Software Project #1

2018년 9월 5일

Due date: 2018년 10월 3일 수요일 23:59:59 까지

본 프로젝트에서는 팀(2~3인) 프로젝트로, 이진 탐색 트리(Binary Search Tree, BST)와 환형 연결 리스트(Circular Linked List), Queue를 이용하여 영어 단어장 프로그램을 구현한다. 이 프로그램은 여러 영어 단어와 한글 뜻을 입력받아 앞으로 외워야할 단어장(TO_MEMORIZE), 현재 외우고 있는 단어장(MEMORIZING), 이전에 외운 단어장(MEMORIZED)으로 구분하여 구축한다. TO_MEMORIZE는 Queue로, MEMORIZING은 BST로, MEMORIZED는 Circular Linked List로 구축한다.

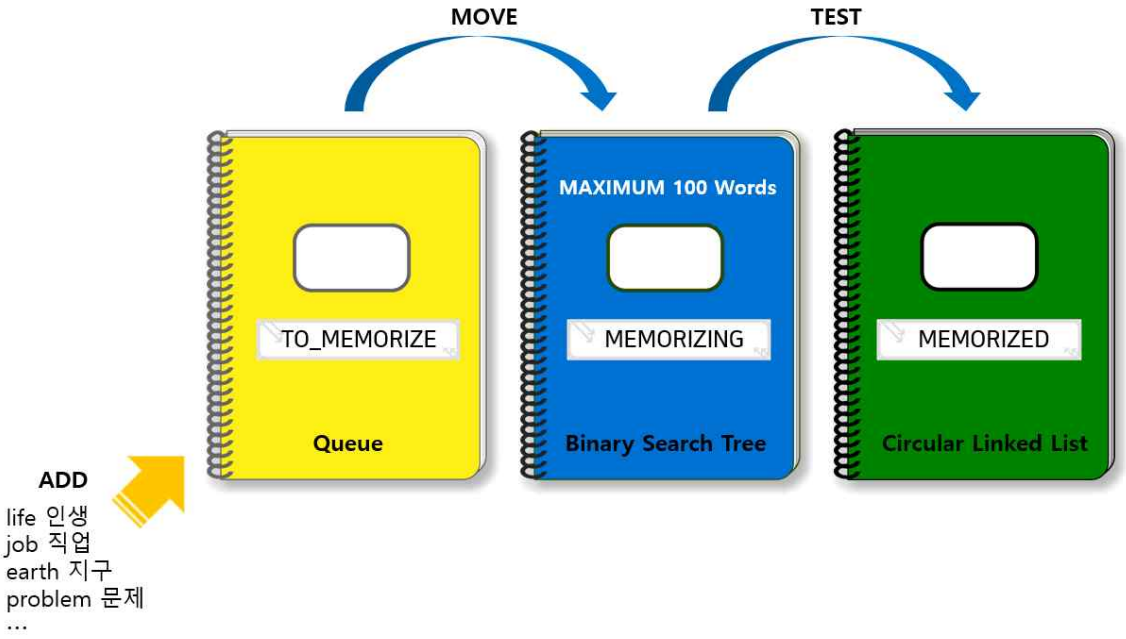

그림 1. 단어장 구분

☐ Program implementation

프로그램에는 TO_MEMORIZE, MEMORIZING, MEMORIZED가 존재하며 각각 Queue, BST, Circular Linked List로 구현되어 있다. 그리고 각 자료구조의 단어 노드에는 영어단어와 한글 뜻이 데이터로 존재한다.

