Data Structure

project 1

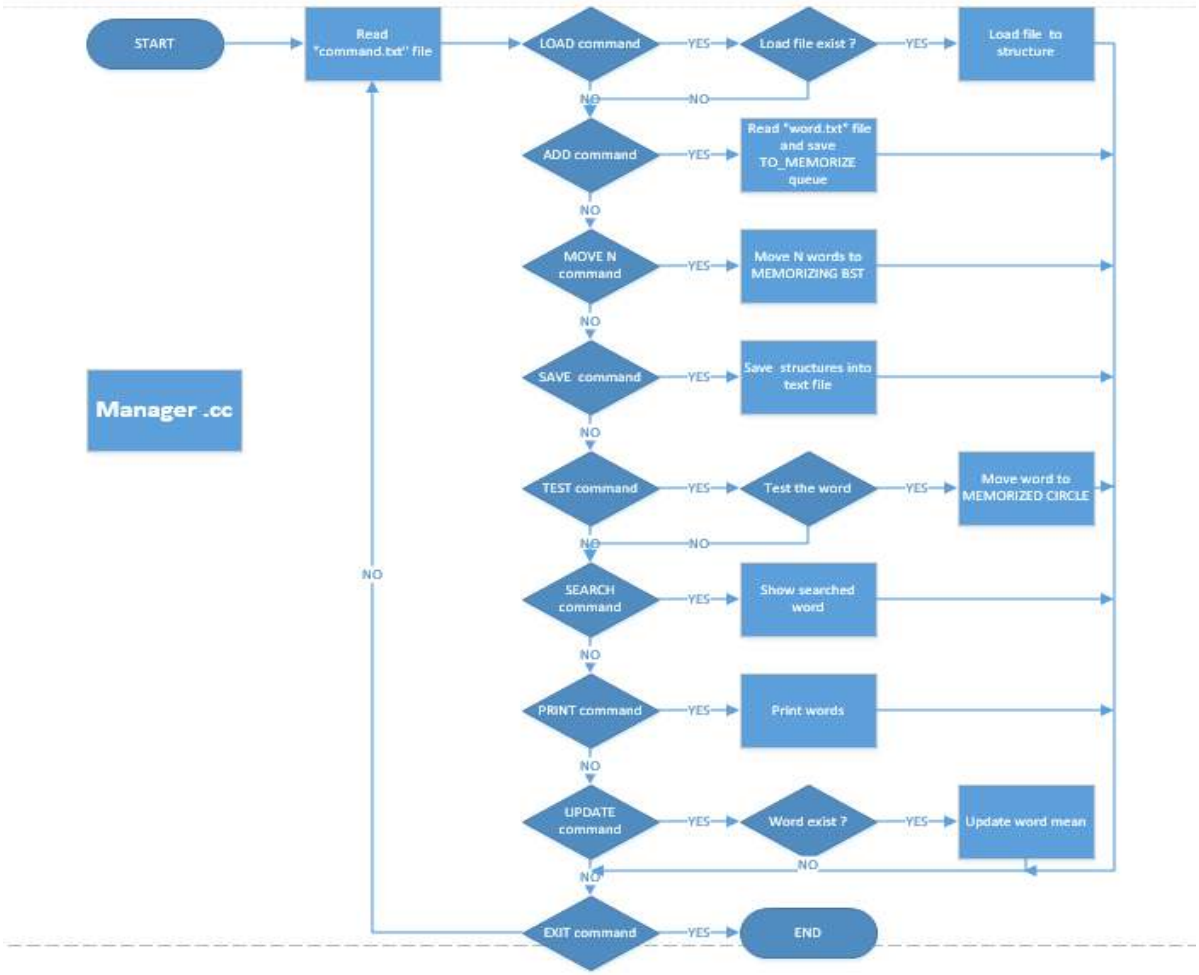
Professor	이기훈 교수님
Department	Computer engineering
TEAM ID	2013722082 이형민
	2013722068 김동현
	2013722080 김동민
DATE	16.10.07(금)

1. introduction

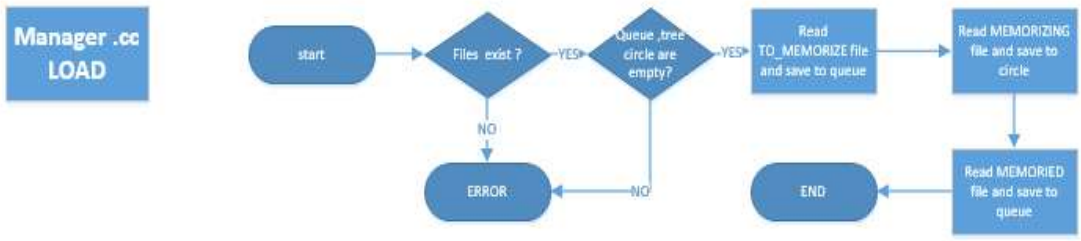
이 프로그램은 3개의 단어장 to_memorize, memorizing, memorized를 구현하는 프로그램입니다. to_memorize는 외울 단어들이 있는 단어장으로 ADD 하여 단어를 Queue 넣습니다.
memorizing은 외우고 있는 단어장으로 MOVE n 하여 to_memorize에 있는 단어를 옮깁니다. 이때 memorizing의 최대 개수는 100개로 100개를 넘으면 error code를 출력합니다.
memorizing은 알파벳 BST로 이루어져 있고 각 BST의 알파벳은 word BST를 가지고 있습니다. memorized는 외운 단어장으로 memorized에 있는 단어는 memorizing에 있는 단어중 TEST 하여 맞으면 memorized로 옮기고 circular Linked List로 구현됩니다. 3개의 단어장은 save하여 text파일로 저장되며 load하여 다시 불러올 수 있습니다.
각 단어장에 있는 단어들을 search, print, update할수 있으며 search는 대소문자 구분없이 단어를 검색할 수 있고 print는 to_memorize와 memorized는 head부터 연결된 순서로 하며
memorizing은 재귀함수를 이용한 pre, in, post의 형태와 재귀함수를 사용하지 않은 pre in post level order로 출력할 수 있습니다. update는 단어의 의미를 수정할 수 있습니다.
마지막으로 exit를 통해 동적할당 받은 메모리들을 반환하여 주고 종료합니다.
모든 command는 command.txt 파일으로 실행되고 그 출력 결과들은 log.txt에 저장됩니다.

