Data Structure

Project 1

Department		데이터 구조	
Student ID	2010720185	윤희제	실습 (A)
	2013722046	고영운	실습 (C)
	2014722047	이민영	실습 (C)

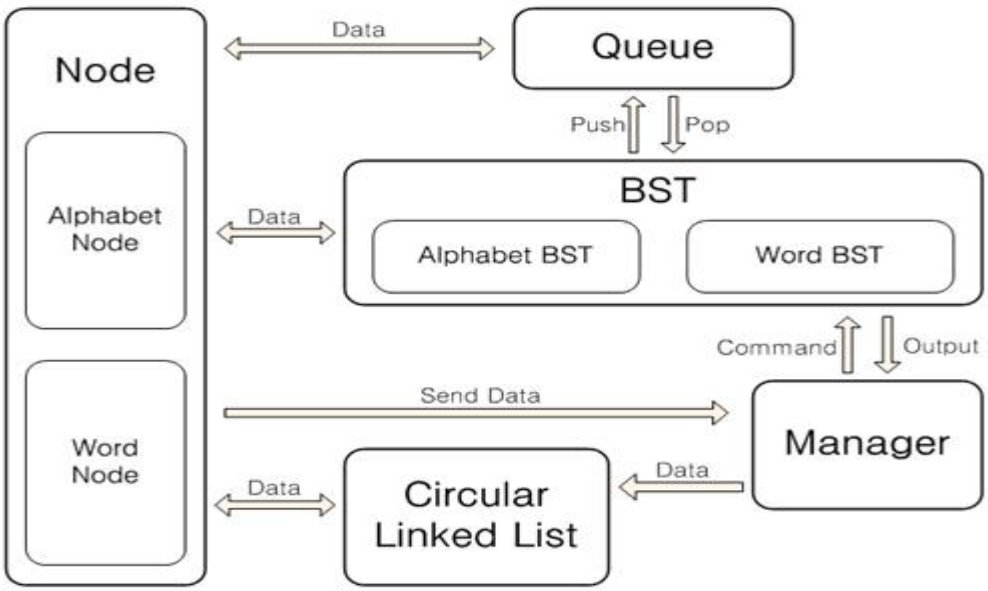
1. Introduction

이번 프로젝트는 영어 단어장 프로그램을 구현하는 것이다. 이진 탐색 트리(Binary Search Tree, BST)와 환형 연결 리스트(Circular Linked List), Queue를 이용한다. 프로그램으로 영어 단어 여러 개와 그 단어의 한글 뜻을 입력 받고, TO_MEMORIZE, MEMORIZING, MEMORIZED로 구분하여 구축하도록 한다. TO_MEMORIZE는 Queue로 구축되며 앞으로 외워야 할 단어가 들어가고, MEMORIZING는 Binary Search Tree로 구축되며 현재 외우고 있는 단어가 들어간다. MEMORIZED는 Circular Linked List로 구축되며 이전에 외운 단어가 들어가게 된다. 그리고 각 자료구조에는 영어단어와 한글 뜻을 데이터로 가지는 단어 노드가 존재한다.

