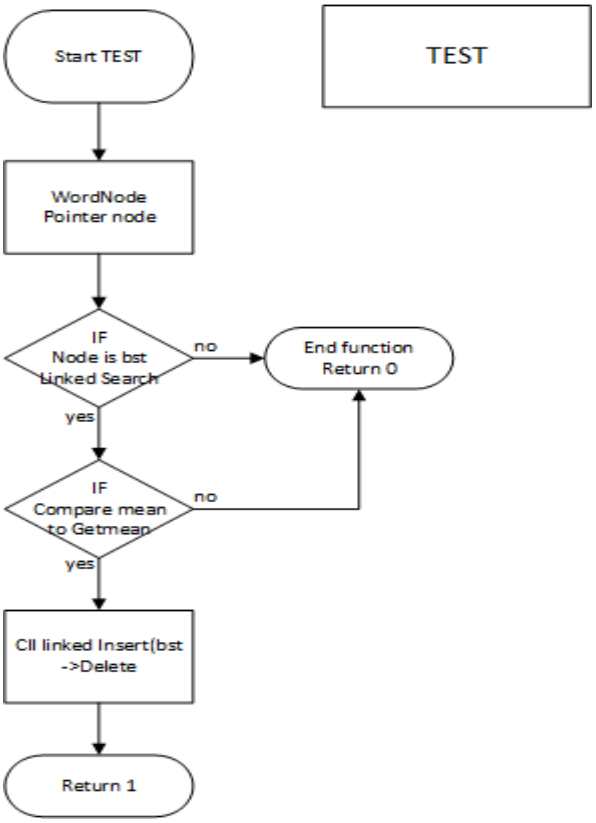

Main
Start
LOAD
MOVE
TEST
PRINT
EXIT
ADD
SAVE
SEARCH
UPDATE
End function
Return 0