2. Flow chart

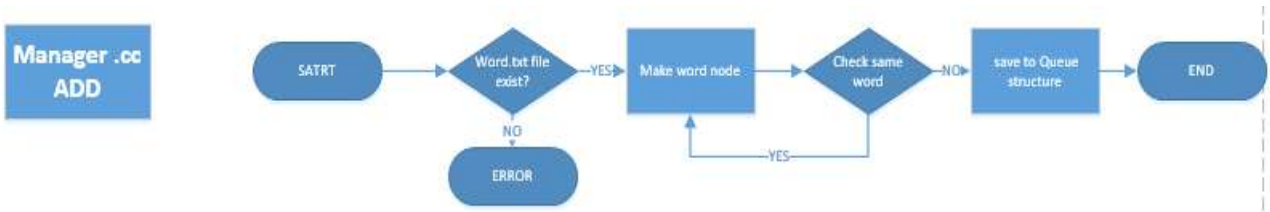
- manager.cc



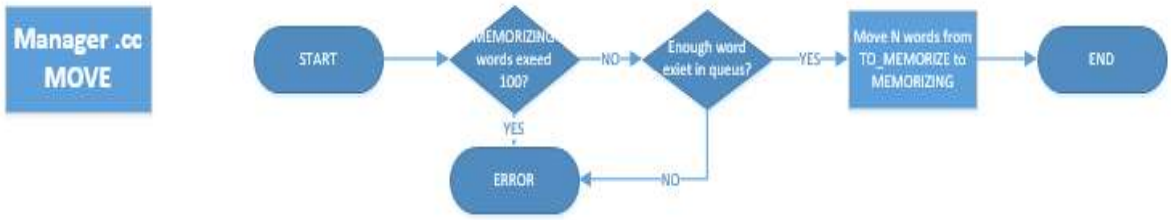
- LOAD



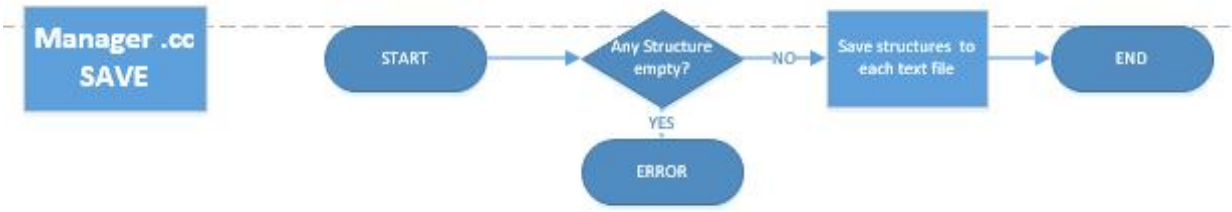
- ADD



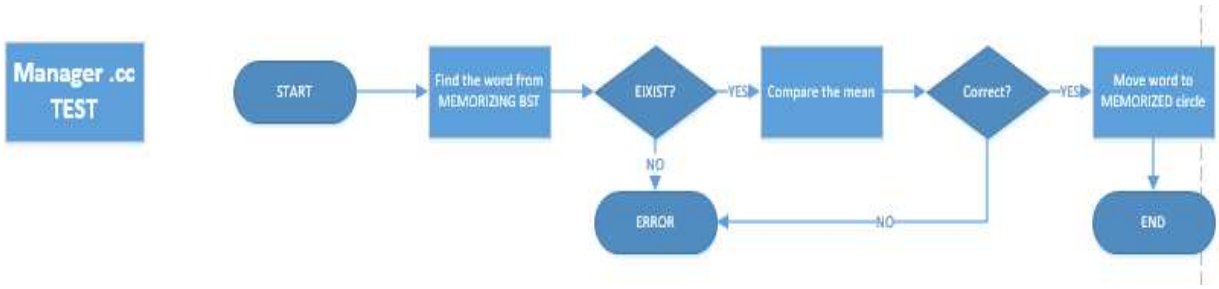
- MOVE



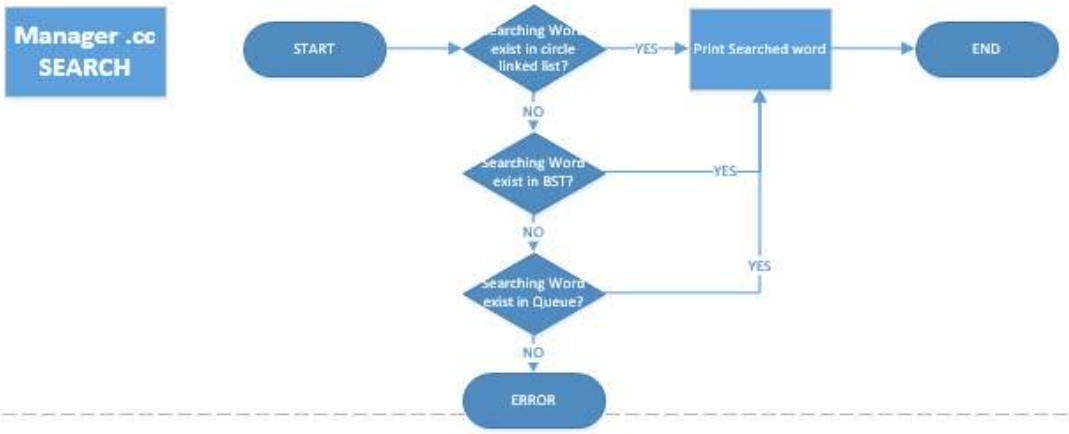
- SAVE



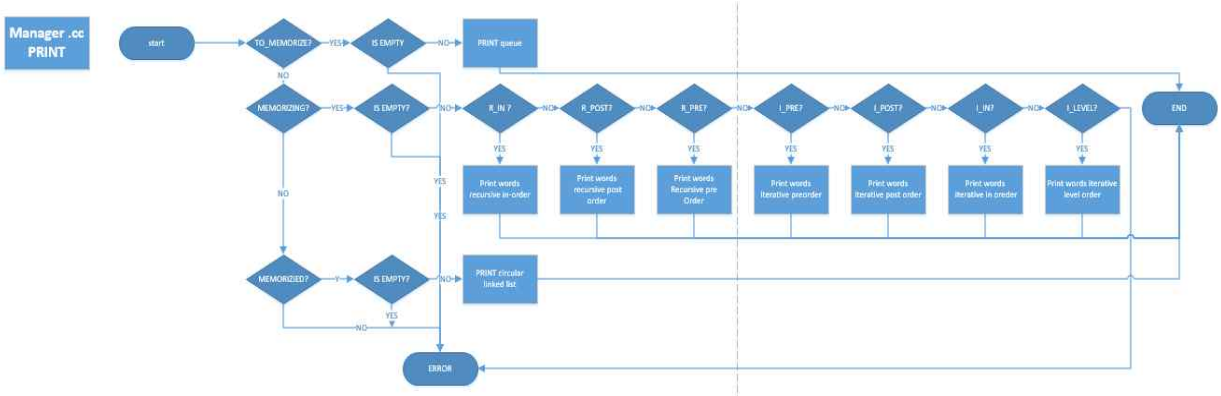
- TEST



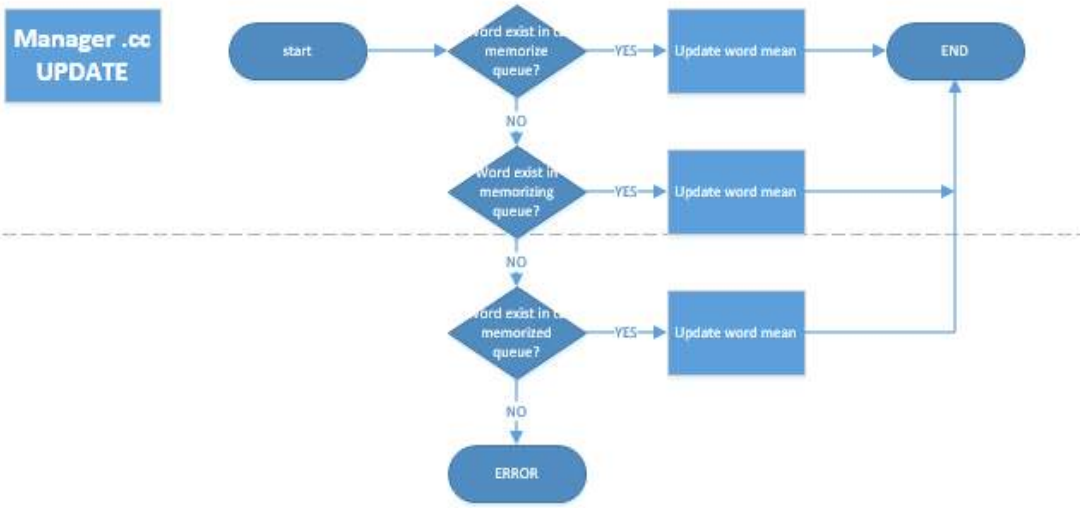
- SEARCH



- PRINT



- UPDATE



3. Algorithm

1. queue

- ~queue()
처음 node인 phead부터 그다음 순서의 node를 가르키는 성질을 가진 queue의 성질을 이용하여 임시 WordNode* 변수 temp와 del을 사용하여 phead의 위치부터 시작 하여 temp = temp->GetNext(); 의 표현을 사용하여 temp를 전진시키고 temp자리에 있던 del을 temp가 NULL이 아닐 때 까지 지우는 방식으로 queue가 사라질 때 실행되는 소멸자를 구현했습니다.

- push
queue에서 값을 insert하는 push의 경우 queue 내에 node가 하나도 존재하지 않을 경우 phead를 새로 받은 node로 지정한뒤 return하지만 그렇지 않을 경우에는 WordNode* 형 변수 temp를 사용하여 temp가 queue내의 node를 GetNext()를 사용하며 돌면서 제일 마지막 node에 temp가 도착 했다면 temp의 SetNext를 노드로 설정을 해주어 마지막node에 잇는 temp의 다음 노드로 새로들어온 노드가 가리켜 질 수 있도록 했습니다. temp가 한칸씩 노드를 거쳐가면서 같은 단어를 가르키는 단어가 있다면 새로들어온 노드를 delete해주고 return하는 형식으로 queue push를 구현 했습니다.

- pop
queue에서 delete에 해당하는 pop의 경우 FIFO의 queue형식을 구현했습니다. 제일 먼저 들어 온 노드가 phead에 해당하고 제일 마지막에 들어온 노드가 제일 끝부분에 해당 하는 노드이므로 WordNode*형 변수 temp를 사용하여 phead의 위치를 GetNext를 사용하여 한칸 옮긴뒤 phead자리에 있는 temp가 가르키는 SetNext의 값을 초기화 시킨뒤 return temp의 표현으로 넘겨줄 수 있도록 설계했습니다.

-Search
queue의 원하는 단어를 받아서 그 단어를 가지고 있는 node를 search하는 기본원리를 가지고 있습니다. 하지만 알파벳이 대문자가 들어올 경우와 소문자가 들어올 경우서로 비교할 수 있는 값이 다르기 때문에 영단어를 무조건 대문자로 바꿔주는 GetBigWord라는 함수를 구현했습니다. 배열에 저장된 단어를 기본함수인 'toupper'를 사용하여 소문자를 대문자로 바꿔주는 함수를 기본적으로 구현한뒤 사용했습니다. 다시 search로 돌아가서 search할 word의 값을 strcpy를 사용하여 pword로 복사한 뒤 WordNode* 형 변수 temp가 phead부터 시작 해서 temp가 NULL값을 가질 때 까지 한칸씩 전진하면서 새로 도착한 노드에 word의 값을 pgetword라는 배열에 strcpy를 사용하여 새로 저장하고 아까 저장한 pword와 pgetword를 구현한 GetBigWord를 사용하여 대문자로 바꿔준 값을 비교했습니다. 만약 찾는 값이 나왔다면 현재 temp노드를 return temp하고 temp가 NULL로 갈 때 까지 못 찾았다면 return 0;로써 함수를 탈출했습니다.

-print