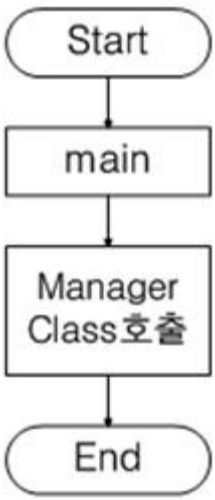
단어가 프로그램에 추가되면 가장 먼저 TO_MEMORIZE로 들어간다. TO_MEMORIZE의 Queue에 새로운 단어가 들어오면, 기본적으로 Push 동작을 하고, MOVE 명령어를 통해 MEMORIZING으로 단어가 옮겨지며 Pop 동작을 한다. 이 때 MEMORIZING 영역에는 단어가 최대 100개까지만 존재할 수 있다. MEMORIZING에는 Alphabet BST와 Word BST가 존재한다. Alphabet node로 이루어져 있는 Alphabet BST는 P, H, X, D, L, T, Z, B, F, J, N, R, V, Y, A, C, E, G, I, K, M, O, Q, S, U, W 순으로 알파벳이 추가되어 프로그램이 시작되면 바로 BST를 구축하게 된다. 각 알파벳 노드가 Word BST를 가지며, MOVE 명령어를 통해 TO_MEMORIZE에서 n개의 단어가 입력되면 Word BST가 구축된다. 여기서 MEMORIZING의 BST를 연결할 때, 부모 노드보다 사전적 순서가 앞쪽인 노드는 왼쪽, 뒤쪽인 노드는 오른쪽 sub tree에 위치해야 한다. 그리고 MEMORIZING에서 TEST 명령어를 통해 단어를 외웠는지 테스트하여 확인되면 MEMORIZED로 옮길 수 있다. MEMORIZED는 Circular Linked List(CLL)로 이루어진다. CLL에는 Head pointer가 존재하는데, 단어가 처음 들어오면 Head Pointer와 단어가 서로를 가리킨다. 그 다음부터 단어가 들어오면 List에 이미 단어가 존재하기 때문에, 새로 들어온 단어는 Head pointer가 가리키는 노드와 바로 뒤의 노드 사이에 연결하게 된다.

