

2016. 10. 07 (금)

Data Structure

Project 1

팀장 : 박상혁 / 설계(1) / 실습 (C) / 2012722048

팀원 : 유원선 / 설계(1) / 실습 (A) / 2012722045

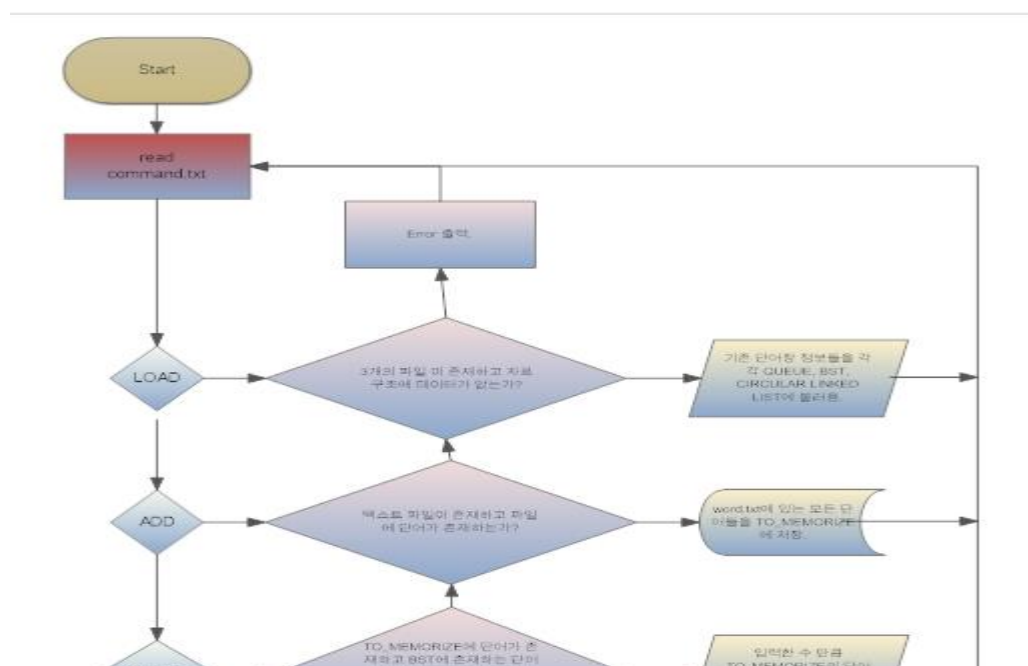
*Introduction

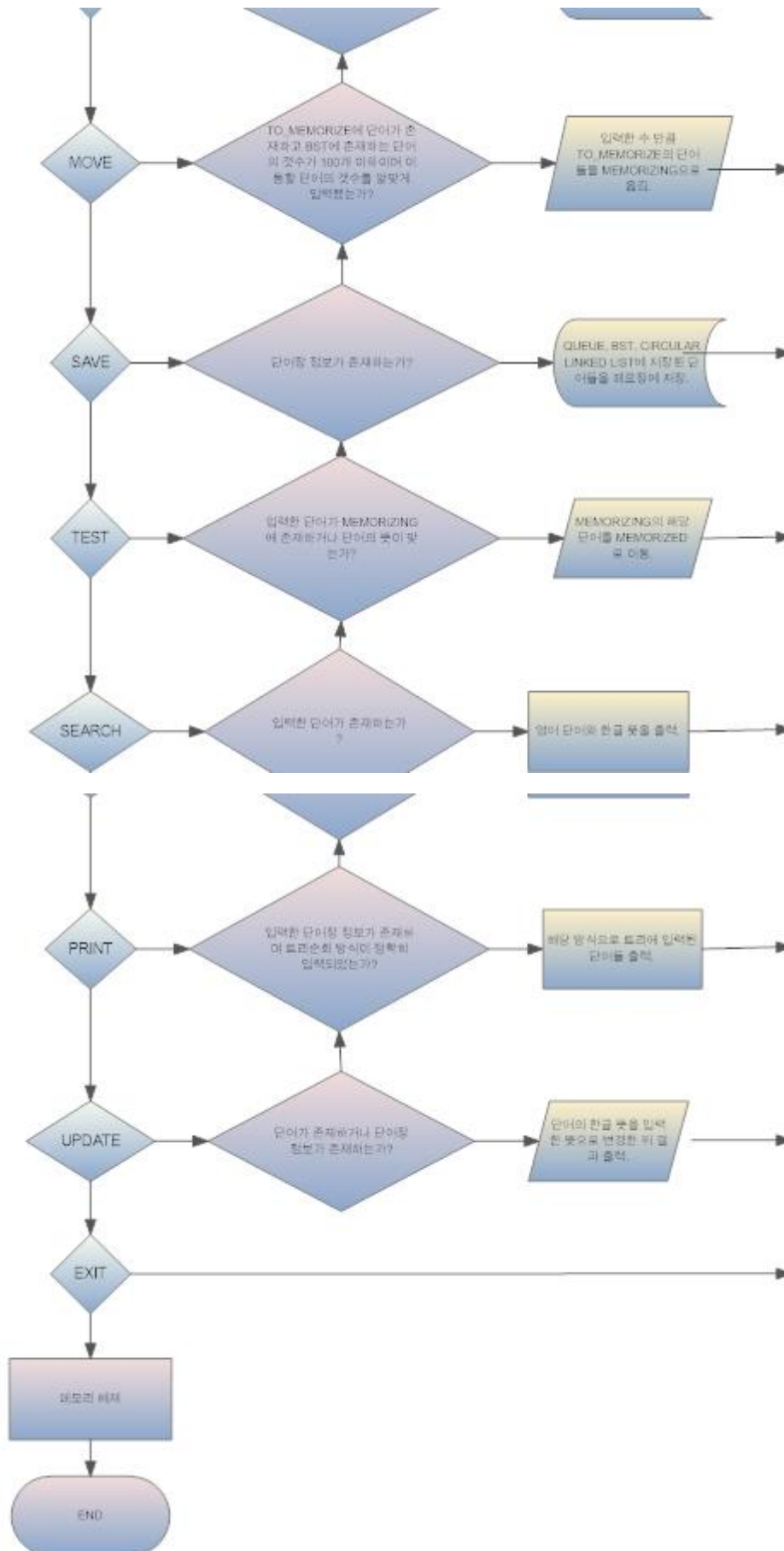
단어의 입출력, 검색, 테스트 등이 가능한 사전 프로그램입니다. 이 프로그램은 3 개의 단어장으로 구성되어 있으며, 각 단어장들은 암기 해야할 단어, 암기중인 단어, 암기가 끝난 단어들을 보관하며, 암기 해야할 단어장은 Queue, 암기중인 단어장은 BST, 암기가 끝난 단어장은 Circular linked list 로 구현되어 있습니다. 이 프로그램은 command.txt 를 통해서 동작하며 해당 텍스트 파일에 존재하는 명령어들을 순서대로 읽으며 해당 명령어들을 수행하고 결과를 출력합니다.