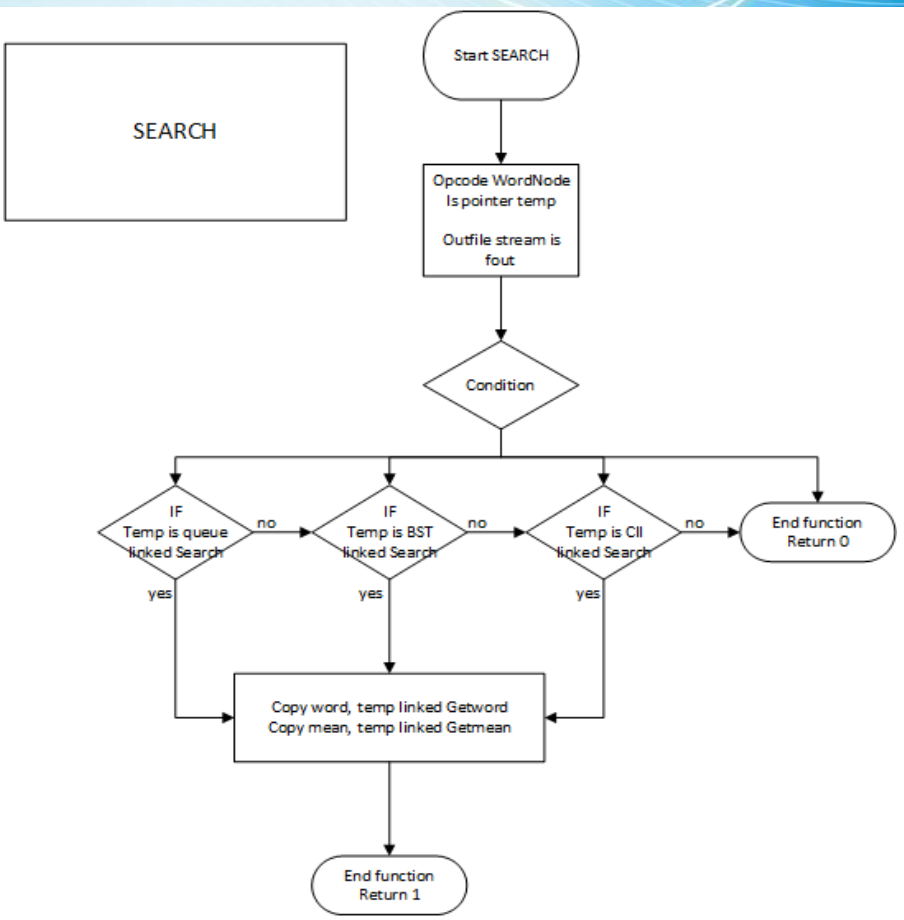

LOAD
LOAD START
CHAR STR1
CHAR STR2
CHAR STR
[100]
Open
"o_memorize_word
IF
OPEN
no
return 0
yes
Empty text
file check
return 0
yes
IF
FIN >> STR
no
Out of
REPETITION
Break
yes
dynamic allocation
wordnode temp
COPY str1,str
TEMP Linked
setword(str1)
COPY str2,str
Queue Linked
Push(temp)
Open file
Memorizing_word"
IF
Open
no
return 0
yes
Empty text
file check
no
delete Queue
yes
IF
FIN >> STR
no
Out of
REPETITION
Break
yes
dynamic allocation
wordnode temp
COPY str1,str
TEMP Linked
setword(str1)
COPY str2,str
Queue Linked
Push(temp)
Open file
Memorized_word"
return 1
IF
Open
no
return 0
yes
Empty text
file check
no
delete Queue
then delete bst
Yes
IF
FIN >> STR
no
Out of
REPETITION
Break
yes
dynamic allocation
wordnode temp
COPY str1,str
TEMP Linked
setword(str1)
COPY str2,str
Queue Linked
Push(temp)