2. Flowchart

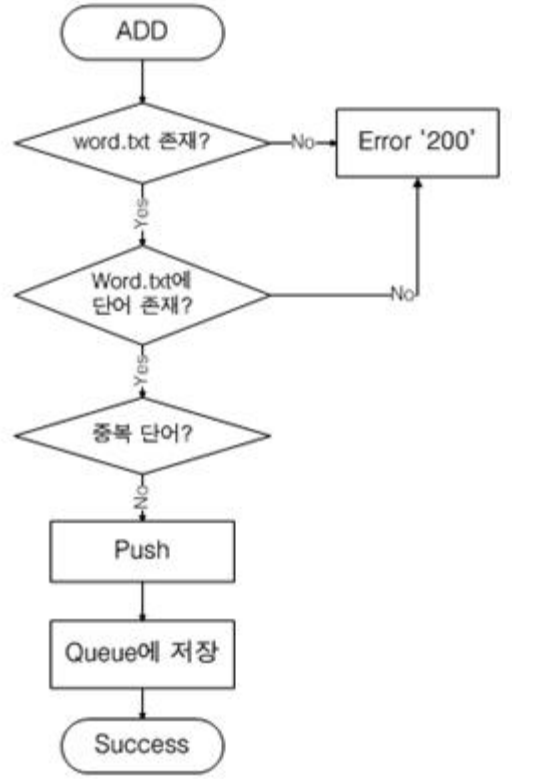
<Overall system class design>

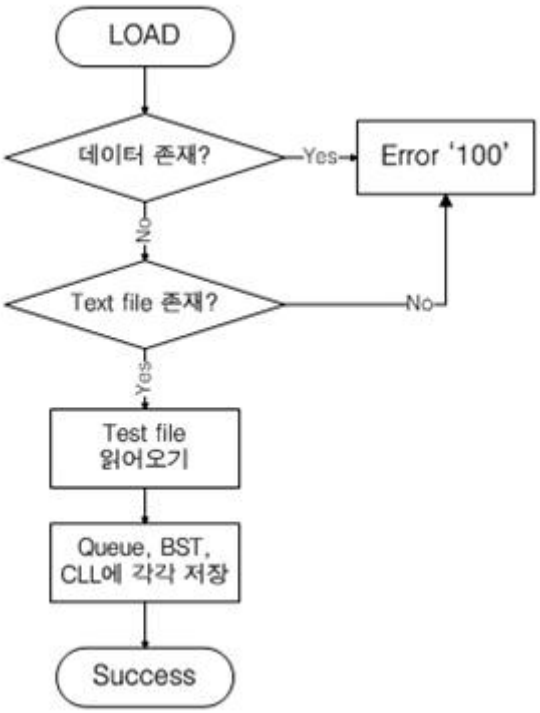


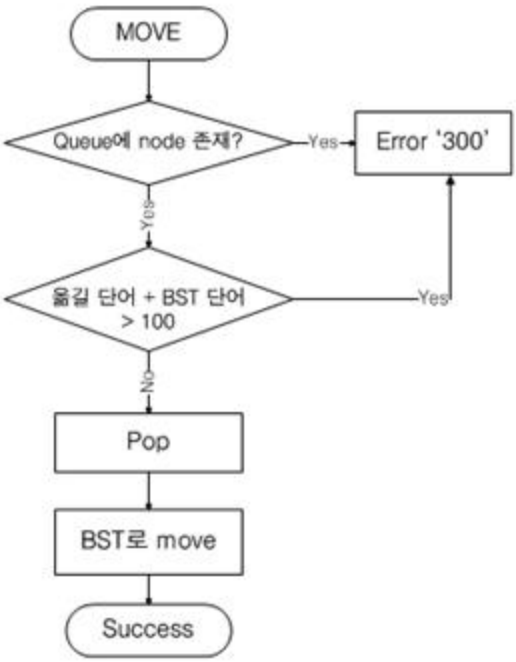
<Main 함수>

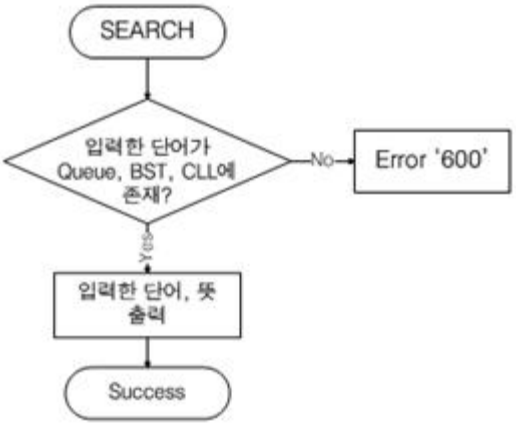


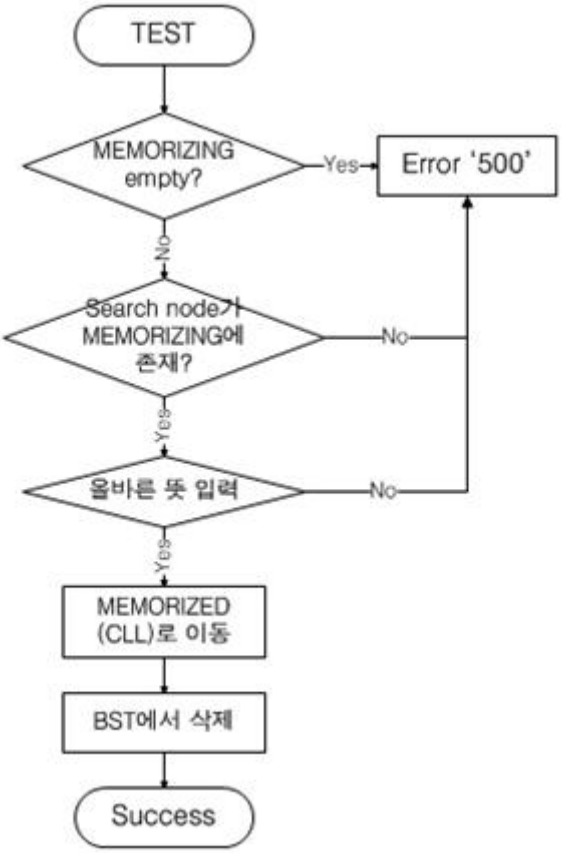
<Manager 함수>


ADD
word.txt 존재?
No
Error '200'
Yes
Word.txt에
단어 존재?
No
Yes
중복 단어?
No
Push
Queue에 저장
Success