queue에서 print를 구현하기 위해서 WordNode* 형 변수 temp를 사용하여 phead자리부터 시작하여 temp가 다음 노드로 한칸씩 전진하면서 노드에 저장된 word와 mean을 Get함수를 사용하여 temp가 NULL이 아닐 때 까지 cout으로 word 와 mean을 출력해주는 방식으로 구현했습니다. 하지만 queue에 node가 없을 경우도 있으므로 맨처음 부분에 조건을 두어 pHead에 값이 없다면 return0 사용하여 탈출하는 방식으로 설계했습니다.

-save
queue에서의 save의 경우 queue에 존재하는 노드가 있다면 파일을 열어서 거기에 해당되는 node의 word와 mean을 파일에 저장하는 방식이므로 "to_memorize_word"라는 파일을 먼저 열고 WordNode* 형 변수 temp를 사용하여 temp가 phead의 자리부터 시작해서 NULL이 아닐때까지 거치는 모든 노드의 word 와 mean을 Get함수를 통해 가져오고 그 값을 open한 파일에 저장하고 저장을 다했다면 다시 file을 닫는 방식으로 구현 했습니다.

2. circular linked list
- ~CircularLinkedList
circularlinkedlist에서의 소멸자의 경우에는 마지막의 노드가 다시 phead를 가르키고 있기 때문에 그분분을 유의하면서 설계했습니다. WordNode* 형 변수 temp와 del을 사용하여 circurlarlinked list안에 아무런 node가 없다면 return0로써 함수를 나가지만 노드가 있다면 queue의 소멸자에서 구현했던 것처럼 phead 노드에 현재 위치한 temp와 del중 temp를 한칸전진한뒤 del의 값을 delete해주고 반복하는 형식입니다. 하지만 위에서 언급한대로 마지막 node의 GetNext가 다시 phead를 가르키고 잇으므로 temp가 마지막 node에 까지 도착했다면(phead==0 || temp->GetNext() == phead)이란 조건을 통하여 마지막 node에 도착한 temp를 구별하고 그 temp를 지우고 break으로 빠져나오는 방식으로 구현했습니다.

-insert
circularlinkedlist의 insert의 경우 queue의 push와 상당히 유사한 부분을 가지고 있습니다. 먼저 새롭게 들어올 node가 원래 있던 노드와 같을 경우 새롭게 들어올 node를 delete한뒤 return 합니다. cll에 node가 하나도 없었던 경우 새롭게 들어올 노드를 phead로 지정한 뒤 SetNext를 통하여 다시 자기자신을 가르키게 설정합니다. cll에 하나 이상의 노드가 있을 경우에는 WordNode* 형 변수 temp를 사용하여 temp를 제일 마지막 노드의 위치로 반복문을 사용하여 옮긴뒤 cll의 성질을 가질 수 있도록 지금까지 마지막 노드였던 temp의 SetNext를 새로들어온 노드로 수정하고 새로들어온 노드의 SetNext를 phead로 하여 cll의 성질을 가질 수 있도록 설계했습니다.

- search
cll의 search의 경우 queue와 같은 방식으로 구현하면 계속해서 무한루프를 돌기 때문에 염두해두면서 설계했습니다. WordNode*형 변수 temp를 사용했습니다. 서치를 수행할 때 cll에 노드가 하나도 없다면 return 0을 통하여 탈출했지만 아니라면 queue서치와 유사한 방식을 사용했습니다. do while문을 사용하여 temp가 cll을 한바퀴 돌아서 다시 phead로 오면 반복문을 탈출 할 수 있도록 설계하고 temp는 반복적으로 노드를 하나씩 거치면서 pword에 저장된 word(strcpy사용)와 pgetword에 저장된 노드의 word를 GetBigword를 통하여 대문자로 바뀐 값을 비교하여 같다면 현재temp노드를 return하는 방식으로 구현했습니다. 만약 찾는 값이 없다면 return 0를 통하여 탈출합니다.

