자료구조2 실습

5주차 과제

제출일	21.10.02	전 공	컴퓨터소프트웨어공학과
과 목	자료구조 2	학 번	20184612
담당교수	홍 민 교수님	이 름	김동민

목 차

1. 이진 트리

이진트리를 이용한 영어사전 프로그램

- **P. 312의 프로그램 8.14를 활용하여 이진 탐색 트리를 이용한 영어 사전 프로그램을 작성하라.**

1.1 문제 분석

조건 1 data.txt파일에 양식, 단어, 뜻이 저장되어 있음

조건 2 I는 데이터를 삽입, d는 삭제, s는 데이터의 탐색을 의미

조건 3 p는 전체 데이터의 출력, q는 프로그램 종료를 의미

조건 4 중복되는 데이터가 삽입되는 경우 기존의 단어에 뜻이 추가되어야 함. 뜻은 연결 리스트의 형태로 크기에 따라 동적할당으로 구현

- 이 문제는 전에 배웠던 리스트를 트리 노드 안에 삽입하여 활용하는 문제이다. 원래의 트리 알고리즘은 같은 단어가 있다면 삽입하지 않았지만 이번 문제는 같은 단어가 있으면 원래 있던 노드의 리스트에 단어의 의미를 삽입한다.

이를 구현하기 위해 트리 노드마다 단어와 리스트가 존재한다. 트리 안의 링크드 리스트는 단어의 의미를 나타내는 문자열을 가지는 리스트이다. 트리의 삽입 연산 외에 검색, 삭제, 출력 알고리즘 또한 구현한다.

파일에는 I, d, s, p, q의 단어가 있고 이것으로 무슨 동작을 할지 명령을 한다. I는 삽입, d는 삭제, s는 검색, p는 출력 동작을 수행한다. q는 바로 프로그램을 종료한다. 여기서 검색은 반복으로 수행하는데, 코드의 길이가 그리 길지 않고 더 빠르기 때문에 반복을 사용한다.

데이터 파일에서 삽입은 I 뒤에 입력할 단어, 의미가 있고 검색과 삭제 연산은 단어만 뒤에 있다. p는 출력만 하기 때문에 p만 존재한다. 삭제와 검색연산은 트리를 돌며 같은 단어 값이 존재한다면 삭제 또는 검색을 하고 어떤 값을 삭제 검색했는지 화면에 출력한다.

모든 동작이 끝나면 트리와 리스트 전체를 돌며 동적할당한 메모리를 해제할 수 있도록 한다.

1.2 소스 코드

- 분할 컴파일로 작성

- common.h (공통 헤더파일)

```c
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동리
    과제 명: 이진트리를 이용한 영어사전 프로그램
*/
//공통으로 사용하는 헤더파일(링크리스트와 트리 선언)
#ifndef COMMON_H              //전처리기 COMMON_H를 define하지 않았다면 endif전까지 포함
#define COMMON_H              //중복 정의를 피하기 위함

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TRUE 1
#define FALSE 0

typedef struct ListNode {           //리스트 노드 구조체
    char *mean;                     //단어
    struct ListNode *link;          //다음 노드를 가리키는 링크
}ListNode;
typedef struct ListType {           //리스트타입
    ListNode*head;                  //가장 앞을 가리키는 head
    ListNode *tail;                 //가장 끝노드를 가리키는 tail
}ListType;

typedef struct Element {            //파일로부터 입력받을때와 키 값을 넘겨줄 때 사용할 구조체
    char *word;
    char *mean;
}Element;

typedef struct TreeNode {           //트리노드
    char *word;                     //단어 문자열 포인터
    ListType *mean_list;            //의미 링크드리스트
    struct TreeNode *left, *right;  //왼쪽 노드와 오른쪽 노드를 가리키는 포인터
}TreeNode;

#endif
```