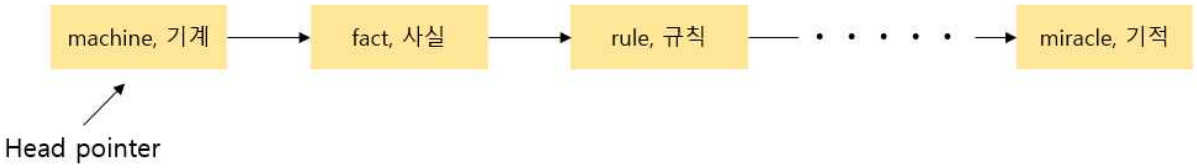

그림 2. TO_MEMORIZE Queue의 예

TO_MEMORIZE의 Queue는 기본적으로 새로운 단어가 들어오면 Push 동작을 하고 MEMORIZING으로 단어를 옮길 때 Pop 동작을 한다. 프로그램에 단어를 추가하게 되면 우선 TO_MEMORIZE에 단어가 모두 들어가게 되며 그 후에 MOVE 명령어를 통해 TO_MEMORIZE의 단어를 MEMORIZING으로 옮길 수 있다. MEMORIZING에는 최대 100개의 단어만 존재해야 한다.

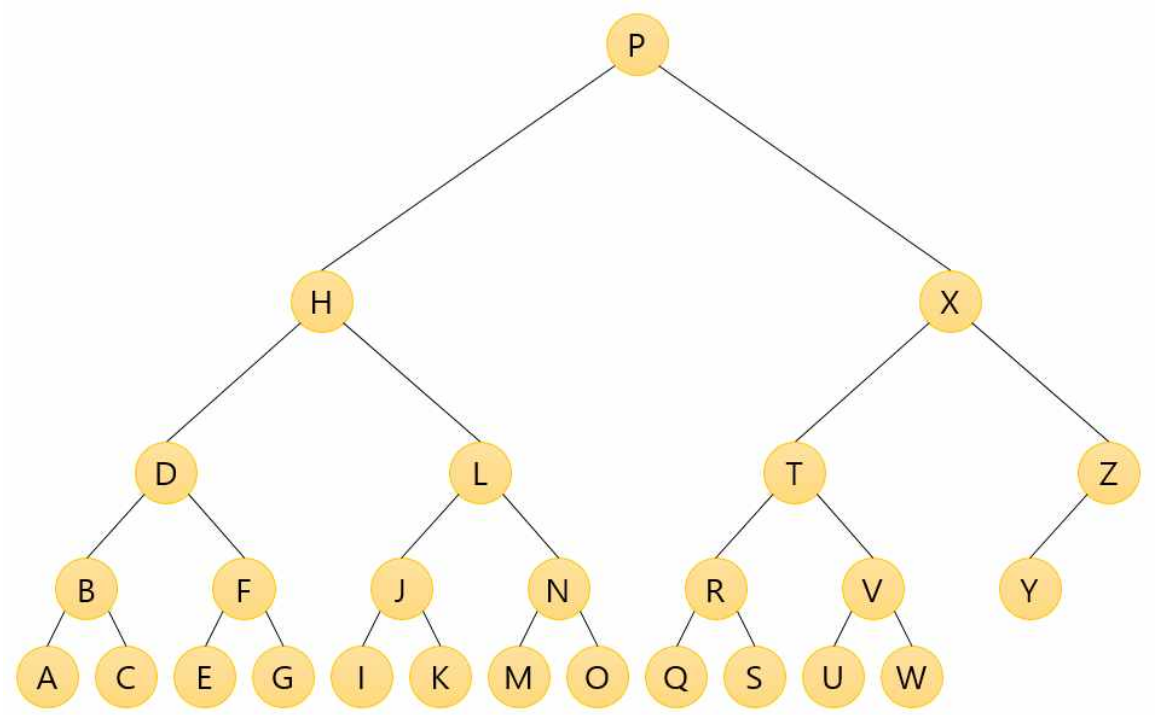

그림 3. MEMORIZING 알파벳 BST의 예

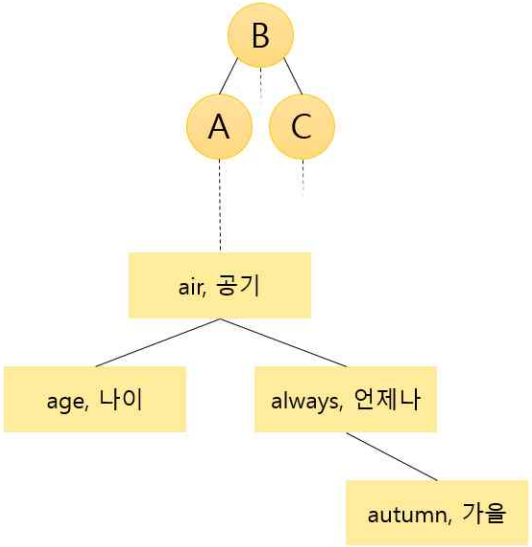

그림 4. MEMORIZING 단어 BST 예

MEMORIZING에는 알파벳 BST와 단어 BST가 존재해야한다. 알파벳 BST는 그림 3과 같이 알파벳 노드로 이루어져있으며 반드시 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 알파벳을 추가하여 위와 같은 알파벳 BST를 구축한다. 알파벳 BST는 프로그램이 시작되면 바로 구축한다. 각각의 알파벳 노드는 그림 4와 같은 단어 BST를 가지고 있으며 MOVE 명령어를 통해 TO_MEMORIZE로부터 n개의 단어를 입력받아 단어 BST를 구축할 수 있다. 