-print
cll의 print구현의 경우 cll에 있는 노드가 없다면 return 0로써 탈출하고 1개 이상의 노드가 있다면 WordNode* 형 변수 temp를 사용하여 search와 같은 반복문 do while을 사용하여 temp가 cll을 한바퀴(temp!=phead조건 사용) 돌면서 한칸한칸 모든 node들에 방문하여 GetWord와 GetMean을 사용하여 출력하는 방식으로 print를 구현했 습니다.

-save
cll에 있는 노드의 word와 mean의 값을 저장하는 save의 경우 cll에 아무런 노드가 없다면 return 0로써 탈출하고 1개 이상의 노드가 있다면 WordNode* 형 변수 temp를 선언한 뒤 저장하려는 파일 "memorized_word.txt"를 연 뒤 서치와 같은 temp 가 cll를 한바퀴만 돌수 있도록 do while(temp != phead)문을 설계하여 temp가 노드 하나하나를 거치면서 각 노드의 word 와 mean을 Get함수를 통하여 open한 파일에 저장할 수 있도록 설계한뒤 파일을 close해주는 방식으로 설계했습니다.

3. WordBST
- WordNode에서는 노드안에 속하는 word와mean을 둘다 담아두기 위해서 자료형 pair를 사용했습니다. pair<char*,char*> wordmean;으로 선언을 하여 wordmean에 char* 형 변수 단어와 단어뜻이 들어 올수 있도록 선언을 해준 동적 할당을 통하여 word, mean이 들어갈 공간을 만들어 주었습니다.

- ~WordBST()
WordBST에서의 소멸자를 구현하기 위해서 재귀함수의 성질을 가지는 postorder_delete함수를 만들어서 사용했습니다. postorder_delete함수는 전달인자로 BST의 root를 전달받아 BST의 root부터 시작하여 리프부터 시작해서 root로 거꾸로 올라오는 성질을 가지는 후위연산을 이용하여 delete를 시작하여 마지막으로 root를 지워주는 방식의 성질을 가지는 postorder_delete함수로써 소멸자를 구현했습니다.

- postorder_delete
postorder_delete함수는 BST의 소멸자를 실행하기위해 생성했습니다. 재귀함수의 성질을 가져서 BST의 왼쪽 제일 끝 리프까지 이동하기위해 계속해서 자신의 함수를 recall하고 리프에서 부터시작 하여 마지막엔 root까지 이동하면서 delete cuurentNode를 통하여 현재 노드를 지워주는 알고리즘을 가지고 소멸자에 이용될 수 있도록 설계했습니다.

- insert
BST에서의 insert는 BST에 노드가 하나도 있지 않을 경우 새로들어온 노드가 root가 되게 해주고 return 시키지만 1개 이상의 노드를 가지고 있을 경우에는 WordNode*형 변수 temp(root의 값을 가지는)를 사용하여 strcmp함수를 가지고 temp와 node를 GetWord했을 때의 값을 비교하여 node가 크다면 오른쪽 node가 작다면temp를 왼쪽으로 이동시킵니다. 그래서 리프와의 비교가 끝났다면 node의 값이 더 작았을 겨우 temp의 SetLeft의 값으로 node를 주고 node의 값이 더 컸을 경우 temp의 SetRight의 값으로 node를 설정한뒤return 으로 탈출합니다. temp가 리프로 값을 비교하며 이동하는 도중 같은 단어를 가지는 node를 찾았다면 temp 노드의 mean을 새롭게 들어올 node의 mean으로 업데이트해준뒤 탈출 할 수있도록 설계했습니다.

- delete
BST에서의 delete의 경우 지워줄 값을 찾기위한 서치의 과정을 먼저 수행합니다. Wordnode*형 변수 temp와 ptemp를 사용해서 strcmp를 이용해 temp가 NULL이 아니고 찾을 word를 찾지 못햇다면 반복하는 반복문을 사용하여 temp의 word와 찾으려는 word를 비교하여 계속해서 값을 비교하면서 한단계씩 내려가는 서치의 과정을 수행합니다. ptemp의 경우 temp의 부모노드를 가지고 있을 수 있도록 값을 정해줍니다. BST에 단어가 없거나 찾는 단어가 존재하지 않을 경우 return 0를 사용하여 탈출하고 그러지 않다면 delete할 노드의 자식노드의 개수를 판단 합니다. 먼저 delete할 노드가 리프노드 일 경우 지우려는 노드가 root인 경우 root 에 NULL값을 넣어 주어 값을 지워주고 ptemp의 어느 쪽 자식노드가 temp인지를 모르므로 어느쪽 자식이 temp인지를 판단한 후에 ptemp가 가르키고 있었던 연결 고리를 Set을 이용해 끊어 준뒤 return temp를 통하여 지우려는 temp를 반환합니다. temp가 왼쪽 자식 노드만 가지고 있을 경우 왼쪽 자식 노드중 가장 큰 값을 가지는 노드로 대체한뒤 return temp를 수행해야 합니다. 따라서 WordNode*형 변수 prevprev,prev,cuur을 사용합니다. 먼저 지우려는 노드가 왼쪽 자식만 가지는 root일 수 있으므로 root의 값을 temp의 Getleft의 로 설정 해주고 return temp를 합니다. 그렇지 않을 경우 변수 curr을 사용하여 temp의 왼쪽 노드중 가장 큰값을 가지는 값을 찾기위서 계속해서 오른쪽 으로 이동을 하게되는 반복문을 거치게 되고 반복문이 종료 될 때는 curr에 NULL값을 가지고 있고 실제적으로 대체되는 노드는 prev에 있게 됩니다(curr의 이전값이prev이므로). 대체할 노드를 prev로 가지는 과정이 끝났다면 연결고리를 수정해주는 작업을 거치게 됩니다. 먼저ptemp의 왼쪽 자식이 temp인지 오른쪽 자식이 temp인지를 판단한 후에 prev가 가르켯던 왼쪽 자식을 prevprev가 가르키게 한 뒤, prev를 temp가 가르키는 것을 가르키게 하고 temp가 가르키던 연결고리를 NULL로 초기화 해주는 방식으로 구현했습니다. temp가 오른쪽 자식만 가르키고 있을 경우는 temp 오른쪽 서브트리에서 제일 작은 값을 가지는 노드로 대

체를 해야 합니다. 따라서 아까 왼쪽 자식만을 가지고 잇는 노드를 지울 때 curr이 temp의 오른쪽 서브트리에서 제일 작은 값을 가지는 노드를 가질 수 있도록 왼쪽으로 이동시켜주는 것과 prev의 오른쪽 자식을 prevprev가 가르키게 해주는 것만 뺀다면 동일한 동작을 수행합니다. 자식 노드를 2개 가질 때 왼쪽 자식만 가질 때 와 동일하게 temp의왼쪽 서브트리에서 가장 큰값을 가지는 노드로 대체를 해주어야 합니다. 따라서 동일하게 수행을 하다가 prev를 temp가 가르켰던 노드와 연결 시켜줄 때 prev가 가르키던 노드를 prevprev가 가르키고 temp의 왼쪽 과 오른쪽 노드 모두를 prev가 가르키고 temp의 연결을 끊어주고 return temp를 하는 방식으로 BST delete를 구현했습니다.