암기중인 단어장은 2 개의 BST 로 구현되어 있으며, 알파벳 BST 를 통해 1 차적으로 들어온 단어를 알파벳 별로 분류하며, 다른 BST 인 단어 BST 를 통해 2 차적으로 단어를 사전적 순서로 배치합니다. BST 는 최대 100 개의 단어들을 저장할 수 있습니다.

이 프로그램에서 사용되는 명령어는 9 개이며 기존의 단어장 정보를 불러오는 LOAD 명령어, 단어 텍스트 파일에 있는 단어 정보를 읽어오는 ADD 명령어, 사용자가 입력한 수만큼 TO_MEMORIZE 의 단어들을 MEMORIZING 으로 옮기는 MOVE 명령어, 현재 단어장 정보를 저장하는 SAVE 명령어, 단어를 외웠는지 테스트하는 TEST 명령어, 단어의 뜻을 찾아 출력하는 SEARCH 명령어, 입력한 단어장에 있는 단어들을 출력하는 PRINT 명령어, 단어의 뜻을 변경하는 UPDATE 명령어, 마지막으로 프로그램 상의 메모리를 해제하며 프로그램을 종료하는 EXIT 명령어로 이루어져 있습니다. 또한 각 명령어들은 명령어에 해당하는 에러 코드들을 가지고 있어서, 해당 명령어가 실행되지 못했을 경우 에러코드를 출력하여 확인이 가능하도록 해야합니다. 모든 출력 결과는 log.txt 에 저장되도록 합니다.

*Flow chart





*Algorithm with pseudo code (visual studio 에서 작성하였습니다)

(1) QUEUE

```
Queue::Queue( )
{
    pHead <- NULL 값으로 설정해 주어 에러를 방지합니다.
}

Queue::~~Queue( )
{
    if ( 큐가 비어있다면 )
        return;

    WordNode * pCur <- 큐를 이동하면서 작업을 수행할 노드 생성.
    while ( pHead가 NULL값이 될때까지 반복 )
    {
        pCur은 항상 큐의 처음 부분에 위치
        pHead는 다음 칸으로 이동
        pCur에 연결되어 있던 다음칸과의 연결을 끊음
        pCur을 삭제
    }
}
```

```
void Queue::Push( WordNode * node )
{
    if ( 큐가 비어있다면 )
    {
        새로 들어온 노드를 pHead로 설정
        return;
    }

    WordNode * pCur = pHead; <- 큐를 이동하며 작업을 수행할 새로운 노드 생성
    while ( pCur의 다음칸이 비어있을때까지 반복 )
    {
        if ( pCur이 가리키고 있는 단어와, 새로 들어온 노드의 단어가 같다면 )
            return;

        pCur은 다음칸으로 이동
    }
    if ( pCur와 새로 들어온 노드의 단어가 같다면 )
        return;
    pCur의 다음칸을 새로 들어온 노드로 설정해 주고 연결
}
```

```

WordNode * Queue::Pop()
{
    if (큐가 비어있다면)
        return 0;

    WordNode * pCur = pHead; <- 큐를 이동하며 작업을 수행할 새로운 노드 생성, 처음 위치는 pHead
    pHead의 위치를 다음칸으로 이동
    pCur가 가리키고 있는 값과 큐의 연결을 끊음
    return pCur;
}

WordNode * Queue::Search(char * word)
{
    if (큐가 비어있다면)
        return 0;

    WordNode * pCur = pHead; <-큐를 이동하며 작업을 수행할 새로운 노드 생성, 처음 위치는 pHead
    while (pCur가 NULL이 될때까지 반복)
    {
        if (pCur가 가리키고 있는 단어와 새로 입력 받은 단어가 같다면)
            return pCur;
        pCur를 다음칸으로 이동
    }
    return 0;
}

