2016-10-07

프로젝트

-데이터 구조 PJ#1-

광운대학교 컴퓨터공학과
2013722059김유성
2013722093김봉효
2013722049변주형

Introduction

1. 제목

영어 단어장 프로그램

2. 주제와 목적

프로젝트의 주제 : Queue, 이진 탐색 트리(Binary Search Tree = 'BST'), 환형 연결 리스트(Circular Linked List)를 구현

본 프로젝트의 목적 : BST와 Queue, Circular Linked List를 각각 구축한 후 최종적으로 하나로 결합해서 영어 단어장으로서 역할을 수행할 수 있도록 구현

여러 개의 영어 단어와 그 단어의 뜻을 입력받아 외워야 할 단어장(TO_MEMORIZE), 현재 외우고 있는 단어장(MEMORIZING), 이전에 외운 단어장(MEMORIZED)의 3개의 단어장을 설계하고 서로 연결해주어 하나의 단어장 프로그램을 구현한다. 이때 각각의 단어장들은 다음의 역할을 수행하도록 설계를 한다.

외워야할 단어장(TO_MEMORIZE) - Queue : 새로운 단어를 저장
외우고 있는 단어장(MEMORIZING) - BST(Binary Search Tree) : 입력받은 단어를 알파벳 순서로 구성
외운 단어장(MEMORIZED) - Circular Linked List : 외운 단어를 저장, 새로 들어온 단어를 가리킬 것

3. 구현할 것

Alphabet BST, Alphabet Node, Circular Linked List, Main, Manager, Queue, Stack, Word BST, Word Node, 9가지 명령어(LOAD, ADD, MOVE, SAVE, TEST, SEARCH, PRINT, UPDATE, EXIT)

4. 원리

프로젝트에 사용 될 단어의 노드는 영어단어와 한글 뜻이 데이터로 존재한다. 새로 들어온 모든 단어는 우선적으로 TO_MEMORIZE의 내부로 들어가야 한다. 단어가 추가되면 Queue는 Push동작을 취하고, 단어를 옮길 때는 Pop동작을 취한다. TO_MEMORIZE에 단어가 모두 들어가게 되면 명령어 MOVE를 사용하여 단어들을 MEMORIZING으로 옮긴다. 단, MEMORIZING의 내부의 단어의 개수는 100개로 제한한다.

명령어 MOVE에 의해서 MEMORIZING에 단어가 넘어오게 되면, MEMORIZING에 존재하는 알파벳 BST와 단어 BST에 의하여 정렬이 된다. 우선 알파벳 순서대로 단어를 분류하고, 그 후 단어끼리 분류가 된다. 앞에서 언급했듯이, MEMORIZING의 내부에는 100개의 단어를 초과할 수 없다.

주의 사항으로 알파벳 BST는 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, W의 순서를 지켜서 추가하고, 알파벳 BST는 반드시 프로그램의 시작과 동시에 바로 구축되어야 한다. 출력이 PRE-ORDER의 방식이기 때문에, 알파벳을 위의 순서대로 추가를 해야 한다.

또한 BST는 다음의 2개의 규칙을 준수해야한다.