-search
BST에서의 서치는 찾으려는 word를 전달받아서 WordNode* 형 변수 temp를 사용하여 root에서부터 strcpy를 사용하여 찾으려는 word를 pword로 , temp노드의 word를 pgetword로 복사한뒤 GetBigWord함수를 사용하여 값을 비교해주면서 temp 가 한단계씩 내려가면서 서치를 수행합니다. 찾으려는 값이 나왔을 경우 return temp를 통해 temp를 반환 해주고 끝나지만 아닐 경우 return 0로써 함수가 종료됩니다.

- inorder
BST에서의 inorder는 inorder방식을 사용하여 값을 출력해주기 위해서 구현합니다. inorder방식으로 값을 출력해주기 위해서 재귀적으로 자기자신을 recall하여 왼쪽서브트리의 리프부터 출력을 시작해서 출력하고 다시 왼쪽서브트리의 오른쪽을 출력을 시작해서 root까지 하고 오른쪽 서브트리의 왼쪽부터 시작해서 오른쪽 까지 출력하는 방식으로 출력이 진행 되게 됩니다.

- preorder
BST에서의 preorder는 preorder방식을 사용한 값의print를 위해서 구현합니다. preorder방식으로 값을 print해주기 위해서 자기자신의 함수를 다시 호출하는 재귀함수의 성질을 이용하여 root부터 시작하여 왼쪽 서브트리를 내려가면서 print하고 그다음 오른쪽 서브트리를 내려가면서 출력하는 방식으로 재귀적인 출력과정을 거치게 됩니다.

- postorder
BST에서의 postorder는 postorder방식을 사용한 값의 print를 위해서 구현합니다. postorder방식으로 값을 print해주기 위해서는 자기자신의 함수를 다시 홀출하는 재귀함수의 성질을 이용하여 root의 왼쪽 서브트리의 리프부터 시작하여 위로 올라가고 root를 재외하고 다시 오른쪽 서브트리의 리프부터 시작하여 위로 올라온뒤 마지막 으로 root를 출력해주는 방식으로 재귀적인 출력과정을 거치게 됩니다.

- nonrec_inorder
BST에서의 nonerec_inorder는 재귀함수를 사용하지 않고 inorder 출력을 나타내야 합니다. 재귀함수를 사용하지 않고 inorder의 출력을 나타내기 위해 스택함수를 사용했습니다. 앞의 queue는 FIFO의 성질을 가지지만 stack의 경우 LIFO의 성질을 가지고 insert에 해당하는 push와 delete에 해당하는 pop을 가진 stack 함수를 먼저 구현했습니다. 스택을 사용하여 root의 왼쪽 서브트리의 왼쪽 리프부터 위로 올라가고 root를 출력한뒤 root의 오른쪽 서브트리의 왼쪽 리프부터 위로 올라가는 출력형태를 띄는 nonrec_inorder를 구현했습니다.

- nonrec_postorder
BST에서의 nonrec_postder는 재귀함수를 사용하지 않고 postorder의 출력을 나타내기 위해 2개의 스택을 사용하여 구현했습니다. s1,s2라는 2개의 stack이 있다고 할 때 root에서부터 시작해서 s1에 각각의 노드를 s2에서 pop했을 때 post_order의 순서가 나올 수 있도록 s1에서 push와 pop을 실시하여서 출력때 s2에서 pop한 순서대로 출력을 실시 하였을 때 postorder의 출력결과가 나올 수 있도록 설계했습니다.

- nonrec_preorder
BST에서의 nonrec_preorder는 재귀함수를 사용하지 않고 preorder를 구현하기 위해서 1개의 스택함수를 사용하여 구현했습니다. 스택에서의 insert에 해당하는 push를 하기전에 먼저 temp의 출력을 실시하여 preorder의 출력때와 같이 root부터 출력을 실시하고 왼쪽 아래 서브트리로 내려가면서 출력을 실시하고 다음 오른쪽 서브트리를 출력하면서 내려가는 preorder와 같은 출력결과를 스택함수를 이용하여 구현했습니다.

- level_order
BST에서의 재귀함수를 사용하지 않고 level order를 구현하기 위해서 앞의 구현했던 queue와 동일한 push와 pop FIFO의 성질을 가지는 queue함수를 구현했습니다. root부터 시작하여 각각의level별로 출력을 실시하는 levelorder의 성질을 구현하기 위해 root를 먼저 출력한 뒤 root의 자식들을 queue에 넣어주고 root의 왼쪽 자식노드를 pop하고 그 자식노드의 자식들을 push로 넣고 root의 오른쪽 자식노드를 pop하고 자식노드를 push로 넣고 pop해주면서 출력하는 방식을 사용하여 root부터 시작하여 level로 출력을 실시해주는 levelorder의 출력을 실시할 수 있게 구현했습니다.

- save
skewed tree로 저장되는 것을 막아주기 위해 preorder의 동작을 활용한 preorder_save라는 함수를 구현했습니다. preorder의 동작을 활용한다면 파일에 출력할 때 skewed tree로 저장되는 것을 막을 수 있기 때문에 재귀함수를 활용한 preorder에 파일에 저장되는 부분을 넣어주어서 preorder로 이동하면서 노드의 정보가 저장될 수 있도록 preoder_save를 이용한 save함수를 구현했습니다.

4. alphabetBST
- AlphabetBST()
AlphabetBST에서의 생성자는 각각의 26개의 알파벳들이 들어갈 수 있는 노드가 위치할 공간을 만들어 주기 위해서 25번의 동적할당하는 과정을 통해 공간을 만들고 새롭게 만든 node에 Set함수를 통해 알파벳을 넣어준 뒤 그 노드를 alphabetBST의 insert에 넣어주므로써 알파벳 BST tree를 생성자에서 생성해줄 수 있도록 설계했습니다.

- ~AlphabetBST()
AlphabetBST에서의 소멸자는 postorder방식의 출력법을 사용하여 postorder _delete라는 함수를 구현하여 사용했습니다. postorder _delete함수는 postorder와 상당히 유사하지만 출력이 아닌 delete의 목적을 두고 있기 때문에 자기자신의 함수를 계속 해서 부르면서 root의 왼쪽 서브트리의 리프부터 위로 올라가면서 delete하고 root의 오른쪽 서브트리의 리프부터 위로 올라가면서 delete한뒤 마지막으로 root를 지워 주므로써 마지막까지 필요한 node를 찾기 위한 link를 끝내지 않고 성공적으로 소멸자의 역할을 수행할 수 있도록 구현했습니다.

- insert
AlphabetBST에서의 insert의 경우 wordBST에서의 insert와 기본적 알고리즘이 같습니다. 생성자를 통해 insert된 각각의 알파벳들을 처음 들어온 알파벳 "P"에 해당되는 알파벳을 root로 만들고 WordNode* 형변수 temp를 사용하여 root에서부터 시작하여 GetAlphabet함수를 사용하여 가져온 알파벳의 비교를 통해 점점 리프쪽으로 이동을 하게되고 리프에서의 비교가 끝났다면 temp의 Set함수를 통해 새로들어올 node의 알벳이 더 작다면 temp의 왼쪽에, 더크다면 temp의 오른쪽에 이어주는 방식 으로 구현을 하였고 만약 같은 같은 알파벳이 들어왔다면 temp 의 SetAlphabet으로 새로들어올 노드가 가진 Alphabet을 Get을 통해 너어주는 방식으로 설계했습니다.