```

```

bool Queue::Print()
{
    if (pHead가 비어있다면)
        return false;

    WordNode * pCur = pHead; <-큐를 이동하며 작업을 수행할 새로운 노드 생성, 처음 위치는 pHead

    cout << "==== PRINT =====" << endl;

    while (pCur가 NULL이 될때까지 반복)
    {
        pCur가 가리키고 있는 노드의 단어와, 뜻을 출력
        pCur를 다음칸으로 이동
    }

    return true;
}

```

```

bool Queue::Save()
{
    if (큐가 비어있다면)
        return false;

    ofstream fout("to_memorize_word.txt"); <- 해당 제목의 텍스트파일 생성하고 열기, 이미 존재한다면 덮어쓰기;

    WordNode * pCur = pHead; <- 큐를 이동하며 작업을 수행할 새로운 노드 생성, 처음 위치는 pHead;
    while (pCur가 NULL이 될때까지 반복)
    {
        pCur가 가리키고 있는 노드의 단어와 뜻을 메모장에 저장
        pCur를 다음칸으로 이동
    }
    메모장 닫기
    return true;
}

```

(2) WordBST

```
WordBST::WordBST()
{
    root <- NULL 값으로 설정해 주어 에러를 방지
}

WordBST::~WordBST()
{
    if (BST가 비어있다면)
        return;
}
```

```
void WordBST::Insert(WordNode * node)
{
    if (BST가 비어있다면)
    {
        BST의 루트를 새로 들어온 노드로 설정
        return;
    }
}
```

```
WordNode * pCur = root; <- BST를 이동하며 작업을 수행할 노드 생성
while (1)
{
    if (새로 들어온 단어가 앞에 위치한다면)
    {
        if (왼쪽 자식이 있다면)
            왼쪽 자식의 위치로 이동

        else 왼쪽 자식이 비어있다면
        {
            새로 들어온 노드를 왼쪽 자식의 위치에 삽입
            return;
        }
    }
    else if (새로 들어온 단어가 뒤에 위치한다면)
    {
        if (오른쪽 자식이 있다면)
            오른쪽 자식의 위치로 이동

        else 오른쪽 자식이 비어있다면
        {
            새로 들어온 노드를 오른쪽 자식의 위치에 삽입
            return;
        }
    }
    else 단어가 같다면 (중복일 경우)
        return;
}
```

```

WordNode * WordBST::Delete()
{
    if (BST가 비어있다면)
        return 0;

    WordNode * pCur = root; <- pCur BST를 탐색할 노드
    WordNode * pPre = pCur; <- pPre = pCur의 부모;
    while (1)
    {
        if (새로 들어온 단어가 앞에 위치한다면)
        {
            if (왼쪽 자식이 있다면)
            {
                pPre = pCur;
                pCur을 왼쪽 자식 위치로 이동
            }
            else 왼쪽 자식이 없다면
                return 0;
        }

        else if (새로 들어온 단어가 뒤에 위치한다면)
        {
            if (오른쪽 자식이 있다면)
            {
                pPre = pCur;
                pCur을 오른쪽 자식 위치로 이동
            }
            else 오른쪽 자식이 없다면
                return 0;
        }
    }
}

```

```

else 단어를 찾았다면
{
    if (자식이 없다면)
    {
        if (pPre 왼쪽 자식 노드가 pCur)
            pPre와 왼쪽 자식의 연결을 끊음

        else pPre 오른쪽 자식 노드가 pCur
            pPre와 오른쪽 자식의 연결을 끊음

        return pCur;
    }

    else if (왼쪽 자식 / 양쪽 자식이 있다면)
    {
        WordNode * pTemp = pCur->GetLeft();      pTemp = 왼쪽 서브트리 중 가장 큰 노드
        WordNode * pTemp2 = pCur;                pTemp2 = pTemp의 부모

        while (pTemp의 오른쪽 값이 NULL이 될 때까지 반복)
        {
            pTemp2 = pTemp;
            pTemp는 오른쪽으로 이동
        }

        if (삭제할 노드가 왼쪽 자식이라면)
            pPre의 왼쪽 자식을 pTemp로 설정

        else
            pPre의 오른쪽을 pTemp로 설정
    }
}

```



```

// 대소문자 구분 X 검색
WordNode * WordBST::Search(char * word)
{
    if (BST가 비어있다면)
        return 0;

    WordNode * pCur = root; <-BST를 이동하며 작업을 수행할 노드 생성.
    while (1)
    {
        if (새 단어가 앞에 위치한다면)
        {
            if (왼쪽 자식이 있다면)
                pCur의 왼쪽 자식의 위치로 이동

            else 왼쪽 자식이 없다면
                return 0;
        }
        else if (새로 들어온 단어가 뒤에 위치한다면)
        {
            if (오른쪽 자식이 있다면)
                pCur의 오른쪽 자식의 위치로 이동
            else 오른쪽 자식이 없다면
                return 0;
        }
        else 단어가 같다면 = 찾았다면
            return pCur; 찾은 단어 반환
    }
}

```

```

bool WordBST::Print(char * order) // PRINT
{
    if (BST가 비어있다면)
        return false;

    if (R_PRE일 경우)
    {
        재귀함수를 통하여 출력
        return true;
    }
    else if (L_PRE일 경우)
    {
        스택을 이용하여

        while (pCur이 NULL 이 될때까지 반복)
        {
            영단어와 한글 뜻을 출력하고
            if (오른쪽 자식이 있다면)    오른쪽 자식 푸쉬
            if (왼쪽 자식이 있다면)    왼쪽 자식 푸쉬
            if (스택의 pHead가 NULL이 아니면)
                return true;
            pCur <- 스택의 Top;
            pop
        }
    }
}