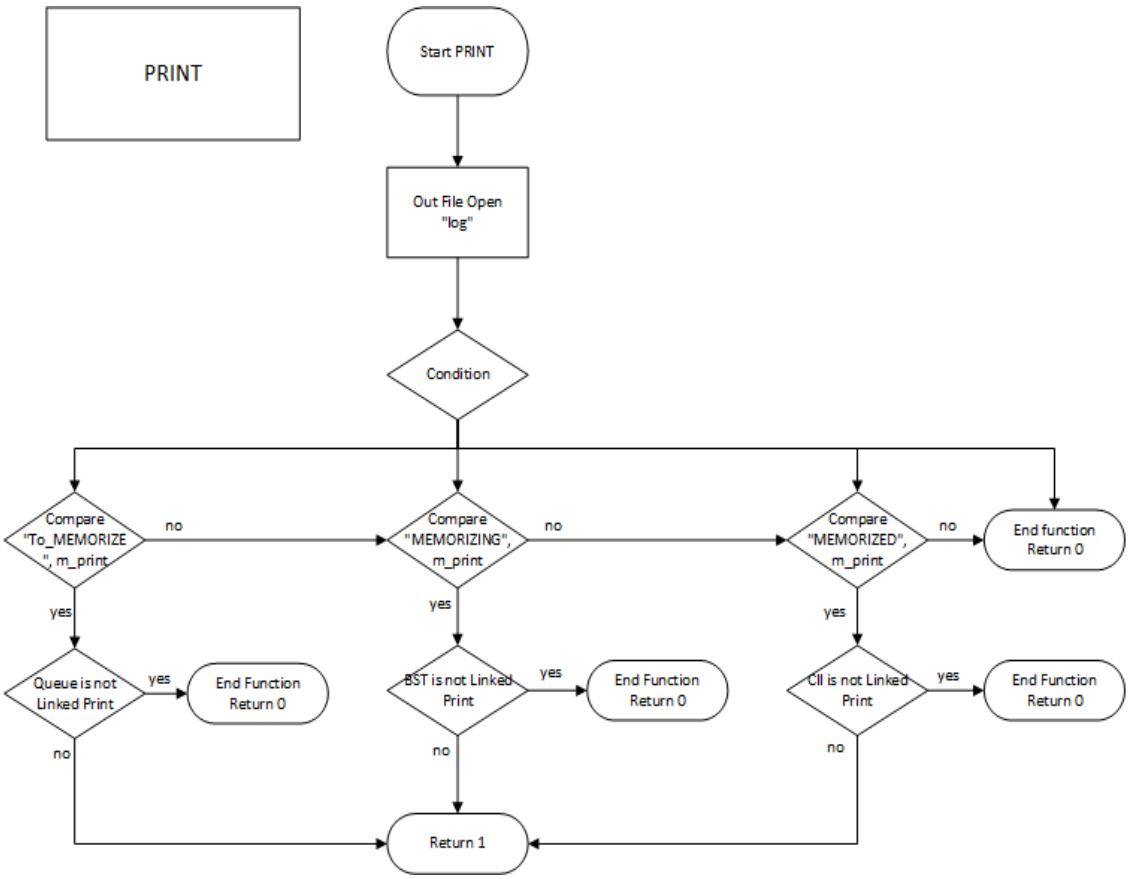

MOVE
Start MOVE
Condition
Repetition
Size Move Word
yes
IF Not
Bst linked IsFUll
no
End Function
Return 0
yes
IF
Temp is queue
linked pop
no
End Function
Return 0
yes
Bst LInked
Insert(temp)
Return 1

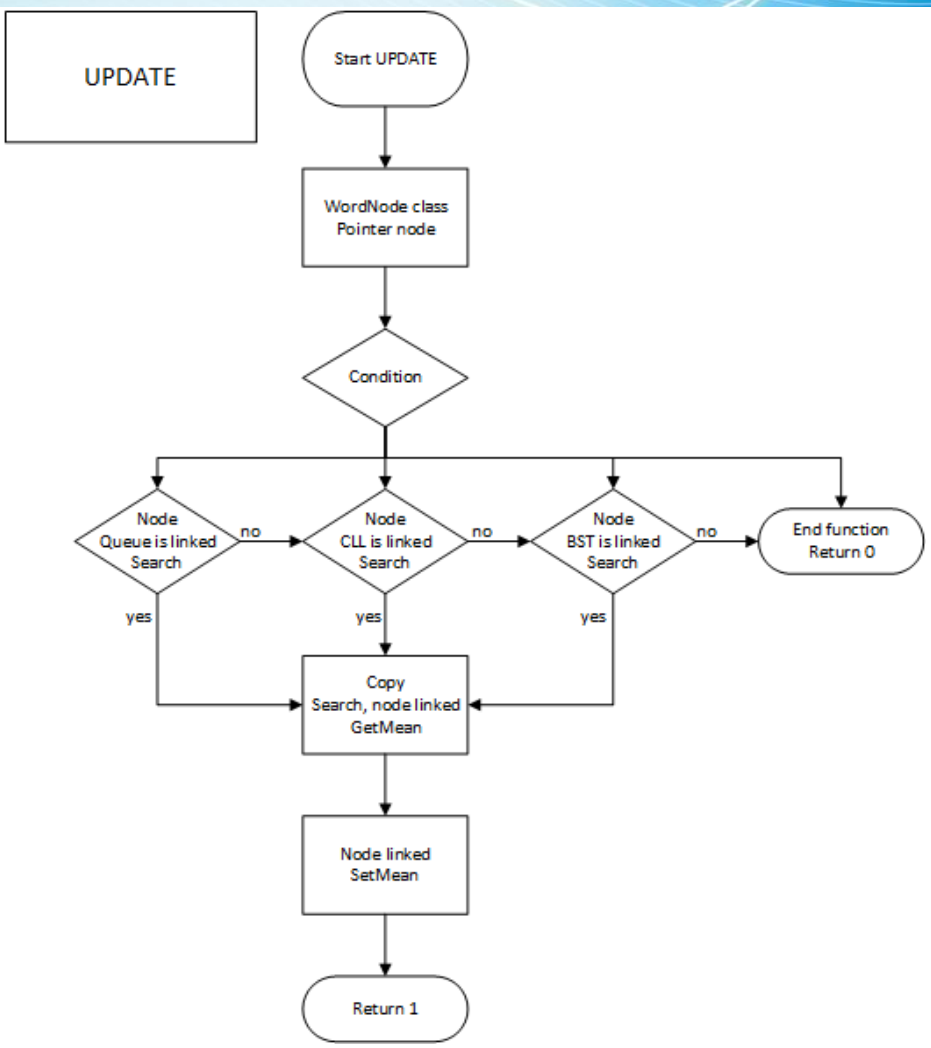

ADD
Open
"word.txt"
Char str1
WordNode temp
Char str[100]
Start ADD
IF file is Open
no
End function
yes
Empty text file check
Yes
IF
Fin>>str
no
Out of Repetiton
Break
Temp is new
WordNode
Copy str1,str
Copy
m_word,str1
IF Exist the Same
Word
yes
Delete Temp
no
Temp Linked
SetMean
Queue
LinkedPush
Close


