데이터 구조 설계

Project #1 : Word Book

팀장 : 2013722095 최재은

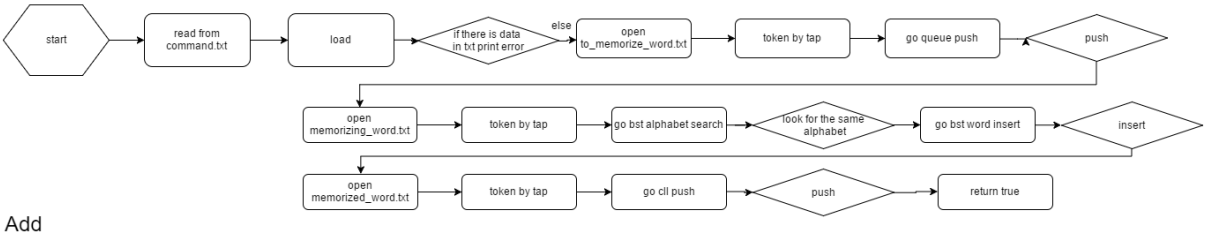
팀원 : 2013722079 김영근

1. Introduction

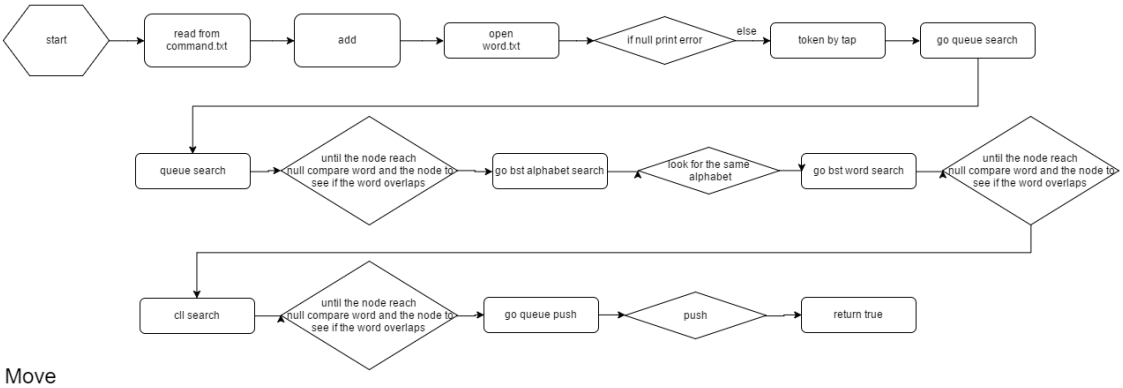
이 프로젝트에서 구현한 프로그램은 단어장 프로그램으로 모든 동작은 command.txt 파일을 읽어와 안에 저장되어 있는 명령어들을 통해서 수행된다. txt형식의 word 파일 등을 읽어오고, 기존에 외우던 단어들의 자료형에 추가, 외우고 있는 단어들의 자료형으로 이동, 시험, 탐색, 출력, 단어의 뜻 변경, 저장하는 기능을 포함하고 있다. 이들은 각각 LOAD, ADD, MOVE, SAVE, TEST, SEARCH, PRINT, UPDATE의 명령문을 통해 수행된다. 외워야 할 단어들은 Queue의 구조, 외우고 있는 단어들은 Binary Search Tree의 구조로, TEST를 완료한 단어들은 Circular Linked list의 구조로 구현된다. 이전에 이미 외우던 내용들이 있는 to_memorize_word.txt 파일에 있는 단어들 또한 Queue 구조에 들어가게 된다. ADD명령어를 통해 word파일을 불러와 Queue에 없는 단어들만 추가하며, MOVE 명령어를 통해서 원하는 수만큼 단어들을 Queue에서 BST로 넘긴다. 이때 TEST를 통해 Pass한 단어들을 CLL로 넘기는 동작을 수행할 수 있다. PRINT 명령어를 통해서 각각 자료형들이 갖고 있는 단어들을 출력할 수 있는데, BST는 Recursive/Iterative Inorder, Recursive/Iterative Preorder, Recursive/Iterative Postorder, Iterative Revelorder의 방식으로 선택 출력할 수 있다. UPDATE 명령어를 통해서 기존에 있던 단어의 뜻을 변경할 수 있고, SEARCH 명령어를 통해서 단어장에 존재하는 단어와 그 뜻을 출력해줄 수 있다. SAVE 명령어를 통해서 각각 자료형에 존재하는 단어들을 to_memorize_word.txt, memorizing_word.txt, memorized_word.txt 파일에 저장해 준다. LOAD 명령어를 통해서 위의 3개 파일들을 읽어와 각각의 자료형으로 구현하는데, 이미 자료형이 존재하는 경우 LOAD 명령어를 실행하면 에러문이 출력된다. 3개의 파일이 존재하지 않는 경우 동일한 에러문이 출력된다. ADD의 경우 word 파일이 존재하지 않거나 파일에 단어가 존재하지 않을 경우 에러코드를 출력한다. MOVE의 경우 입력되는 수가 1~100이 아니거나, queue에 저장할 단어가 없는 경우, 입력받은 수와 BST에 존재하는 단어들의 수가 100을 넘어갈 경우에도 에러코드를 출력한다. SAVE는 단어장 정보가 존재하지 않을 경우 알맞은 에러를 출력한다. TEST는 찾는 단어가 존재하지 않거나 뜻이 다른 경우 알맞은 에러코드를 출력한다. SEARCH는 찾는 단어가 존재하지 않을 경우 에러문을 출력하고 PRINT의 경우 단어장의 정보가 존재하지 않을 경우 에러코드를 출력한다. UPDATE 또한 변경할 단어가 존재하지 않을 경우 에러문을 출력한다. EXIT 문이 입력되면 프로그램 상의 메모리를 해제하고 프로그램을 종료한다. 화면에 출력되는 모든 내용들은 log.txt 파일에 저장되어야 한다.