여기서 MEMORIZING에 존재하는 단어 노드는 최대 100개까지만 존재할 수 있음을 명심한다. TEST 명령어를 통해 단어를 외웠는지 테스트할 수 있으며 단어를 외운 것이 확인되면 MEMORIZING의 해당 단어를 MEMORIZED로 옮긴다.

MEMORIZING에 사용되는 BST를 연결하는 규칙은 다음과 같다.

※ BST 연결 규칙

① 부모 노드보다 단어의 사전적 순서가 작은 노드는 왼쪽, 큰 노드는 오른쪽 서브 트리에 반드시 위치한다.

② 노드를 제거할 때, **양쪽 자식 노드가 모두 존재할 경우에는 왼쪽 자식 노드 중 가장 큰 노드를 제거되는 노드 위치로 이동**시키며, 왼쪽 자식노드가 존재하지 않을 경우에는 오른쪽 자식 노드 중 가장 작은 노드를 제거되는 노드 위치로 이동시킨다.

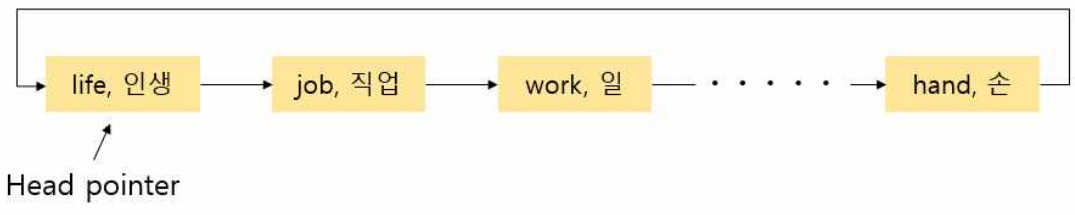

그림 5. MEMORIZED의 Circular Linked List 예

MEMORIZED의 Circular Linked List는 그림5와 같이 구성된다. 처음 단어가 들어올 때는 Head pointer가 들어온 단어를 가리키고 들어온 단어는 Head pointer를 가리키게 된다. List에 단어가 이미 존재할 경우에는 새로 들어온 단어를 Head pointer가 가리키는 노드(그림 5에서 <life, 인생>)와 바로 뒤의 노드(그림 5에서 <hand, 손>) 사이에 연결한다.