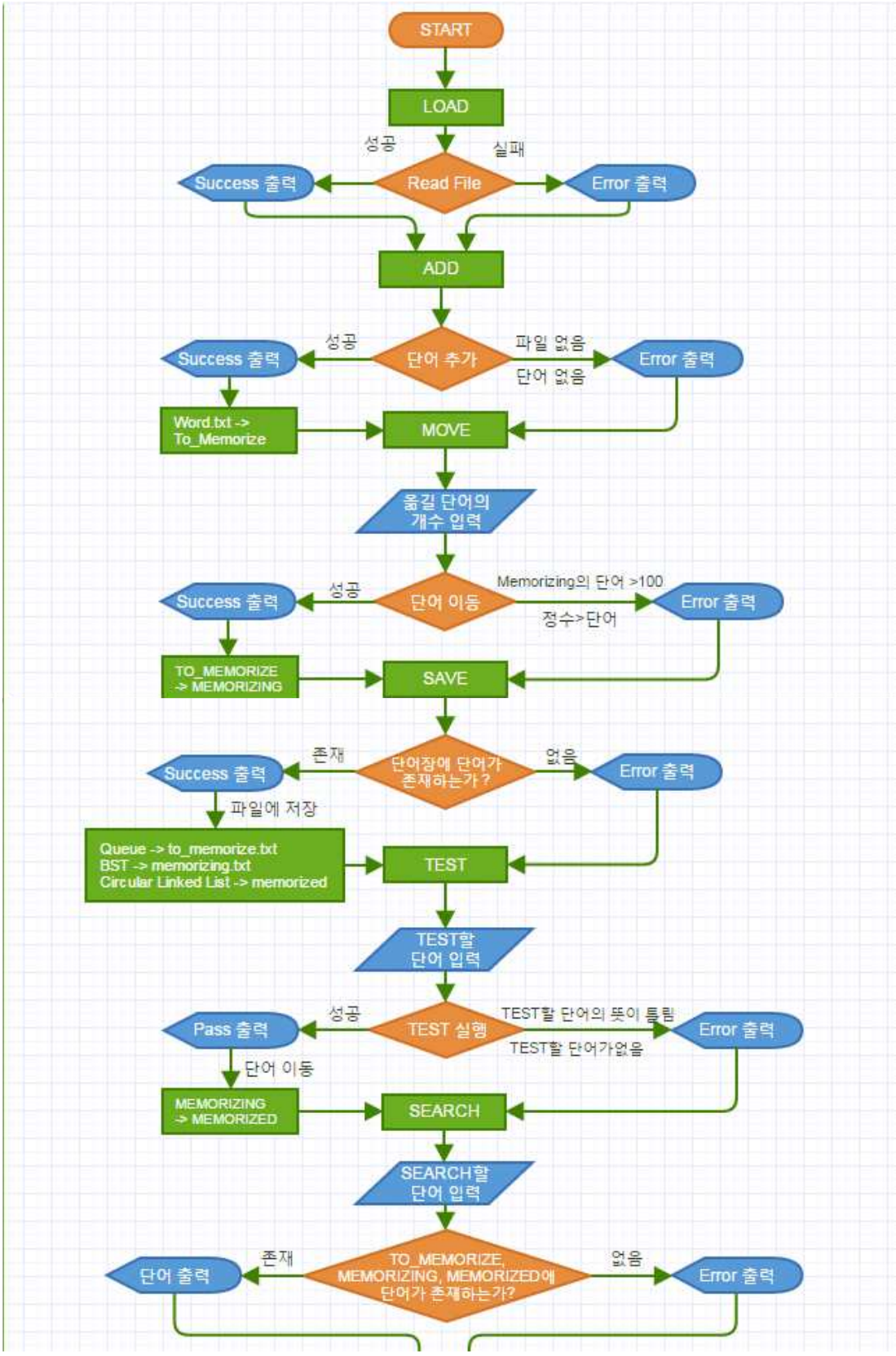
-부모 노드보다 단어의 사전적 순서가 작은 노드는 왼쪽으로, 큰 노드는 오른쪽 서브트리에 반드시 위치할 것
- 노드를 제거할 때, 양쪽 자식 노드가 모두 존재할 경우에는 왼쪽 자식 노드 중 가장 큰 노드를 제거되는 노드 위치로 이동시키며, 왼쪽 자식노드가 존재하지 않을 경우에는 오른쪽 자식 노드 중 가장 작은 노드를 제거되는 노드 위치로 이동시킨다.

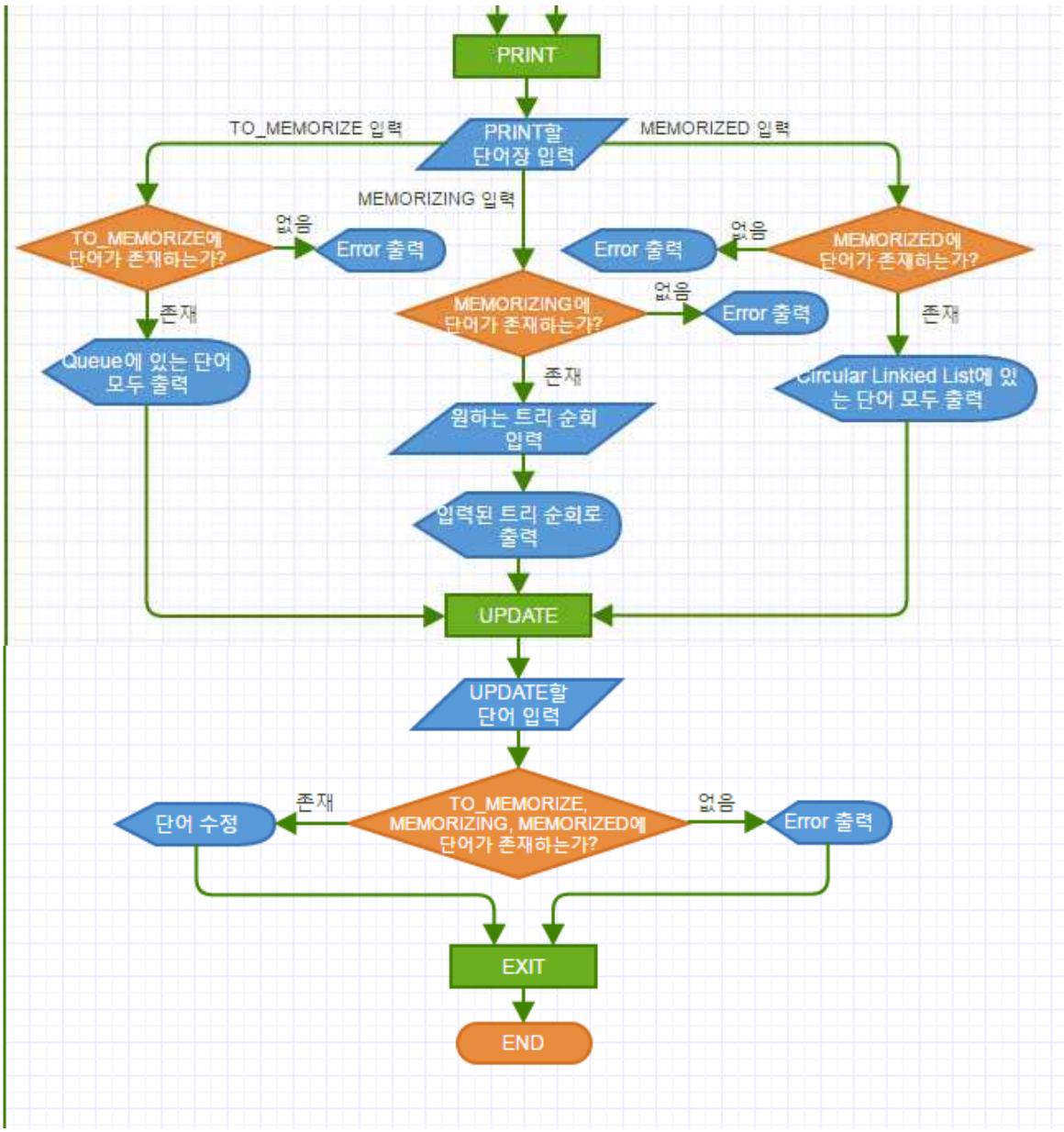
MEMORIZING의 단어들은 명령어 TEST에 의하여 테스트를 할 수 있다. 이때 테스트에 통과한 단어는 외운 것으로 간주되며 해당 단어를 MEMORIZED로 옮긴다. MEMORIZED로 들어온 단어들은 Circular Linked List의 형태로 저장되어야 한다. Circular Linked List란 저장된 마지막 단어에서 다음으로 넘어가면 다시 처음 단어가 되는 형태이다.