```

```

else if (R_IN일 경우)
{
    재귀함수를 통하여 출력
    return true;
}
else if (I_IN일 경우)
{
    스택을 이용하여

    while (1) {
        while (pCur이 NULL 이 될때까지 반복)
        {
            pCur값을 푸쉬
            pCur의 위치를 왼쪽 자식의 위치로 이동
        }
        if (스택의 pHead가 NULL이 아니면)
            return true;
        pCur <-스택의 Top;
        pop
        pCur가 가리키는 노드의 영단어, 뜻 출력
        pCur의 위치를 오른쪽 자식의 위치로 이동
    }
}
else if (R_POST)
{
    재귀함수를 통하여 출력
    return true;
}
}

```

```

else if (I_POST일 경우)
{
    스택을 이용하여

    while (1)
    {
        if (pCur가 leaf 라면)
        {
            pCur가 가리키는 노드의 영단어, 뜻 출력
            pTemp = pCur;
            pCur <-스택의 Top;
            pop;
            continue;
        }

        pCur를 스택에 push
        if (pCur가 leaf 라면)
        {
            s.Pop();
            pCur가 가리키는 노드의 영단어, 뜻 출력
            pTemp = pCur;
            pCur <-스택의 Top;
            pop;
            if (pTemp == root)
                return true;
            else
                continue;
        }
    }
}

```

```

        if (오른쪽 자식이 있다면)    오른쪽 자식 푸쉬
        if (왼쪽 자식이 있다면)    왼쪽 자식 푸쉬
        pCur <-스택의 Top;
        pop
    }
}
else if (ILLEVEL일 경우)
{
    큐를 이용하여

    while (pCur이 NULL 이 될때 까지 반복)
    {
        pCur가 가리키는 노드의 영단어, 뜻 출력
        if (왼쪽 자식이 있다면)    왼쪽 자식 푸쉬
        if (오른쪽 자식이 있다면)    오른쪽 자식 푸쉬
        if (큐의 pHead가 NULL이 아니라면)    return true;
        pCur <-큐의 Top;
        pop
    }
}

return false;    // 일치하는 order가 없는 경우 -> 매니저 클래스에서 에러 처리
}

```

```

bool    WordBST::Save()                                // SAVE
{
    if (BST가 비어있다면)                                // 단어가 하나도 없으면
        return false;

    재귀함수를 통한 저장
    return true;
}

```

```

void    WordBST::Save_PreOrder(WordNode * node)
{
    ofstream fout("memorizing_word.txt", ios::app);    <- 해당 텍스트 파일에 덮어 쓰지 않고 이어쓰기

    스택 및 탐색 노드 생성

    while (pCur이 NULL이 될때 까지 반복)
    {
        pCur 노드의 영단어 뜻 출력 및 텍스트 파일에 저장

        if (오른쪽 자식이 있다면)    오른쪽 자식 푸쉬
        if (왼쪽 자식이 있다면)    왼쪽 자식 푸쉬
        if (스택의 pHead가 NULL이 아니라면)
            return;
        pCur <-스택의 Top;
        pop
    }
    fout.close();
}

```

(3) Circular Linked List

```
CircularLinkedList::CircularLinkedList()
{
    pHead <- NULL 값으로 설정해 주어 에러발생을 방지
}

CircularLinkedList::~~CircularLinkedList()
{
    if (pHead가 NULL이면)
        return;

    pCur, pHead 생성 및 초기화

    do {
        pCur = pHead;
        pHead를 다음 노드 위치로 이동
        delete pCur;
    } while (pHead 와 pTemp가 같지 않다면 반복);
}
```

```
void CircularLinkedList::Insert(WordNode * node)
{
    if (pHead가 NULL이면)
    {
        pHead = node;
        환형 링크드 리스트 이므로 자기 자신을 가리킴
        return;
    }

    WordNode * pCur = pHead;
    while (pCur가 가리키는 노드가 pHead가 아닐경우 반복)
        pCur를 계속 이동
    pCur의 다음노드를 새로 들어온 노드로 설정
    새로 들어온 node의 다음을 pHead로 설정
}

WordNode * CircularLinkedList::Search(char * word)
{
    if (pHead가 NULL이면)
        return 0;

    WordNode * pCur = pHead;
    do {
        if (_stricmp(pCur->GetWord(), word) == 0)
            return pCur;
        pCur = pCur->GetNext();
    } while (pCur != pHead);
    return 0;
}
```

```

bool CircularLinkedList::Print() // PRINT
{
    if (pHead가 NULL이면)
        return false;

    WordNode * pCur = pHead;
    cout << "==== PRINT =====> endl;
    do {
        pCur가 가리키는 노드의 영단어 뜻 출력
        pCur는 옆 노드로 이동
    } while (pCur 와 pHead 같지 않으면 반복);
    cout << "====> endl << endl;
    return true;
}

bool CircularLinkedList::Save() // SAVE
{
    if (pHead가 NULL이면)
        return false; // 단어장 비어있으면 false

    ofstream fout("memorized_word.txt"); 해당 이름의 텍스트 파일 생성 및 열기

    WordNode * pCur = pHead;
    do {
        pCur가 가리키는 노드의 영단어 뜻 텍스트 파일에 저장
        pCur는 옆 노드로 이동
    } while (pCur 와 pHead 같지 않으면 반복);
    파일 닫기
    return true;
}