LOAD
데이터 존재?
Yes
Error '100'
No
Text file 존재?
No
Yes
Test file
읽어오기
Queue, BST,
CLL에 각각 저장
Success


MOVE
Queue에 node 존재?
Yes
Error '300'
Yes
옮길 단어 + BST 단어
> 100
Yes
No
Pop
BST로 move
Success


SEARCH
입력한 단어가
Queue, BST, CLL에
존재?
No
Error '600'
Yes
입력한 단어, 뜻
출력
Success


TEST
MEMORIZING
empty?
Yes
Error '500'
No
Search node가
MEMORIZING에
존재?
No
Yes
올바른 뜻 입력
No
Yes
MEMORIZED
(CLL)로 이동
BST에서 삭제
Success

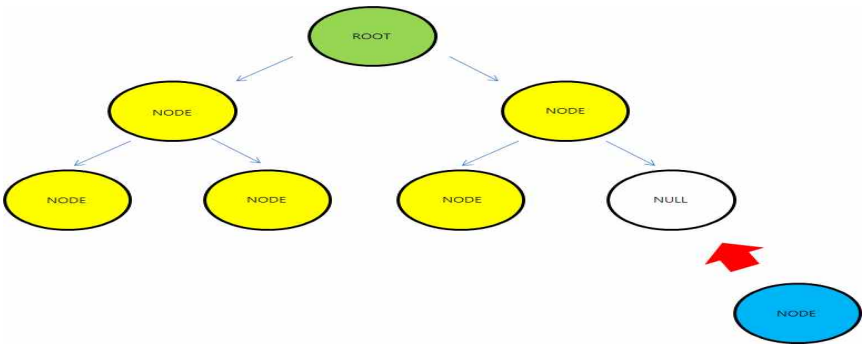
3. Algorithm

BST - Binary Search Tree (AlphabetBST & WordBST)

BST 즉, 이진탐색트리는 특정 Data를 지닌 Node가 입력 되었을 때 최초의 Root node를 기준으로 Data 값의 비교를 통하여 Data가 작으면 왼쪽, Data가 크면 오른쪽에 자식 Node를 정렬하여 준다.

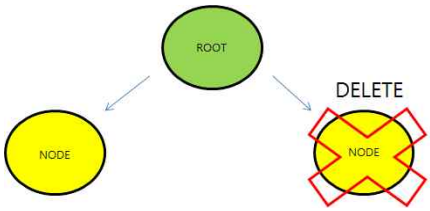
-INSERT & SEARCH

SEARCH 와 INSERT는 동일한 과정을 거친다. NODE가 없을 시에는 ROOT로 바로 삽입이 되며,
기존에 노드가 삽입 되어 있을 경우는 왼쪽 자식 노드 부터 검색을 시작하여 오른쪽 자식 노드 까지
검색을 시작한다. 이때 NULL 값이 나오면 검색을 종료하게 된다. INSERT와 SEARCH는 다음의
동일한 과정을 거친 후 NULL 값 발견 시 NODE INSERT를 진행한다.