2. Flowchart

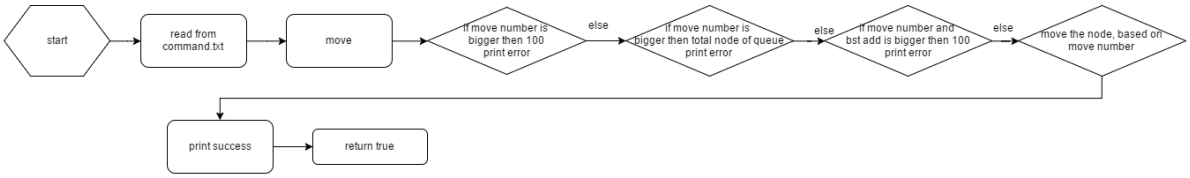
Load



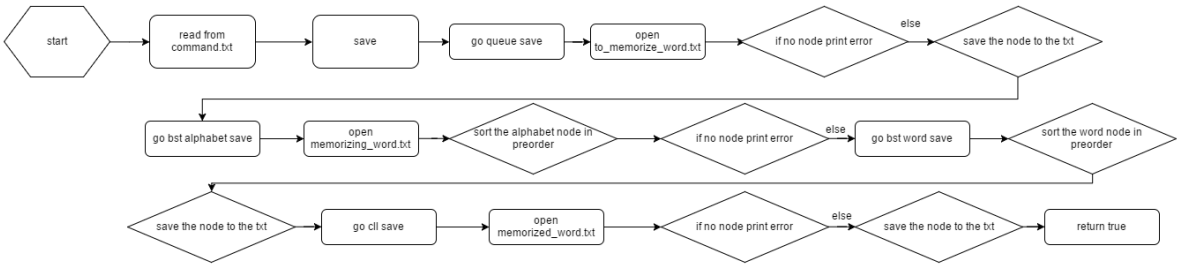
Add



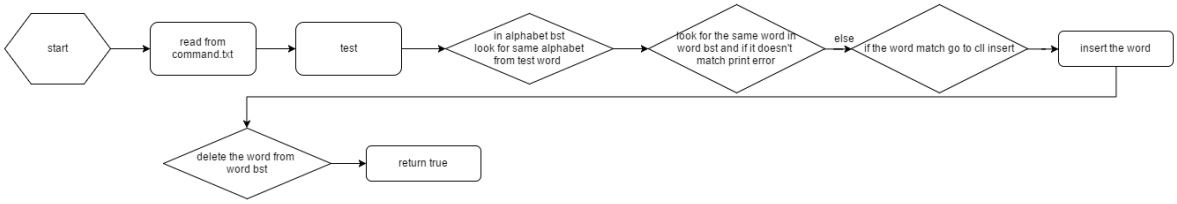
Move

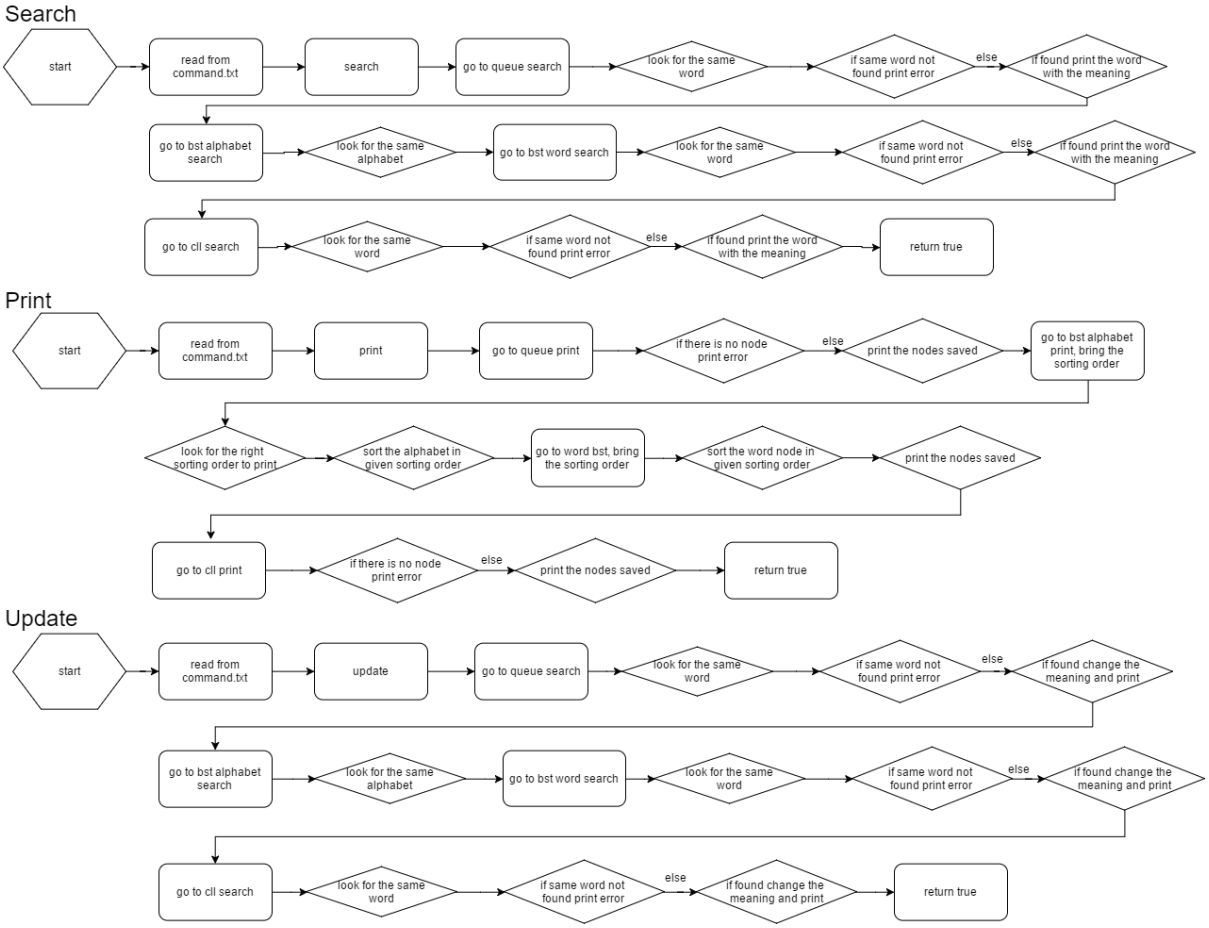


Save



Test





3. Algorithm

1) main

- Manager class의 run function을 실행한다.
- 실행할 파일의 이름을 run함수에 넣어준다.'

2) Manager

* run

- PHXDLTZBFJNRVYACEGIKMOQSUW의 순으로 알파벳 BST를 구성한다.
- for문을 통해서 26개의 알파벳 노드를 만들고 이는 AlphabetBST의 Insert 함수를 통해 BST 형태로 구현된다.
-ifstream으로 command 파일을 열고, 파일의 끝이 나올 때까지 반복문을 실행
- getline 함수를 통해 앞서 선언된 배열에 파일상의 한 줄을 저장
- strtok 함수를 이용해 'space' 기준으로 배열을 자르는데, 첫 token이 'EXIT'면 프로그램 종료
- 토큰 3개를 입력 받아 해당 명령문의 함수들을 실행하는 ComImple 함수 실행

3) LOAD

- 현재 자료형이 존재하는지 확인을 위해서 queue의 pHead가 NULL인지, BST의 wordcnt가 0인지, cll의 pHead가 NULL인지를 3중 if문을 통해서 검사. if문에 해당되지 않는 경우 알맞은 에러코드(100)을 출력함.