본 프로젝트에서는 Head pointer가 처음으로 들어온 단어를 가리키게 되고, 들어온 단어는 Head pointer를 가리키게 된다. List에 단어가 존재하는 경우, 즉 저장된 단어가 있으면 새로 들어온 단어는 Head pointer가 가리키는 단어와 바로 뒤의 노드 사이에 연결 되어야 한다.

Flowchart

다음의 순서대로 동작한다.




PRINT
PRINT할 단어장 입력
TO_MEMORIZE 입력
MEMORIZING 입력
MEMORIZED 입력
TO_MEMORIZE에 단어가 존재하는가?
없음
Error 출력
존재
Queue에 있는 단어 모두 출력
MEMORIZING에 단어가 존재하는가?
없음
Error 출력
존재
원하는 트리 순회 입력
입력된 트리 순회로 출력
MEMORIZED에 단어가 존재하는가?
없음
Error 출력
존재
Circular Linkied List에 있는 단어 모두 출력
UPDATE
UPDATE할 단어 입력
TO_MEMORIZE, MEMORIZING, MEMORIZED에 단어가 존재하는가?
존재
단어 수정
없음
Error 출력
EXIT
END

Algorithm

프로젝트 구현에 쓰인 알고리즘은 Queue, Binary Search Tree, Circular Linked List와 9가지 명령어를 사용하였다.

Queue는 다음의 동작을 실행한다.

- Push : TO_MEMORIZE에 있는 저장된 단어를 Queue로 옮길 것. 이때 단어들은 읽어 들인 순서대로 가장 처음으로 들어온 단어가 아래에서부터 쌓이도록 저장해야 한다.

- Pop : Queue에 저장된 단어들을 MEMORIZING으로 옮겨줄 것. 단어를 옮기는 순서는 위에서부터 아래로 차례대로 옮겨야 한다. 단, MEMORIZING에 단어의 수가 100개가 초과하거나 더 이상 옮길 단어가 없으면 'Pop'을 수행할 수 없다.

Binary Search Tree는 자식노드가 최대 2개가 되어야 하며, 왼쪽 자식이 부모보다 작고 오른쪽 자식이 부모보다 크도록 설계를 한다.

Circular Linked List는 Head Pointer를 사용하여 MEMORIZED에 처음으로 단어가 들어오면 그 단어를 가리키도록 설정한다. 이후 추가로 단어가 들어오면 기존의 Head Pointer가 가리키고 있는 단어와 바로 이전의 단어의 사이에 저장하고, Head Pointer는 새로운 단어를 가리키도록 설정해야 한다. 이러한 동작을 취하면 최종적으로 처음 저장된 단어와 마지막으로 저장된 단어가 서로 연결된다.

9가지 명령어는 다음의 기능을 실행한다.

- LOAD : 기존의 단어장 정보를 불러올 것. 텍스트 파일에 단어장 정보가 이미 존재할 경우 텍스트 파일을 읽어 TO_MEMORIZE Queue, MEMORIZING BST, MEMORIZED Circular Linked List의 자료에 이전과 동일한 연결 순서를 가지도록 저장할 것. 단, 3개의 텍스트 파일('to_memorize_word.txt', 'memorizing_word.txt', 'memorized_word.txt')이 존재하지 않거나 이미 데이터가 들어가 있다면 에러 코드를 출력할 것.