- print(+여러가지 print의 방식)
AlphabetBST에서의 print의 경우 print방식에 해당하는 order를 넘겨받아서 strcmp함수를 통해 order와 print방식이 같은 경우에 각각에 print방식에 맞는 조건문으로 들어가서 print함수로 order와 root를 넘겨주는 형태로 구현했습니다. 여러 가지 print의 방식은 wordBST에서의 여러 가지 print방식과 같은 알고리즘을 가지고 있습니다. 하지만 AlphabetBST에서의 print는 알파벳들을 출력하는 것이 아닌 해당 알파벳의 subBST에 속하는 wordBST의 print를 출력해야 하므로 wordBST에서의 여러 가지 print함수의 cout으로 출력했던 라인에 GetBST를 통하여 wordBST에 접근한뒤 wordBST에서의 print함수로 접근한뒤 order를 전해주는 방식으로 AlphabetBST에서의 여러 가지 print함수들을 설계하여 넘겨받은 order를 wordBST에서 그러한 방식으로 사용될 수 있도록 했고 Alphabet BST와 wordBST가 같은 방식으로 출력되는 결과물을 확인 할 수 있도록 설계했습니다.

- search
AlphabetBST에서의 search의 경우 원하는 alphabet을 건네 받으면 AlphabetBST를 WordNode* 형 변수 temp를 통해 값을 비교해가면서 한칸 한칸씩 내려가면서 찾는 알파벳을 탐색하고 찾았다면 return temp를 통해서 건네주는 역할을 수행할 수 있도록 설계했습니다. 건네 받는 알파벳의 경우 소문자가 나올 수 있고 대문자가 나올 수도 있기 때문에 아스키코드의 값 중 소문자와 대문자 값의 차이를 이용하여 소문자를 대문자로 바꿔 줄 수 있도록 설계했습니다.

- save
AlphabetBST에서의 save함수는 Inorder_save함수를 사용하여 구현했습니다. AlphabetBST의 save의 경우 AlphabetBST의 subBST에 속하는 wordBST의 값을 열린 파일에 저장할 수 있도록 수행해야하므로 GetBST함수를 사용하여 subBST의 저장을 가르킬 수 있도록 하였고 저장 방식으 inorder 출력방식을 사용하여 inorder의 방식대로 노드를 방문하면서 출력 대신 subBST의 저장을 실시하는 방식으로 inorder_save를 구현하였고 save에서는 구현된 inorder_save를 활용하여 구현했습니다.

- postdorder_delete
소멸자에서 설명을 했으므로 생략하겠습니다.

- WordCnt
AlphabetBST에서의 getwordcnt는 wordBST의 insert로 들어간 단어의 개수를 확인하기 위해서 생성했습니다. BST안의 단어의 개수가 총 100개 넘게 들어가면 안되므로 wordBST에서의 wordcnt를 활용하기 위해 return Wordcnt를 가지는 get함수와 wordcnt의 값을 정해주는 set을 사용하여 구현했습니다.

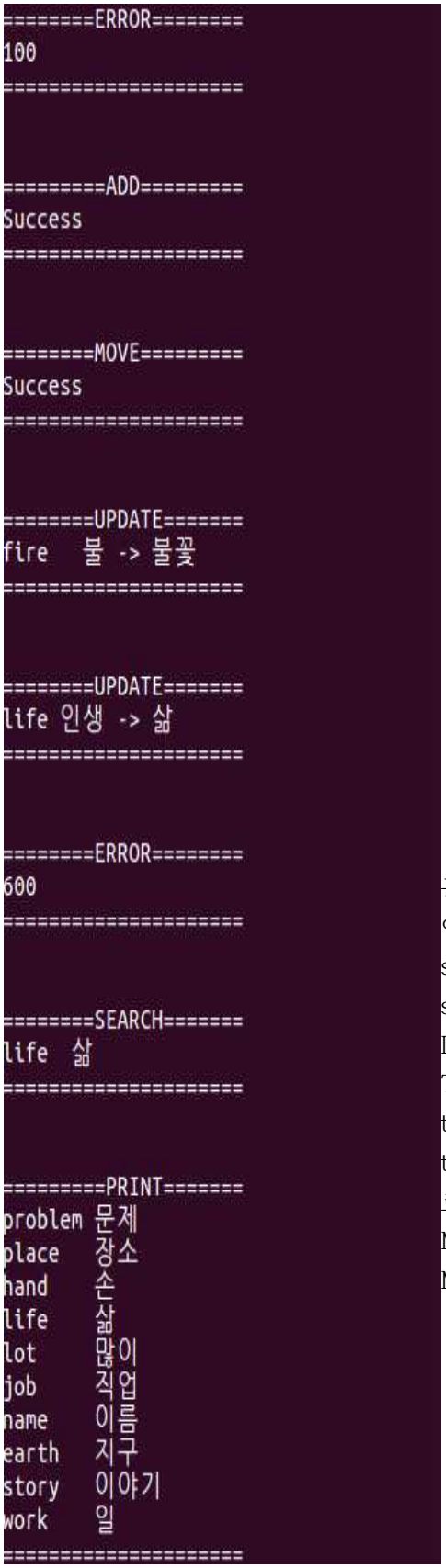
(뒷장에서 계속됩니다.)

4. Result scream

```
dongmin@ubuntu:~/project1$ ./wordbook
========ERROR========
100
====================


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


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


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


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


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


=========PRINT=======
earth   지구
hand    손
job     직업
life    인생
lot     많이
name    이름
place   장소
problem 문제
story   이야기
work    일
========ERROR========
400
====================
```

```
LOAD
ADD
MOVE 10
PRINT MEMORIZING R_PRE
PRINT MEMORIZING I_PRE
PRINT MEMORIZING R_IN
PRINT MEMORIZING I_IN
SAVE
EXIT
```

처음에 LOAD 하므로 Eorror출력
그후 ADD, MOVE 10 success
R_PRE, I_PRE, R_IN, I_IN 방식으로
출력




```
LOAD
ADD
MOVE 10
UPDATE fire 불꽃
UPDATE life 삶
SEARCH lif
SEARCH life
PRINT MEMORIZING I_LEVEL
MOVE 80
PRINT TO_MEMORIZE
MOVE 10
TEST life       인생
TEST life       삶
TEST problem    문제
TEST story      이야기
TEST lot        많이
TEST name       이름
TEST hand       손
TEST place      장소
TEST work       일
TEST make       만들다
PRINT MEMORIZED
MOVE 10
SAVE
EXIT
```