```

(4) Manager

```

Manager::Manager()
{
    cll = new CircularLinkedList;
    bst = new AlphabetBST;
    queue = new Queue;
}

Manager::~Manager()
{
    delete cll;
    delete bst;
    delete queue;
}

```



```

void Manager::run(const char * command)
{
    ifstream fin(command);
    if (파일 열지 못하면)
        return;

    char ch[26] = { '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' };

    for (int i = 0; i < 26; i++)
    {
        알파벳 BST 초기화 작업
    }

    // str1:명령어 한 줄   str2:명령어 str3:인자 str4:인자
    char line[64] = {}, cmd[32] = {}, param1[32] = {}, param2[32] = {}, param3[32] = {};
    int num = 0;
    int i, j, k, l = 0;

    while (파일의 끝날때까지 반복)
    {
        명령어를 읽고 str에 저장
    }
}

```

```

if (명령어가 LOAD 라면)
{
    if (인자가 많은 경우)
        cout << "===== ERROR =====" << endl << "100" << endl << "===== " << endl << endl;
    else if (명령어 실행과 동시에 에러 판단)
        cout << "===== ERROR =====" << endl << "100" << endl << "===== " << endl << endl;
    else
        cout << "===== LOAD =====" << endl << "Success" << endl << "===== " << endl << endl;
}
else if (명령어가 ADD 라면)
{
    if (인자가 많은 경우)
        cout << "===== ERROR =====" << endl << "200" << endl << "===== " << endl << endl;
    else if (명령어 실행과 동시에 에러 판단)
        cout << "===== ERROR =====" << endl << "200" << endl << "===== " << endl << endl;
    else
        cout << "===== ADD =====" << endl << "Success" << endl << "===== " << endl << endl;
}
else if (명령어가 MOVE 라면)
{
    num = atoi(param1);
    if (인자가 많거나 적은 경우)
        cout << "===== ERROR =====" << endl << "300" << endl << "===== " << endl << endl;
    else if (MOVE(num) == false)
        cout << "===== ERROR =====" << endl << "300" << endl << "===== " << endl << endl;
    else
        cout << "===== MOVE =====" << endl << "Success" << endl << "===== " << endl << endl;
}
}

```

```

else if (SAVE 명령어)
{
    if (인자가 많은 경우)
        cout << "===== ERROR =====" << endl << "400" << endl << "===== " << endl << endl;
    else if (SAVE() == false)
        cout << "===== ERROR =====" << endl << "400" << endl << "===== " << endl << endl;
    else
        cout << "===== SAVE =====" << endl << "Success" << endl << "===== " << endl << endl;
}
else if (TEST 명령어)
{
    if (인자가 많은 경우)
        cout << "===== ERROR =====" << endl << "500" << endl << "===== " << endl << endl;
    else if (인자가 적은 경우)
        cout << "===== ERROR =====" << endl << "500" << endl << "===== " << endl << endl;
    else if (실패했을 경우)
        cout << "===== ERROR =====" << endl << "500" << endl << "===== " << endl << endl;
    else
        cout << "===== TEST =====" << endl << "Pass" << endl << "===== " << endl << endl;
}
else if (SEARCH 명령어)
{
    if (인자가 적거나 많은 경우)
        cout << "===== ERROR =====" << endl << "600" << endl << "===== " << endl << endl;
    else if (실패했을 경우)
        cout << "===== ERROR =====" << endl << "600" << endl << "===== " << endl << endl;
}
}

```

```

else if (PRINT 명령어)
{
    if(인자가 적은 경우)
        cout << "===== ERROR =====" << endl << "700" << endl << "===== " << endl << endl;
    else if (인자가 많은 경우)
        cout << "===== ERROR =====" << endl << "700" << endl << "===== " << endl << endl;
    else if (인자가 적은 경우)
        cout << "===== ERROR =====" << endl << "700" << endl << "===== " << endl << endl;
    else if (인자가 많은 경우)
        cout << "===== ERROR =====" << endl << "700" << endl << "===== " << endl << endl;
    else if (인자가 많은 경우)
        cout << "===== ERROR =====" << endl << "700" << endl << "===== " << endl << endl;
    else if (실패했을 경우)
        cout << "===== ERROR =====" << endl << "700" << endl << "===== " << endl << endl;
}
else if (UPDATE 명령어)
{
    if (인자가 많은 경우)
        cout << "===== ERROR =====" << endl << "800" << endl << "===== " << endl << endl;
    else if (인자가 적은 경우)
        cout << "===== ERROR =====" << endl << "800" << endl << "===== " << endl << endl;
    else if (실패했을 경우)
        cout << "===== ERROR =====" << endl << "800" << endl << "===== " << endl << endl;
}
else if (EXIT 명령어)
{
    if (인자가 많은 경우)
        continue;
    else
        return;
}
}

```

```

}
else
    continue;
}
// 일치하는 명령어가 없을 경우에 대한 예외처리
// 인식할 수 있는 다음 명령어가 올 때까지 반복문 실행 (파일이 끝나기 전까지)
}
fin.close();

```

```

bool Manager::LOAD()
{
    if (큐에 단어 있다면)
        return false;
    if (bst에 단어가 있다면)
        return false;
    if (c11에 단어가 있다면)
        return false;

    char word[32] = {};
    char mean[32] = {};

    ifstream fin("to_memorize_word.txt");

    if (파일 열기에 실패했을 경우)
        return false;

    while (파일이 끝날때까지)
    {
        WordNode * pNew = new WordNode;

        if (파일이 비어있으면)
            break;

        파일을 읽어서
        pNew에 단어 설정
        pNew에 뜻 설정

        queue에 삽입
    }
}