사용 예) LOAD

- ADD : 단어가 저장되어 있는 텍스트 파일을 읽어서 존재하는 모든 단어들을 TO_MEMORIZE에 저장할 것. 단, 텍스트 파일('word.txt')이 존재하지 않거나 이미 데이터가 들어가 있다면 에러 코드를 출력할 것.

사용 예) ADD

- MOVE : 1~100사이의 정수를 입력한다. 입력한 정수만큼 TO_MEMORIZE의 단어들을 MEMORIZING으로 옮길 것. 이때 옮겨지는 단어와 MEMORIZING에 존재하는 단어와의 합이 100을 초과할 수 없다. 입력한 정수만큼 옮길 단어가 존재하지 않거나 MEMORIZING의 단어와 합이 100을 초과하면 에러 코드를 출력할 것

사용 예) MOVE 100

- SAVE : 명령어가 입력되면 현재의 단어장(Queue, BST, Circular Linked List)에 저장된 정보들을 각각의 텍스트 파일('to_memorize_word.txt', 'memorizing_word.txt', 'memorized_word.txt')에 저장할 것. 이때 MEMORIZING 텍스트 파일이 Skewed Binary Tree가 되므로 주의할 것. 단어장 정보가 존재하지 않으면 에러 코드를 출력할 것.

사용 예) SAVE

- TEST : MEMORIZING에 입력한 단어가 존재하는지, 뜻이 일치하는지를 확인하고 통과하면 해당 단어를 외운 단어로 간주하고 MEMORIZED로 이동시킬 것. 단, 단어가 존재하지 않거나 뜻이 틀릴 경우 에러 코드를 출력할 것.

사용 예) TEST apple 사과

- SEARCH : TO_MEMORIZE, MEMORIZING, MEMORIZED에 입력한 단어가 존재하면 해당 단어와 뜻을 출력할 것. 단어가 없을 경우 에러 코드를 출력할 것.

사용 예) Search apple

- PRINT : 입력에 따라서 단어들을 출력할 것. TO_MEMORIZE를 입력하면 Queue에 있는 모든 단어를 Header부터 순서대로 전부 출력할 것. MEMORIZING을 입력하면 추가로 "R_PRE", "I_PRE", "R_IN", "I_IN", "R_POST", "I_POST", "I_LEVEL"을 입력받아 각각의 트리순회 방법대로 출력할 것. MEMORIZED를 입력하면 외운 순서대로 모든 단어를 출력할 것. 단, 입력한 단어장 정보가 존재하지 않으면 에러 코드를 출력할 것.

사용 예) PRINT MEMORIZING I_PRE

출력방식은 다음과 같다.

R_PRE: Recursive pre-order(재귀함수를 이용한 pre-order)
I_PRE: Iterative pre-order(반복문을 이용한 pre-order, 재귀함수 사용금지)
R_IN: Recursive in-order(재귀함수를 이용한 in-order)
I_IN: Iterative in-order(반복문을 이용한 in-order, 재귀함수 사용금지)
R_POST: Recursive post-order(재귀함수를 이용한 post-order)
I_POST: Iterative post-order(반복문을 이용한 post-order, 재귀함수 사용금지)
I_LEVEL: Iterative level-order(반복문을 이용한 level-order)

- UPDATE : 입력한 단어가 TO_MEMORIZE, MEMORIZING, MEMORIZED에 존재할 경우, 단어의 한글 뜻을 입력한 뜻으로 변경하고 결과를 출력할 것. 단어가 존재하지 않거나 단어장 정보가 존재하지 않으면 에러 코드를 출력할 것.

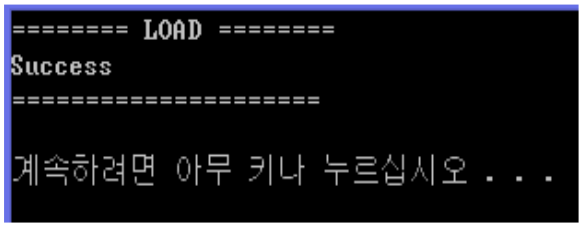
사용 예) UPDATE apple 사과

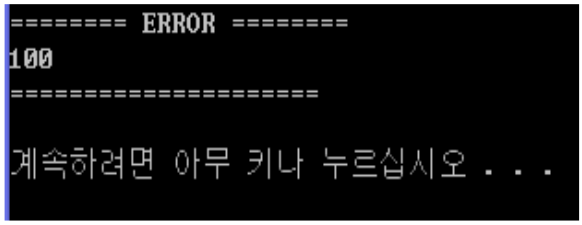
- EXIT : 프로그램 상의 모든 메모리를 해제하며, 프로그램을 종료할 것.

ERROR코드는 다음과 같다.