- 3개의 text파일(to_memorize_word.txt, memorizing_word.txt, memorized_word.txt)을 open
- to_memorize_word 파일 열기에 실패할 경우 Successflag를 0으로 함.
- 열기 성공 시, 파일의 끝이 나올 때까지 반복문을 실행.
- 한 줄을 읽어 배열에 저장하는데 배열의 첫 칸이 NULL값이고 파일의 시작이라면 반복문을 탈출함. 아니라면 WordNode* node에 WordNode형식으로 동적 할당을 해주고 token으로 잘라 WordNode class의 Setword/SetMean 함수에 넣어준다.
- 이후 node를 queue에 push해준다.
- memorizing_word.txt. 파일열기에 실패한 경우 Successflag를 0으로 한다.
- 위와 동일한 방식으로 node에 단어와 뜻을 setting하는데, bst의 serch함수를 통해 알파벳 BST를 순회하며 알맞은 알파벳 노드를 찾고, 알파벳 노드의 wordBST에 insert한다.
- memorized 파일 열기에 실패한 경우 successflag를 0으로 한다.
- 위와 동일한 방식으로 node에 단어와 뜻을 Setting하고, node를 cll에 insert한다.
- 위의 동작을 수행하고 SuccessFlag가 0이라면 알맞은 에러코드를 출력한다.
- SuccessFlag가 1이라면 Success를 출력한다.