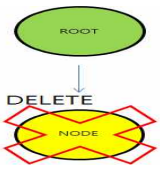


-DELETE

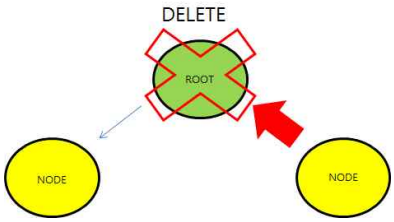
CASE1 : 2개의 자식 노드 중 1개 삭제



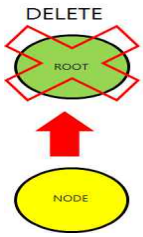
CASE2 : 1개의 자식 노드 삭제



CASE3 : 2개의 자식 노드를 가진 부모 노드 삭제
- DATA가 큰 자식 NODE[우측] 로 대체

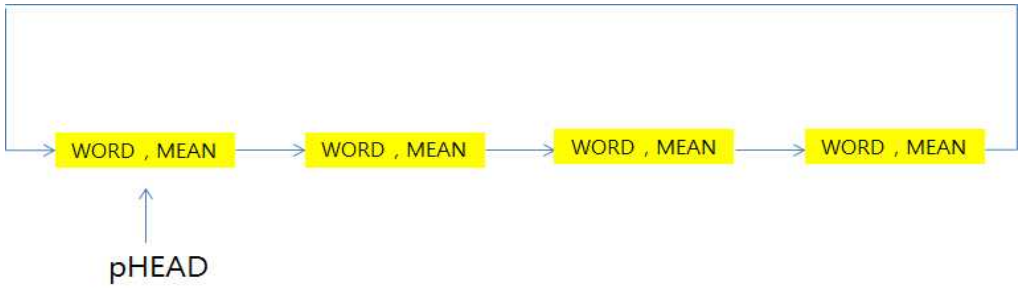


CASE4 : 1개의 자식 노드를 가진 부모 노드 삭제
- 자식 NODE로 대체



NODE (AlphabetNode & WordNode)			
GetAlphabet()	알파벳 노드 DATA 반환 함수	SetAlphabet()	알파벳 노드 DATA SET 함수
GetBST()	BST 노드 DATA를 반환 함수	SetLeft()	좌측 노드 SET 함수
GetLeft()	좌측 노드를 반환 함수	SetRight()	우측 노드 SET 함수
GetRight()	우측 노드를 반환 함수	SetNext()	다음 노드 SET 함수
GetNext()	다음 노드를 반환 함수		

CLL - Circular linked list
[MEMORIZED를 구성하는데 사용]

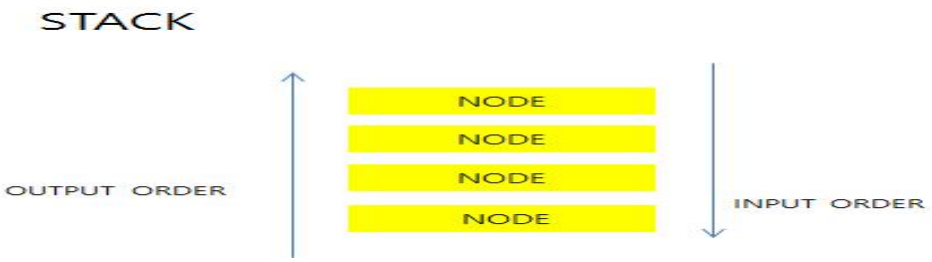


CASE1 : 단어 처음 삽입 시
처음 단어 삽입 시 HEAD POINTER는 삽입된 단어를 가리키며, 삽입이 완료된 단어(TEMP)는 HEAD POINTER를 가리킨다.
CASE2 : 삽입된 단어 존재 시
이미 삽입된 단어가 존재할 경우 새로 삽입된 단어는 HEAD POINTER가 가리키는 단어와 바로 다음 노드 사이를 연결해 준다.[GetNext, SetNext]
CASE3 : SEARCH
단어를 찾기 위해 SEARCH(char* word)에 단어가 입력되어진다. 이는 WORD 부분에 해당하는 부분을 검색 하는 방식인데 삽입과 마찬가지로 기존 cll에 삽입된 단어 존재 여부에 따라 두 가지 경우가 되며 삽입되는 방식처럼 HEAD POINER를 가리키는 노드들을 검색하여 준다.