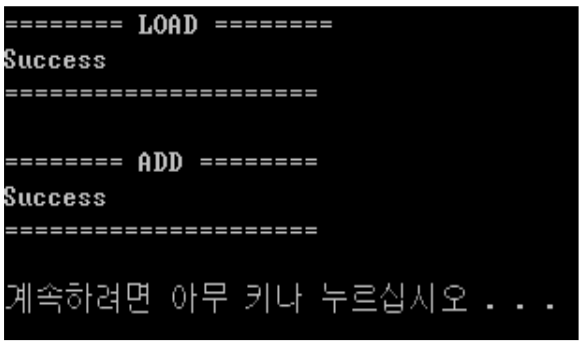
동작	에러코드
LOAD	100
ADD	200
MOVE	300
SAVE	400
TEST	500
SEARCH	600
PRINT	700
UPDATE	800

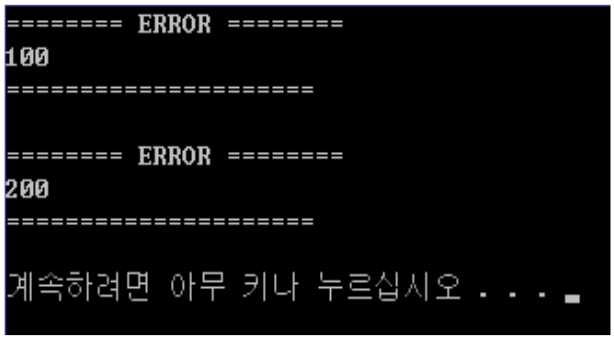
Result Screen



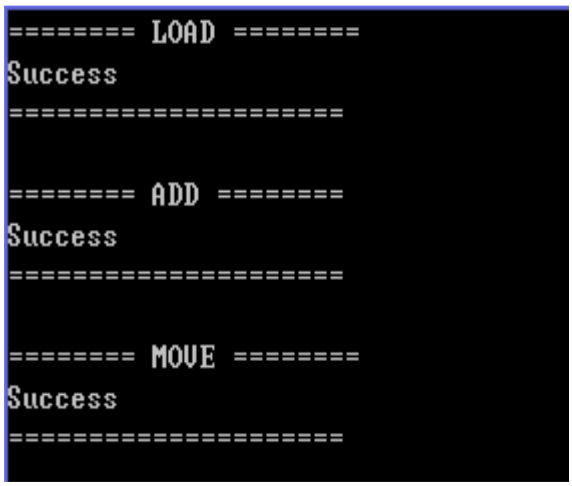


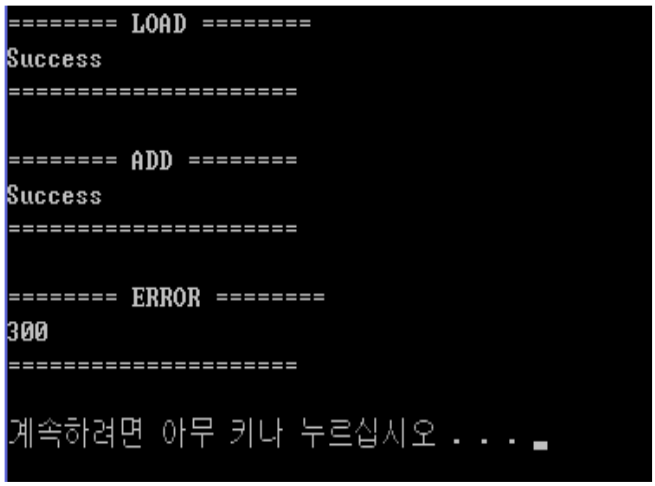
명령어 LOAD가 정상적으로 실행되면 'Success'가 출력되고, 3개의 텍스트 파일이 존재하지 않거나 자료구조에 이미 데이터가 존재할 경우 ERROR 코드 '100'이 출력된다.