4) ADD

- word.txt 파일을 읽어들임. Tok1, tok2를 NULL로 초기화함.
- 실패 시 에러코드 200을 출력하고 false값 반환.
- 성공 시, 파일의 끝까지 반복문을 실행.
- WordNode* node를 WordNode 형으로 동적할당.
- 한 줄씩 읽어 배열에 저장하는데, int first가 0이고 배열의 첫 칸이 NULL값이라면 반복문을 탈출함. int first가 1이고 배열의 첫 칸이 NULL값이라면 에러코드 200을 출력.
- 배열을 '₩t'을 기준으로 자른 tok1(단어), tok2(뜻)을 node에 저장.
- 자료형 queue,bst,cll에 해당 단어가 있는지 확인, 있으면 동적 할당된 node를 delete함.
- 없으면 queue에 push하고 success를 출력

5) MOVE

- int형 변수 number를 선언하고 tok2(run에서 저장되어있는)를 atoi함수를 통해 int형으로 변환시켜서 number에 저장.
- number가 100을 넘는, queue에 있는 노드의 개수가 number보다 적은, bst에 있는 Wordnode의 개수+number가 100을 넘는 경우에는 에러코드 300 를 출력
- 위의 조건에 해당되지 않는다면 number만큼 반복문을 실행시킴.
- 반복문의 내용은 다음과 같음. queue에서 pop시킨 wordnode를 bst에서 알맞은 알파벳 노드를 찾고, 그 알파벳 노드의 wordbst에 insert함. Bst의 WordCnt를 1증가 시킴.
- 위의 반복문이 끝나면 Success를 출력함.

6) SAVE

- Queue의 save 함수(to_memorize_word파일을 열고 pHead가 NULL이면 파일 닫음, 아니라면 반복문을 통해 open한 파일에 queue의 node들의 단어와 뜻을 씀.)를 실행.
- bst의 save함수(알파벳 노드를 preoder로 순회하면서 각 알파벳 노드의 wordBST에 있는 노드들의 단어와 뜻을 open한 파일에 씀. WordBST를 순회하는 방법 또한 preorder)를 실행.
- CLL의 save함수(memorized_word파일을 열고 pHead가 null이면 파일 닫음, 아니라면 반복문을 통해 open한 파일에 cll의 node들의 단어와 뜻을 씀.)를 실행.
- 위의 함수들이 모드 true를 반환하면 success를 출력하고, 하나라도 false를 반환하면 에러코드 400을 출력

7) TEST

- run에서 저장된 tok2(단어)와 tok3(뜻)를 활용
- bst의 search를 통해서 해당되는 wordnode를 찾음, 못 찾으면 에러코드 500출력
- 찾았을 때 조건문을 통해 그 wordnode의 뜻과 tok3를 비교
- 서로 같으면 pass출력, 다르면 에러코드 500출력

8) SEARCH

- run에서 저장된 tok2를 활용
- queue, bst, cll의 search를 통해서 해당되는 wordnode를 찾음. 못 찾으면 에러코드 600출력
- 찾으면 그 wordnode의 단어와 뜻을 출력해줌.

9) UPDATE

- run에서 저장된 tok2, tok3를 활용
- queue, bst, cll의 search를 통해서 해당되는 wordnode를 찾음. 못 찾으면 에러코드 800출력
- 찾으면 그 wordnode의 뜻을 setmean함수를 통해 tok3로 저장.

10) PRINT

- run에서 저장된 tok2(어떤 자료형의 단어들을 출력할지), tok3(어떤 형태로 출력할지)를 활용
- ChoosePrint함수에 tok2, tok3를 넘겨줌.