Start SAVE
SAVE
Initialize
Error is true
Not
Queue save
yes
Error is False
no
Not
Bst save
yes
Error is False
no
Not
Cll save
yes
Error is False
no
Return Error


Start TEST
TEST
WordNode
Pointer node
IF
Node is bst
Linked Search
no
End function
Return 0
yes
IF
Compare mean
to Getmean
no
yes
Cll linked Insert(bst
->Delete
Return 1


SEARCH
Start SEARCH
Opcode WordNode
Is pointer temp
Outfile stream is
fout
Condition
IF
Temp is queue
linked Search
no
IF
Temp is BST
linked Search
no
IF
Temp is CII
linked Search
no
End function
Return 0
yes
yes
yes
Copy word, temp linked Getword
Copy mean, temp linked Getmean
End function
Return 1


PRINT
Start PRINT
Out File Open
"log"
Condition
Compare
"To_MEMORIZE
", m_print
no
Compare
"MEMORIZING",
m_print
no
Compare
"MEMORIZED",
m_print
no
End function
Return 0
yes
yes
yes
Queue is not
Linked Print
yes
End Function
Return 0
BST is not Linked
Print
yes
End Function
Return 0
CII is not Linked
Print
yes
End Function
Return 0
no
no
no
Return 1


UPDATE
Start UPDATE
WordNode class
Pointer node
Condition
Node
Queue is linked
Search
no
Node
CLL is linked
Search
no
Node
BST is linked
Search
no
End function
Return 0
yes
yes
yes
Copy
Search, node linked
GetMean
Node linked
SetMean
Return 1

3. Algorithm

영어 단어장 프로그램 알고리즘

[Main]

프로그램이 시작되면 Main함수에서 manager 클래스의 객체인 manager를 생성한다. Manager 클래스의 멤버 함수인 run을 실행하여 명령어가 저장되어 있는 command.txt를 읽어온다. 저장 되어 있는 명령어를 순차적으로 실행하며, 명령어가 저장된 텍스트 파일에서 'EXIT'를 읽으면 프로그램을 종료한다.

[LOAD]

단어장의 정보를 불러와 노드를 생성하며 단어 정보를 단어장에 세팅한다. 파일 객체를 생성해서 to_memorize_word.txt가 존재 하지 않으면 에러를 출력하고, 존재하면 파일의 끝까지 반복하여 단어 노드의 객체 포인터를 동적 할당하여 생성해서 문자열을 복사한 후 단어와 뜻을 설정하고 파일을 닫는다. memorized_word.txt가 존재하지 않으면 에러를 출력하고, 존재하면 파일의 끝까지 반복하여 단어 노드의 객체 포인터를 동적 할당하여 생성해서 문자열을 복사한 후 단어와 뜻을 설정하고 파일을 닫는다. memorizing_word.txt가 존재하지 않으면 에러를 출력하고, 존재하면 파일의 끝까지 반복하여 단어 노드의 객체 포인터를 동적 할당하여 생성해서 문자열을 복사한 후 단어와 뜻을 설정하고 파일을 닫는다.

[ADD]

ADD 함수가 실행되면 word.txt파일 연다. txt파일이 존재하지 않거나, txt 파일의 내용이 비어 있으면 에러 코드를 출력한다. word.txt파일에 단어가 존재하면 단어 노드의 객체 포인터를 생성한다. 단어 노드를 동적 할당하여 생성하고 SEARCH함수가 실행 돼서 중복된 단어가 있으면 동적 할당을 해제한다.

[MOVE]

MOVE함수가 실행되면 단어 노드의 객체 포인터를 생성하고, 이동할 단어 수가 Queue에 저장된 단어 수보다 크면 함수를 종료하고, 입력한 단어 수만큼 반복하는데, 만약 Binary Search Treed의 노드가 full Binary Search Tree가 아니고, Queue에 입력한 수만큼의 단어가 존재할 경우 단어 노드가 삽입되어 단어를 저장한다.