표 1. 명령어 사용 예 및 기능 설명

명령어	명령어 사용 예 및 기능
LOAD	사용 예) LOAD 기존의 단어장 정보를 불러오는 명령어로, 텍스트 파일에 단어장 정보가 존재할 경우 텍스트 파일을 읽어 TO_MEMORIZE Queue, MEMORIZING BST, MEMORIZED Circular Linked List 자료구조에 이전과 동일한 연결 순서를 가지도록 저장한다. 만약 3개의 텍스트 파일이 존재하지 않거나 자료구조에 이미 데이터가 들어가 있을 경우에 알맞은 에러 코드를 출력한다. 사용되는 텍스트 파일의 이름은 아래와 같으며 파일 이름을 수정하지 않는다. TO_MEMORIZE 텍스트 파일: to_memorize_word.txt MEMORIZING 텍스트 파일: memorizing_word.txt MEMORIZED 텍스트 파일: memorized_word.txt
ADD	사용 예) ADD 단어 텍스트 파일에 있는 단어 정보를 읽어오는 명령어로, 단어 텍스트 파일에 존재하는 단어들을 TO_MEMORIZE에 모두 저장한다. 단어 텍스트 파일이 존재하지 않거나 파일에 단어가 존재하지 않을 경우 알맞은 에러 코드를 출력한다. 사용되는 단어 텍스트 파일의 이름은 아래와 같으며, 파일의 이름은 절대 수정하지 않는다. 단어 텍스트 파일: word.txt
MOVE	사용 예) MOVE 100 사용자가 입력한 수 만큼 TO_MEMORIZE의 단어들을 MEMORIZING으로 옮기는 명령어로, 1~100 사이의 정수를 입력받고 입력받은 수와 MEMORIZING 단어 수의 합이 100을 넘어가지 않도록 한다. TO_MEMORIZE에 단어가 없는 경우, 입력받은 수와 MEMORIZING의 단어 수의 합이 100을 넘어가게 되는 경우, 사용자가 입력한 수 만큼 단어가 존재하지 않을 경우 등이 발생하면 에러 코드를 출력한다.
SAVE	사용 예) SAVE 현재 단어장 정보를 저장하는 명령어로, 각각 Queue, BST, Circular Linked List에 저장된 단어들을 TO_MEMORIZE, MEMORIZING, MEMORIZED 텍스트

	파일에 저장한다. 사용되는 텍스트 파일의 이름은 LOAD 명령어와 동일하며 단어장 정보가 존재하지 않을 경우 알맞은 에러 코드를 출력한다.
TEST	사용 예) TEST apple 사과 단어를 외웠는지 테스트하는 명령어로, MEMORIZING에 입력한 단어가 있는지 찾아 뜻이 맞는지 확인하고, 맞을 경우 해당 단어를 MEMORIZING에서 MEMORIZED로 이동시킨다. 입력한 단어가 MEMORIZING에 존재하지 않거나 단어의 뜻이 틀릴 경우 알맞은 에러 코드를 출력한다.
SEARCH	사용 예) SEARCH apple 단어의 뜻을 찾아 출력하는 명령어로, TO_MEMORIZE, MEMORIZING, MEMORIZED에 입력한 단어가 존재할 경우, 영어 단어와 한글 뜻을 출력해준다. 만약 입력한 단어가 없을 경우 알맞은 에러 코드를 출력한다.
PRINT	사용 예) PRINT TO_MEMORIZE 사용 예) PRINT MEMORIZING I_PRE 입력한 단어장에 있는 단어들을 출력하는 명령어로, TO_MEMORIZE를 입력할 경우 TO_MEMORIZE Queue에 있는 단어들을 Header부터 순서대로 전부 출력한다. MEMORIZING의 경우 Recursive Preorder 트리순회(Tree Traversal) 방법으로 단어들을 전부 출력한다. MEMORIZED를 입력할 경우 MEMORIZED Circular Linked List의 단어들을 외운 순서대로 전부 출력한다. 만약 입력한 단어장 정보가 존재하지 않을 경우 알맞은 에러 코드를 출력한다.
UPDATE	사용 예) UPDATE apple 사과 단어의 뜻을 변경하는 명령어로 TO_MEMORIZE, MEMORIZING, MEMORIZED에 입력한 단어가 존재할 경우 단어의 한글 뜻을 입력한 뜻으로

	변경한 뒤 결과를 출력해준다. 만약 단어가 존재하지 않거나 단어장 정보가 존재하지 않을 경우 알맞은 에러 코드를 출력한다.
EXIT	사용 예) EXIT 프로그램을 종료한다.