불 -> 불꽃으로 update

인생 -> 삶으로 update 성공

search lif 는 fail

search life 는 success

I_LEVEL로 출력

TO-MEMORIZE 출력

test 인생 삶으로 바뀌었으므로 fail

test life ~ make 는 test 후 pass하여 MEMORIZED 로 옮겨짐

MEMORIZED 출력

MOV 10 는 100개를 넘으므로 error

=========PRINT=======
make 만들다
different 다른
important 중요한
person 사람
clothes 옷
movie 영화
activity 활동
example 예
letter 편지
fire 불꽃
part 부분
plan 계획
plant 식물
fun 재미
listen 듣다
learn 배우다
each 각각
same 같은
bird 새
trip 여행
vacation 휴가
summer 여름
course 강좌
spring 봄
autumn 가을
winter 겨울
space 공간
street 거리
paper 종이
newspaper 신문
face 얼굴
mind 마음
volunteer 자원봉사자
visit 방문
start 시작
light 빛
enjoy 즐기다
watch 시계
win 이기다
clean 깨끗한
famous 유명한
special 특별한
just 단지
nature 자연
restaurant 식당
group 집단
habit 습관
culture 문화
information 정보
culture 문화
information 정보
advertisement 광고
science 과학
gene 유전자
war 전쟁
store 가게
sound 소리
fly 날다
easy 쉬운
poor 가난한
fast 빨리
back 뒤
always 언제나
history 역사
state 상태
soldier 군인
village 마을
office 사무실
island 섬
piece 조각
rock 바위
line 선
cook 요리사
end 끝
remember 기억하다
wear 입다
send 보내다
hot 뜨거운
early 일찍
often 자주
sometimes 때때로
pet 애완동물
vegetable 채소
leaf 잎
forest 숲
area 지역
neighbor 이웃
art 미술
poem 시
subject 주제
bottle 병
machine 기계
fact 사실
rule 규칙
die 죽다
busy 바쁜
sick 아픈
health 건강
holiday 공휴일

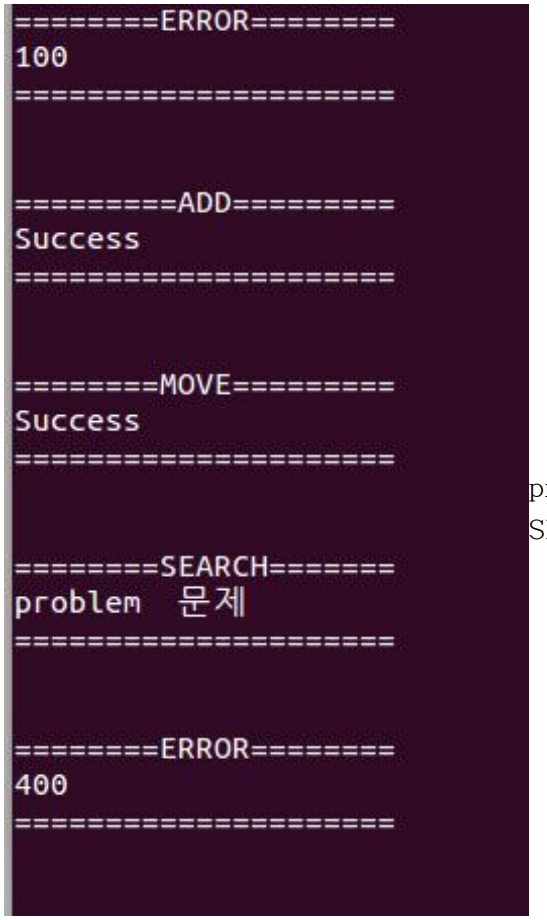
holiday 공휴일
goal 목표
order 순서
result 결과
half 절반의
decide 결정하다
choose 선택하다
difficult 어려운
foreign 외국의
brain 뇌
voice 목소리
opinion 의견
age 나이
price 가격
dish 접시
subway 지하철
bear 곰
human 인간
buy 사다
sell 팔다
follow 따르다
miss 놓치다
close 닫다
delicious 맛있는
sad 슬픈
however 그러나
diary 일기
cartoon 만화
event 사건
toy 장난감
weight 무게
smart 영리한
president 대통령
arm 팔다
meal 식사
skill 기술
contest 경쟁
prize 상품
chance 기회
shape 모양
wall 벽
smell 냄새
judge 판단하다
cover 덮개
catch 잡다
wash 씻다
dark 더움
dirty 더러운
tired 피곤한
upset 속상한

upset 속상한
environment 환경
pollution 오염
lake 호수
desert 사막
insect 곤충
accident 사고
collage 대학
advice 조언
hobby 취미
mark 표시
post 우편
receive 바다
add 더하다
physical 신체의
modern 현대의
pretty 예쁜
clerk 사무원
foot 발
company 회사
factory 공장
garage 차고
palace 궁전
hole 구멍
bill 청구서
seed 씨
medicine 약
meaning 의미
race 경주
collect 모으다
heavy 무거운
useful 유용한
maybe 아마
sense 느낌
image 이미지
map 지도
type 유형
project 계획
traffic 교통
safety 안전
spaceship 우주선
rainbow 무지개
seat 좌석
magic 마술
secret 비밀
luck 행운
success 성공
coin 동전
goods 상품
address 주소

address 주소
circle 원
mix 섞다
library 도서관
museum 박물관
ocean 바다
sentence 문장
proverb 속담
invention 발명
trash 쓰레기
moment 순간
product 제품
ring 반지
date 날짜
peace 평화
storm 폭풍우
disease 병
ghost 유령
bone 뼈
wisdom 지혜
candle 초
zoo 동물원
scene 장면
memory 기억
recipy 조리법
wheel 바퀴
kid 아이
smoke 연기
cute 귀여운
already 이미
hometown 고향
yard 마당
bridge 다리
fairy 요정
flag 깃발
pain 고통
guide 안내자
promise 약속
bake 굽다
background 배경
distance 거리
behavier 행동
danger 위험
object 물건
tear 눈물
honor 명예
expensive 비싼
ability 능력
action 행동
pride 자랑
pride 자랑
slave 노예
rope 밧줄
tip 조언
sunrise 일출
garbage 쓰레기
cash 현금
police 경찰
dictionary 사전
adult 어른
brick 벽돌
temple 사원
wing 날개
purpose 목적
god 신
guest 손님
diver 잠수부
cell 세포
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
==========

```
========ERROR========
500
====================

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

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

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

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

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

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

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

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

========TEST=========
Pass
====================
```

```
=========PRINT=======
life     삶
problem 문제
story    이야기
lot      많이
name     이름
hand     손
place    장소
work     일
fire     불꽃
====================

========ERROR========
300
====================

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