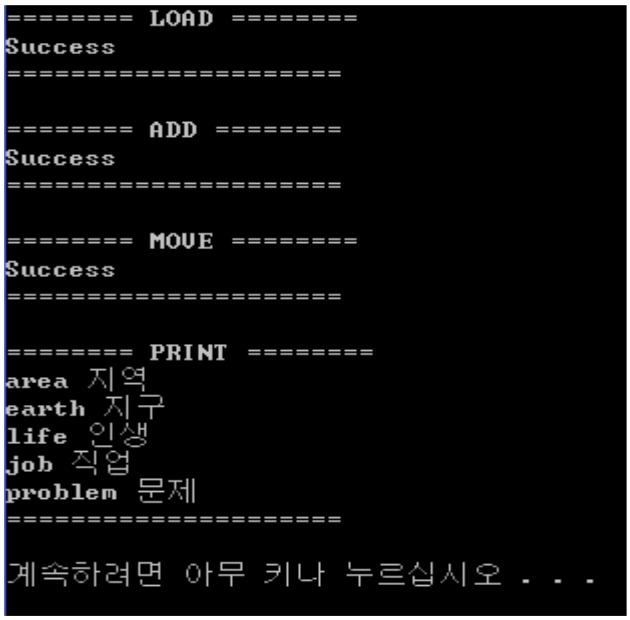


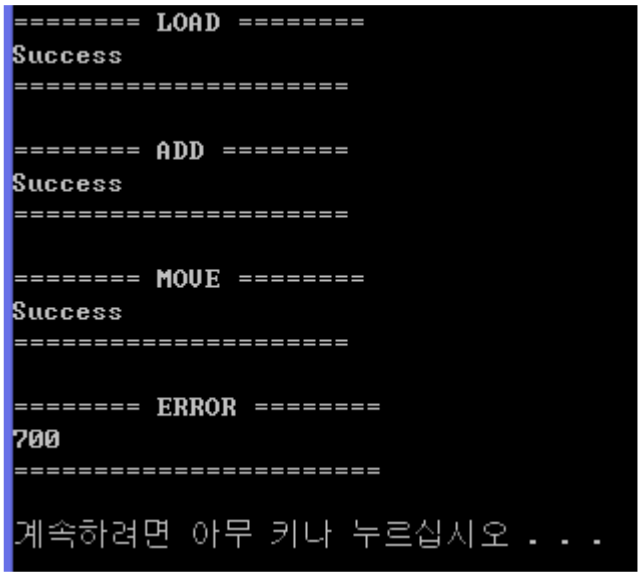
명령어 ADD가 정상적으로 실행되면 'Success'가 출력되고, 단어 텍스트 파일이 존재하지 않거나 파일에 단어가 없으면 ERROR 코드 '200'이 출력된다.



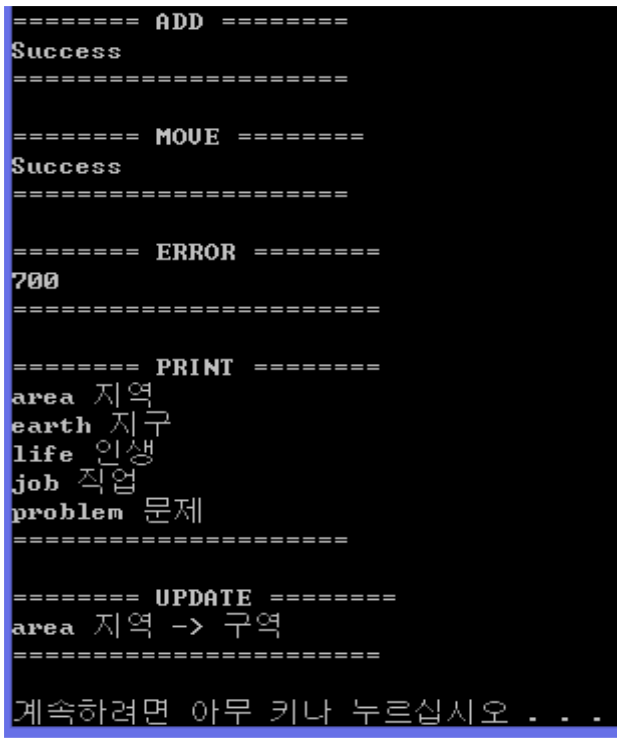


명령어 MOVE가 정상적으로 실행되면 'Success'가 출력되고, 단어가 옮길 정수보다 적거나 MEMORIZING에 단어가 100개가 초과되면 ERROR코드 '300'이 출력된다.