[SAVE]

에러를 확인하는 변수를 true로 초기화하고, 각 자료구조에 저장할 단어장 정보가 없을 경우 에러를 false로 바꾼다. 에러를 반환하고 종료 한다.

[TEST]

TEST명령어를 command.txt에서 읽어오면 Manager 클래스에 정의된 TEST 함수를 실행하여 WordNode의 객체 포인터 node가 생성되고, Alphabet BST에서 단어를 찾았을 때 문자열 비교 함수로 단어의 뜻을 비교하여 같으면 Binary Search Tree에서 삭제하고 Circular Linked List에 단어가 삽입된다.

[SEARCH]

단어 노드의 객체 포인터와 파일 객체를 생성하고, 자료구조에서 단어를 검색해서 찾으면 해당 단어를 출력한다.

[PRINT]

log.txt를 열고, 조건문을 사용하여 Recursive와 Iterative 방식 중 하나를 선택하고 Preorder, Inorder, Postorder, Levelorder 중 또 하나를 선택하여 출력 방식을 결정하고 그것을 PRINT 명령어 뒤에 입력하여 해당 단어장에 저장된 단어를 출력한다.

[UPDATE]

UPDATE 명령어를 command.txt에서 읽어오면 Manager 클래스에 정의된 UPDATE 함수를 실행하여 WordNode를 가리키는 객체 포인터 node가 생성되고, if-else 조건문으로 Queue, Circular Liked List, Binary Search Tree에서 단어를 찾고, 뜻을 갱신한다.

4. Result Screen

command.txt

```
LOAD word.txt
ADD
MOVE 50
PRINT MEMORIZED
PRINT MEMORIZING R_IN
UPDATE part 역할
SEARCH part
TEST part 역할
PRINT MEMORIZED
SAVE
EXIT
```

실행결과 콘솔 창

```
======== ERROR ========
100
=======================
======== ADD ========
Success
======================
======== MOVE ========
Success
======================
======== ERROR ========
700
=======================
======== PRINT ========
activity        활동
autumn  가을
bird    새
clean   깨끗한
clothes 옷
course  강좌
different       다른
each    각각
earth   지구
enjoy   즐기다
example 예
vacation        휴가
visit   방문
volunteer       자원봉사자
watch   시계
win     이기다
winter  겨울
work    일
======================
======== UPDATE ========
part 부분 -> 역할
=======================
======== SEARCH ========
part 역할
=======================
======== TEST ========
Pass
======================
======== PRINT ========
part    역할
======================
======== SAVE ========
Success
======================
```