☐ Requirements in implementation

- ✓ 모든 명령어는 command.txt에 저장하여 순차적으로 읽고 처리한다.
- ✓ 모든 명령어는 반드시 대문자로 입력한다. (단어 제외. ex, SEARCH apple)
- ✓ TEST, SEARCH, UPDATE 시 단어의 대소문자를 구분하지 않는다.
  (ex, SEARCH apple == SEARCH APPLE)
- ✓ 명령어에 인자(Parameter)가 모자라거나 필요 이상으로 입력받을 경우 에러 코드를 출력한다.
- ✓ 단어 노드 클래스(WordNode Class)는 반드시 pair.java 객체를 이용하여 단어와 뜻을 저장한다.
  - `Pair<String,String> wordmean = new Pair<String, String>();`
  - wordmean.first => 영어 단어 (ex, apple)
  - wordmean.second => 단어 뜻 (ex, 사과)
- ✓ 단어장 프로그램에는 중복된 단어가 존재하지 않아야한다.
  - ADD할 때 중복된 단어가 존재할 경우 에러를 발생시키지 않고 중복된 단어를 제외한 나머지 단어들을 TO_MEMORIZE에 저장한다.
- ✓ MEMORIZING에는 100개 이하의 단어가 존재해야한다.
- ✓ 예외처리에 대해 반드시 에러 코드를 출력한다.
- ✓ 출력은 “출력 포맷”을 반드시 따라한다.
- ✓ log.txt 파일에 출력 결과를 반드시 저장한다.
  - log.txt가 이미 존재할 경우 텍스트 파일 가장 뒤에 이어서 추가로 저장한다.
- ✓ 읽어야할 텍스트 파일이 존재하지 않을 경우 해당 텍스트 파일을 생성 진행하도록 한다.

□ 동작별 에러 코드

동작	에러 코드
LOAD	100
ADD	200
MOVE	300
SAVE	400
TEST	500
SEARCH	600
PRINT	700
UPDATE	800

□ 출력 포맷

기능	출력 포맷	설명
에러(ERROR)	======== ERROR ======== 100 ======================	에러 발생 시 출력 포맷에 맞춰 에러 코드를 출력
LOAD	======== LOAD ======== Success ======================	단어장 정보를 읽어오는데 성공 시 출력 포맷에 맞춰 결과를 출력 텍스트 파일 정보가 없을 시 에러 출력
ADD	======== ADD ======== Success ======================	단어 추가 성공 시 출력 포맷에 맞춰 결과를 출력 텍스트 파일이 존재하지 않거나 텍스트 파일에 단어가 없을 경우 에러 출력
MOVE	======== MOVE ======== Success ======================	단어 이동 성공 시 출력 포맷에 맞춰 결과를 출력 TO_MEMORIZE에 단어가 없거나 MEMORIZING에 단어 수가 100을 초과하게 될 경우 에러 출력
SAVE	======== SAVE ========	단어장 정보 저장 성공 시 출력