11) ChoosePrint

- 조건문을 통해 해당 자료형의 Print함수를 실행
- tok2가 'TO_MEMORIZE'이면 queue의 print함수 실행
- tok2가 'MEMORIZING'이면 bst의 print함수에 tok3를 넘겨줌
- tok2가 'MEMORIZED'면 cll의 print함수 실행

12) ComImple

- run 함수에서 넘겨 받은 tok1, tok2, tok3를 활용(아래에서 실행하는 함수들은 manager class의 함수들임.)
- tok1이 LOAD이면 LOAD함수를 실행, 만약 tok2가 null값이 아니라면 (command에서 LOAD 뒤에 추가적인 단어가 존재하면) 에러코드 100 출력
- tok1이 ADD면 ADD함수를 실행, 만약 tok2가 null값이 아니라면 에러코드 200출력
- tok1이 MOVE면 MOVE함수를 실행, 만약 tok3가 null값이 아니라면 에러코드 300출력
- tok1이 SAVE면 SAVE함수를 실행, 만약 tok2가 null값이 아니라면 에러코드 400 출력

- tok1이 TEST면 TEST함수를 실행
- tok1이 SEARCH면 SEARCH함수를 실행
- tok1이 PRINT면 PRINT함수를 실행
- tok1이 UPDATE면 UPDATE함수를 실행

13) Queue::push
- pHead가 null값이면 입력받은 노드를 pHead가 가리키게 함
- 아니라면 반복문을 통해 Wordnode* temp를 Queue의 끝까지 이동시키고 temp의 pNext를 노드로 한다. 노드의 pNext는 NULL로 하낟.

14) Queue::pop
- temp가 NULL이면 NULL값을 반환
- pHead가 그 다음노드를 가리키게하고 temp의 링크를 끊은 다음 temp(node)를 반환한다.

15) Queue::up
- toupper함수를 이용해서 입력 받은 char* word의 글자들을 대문자로 배열에 저장한 다음 그 배열을 반환한다.

16) Queue::Search
- queue link를 순회하면서 입력 받은 char* word의 글자들을 up에 넣어 배열에 저장한다.
- 현재 순회하는 노드의 단어를 up에 넣어 다른 배열에 저장한다.
- 두 배열을 비교해서 서로 같을 때까지 다음 노드로 넘어가며, 찾으면 해당 노드를 반환한다.

17)Queue:: PRINT
- queue link에 노드가 없으면 에러코드 700을 출력
- 만약 노드가 pHead뿐이면 pHead를의 단어와 뜻을 출력한다.
- 그 이외의 경우 순회포인터가 NULL을 가리킬때까지 순회하면서 노드의 단어와 뜻을 출력함.

18)Queue::Save
- to_memorize_word 파일을 쓰기모드로 연다.
- 노드가 없으면 파일을 닫고 false 반환
- 그 이외의 경우 반복문을 통해서 순회포인터가 null값을 가리킬 때까지 순회하면서 노드의 단어와 뜻을 파일에 쓴다.

19) Queue::Cntqueue
- 반복문을 통해 순회포인터가 null값을 가리킬 때까지 순회하면서 cntqueue값을 1씩 증가시키고 반복문을 탈출하면 cntqueue값을 반환한다.

20) AlphabetBST::Insert
- 알파벳 노드가 없으면 해당 노드를 root로 하고 종료
- 노드의 알파벳과 순회 포인터의 알파벳을 비교. 노드의 알파벳이 더 작을 때 순회포인터가 가리키는 노드의 왼쪽자식이 없으면 노드를 순회포인터의 왼쪽자식으로 셋팅하고 반복문 탈출. 왼쪽자식이 있으면 순회포인터를 왼쪽자식으로 이동시킴
- 노드의 알파벳이 더 큰 경우 순회포인터가 가리키는 노드의 오른쪽 자식이 없으면 노드를 순회포인터의 오른쪽 자식으로 셋팅하고 반복문 탈출. 오른쪽 자식이 있으면 순회포인터를

오른쪽 자식으로 이동시킴.

21) AlphabetBST::Saveroot
- 입력된 노드의 pSnext를 설정하여 일직선 형태의 링크로 만듦.
- 만약에 일적선 형태의 링크에 노드가 없으면 입력된 알파벳 노드를 Sroot로 함.

22) AlphabetBST::Print
- APreorder함수를 통해서 모든 알파벳 노드들을 일직선 형태의 링크(pSnext)로 만들어줌
- pSnext링크를 순회하면서 wordNode의 존재여부. 있으면 nodeis변수를 1로 함.
- nodeis변수가 0이면 에러코드 700을 출력함
- 이후 조건문에서 비교를 통해 알맞은 order함수들(R_IN, I_IN, R_PRE 등)을 호출함.

23) AlphabetBST::Search
- 순회포인터 temp를 알파벳BST의 root로 설정
- 입력받은 char변수가 temp의 알파벳보다 작으면 순회포인터를 그 왼쪽자식으로 이동
- 크면 순회포인터를 오른쪽 자식으로 이동, 서로 같으면 해당 알파벳 노드를 반환.

24) AlphabetBST::APreorder
- 전위순회방식으로 driver와 workhorse가 나누어져 있다.
- 전위순회하면서 노드를 현재 노드를 Saveroot함수에 넣어준다.