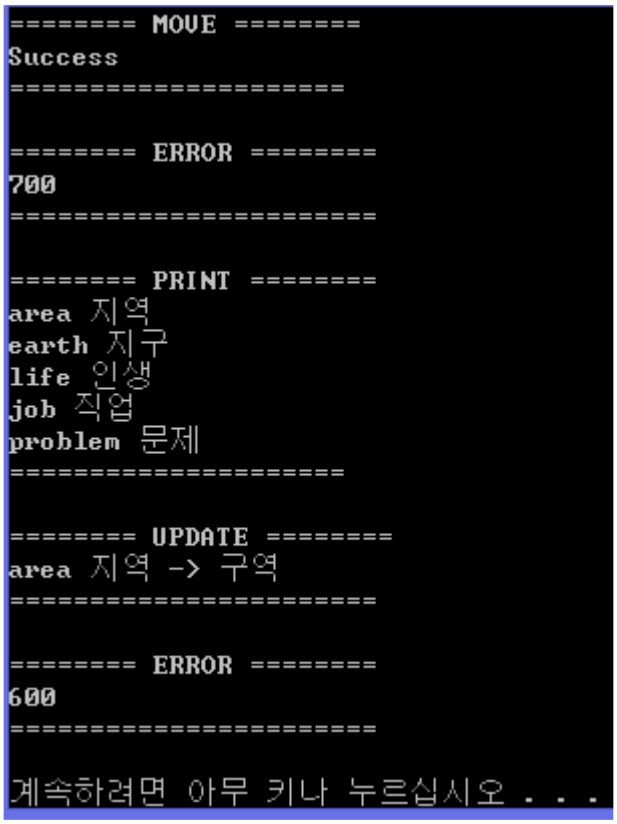




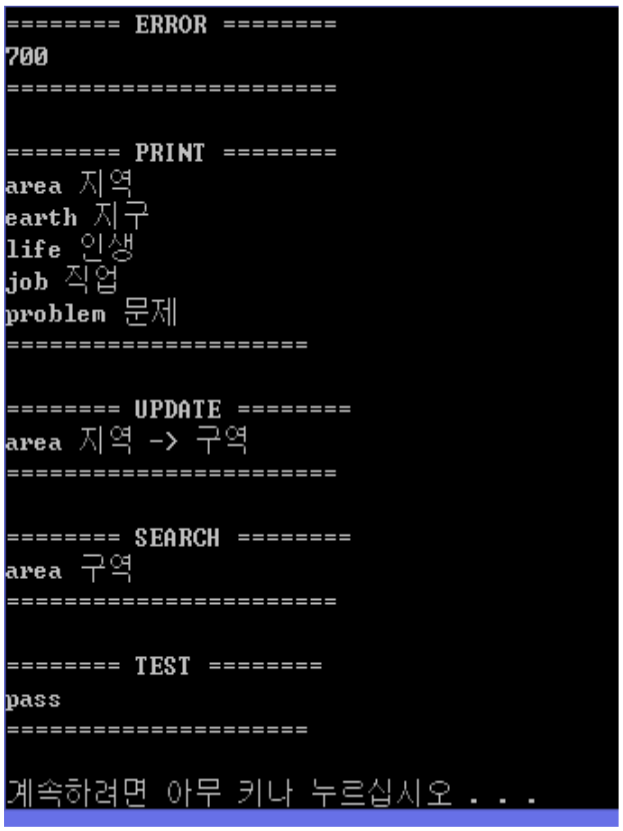
명령어 PRINT가 정상적으로 실행되면 입력에 따라서 TO_MEMORIZE를 입력하면 Queue에 있는 모든 단어를 순서대로 전부 출력하고, MEMORIZING을 입력하면 추가로 “R_PRE”, “I_PRE”, “R_IN”, “I_IN”, “R_POST”, “I_POST”, “I_LEVEL”중 하나를 입력하고, 입력받은 트리순회 방법대로 출력한다. MEMORIZED를 입력하면 외운 모든 단어를 출력한다. 입력한 단어장의 정보가 존재하지 않으면 ERROR 코드 ‘700’이 출력된다.

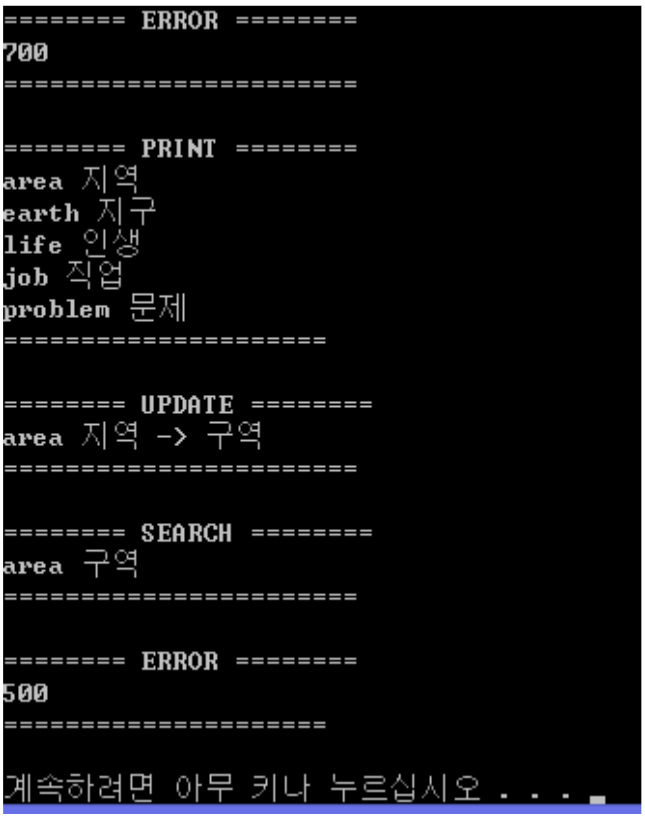


명령어 UPDATE가 정상적으로 실행되면 입력한 단어가 존재하면 그 뜻을 입력한 값으로 바꿔준다. 단어가 존재하지 않으면 ERROR 코드 ‘800’이 출력된다.