```
LOAD
ADD
MOVE 10
SEARCH PrObLeM
SAVE
EXIT
```

problem을 대소문자를 섞어 PrObLeM으로 SEARCH하여도 검색하여 출력합니다.

5. Consideration

이름	프로젝트에서 맡은 역할	본인 스스로 생각하는 자신의 점수
이형민	Alphabet BST Print R_Pre, I_Pre Manager LOAD,ADD,TEST	10
고찰	팀프로젝트를 진행해 나가는데 있어서 각각 다르게 세명이서 설계한 코드들을 가지고 하나의 프로그램으로써 동적을 수행하기위해 합치는 과정에서 생각보다 많은 시간이 걸렸습니다. 같은 목적을 위해 설계한 프로그램임에도 불고하고 많은 에러들이 발생해서 합치는 과정에서 애를 많이 먹었습니다. 또 비주얼 스튜디오에서 에러 없이 돌아가는 코드를 리눅스로 옮겨서 실행했을 때 원했던 결과들이 나오지 않았습니다. 예상치 못했던 메모리 에러들이나 잘 못된 NULL 참조값으로 인해서 생긴 에러들 이였는데 segmentation fault가 발생하고 프로그램이 꺼지는하는 경우도 허다 했습니다. 코드를 하나하나 다시보면서 잘못된 것들을 수정해나가는 과정에 있어서 리눅스를 다루는 법도 새롭게 많이 익힌것 같습니다. 프로젝트를 처음 진행할때 프로젝트 자체를 이해하지 못했습니다. 큐 와 BST와 원형linkedlist가 각각 사용된다는 것을 알았는데 매니져의 쓰임이 어떻게 되는지 delete에서는 왜 return으로 노드를 넘겨줘야하는지 하는 기초적인 것들 부터 시작해서 command파일을 읽어와서 파일에 있는 명령대로 프로그램을 실행시키는 것 까지 전체적인 큰그림을 보지 못한채 무턱대고 프로젝트에 대한 코딩을 시작해서 그랬던 것이라고 생각합니다. 이전 까지 homework로 나왔던 간단한 프로그램 코딩과 같은 경우에는 일정한 목적을 가지고 부분을 나누지 않고 코딩을 해도 문제를 이해하는데 있어서 무리가 없었지만 이번 프로젝트와 같은 규모가 이전보다 커진 문제에서는 먼저 문제를 이해한후에 코딩을 실시하는 것이 더 중요하다는 깨달음을 가지게 되었습니다. 스택과 큐의 많은 활용방법에 대해서도 익혔습니다. 여러가지 방식의 print를 수행해내는 과정에 있어서 재귀함수를 활용하는 것이 아닌 다른 방식으로 print를 재귀함수를 썻을 때와 같은 결과값을 가지도록 코딩을 하는데 스택과 큐의 저장	

	원리인 LIFO,FIFO를 활용한 방법들을 생각했을 때 대단하다는 생각을 했습니다. 특히 2개의 스택을 사용한 print 방법은 스택을 2개로 활용할 생각도 못하고 스택을 1개만 써서 어떻게든 해보려고 팀원들과 많은 시간들을 보냈는데 '한번 2개를 써볼까?' 라는 팀원의 말대로 2개로써 설계를 해서 됬을 때 팀프로젝트의 의미도 다시한번 생각하게 되었습니다. 수많은 예외처리에 대해서도 생각해 보게 됬습니다. 이번 프로젝트를 진행하면서 지금까지 homework를 수행했던 코드들이 얼마나 엉망인 code인지를 알게되었습니다. 수많은 가능한 경우의 수들을 두고 해당하는 경우에 대해 예외처리를 실시하는 부분 하나하나가 엉망인 code를 하나하나 고쳐나가는 과정 같은 느낌도 들었습니다. 비록 이번1차 프로젝트만 팀을 정해서 하는 팀프로젝트 였지만 다음애도 팀프로젝트의 기회가 있다면 같은 팀을 이뤄해보고 싶습니다.	
김동현	Word BST Print R_in, I_in Manager RUN, MOVE, SEARCH,	10
고찰	이번 프로젝트는 리눅스 상에서 구현을 했다. 하지만 불편한 개발환경으로 처음에는 visual studio를 이용해서 개발을 하기 시작했다. 그리고 코드를 모두 작성하고 세명의 코드를 모두 모아서 visual studio에서 잡을 수 있는 오류들을 모두 잡기 시작했다. 어느정도 에러들이 잡혀갔을 쯤 리눅스에 올려서 프로그램을 실행했을 때 컴파일 조차 되지 않는 상황이 발생하였다. 구글을 통해서 비쥬얼 스튜디오와 g++ 컴파일러의 차이점을 공부하고 안되는 부분들을 찾아 내기 위해서 많은 시간들을 보냈다. 특히 g++ 에서는 비주얼 스튜디오에서 한줄 단위 프로시저 실행 디버깅이 없어서 어디서 값이 제대로 읽히는지 잘 쓰이긴 하는지 잘 전달됐는지 header는 잘 있는지 확인하기 상당히 어려웠다. 직접 화면에 cout을 이용해서 출력하는 방법을 통해서 위의 오류들을 잡기는 했지만 g++에도 디버깅이 가능하다는 점을 구글을 통해서 알게 되었다. 이번 프로젝트를 마치고 앞으로는 리눅스 환경에서 계속 개발할 예정이기 때문에 g++용 디버깅 명령어와 디버깅 방법들을 좀더 공부할 필요를 느꼈다. 그리고 무엇보다도 방어적인 코딩의 중요성을 매우 느꼈는데 리눅스 검정 화면에서 segmentation fault 딱 한줄이 뜨고 프로그램이 죽을	

	경우엔 정말 앞이 깜깜해 졌다. 이는 분명 null 포인터의 주속값을 참조하려다가 발생한 오류들임이 틀림없었지만 어느 부분에서 잘못 접근하는지 알기가 매우 어려웠다 .그리하여서 bst CLL queue 를 나누어서 따로 인서트 해보고 서치해보고 각각의 함수들으 모두 돌려 보고 다시 끼워 맞추는 등 여러가지의 작업을 했다. 처음부터 셋다 방어 적인 코딩을 했다면 아마 프로젝트를 좀더 일찍 끝낼수 있었을 것이다. 특히 node->getnext는 조심해야 하는 점과 구조체가 비어있는 경우에 예외처리들을 잘 해줘야 겠다고 느꼈다. 팀프로젝트를 하면서 느낀점은 일단 팀원이 기존에 학교를 같이 다니던 동기들이였고 서로 친하기 때문에 약속시간에 모이는데 어려움은 없었다.하지만 세명의 코딩 스 일과 또 실력들이 좀 차이가 나서 어려움이 있었다. 하지만 적절히 일을 배분하여 좋게 마무리 할 수 있었던것 같다.	
김동민	Stack, Queue, circular linked list Print R_post, I_post, I_Level Manager UPDATE,PRINT,SAVE	10
고찰	이번 프로젝트를 구현하면서 겪었던 어려움은 처음 써보았던 리눅스와 프로그램을 팀프로젝트로 진행하였던 것입니다. 리눅스를 처음해보니 불편하고 프로젝트파일을 실행시키는것도 잘하지못해 팀원의 도움을 받아 하였습니다. 또한 리눅스는 컴파일이 visual과 다르게 segment fault가 나는 오류가 발생하여서 어떻게 해결해야할지 막막했습니다. 그래서 다시 visual에서 컴파일하면서 cout으로 결과를 눈으로 확인하면서 오류를 수정해 나갔습니다. 특히 리눅스에서 strcpy를 사용할때 기존의 저의 스타일은 0이거나 -1이거나 1로 했는데 이렇게 구현을 하면 리눅스 상에서는 제대로 작동 안할수있단걸 몰랐습니다. 하지만 팀프로젝트기 때문에 조원들간의 교류를 통해 오류를 빠르게 알수있었고 잘 해결할수있었습니다. 기존의 친구들이 저와 코딩스타일이 달라 친구들의 코드를 이해하고 그 코드를 이용해서 코드를 구현하는데 상당히 어려움이있었습니다. 그래서 친구들에게 방어적인 코딩 스타일을 배웠고 예외처리에 대한 중요성에 대해 알게되었습니다. 혼자 구현할땐 몰랐었는데 다른 친구들과 맞춰보니 기존의 작성했던 코드에서 제가 찾지 못했던 예외처리들 때문에	

	프로그램이 중간에 멈춰 힘든점들이 많았지만 셋이 힘을 모아 열심히 visual 을 이용해서 디버깅을 하고 오류를 모두 해결했습니다. 그러면서 자연스럽게 제 실력도 늘고 과제를 성공적으로 마친것같습니다.	