25) AlphabetBST::;Preorder
- 전위순회 방식으로 driver와 workhorse가 나누어져 있다.
- 전위순회하면서 현재 알파벳 노드의 WordBST의 Print함수를 실행한다.
- 그 프린트 함수에는 R_PRE라는 문자열이 들어간다.

26) AlphabetBST::Inorder
- 중위순회 방식으로 driver와 workhorse로 나누어져 있음
- 중위순회를 하면서 현재 알파벳 노드의 WordBST의 Print함수를 실행한다.
- 그 프린트 함수에는 R_IN이라는 문자열이 들어간다.

27) AlphabetBST::Postorder
- 후위순회 방식으로 driver와 workhorse로 나누어져 있음
- 후위순회를 하면서 현재 알파벳 노드의 WordBST의 Print함수를 실행한다.
- 그 프린트 함수에는 R_POST라는 문자열이 들어간다.

28) AlphabetBST::I_PRE
- 전위순회 방식으로 스택을 이용해 알파벳 노드들의 단어들을 출력하는 방식
- 현재 노드의 wordbst의 print함수를 실행하고(I_PRE라는 문자열을 넣어줌) 스택에 현재 노드의 오른쪽 자식을 넣고 왼쪽자식을 넣는다.
- stack을 pop하고 pop한 노드의 오른쪽 자식, 왼쪽자식 순으로 stack에 저장
- 위의 두 단계를 반복문을 통해서 실행.

29) AlphabetBST::I_IN
- 중위순회 방식으로 스택을 이용하여 알파벳 노드들의 단어들을 출력하는 방식
- 반복문(현재노드)을 통해서 현재노드를 스택에 넣고 순회포인터를 그 왼쪽자식으로 이동
- 순회포인터(현재노드)가 stack의 최상부 노드(top)을 가리키게 하고 그것을 pop함.
- pop한 노드의 wordbst의 print함수를 실행하고(I_IN이라는 문자열을 받음) 순회포인터를

그 오른쪽 자식으로 이동시킴.

- 위의 내용을 무한루프 반복문으로 반복하는데 stack이 비어있으면 반복문을 종료함.

30) AlphabetBST::I_POST

- 후위순회 방식으로 스택을 이용하영 알파벳 노드들의 단어들을 출력하는 방식
- 이전 노드인 prevNode포인터가 null값이거나 순회포인터가 prevNode의 왼쪽, 오른쪽 자식인 경우 왼쪽자식이 있으면 왼쪽 자식을 넣고, 오른쪽 자식만 있으면 오른쪽 자식을 스택에 넣는다. leaf노드인 경우 그 노드의 WordBST의 단어들을 출력해준다.
- 만약 prevNode가 순회포인터의 왼쪽자식일 때, 순회포인터의 오른쪽 자식이 있으면 스택에 넣어주고, 없으면 그 노드의 단어들을 출력하고 pop을 실행함.
- 만약 prevNode가 순회포인터의 오른쪽자식일 때, 순회포인터의 단어들을 출력해준다.
- prevNode 포인터를 현재 순회포인터가 가리키는 노드로 이동시킨다.
- 위의 내용들은 stack이 빌 때까지 반복문으로 실행된다

31) AlphabetBST::I_LEVEL

- Queue를 이용하여 level별로 알파벳 노드들을 저장하고 그 단어들을 출력해줌.
- 자기자신의 단어들을 출력하고, 왼쪽 자식 오른쪽 자식순으로 queue에 저장
- pop실행해서 그 알파벳 노드의 단어들을 출력하고 그 자식들을 왼쪽, 오른쪽 순으로 저장.
- 위의 내용을 반복문으로 queue가 빌 때까지 실행시킴.

32) WordBST::WPostorder

- 소멸자에서 노드를 후위순회로 순회하면서 현재 순회포인터가 가리키는 노드의 동적할당을 해제하는 역할을 수행함. Driver, workhorse로 나누어져 있음.

33) WordBST::Insert

- AlphabetBST의 Insert함수와 동일한 방식으로 진행되지만 마지막에 순회포인터의 단어와 입력받은 노드의 단어를 비교하여 순회포인터가 가리키는 노드의 자식으로 설정함.

34) WordBST::low

- 반복문과 tolower 함수를 사용하여 입력받은 문자열을 배열에 모두 소문자로 저장한 뒤 그 배열을 반환해줌.