QUEUE
[TO_MEMORIZE를 구성하는데 사용]

CASE1 : 단어 처음 삽입시[PUSH]
Queue에 NODE가 존재 하지 않을 경우 노드 Queue에 그대로 삽입
CASE2 : 삽입된 단어 존재시[PUSH]
이미 삽입된 단어가 존재할 경우 새로 삽입된 단어는 HEAD POINTER가 가리키는 단어 바로 다음 노드에 삽입된다.[GetNext, SetNext]
CASE3 : POP
FIFO 즉, 선입 선출로 삽입이 되면 먼저 삽입된 노드는 앞으로 밀리게 되며 이를 마지막 삽입된 노드 역순으로 추적을 하여 처음 삽입된 노드를 제거하여 준다.

STACK



CASE1 : 단어 처음 삽입시[PUSH]
Stack에 NODE가 존재 하지 않을 경우 노드 Stack에 그대로 삽입
CASE2 : 삽입된 단어 존재시[PUSH]
이미 삽입된 단어가 존재할 경우 새로 삽입된 단어는 기존에 삽입된 노드 위에 위치하게 된다.
CASE3 : POP
FILO 즉, 선입 후출로 삽입이 되면 먼저 삽입된 노드는 밑으로 가라앉게 되며 이를 마지막 삽입된 노드를 제거하여 준다. 즉, 스텍에 저장된 반대순서 최근 NODE부터 제거하여 준다.

4. Result Screen

5. Consideration

이름	프로젝트에서 맡은 역할	스스로 생각하는 자신의 점수 (10)
고찰		
고찰		
이민영	보고서-알고리즘작성 소스코드- SEARCH 및 알고리즘	5
고찰	코딩은 간만에 진행하였다. 조장임에도 불구하고 프로그래밍 실력이 부족하여 조원들에게 짐을 떠맡긴거 같아 미안하고 다음과제 전까지 꾸준히 프로그래밍 연습을 통하여 보다 구조적 이해를 높이고 프로그래밍 실력을 올리도록 하겠다.	

이름	프로젝트에서 맡은 역할	스스로 생각하는 자신의 점수 (10)
고찰		
고찰		
고영운	전체적인 프로그램 코딩 및 디버깅	6
고찰	프로그래밍 프로젝트를 팀을 구성해서 진행한 건 처음 인데 처음에 우려했던 것과 달리 서로 협업하여 프로젝트를 진행할 수 있어서 많은 도움이 되었다. 내가 프로그래밍 실력이 부족하여 비록 프로젝트를 완성하는데는 실패했지만 , 다음 프로젝트를 진행하는데 있어 큰 도움이 될 거 같다. 1학기 고급프로그래밍설계에서 다뤘던 자료구조를 좀 더 세세하게 구현해볼 기회를 가질 수 있게 되어 내 프로그래밍 실력에 도움이 되었다.	

이름	프로젝트에서 맡은 역할	스스로 생각하는 자신의 점수 (10)
고찰		
고찰		
이민영	보고서-Introduction, Flowchart작성 소스코드 전체적인 오류 수정	6
고찰	코딩을 팀플로 진행한 건 처음이었는데 내가 부족한 부분을 많이 얻고 역할을 나눠서 맡아서 효율적으로 하니까 혼자 하는 것보다 훨씬 도움 됐다. 다만 과제를 좀 늦게 시작해서 시간이 부족했던 점은 있었다. BST 코드도 처음 구현하고 리눅스 시스템도 처음 사용 해본 거라서 Visual에서 돌릴 때는 잘 돌아가는데 리눅스에서는 오류가 나는 부분도 있어서 수정이 많이 필요했다. 예를 들어 visual에서 평범하게 사용하던 _s가 붙은 함수들이라던지, NULL 문자를 꼭 ₩0으로 표기해줘야 하는 것 등등이 있었다. 아직 자료구조 자체도 익숙하지 않을 뿐더러, 링크드리스트나 BST, 리눅스에 관해서는 공부를 더 많이 하고 이번 시행착오를 통해 다음 과제를 더 효율적으로 수행해야 할 것 같다.	