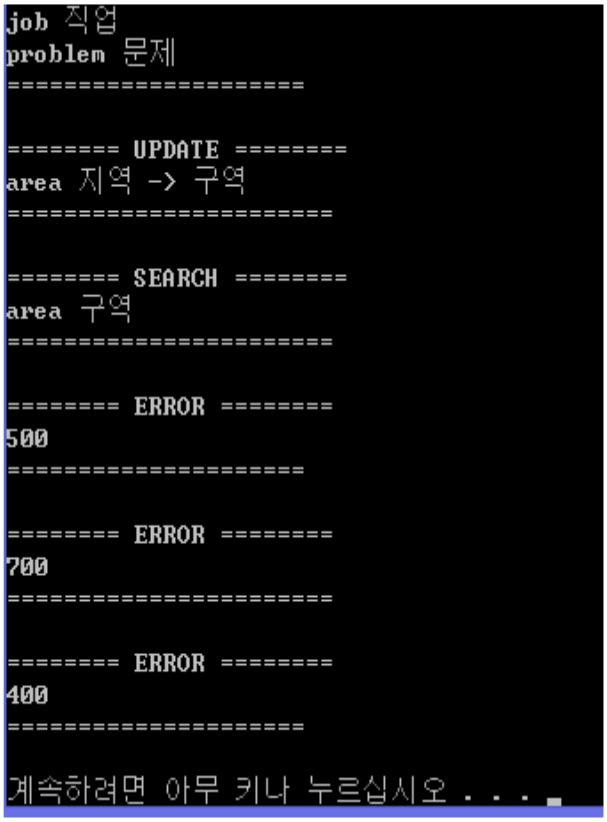


명령어 SEARCH가 정상적으로 실행되면 입력한 단어가 존재하면 그 뜻을 출력한다. 단어가 존재하지 않으면 ERROR 코드 '600'이 출력된다.





명령어 TEST가 정상적으로 실행되면 입력한 단어를 TEST하고 통과하면 ‘PASS’가 출력되고, 통과하지 못하면 ERROR 코드 ‘500’이 출력된다.



명령어 SAVE가 정상적으로 실행되면 입력한 단어를 파일에 저장하고 ‘Success’가 출력되고, 실패하면 ERROR 코드 ‘400’이 출력된다.

Consideration

-팀장 김봉효

프로젝트를 막상 진행하면서 첫 느낌은 턱 막혔었다. 이 문제를 보자마자 무엇부터 해나가야 할지... 우선 문제를 이해하자로 시작하여 문제를 오랫동안 보고 각 기능이 어떤 기능을 갖고 있는지부터 머릿속에 채웠던 것 같다. 또한 팀원들과 이 부분을 어떻게 해결해야하는지 많은 이야기를 나눈 것 같다. 내가 맡은 부분이 queue 부분과 manage 함수의 add부분을 코딩하기 위해 각종 책과 인터넷을 보고 공부를 하고 또한 팀원들에게 물어볼 수 있어서 좋았다.

이 프로젝트를 진행하면서 좋았던 점은 누군가에게 모르는 것을 물어보면서 배워갈 수 있는 점이 좋았다. 그러나 내가 누군가에게 짐이 되고 있는지는 아는가? 이 기회를 통해 책임감을 갖게 되고 좀 더 매진하게 되는 장점이 있었던 것 같다. circular linked list부분과 BST 부분을 마치 과외 받듯이 잘 알아가는 것에 대해 만족스러웠다.

프로젝트에서 맡은 역할 : Queue, Manager.cc구현
본인 스스로 생각하는 자신의 점수(10) : 5

-팀원 김유성

프로젝트를 시작하면서, 구현하고자 하는 단어장이 어떻게 동작되며, 무엇이 필요한지에 대해서 알아야 했다. 토론 후 팀원들은 각자 분야를 나누어서 코드를 구현하였으며, 문제가 발생하면 서로의 의견을 주고받으며 프로젝트를 진행하였다. 모르는 부분은 각종 서적이나 인터넷을 참고하거나 팀원들에게 물어가며 하나하나씩 해결하였다.

Queue의 동작인 'Pop'과 'Push'에 대해서 명확하게 몰랐던 부분을 팀원을 통하여 알게 되었으며, 그것의 동작원리 또한 알 수 있게 되었다. BST를 구현하는 방법에 대해서 정확하게는 몰랐었는데, 팀 프로젝트가 아닌 혼자서 설계하였다면 절대로 해결할 수 없었을 것이다. Circular Linked List의 코드는 기존의 내가 알고 있었던 방식과 조금 다르게 설계가 되어서 이해하는데 문제가 조금 있었는데, 다른 팀원에게 도움을 받아서 이해할 수 있었다.

프로젝트에서 맡은 역할 : Circular Linked List 구현
본인 스스로 생각하는 자신의 점수(10) : 5

-팀원 변주형

이번 프로젝트는 3개의 파일 구조에 입력, 삭제, 검색, 출력, 순회 등을 구현하는 것으로 AlphabetBST 트리 안에 WordBST트리가 들어있는 2중 트리 구조와 파일 입출력 부분에서 예외처리를 하는 부분이 구현을 하는데 상당히 까다로웠다. 또한 리눅스환경에서 컴파일을 할 때 디버깅이 익숙하지 않아서 오류가 발생했을 때 대처하기 매우 어려웠다.

처음으로 코딩을 팀프로젝트로 진행해봤는데 특정 함수나 클래스를 구현함에 있어서 혼자서 모든 코드를 짜는게 아닌 팀원 별로 나눠서 코딩하다 보니 그 클래스나 함수에 대해서 올바르게 구현했다 하더라도 다른 팀원이 짠 코드들과 호환이 안되는 경우들이 발생하였다. 구현을 시작하기 전 팀원들끼리의 충분한 의사소통을 통해 구현방향성을 정확히 잡아야 이러한 일을 사전에 방지할 수 있을 것 같다.

프로젝트에서 맡은 역할 : WordBST tree, AlphabetBST tree, Manager.cc 구현
본인 스스로 생각하는 자신의 점수(10) : 7