35) WordBST::Delete

- 반복문과 strcmp함수를 이용해 입력받은 문자열과 동일한 단어를 갖는 노드를 탐색
- 입력받은 문자열은 low함수를 통해서 모두 소문자로 바뀌어 배열에 저장됨.
- 만약 찾는 단어가 존재하지 않으면 null값을 반환함.
- 찾는 단어가 leaf노드라면 해당 노드와 부모 노드 간의 링크를 끊고 해당 노드를 반환한다.
- 찾는 노드가 오른쪽 자식만 갖는 경우에서 그 노드가 root이면 root 포인터가 그 오른쪽 자식을 가리키게 하고 노드의 오른쪽 링크를 끊은뒤 그 노드를 반환한다.
- root가 아닐 경우 이전노드의 오른쪽 링크를 순회포인터의 오른쪽 자식으로 하고 순회포인터의 오른쪽 링크를 끊은 뒤 그 노드를 반환한다.
- 찾는 노드가 왼쪽 자식만 가진 경우 반복문을 통해서 그 왼쪽 subtree에서 가장 큰 값을

가진 노드를 찾아서 링크를 끊는다.

- 잘라준 노드의 왼쪽 링크를 찾는 노드의 왼쪽 자식으로 연결하고, 찾는 노드의 왼쪽 링크를 끊는다. (만약 찾는 노드가 root면 root pointer가 잘라준 노드를 가리키게 한다.)
- 찾는 노드를 반환시켜준다.
- 만약 찾는 노드가 자식을 둘 다 가진 경우에 위의 방식과 동일하게 진행하여 왼쪽 subtree에서 가장 큰 값을 갖는 노드를 찾은 뒤 찾는 노드의 자리를 대체한다.
- 이때 찾는 노드가 root인 경우 root 포인터가 가장 큰 값을 가진 노드를 가리키게 한다.

36) WordBST::Search

- root가 null값을 가리키면 null을 반환함.
- 입력받은 문자열과 순회포인터가 가리키는 노드의 단어를 up함수를 통해서 각각의 배열에 저장하고 strcmp함수를 통해 비교하여 순회 포인터의 이동 루트를 지정.
- 서로 같으면 순회포인터가 가리키는 노드를 반환한다.

37) WordBST::Save

- memorizing _word.txt파일을 이어쓰기 모드로 엶.
- Saveroot 함수를 통해서 linked list형태로 약식 설정된 노드들을 순회하면서 순회포인터가 가리키는 노드들의 단어와 뜻을 파일에 씀.

38) WordBST::Print

- 문자열을 넘겨받는데 그 문자열과 order들을 비교하여 동일한 order로 wordnode들의 단어와 뜻을 출력해줌. order함수들은 AlphabetBST에서의 함수들과 거의 다르지 아니함.

**** WordBST의 order 함수들은 해당 노드들의 단어와 뜻을 출력해주는 역할을 함.

39) CLL::Insert

- 넘겨받은 노드를 링크에 추가하는 역할을 수행
- 노드가 없으면 입력받은 노드를 pHead로 설정하고 pNext가 자신을 가리키게함.
- 노드가 하나일 경우 pHead의 pNext를 입력받은 노드로 설정하고 노드가 pHead를 가리키게함.
- 그 외에는 순회포인터를 pHead의 전 노드로 반복문을 통해 이동시키고, 순회 포인터가 입력받은 노드를 가리키게 한 다음 입력받은 노드가 pHead를 가리키게 한다.

40) CLL::Print

- 반복문을 통해서 링크를 순회하는데 이때 순회포인터의 단어와 뜻을 출력한다.
- 출력 이후에 조건문을 통해 순회포인터의 다음이 pHead인지 확인하여 반복문을 탈출.

41) CLL::Search

- up함수를 통해서 입력받은 문자열과 순회포인터가 가리키는 노드의 단어를 각각 배열에 저장해준다. 순회하면서 동일한 값을 찾은경우 순회포인터를 반환해준다.
- 한 바퀴를 다 도는 동안에 찾지 못하였다면 null값을 반환시킨다.

42) CLL::Save

- memorized_word.txt파일을 쓰기모드로 엶.
- 반복문을 통해서 링크를 순회하면서 순회포인터가 가리키는 노드의 단어와 뜻을 연 파일에 쓰는데 조건문을 통해 순회포인터의 다음이 pHead인지 확인하여 반복문을 탈출

4. Result Screen

- 다음과 같은 명령어를 수행합니다.

```
LOAD
ADD
LOAD
PRINT MEMORIZING
PRINT MEMORIZED
MOVE 10
UPDATE liFe 삶
TEST life 인생
TEST liFe 삶
SEARCH liFE
PRINT MEMORIZED
PRINT MEMORIZING R_IN
SAVE
EXIT
```