```

```

    }
    파일 닫기

    fin.open("memorizing_word.txt");
    if (열기에 실패했을 경우)
        return false;

    while (파일이 끝날때 까지)
    {
        WordNode * pNew = new WordNode;

        if (파일이 비어있으면)
            break;

        파일을 읽어서
        pNew에 단어 설정
        pNew에 뜻 설정

        BST에 노드 삽입
        단어 카운트 +1
    }
    파일 닫기

    fin.open("memorized_word.txt");
    if (파일 열기에 실패했을 경우)
        return false;

    while (파일이 끝날때 까지)
    {
        WordNode * pNew = new WordNode;

```

```

        if (파일이 비어있으면)
            break;

        파일을 읽어서
        pNew에 단어 설정
        pNew에 뜻 설정

        CircularLinkedList에 삽입
    }
    파일 닫기
    return true;
}

/* word.txt에서 큐로 단어 Push */
bool Manager::ADD()
{
    char word[32] = {};
    char mean[32] = {};

    ifstream fin("word.txt");
    if (파일 열기에 실패했을 경우)
        return false;

    if (파일이 비어있으면) // 파일이 비어있으면
        return false;

    while (파일이 끝날때 까지)
    {
        WordNode * pNew = new WordNode;

```



```

    파일을 읽어서
    pNew에 단어 설정
    pNew에 뜻 설정

    if(bst, c11에 단어가 중복되지 않는다면)
        큐에 Push
    }
    파일 닫기
    return true;
}

```

```

bool Manager::MOVE(int num)
{
    int queueCnt = 0;    큐에 저장된 단어 수
    WordNode * pCur = queue->pHead;
    while (pCur != NULL)    아니면
    {
        pCur은 옆 노드로 이동
        큐 카운트 +1
    }

    if (큐에 단어가 없는 경우)
        return false;
    else if (입력한 수와 BST의 단어 수 합이 100개를 넘는 경우)
        return false;
    else if (입력한 만큼 단어가 존재하지 않는 경우)
        return false;

    pCur = 0;
    char str[32] = {};

    큐에 충분한 단어가 있고 입력한 수와 BST 단어의 합이 100개 이하인 경우
    for (int i = 0; i < num; i++)
    {
        BST에 단어 추가
    }
    return true;
}

```

```

bool Manager::SAVE()
{
    int err = 0;
    if (세 단어장 모두 Save 실패할 경우)
        err++;

    if (err > 0)
        return false;
    else
        return true;
}

BST단어장에서 찾은 후 일치하면 환형리스트로 이동
bool Manager::TEST(char * word, char * mean)
{
    WordNode * pCur는 검색 결과
    if (단어 못 찾으면)
        return false;
    else if (뜻이 다르면)
        return false;
    else 뜻이 같으면 BST에서 제거, 환형 리스트에 INSERT
    {
        BST에서 단어 제거
        환형 리스트에 추가
        BST의 단어 수 감소
        return true;
    }
}

```

```

세 단어장에서 단어 검색
bool Manager::SEARCH(char * word)
{
    WordNode * pCur = 큐에서 검색한 결과
    if (단어 찾았으면)
    {
        찾은 단어 출력
        return true;
    }

    pCur = BST에서 단어 검색한 결과
    if (단어 찾았으면)
    {
        찾은 단어 출력
        return true;
    }

    pCur = 환형 리스트에서 검색한 결과
    if (단어 찾았으면)
    {
        찾은 단어 출력
        return true;
    }

    단어 못 찾았으면
    return false;
}

```

```

bool Manager::PRINT(char * name, char * order)
{
    if (큐에 대한 출력을 요구할 경우)
        return queue->Print();           출력과 동시에 결과 (성공 true / 실패 false) 반환
    else if (BST에 대한 출력을 요구할 경우)
        return bst->Print(order);        출력과 동시에 결과 (성공 true / 실패 false) 반환
    else if (환형리스트에 대한 출력을 요구할 경우)
        return cli->Print();             출력과 동시에 결과 (성공 true / 실패 false) 반환

    return false;
}

```

세 단어장에서 검색 후 뜻 변경

```

bool Manager::UPDATE(char * word, char * mean)
{
    WordNode * pCur = 큐에서 검색한 결과
    if (단어 찾았으면)
    {
        새로운 뜻 설정 후 출력
        return true;
    }

    pCur = BST에서 단어 검색한 결과
    if (단어 찾았으면)
    {
        새로운 뜻 설정 후 출력
        return true;
    }
}

```

```

pCur = 환형 리스트에서 검색한 결과
if (단어 찾았으면)
{
    새로운 뜻 설정 후 출력
    return true;
}

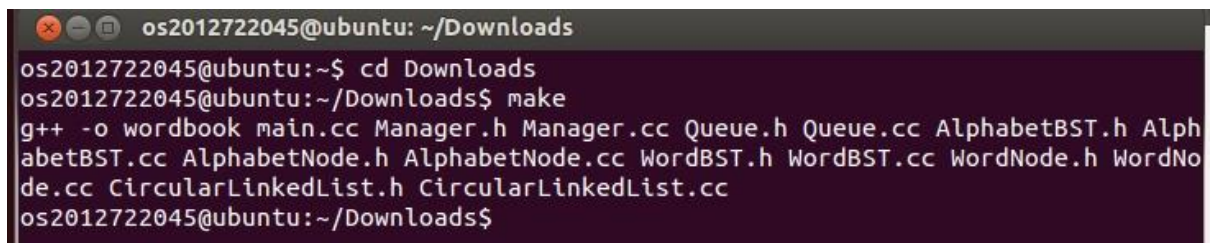
단어 못 찾았으면
return false;
}
#endif