실행결과 txt화면

```
======== ERROR ========
100
====================

======== ADD ========
Success
====================

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

======== ERROR ========
700
====================

======== PRINT ========
activity          활동
autumn    가을
bird      새
clean     깨끗한
clothes   옷
course    강좌
different         다른
each      각각
earth     지구
enjoy     즐기다
example   예
vacation          휴가
visit     방문
volunteer         자원봉사자
watch     시계
win       이기다
winter    겨울
work      일
====================

======== UPDATE ========
part 부분 -> 역할
====================

======== SEARCH ========
part 역할
====================

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

======== PRINT ========
part      역할
====================

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

TO_MEMORIZE

crowd	군중
dig	파다
wood	나무
cave	동굴
tomb	무덤
tool	도구
satelite	위성
wealth	재산
attention	주의
fold	접다
lawyer	변호사
blood	피곤한
gun	총
spirit	정신
route	길
pole	막대기
rubber	고무
root	뿌리
structure	구조
vote	투표
patient	환자
quick	빠른
captain	선장
bucket	양동이
cage	새장
kite	연
miracle	기적

248행, 11열

MEMORIZING

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

49행, 12열

MEMORIZED

part 역할

1행, 1열

5. Consideration

나승학(2012722020)

이름	프로젝트에서 맡은 역할	본인 스스로 생각하는 자신의 점수(10)
나승학	**보고서 작성 및 수정(Introduction** - Program Explanation, Instruction Output Format & description, Instruction Function & Example, **Algorithm), header파일 주석 일부**	**10**
고찰	**이번 1차 프로젝트를 진행하면서, 지난 학기에 '고급프로그래밍설계 및 실습'을 수강하지 않았고 코딩 실력이 좋지 않아서 프로그램 코드 작성보다는 보고서를 담당하여 작성 및 통합하였고 주석 일부를 작성하였다. 코드를 작성하지 않았지만 프로젝트를 진행하면서 우분투 상에서 동작을 팀원과 같이 확인하고 예외처리 동작에 대해 고민하고 코드에 주석을 달아보면서 리눅스 명령어의 사용법, 우분투 상에서 디버깅 및 컴파일 방법, Queue, Binary Search Tree, Circular Linked List 등의 자료 구조를 코딩하는 방법 등을 배우는 과정을 통해 리눅스 우분투의 명령어 사용법과 컴파일 및 디버깅 방법에 대해 알게 되었다. 또한, 팀 프로젝트를 통하여 협업의 중요성과 역할 분담의 중요성에 대해 느낄 수 있었다. 다음 프로젝트부터는 소스 코드 작성은 개별로 작성해야 하므로 1차 프로젝트의 경험을 바탕으로 2차 프로젝트를 준비 할 것이다.**	

송민규(2013722042)

이름	프로젝트에서 맡은 역할	본인 스스로 생각하는 자신의 점수(10)
송민규	**코드 전체 작성, header파일 주석 일부**	**10**
고찰	**이번 과제를 하면서 리눅스를 처음 사용하게 되었는데, CUI에 익숙하지 않다 보니 아무래도 처음에는 불편함을 느꼈다. 그런데 운영체제의 특성상 코딩을 하는 데 있어서 컴파일이 굉장히 간편했다. 다만 단순히 VIM으로 디버깅을 하려니, 중간 결과값을 확인할 수 없어서 gdb 등의 툴을 이용해야 했는데, 그보다 좀더 익숙한 방식을 사용하는 이클립스를 이용해서 디버깅을 했다. 처음 과제를 받았을 때는 visual studio를 이용해서 코드를**	

	짰는데, 이걸 리눅스로 가져와서 실행해보니 컴파일이 되지 않았다. 알고보니 운영체제마다 문자열을 처리하는 방식이 달랐기 때문이었다. 윈도우에서는 문자열 끝이 ₩n으로 되어있는데, 리눅스에서는 문자열 끝이 ₩r₩n으로 이루어져 있어서 텍스트파일에서 파일을 읽을 때 제대로 동작하지 않았다. 그리고 한글 문자열을 처리할 때, 윈도우에서는 텍스트파일이 ANSI 방식으로 인코딩 되어있었는데 리눅스에서는 UTF-8 방식의 인코딩이 기본 설정으로 되어 있어서 UNICODE와 UTF-8, UTF-16등의 한글 인코딩 방식에 대해서 깊게 배우는 계기가 되었다.

주정인(2013722006)

이름	프로젝트에서 맡은 역할	본인 스스로 생각하는 자신의 점수(10)
주정인	보고서(Introduction – ADTs, Flow Chart), cc파일 주석작성, 프로젝트에 전반적인 Skelton Frame과 Load, Search, Add, Delete, Test, Move의 Direction code 제시	10
고찰	Binary Search Tree, Circular Linked List, Tree travel등 프로젝트 수행 중에 좀더 보완할 수 있었다. Circular Linked List를 학습하면서 Singly Linked List와의 차이점을 알 수 있었고 왜 Circular Linked List의 node 삭제, 삽입에 해당 Node 가 어떻게 연결 되는지, 왜 NULL 값으로 연결이 안되는지에 대해 알 수 있었다. 프로젝트를 수행하면서 코딩을 하는 알고리즘을 먼저 학습하고 토의를 통해 어떻게 방향을 잡아야 하고 코딩을 해야하는지, 또는 어떤 역할을 해야하는지에 대해 팀 프로젝트의 장점을 느꼈지만 팀의 구성원의 사는 곳이 다르고 수업 시간이 다르다 보니 시간과 장소를 잡아 토의하기에 다소 불편함이 있었다.	