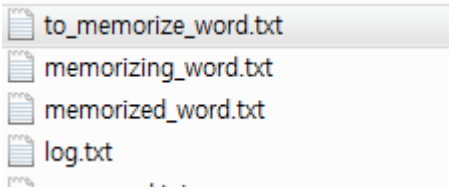
- 다음과 같은 결과를 보입니다.
- 처음 시작 3개 TXT파일이 존재하지 않으므로 에러 코드 100출력 중간에 이미 자료형이 존재하므로에러 코드 100을 출력합니다.
- 넘어간 자료나 TEST한 단어가 없으므로 에러코드 700을 각각 출력
- TEST에서 알맞은 에러코드를 출력해주었고 PASS하고 나서 정상적으로 CLL에 넘어간 것을 확인할 수 있습니다.
- 10개의 단어가 BST로 넘어왔고 1개를 TEST하여 PASS하였으므로 9개가 남은 것을 확인할 수 있습니다. INORDER방식으로 출력되었습니다.

```
========ERROR========
100
======================
========ADD========
Success
====================
========ERROR========
100
======================
======== PRINT ========
700
======================
======== PRINT ========
700
======================
========MOVE=========
Success
======================
========UPDATE========
life 인생 -> 삶
======================
========ERROR========
500
======================
========TEST========
 PASS
====================
========SEARCH========
life 삶
======================
```

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

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

- 정상적으로 파일들이 저장되었습니다.



5. Consideration

<2013722079 김영근>

먼저 프로젝트를 처음 받고 이 것을 어떻게 구현해야 할지 생각했다. 이진 탐색 트리를 이용한 영어 단어장을 구현 할 것을 요구했다. 크게 Queue, Binary Search Tree, Circular Linked List로 구성 되어 있었으며 BST안에 또 다른 BST를 구현하여 이중 트리를 구현하도록 요구되었다. 전체적인 틀이 잡혀 있었기 때문에 각자 파트를 나눠 함수들을 구현하고 그 함수들을 다시 합쳐 컴파일 해보고 오류를 잡으며 협력하여 프로젝트를 구현하였다. 마지막으로 비주얼로 작성한 코드를 리눅스에서 돌려서 실행이 되나 확인 해야 했는데 이때 까지는 리눅스를 사용한적이 없었기 때문에 리눅스 사용법을 이해하고 어떤 코드는 리눅스에서 호환이 안되어 오류가 뜨는지 알게 되었다. 이번학기 처음 하는 프로젝트였는데 지금까지는 개인으로 프로젝트를 만들고 제출하는 것 이었지만 이번은 팀프로젝트여서 많이 낯설었던 거 같다. 하지만 서로 협동하여 프로젝트를 만든다는 것이 신선했으며 한편으로는 만약 서로 맞지 않으면 프로젝트를 진행하는데 많은 어려움이 있을 수 있겠다는 생각을 하였다. 생각해 보면 졸업을 하고 사회로 나가 회사를 다니면 지금처럼 프로 프로젝트를 할 때 서로 파트를 나누어 구현하고 이를 합쳐서 하나의 프로젝트를 완성 시킬 것이기 때문에 이번 기회가 나중에 좋은 경험으로 남아 나중에 낯설지 않게 서로 도와가며 프로젝트를 만들 수 있을 것 같다.

<2013722095 최재은>

프로젝트에 처음 착수하고 나서 제일 먼저 한 일은 일단 쉬운 부분을 끝내 놓자 였습니다. 김영근 학생이 Queue를 구현하고 제가 CLL을 맡아서 구현하였습니다. 이후 가장 큰 파트를 차지하는 BST를 구현하는 부분에 있어서 서로 임무분배를 하는 것이 꽤나 고난이었던 것 같습니다. 서로 간략하게 나마 나눈 부분을 수행하면서 먼저 끝난 사람이 선행적으로 필요한 함수를 구현하는 것을 서로 반복하다 보니 조금씩 코딩 스타일이 달라 이를 분석하는 데에 시간이 또 소요된다는 것을 배웠습니다. 처음 해보는 팀단위 과제였는데, 고난이 있었던 만큼 서로에게 많은 이득을 주었다고 생각합니다. 해당 프로그램을 진행하면서 가장 고통스러웠던 부분은 리눅스였는데, 비주얼 스튜디오에서 되는 것이 리눅스에서 안되니 참 난해할 뿐이었습니다. 처음 써보는 프로그램인데다가 비주얼 스튜디오처럼 디버깅을 한줄한줄 할 수가 없어 좀 힘들었던 것 같습니다. 막힌 부분을 찾으려고 대강 짐작가는 곳에 cout을 해가면서 막힌 부분을 풀어나갔습니다. 그래도 막상 나중이 되어보니 나름 괜찮은 프로그램인 듯 합니다. 실무에서는 리눅스를 주로 쓴다고 하는데 잘 익혀나가야겠습니다. 이번 프로젝트에서 팀으로 과제를 수행하면서 배운 나름의 노하우를 사회에 나가서도 잘 써먹을 수 있었으면 좋겠습니다. 보고서에서 김영근 학생이 출력과 FLOW CHART를 작성하였고, 제가 나머지를 수행하였습니다.

6. 기여도

최재은 : 60%

김영근 : 40%