```

*Result Screen

(1) 해당 command 로 동작 시켰을 때의 결과 화면입니다.

```
LOAD word.txt
LOAD
ADD 20
ADD
MOVE 101
MOVE 100
SEARCH fIrE
SEARCH aREa
SEARCH DATASTRUCTURE
UPDATE activITy 행동
UPDATE activity
UPDATE DATASTRUCTURE 데이터구조
TEST aCtIvItY 행동
PRINT MEMORIZED R_IN
PRINT MEMORIZED
SAVE
EXIT
```



```
os2012722045@ubuntu: ~/Downloads
os2012722045@ubuntu:~$ cd Downloads
os2012722045@ubuntu:~/Downloads$ make
g++ -o wordbook main.cc Manager.h Manager.cc Queue.h Queue.cc AlphabetBST.h Alph
abetBST.cc AlphabetNode.h AlphabetNode.cc WordBST.h WordBST.cc WordNode.h WordNo
de.cc CircularLinkedList.h CircularLinkedList.cc
os2012722045@ubuntu:~/Downloads$
```



```
os2012722045@ubuntu: ~/Downloads
os2012722045@ubuntu:~$ cd Downloads
os2012722045@ubuntu:~/Downloads$ ./wordbook
===== ERROR =====
100
=====

===== LOAD =====
Success
=====

===== ERROR =====
200
=====

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

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

===== MOVE =====
Success
```

위의 사진은 make 명령어를 사용하여 컴파일을 거친 후 실행파일을 만든 사진입니다.

왼쪽의 사진은 생성된 wordbook 파일을 실행하였을 때의 화면이며, 명령어가 올바른 형식으로 입력되지 않았을 경우 ERROR메시지와 해당 명령어의 에러코드를 출력합니다.

```
os2012722045@ubuntu: ~/Dow
===== MOVE =====
Success
=====

===== SEARCH =====
fire 불
=====

===== SEARCH =====
area 지역
=====

===== ERROR =====
600
=====

===== UPDATE =====
activity 활동 -> 행동
=====

===== ERROR =====
800
=====

os2012722045@ubuntu: ~/Dow
800
=====

===== ERROR =====
800
=====

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

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

===== PRINT =====
activity 행동
=====

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

os2012722045@ubuntu: ~/Downloads$
```

```
log.txt x
===== ERROR =====
100
=====

===== LOAD =====
Success
=====

===== ERROR =====
200
=====

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

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

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

===== SEARCH =====
fire 불
=====

log.txt x
===== SEARCH =====
area 지역
=====

===== ERROR =====
600
=====

===== UPDATE =====
activity 활동 -> 행동
=====

===== ERROR =====
800
=====

===== ERROR =====
800
=====

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

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

===== PRINT =====
```



```

===== PRINT =====
activity 행동
=====

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

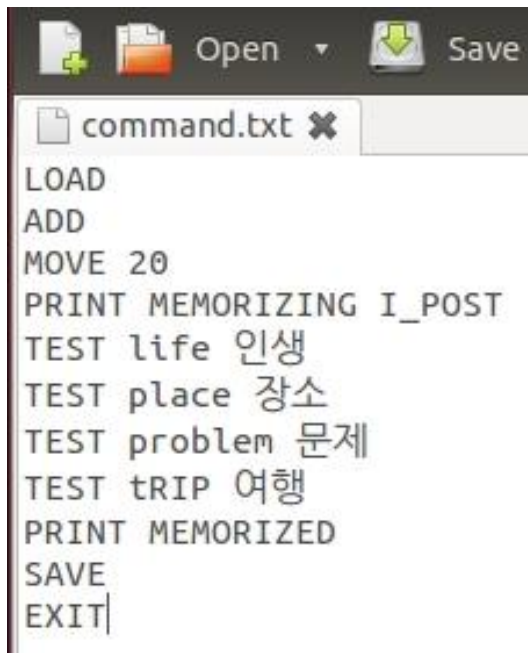
```

결과 화면 출력이 끝난 후,

log.txt 에 출력 내용이 저장된 사진입니다. 출력 되었던 내용 그대로 txt파일에 저장된 것을 확인할 수 있습니다.

(2) 해당 command 로 동작 시켰을 때의 결과 화면입니다.