	Success ========== 포맷에 맞춰 결과를 출력 저장할 단어장 정보가 없을 시 에러 출력	
TEST	======== TEST ======== Pass ========== TEST 통과 시 출력 포맷에 맞춰 결과를 출력 없는 단어를 입력하거나 뜻이 틀리면 에러 출력	
SEARCH	======== SEARCH ======== area 구역 ========== 단어 탐색에 성공 시 출력 포맷에 맞춰 결과를 출력 단어가 없을 경우 에러 출력	
PRINT	======== PRINT ======== activity 활동 advertisement 광고 always 언제나 ... ========== 해당 단어장 정보가 존재할 시 출력 포맷에 맞춰 결과를 출력 해당 단어장 정보가 없을 경우 에러 출력	
UPDATE	======== UPDATE ======== area 지역 -> 구역 ========== 업데이트할 단어가 존재할 경우 출력 포맷에 맞춰 결과를 출력 업데이트할 단어가 존재하지 않을 경우 에러 출력	

□ 동작 예시

command.txt
LOAD word.txt ADD MOVE 100 PRINT MEMORIZED PRINT MEMORIZING UPDATE area 구역 SEARCH area TEST area 구역 PRINT MEMORIZED SAVE EXIT

실행 후 log.txt

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

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

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

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

======== PRINT ========
problem 문제
place 장소
person 사람
part 부분
paper 종이
piece 조각
...
=======================

======== UPDATE ========
area 지역 -> 구역
==========================

======== SEARCH ========
area 구역
==========================

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

======== PRINT ========
area 구역
=======================

======== SAVE ========
Success
=======================

======== EXIT ========
Success
=======================
```

□ 구현 시 반드시 정의해야하는 Class

- ✓ AlphabetNode.java – 알파벳 노드 클래스
- ✓ AlphabetBST.java – 알파벳 BST 클래스
- ✓ WordNode.java – 단어 노드 클래스
- ✓ WordBST.java – 단어 BST 클래스
- ✓ CircularLinkedList.java – 환형 연결 리스트 클래스(Circular Linked List)
- ✓ Queue.java – 큐 클래스(Queue)
- ✓ Manager.java – Manager 클래스(다른 클래스들의 동작을 관리하여 프로그램을 전체적으로 조정하는 역할을 수행)

□ Files

- ✓ word.txt : 프로그램에 추가할 단어들이 저장되어 있는 파일
- ✓ command.txt : 프로그램을 동작시키는 명령어들을 저장하고 있는 파일
- ✓ to_memorize_word.txt : TO_MEMORIZE의 단어들을 저장하고 있는 파일
- ✓ memorizing_word.txt : MEMORIZING의 단어들을 저장하고 있는 파일
- ✓ memorized_word.txt : MEMORIZED의 단어들을 저장하고 있는 파일
- ✓ log.txt : 프로그램 출력 결과를 모두 저장하고 있는 파일

□ 제한사항 및 구현 시 유의사항

- ✓ 반드시 제공되는 압축파일(skel_sp_p1.zip)를 이용하여 구현하며 작성된 소스 파일의 이름과 클래스와 함수 이름 및 형태를 절대 임의로 변경하지 않는다.
- ✓ 클래스의 함수 및 변수는 자유롭게 추가 구현이 가능하다.
- ✓ 제시된 Class를 각 기능에 알맞게 모두 사용한다.
- ✓ 프로그램 구조에 대한 디자인이 최대한 간결하도록 고려하여 설계한다.
- ✓ 채점 시 코드를 수정해야 하는 일이 없도록 한다.
- ✓ 주석은 반드시 작성한다. (없으면 감점)

□ 제출기한 및 제출방법

✓ 제출기한
- 2018년 10월 3일 수요일 23:59:59 까지 제출

✓ 보고서 작성 형식
- 제공된 보고서 양식파일 사용 (하드카피는 제출하지 않음)
- 팀원 기여도를 반드시 작성

✓ 제출방법
- 소스코드와 보고서 파일(pdf)을 함께 압축하여 u-campus에 제출
  □팀장: 소스코드 및 보고서 제출 / 팀원: 보고서만 제출
  □확장자가 .java/.pdf가 아닌 파일은 제출하지 않음
  □보고서 파일 확장자가 pdf가 아닐 시 감점

✓ 제출 형식
- 팀No_학번_이름_p1.zip