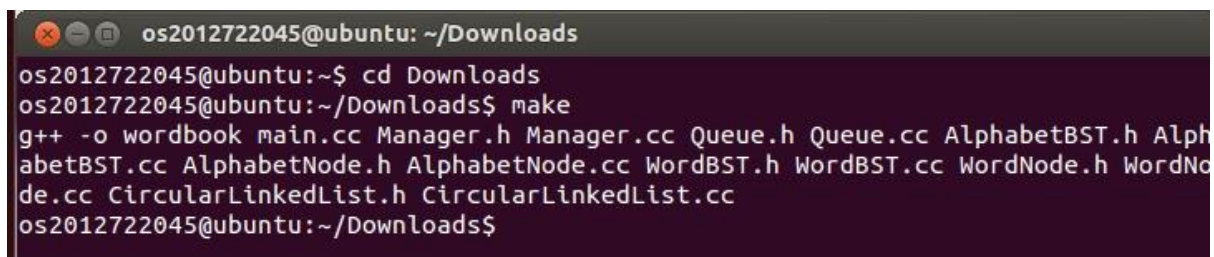
두번째 동작은 TO_MEMORIZE, MEMORIZING, MEMORIZED 파일을 초기화 후 동작시켰습니다.



```

LOAD
ADD
MOVE 20
PRINT MEMORIZING I_POST
TEST life 인생
TEST place 장소
TEST problem 문제
TEST tRIP 여행
PRINT MEMORIZED
SAVE
EXIT|

```



```

os2012722045@ubuntu: ~/Downloads
os2012722045@ubuntu:~$ cd Downloads
os2012722045@ubuntu:~/Downloads$ make
g++ -o wordbook main.cc Manager.h Manager.cc Queue.h Queue.cc AlphabetBST.h AlphabetBST.cc AlphabetNode.h AlphabetNode.cc WordBST.h WordBST.cc WordNode.h WordNode.cc CircularLinkedList.h CircularLinkedList.cc
os2012722045@ubuntu:~/Downloads$

```

위의 사진은 make 명령어를 사용하여 컴파일을 거친 후 실행파일을 만든 사진입니다.

```
os2012722045@ubuntu:~$ cd
os2012722045@ubuntu:~/Dow
g++ -o wordbook main.cc M
abetBST.cc AlphabetNode.h
de.cc CircularLinkedList.
os2012722045@ubuntu:~/Dow
===== LOAD =====
Success
=====
===== ADD =====
Success
=====
===== MOVE =====
Success
=====
===== PRINT =====
activity 활동
clothes 옷
example 예
earth 지구
fire 불
different 다른
important 중요한
job 직업
movie 영화
make 만들다
name 이름
letter 편지
lot 많이
life 인생
hand 손
story 이야기
work 일
person 사람
place 장소
problem 문제
=====
===== TEST =====
Pass
=====
===== TEST =====
Pass
=====
```

왼쪽의 사진은 생성된 wordbook 파일을 실행하였을 때의 화면이며, 명령어가 올바른 형식으로 입력되지 않았을 경우 ERROR메시지와 해당 명령어의 에러코드를 출력합니다.

아래의 사진은 결과 화면 출력이 끝난 후, log.txt 에 출력 내용이 저장된 사진입니다. 출력 되었던 내용 그대로 txt파일에 저장된 것을 확인할 수 있습니다.

```
os2012722045@ubuntu:~/Downloads$ cat log.txt
===== TEST =====
Pass
=====

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

===== ERROR =====
500
=====

===== PRINT =====
life 인생
place 장소
problem 문제
=====

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

===== LOAD =====
Success
=====

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

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

===== PRINT =====
activity 활동
clothes 옷
example 예
earth 지구
fire 불
different 다른
important 중요한
job 직업
movie 영화
make 만들다
name 이름
letter 편지
lot 많이
Loading file /home/os2012722045/Downloads/log.txt
```

```
log.txt (~/Downloads)
name 이름
letter 편지
lot 많이
life 인생
hand 손
story 이야기
work 일
person 사람
place 장소
problem 문제
=====

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

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

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

===== ERROR =====
500
=====

===== PRINT =====
life 인생
place 장소
problem 문제
=====

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


*Consideration

이름 : 유원선

맡은 역할 : Queue, WordNode, WordBST, CircularLinkedList, AlphabetNode, 보고서 작성, 에러수정

프로젝트에 대한 고찰 : 데이터 구조의 첫 팀 프로젝트였습니다. 처음에는 쉬울 것 같았으나 코드를 짜면서 헤매는 부분이 있어서 팀원의 도움으로 프로젝트를 완성시킬 수 있었습니다. 팀 프로젝트였기 때문에 제가 못했던 부분을 도움을 얻어서 일을 진행할 수 있다는 것이 좋은 점이었다고 생각하며, 이번 프로젝트를 진행하면서 어려웠던 점은 특별히 없었습니다. 다음 프로젝트도 열심히 하여 제출하겠습니다.

이름 : 박상혁

맡은 역할 : 위에서 언급한 코드 공동 구현, 이외에 BST, 매니저 클래스 구현, 동작 테스트 및 수정, 리눅스 컴파일

고찰 : 코딩 프로젝트가 늘 그렇듯 쉬운 편은 아니지만, 곰곰히 생각하며 구현하면 작성하지 못할 코드는 없는 것 같다. 이번 프로젝트도 중간마다 디버깅없이 프로젝트를 완성한 후에 한 번에 에러를 잡으려 하니 막막했지만, 차근차근 풀어간 결과 완성 할 수 있었다. 다만 여러번 테스트를 통해 발견한 모든 에러를 잡았음에도 아직 내가 모르는 에러가 나올 수 있을까 싶어 조금 걱정된다. 프로젝트를 하면서 시간관리의 중요성을 다시 한 번 느꼈다. 이번 팀프로젝트에서 다행인 것은 난이도가 아주 어렵진 않아서 핵심적이 코드들을 구현함에 있어 큰 부담이 되지 않았던 것과 팀원과 친분이 있는 사이여서 서로 부담없이 의견교환 및 프로젝트를 진행할 수 있었던 점이다.

본인 스스로 생각하는 자신의 점수(10) : 9 점 이상