- Tree_func.h (트리 함수 헤더파일)

```c
//트리 함수 헤더파일

#ifndef TREE_FUNC_H
#define TREE_FUNC_H
#include "common.h"                         //공통 헤더파일 common.h포함

int compare(char *s1, char *s2);            //문자열 비교 함수
void Print_Tree(TreeNode *, int);           //트리전체 출력함수
TreeNode * new_node(Element);               //새로운 노드 생성 함수
TreeNode * Insert_Tree_Node(TreeNode*, Element);        //노드 삽입 함수
TreeNode* delete_node(TreeNode*, char* key, int *check);    //노드 1개 삭제 함수
TreeNode *min_value_node(TreeNode*);                        //이진 탐색 트리에서 최소 키값을 가지는 노드 찾음
TreeNode* search(TreeNode*, char*);                         //이진 트리 탐색 함수
int Get_Node_Count(TreeNode*);                              //노드의 개수를 반환하는 함수
void Delete_Tree(TreeNode*);                                //트리 노드를 전체 삭제하는 함수

#endif // !TREE_FUNC_H
```

- List_func.h (리스트 함수 헤더파일)

```c
//링크리스트 함수 헤더파일
#ifndef LIST_FUNC_H
#define LIST_FUNC_H
#include "common.h"

void Insert_List_Node(ListType*, char*);    //리스트에 삽입하는 함수
ListType* create( );                        //리스트 생성 함수
void Print_List(ListType*);                 //리스트 출력 함수
void Delete_List(ListType*);                //리스트 삭제함수

#endif // !LIST_FUNC_H
```

- main.c (main함수 구현)

```c
// 메인함수 소스코드
#include "common.h"
#include"List_func.h"
#include"Tree_func.h"
int Interface(FILE*, TreeNode**, char c);

int main() {
    FILE *fp;
    TreeNode *root=NULL;              //트리 노드
    char c;

    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open₩n");
        return 0;
    }
    while (!feof(fp)) {
        fscanf(fp, "%c", &c);         //데이터로부터 i,d,s,p값들에 하나 읽어옴
        if (!Interface(fp, &root, c))break;     //Interface에 값 전달 만약 q가 입력되었다면 프로그램 종료
    }
    Delete_Tree(root);                //트리 노드 전체 삭제
    fclose(fp);
    return 0;
}
int Interface(FILE*fp , TreeNode**root, char c) {                   //메인함수로부터 전달받은 값에 따라 맞는 출력문과 함수 호출
    char word[10], mean[10];
    int check = FALSE;
    TreeNode *tmp;                                                  //임시 트리 노드 변수
    Element element;                                                    //삽입 시 트리에 전달할 키값
    switch (c) {
    case 'q':
        printf("%c₩n", c);
        return FALSE;
    case 'i':                     //삽입 연산
        printf("%c₩n", c);
        fscanf(fp, "%s%s", word, mean);
        element.word = (char*)malloc(sizeof(char)*(strlen(word) + 1));      //입력받은 문자열 길이 +1로 동적할당
        element.mean = (char*)malloc(sizeof(char)*(strlen(mean) + 1));
        strcpy(element.word, word);                                 //문자열 복사
        strcpy(element.mean, mean);
        *root = Insert_Tree_Node(*root, element);                       //트리 노드에 삽입
        printf("₩n");
        break;
    case 'd':                     //삭제 연산
        check = FALSE;                                              //삭제할 단어가 존재하는지 확인하는 변수
        printf("%c₩n", c);
        fscanf(fp, "%s", word);                                     //삭제할 단어 입력
        *root = delete_node(*root, word, &check);                       //삭제 연산 수행
        if (!check) printf("'삭제할 단어 %s이(가) 존재하지 않습니다.₩n", word);  //단어가 존재하지 않은다면
        printf("₩n");
        break;
    case 's':                     //검색 연산
        printf("%c₩n", c);
        fscanf(fp, "%s ", word);

        tmp = search(*root, word);                          //검색 연산 수행결과를 tmp에 담음
        if (tmp == NULL) {                                  //만약 검색한 단어가 없다면 NULL이 리턴으로 메시지 출력하고 바로 종료
            printf("%s 단어가 존재하지 않습니다.₩n₩n", word);
            break;
        }
        printf("단어: %s₩n", tmp->word);                    //검색된 단어 출력
        printf("뜻미: ");
        Print_List(tmp->mean_list);                         //단어의 뜻이 여러가지 일 수도 있으므로 리스트 출력
        printf("₩n₩n");
        break;
    case 'p':
        printf("%c₩n", c);                                  //트리노드 전체 출력
        Print_Tree(*root, Get_Node_Count(*root));           //트리노드가 출력되서 뜬 리스트도 같이 출력됨
        printf("₩n₩n");
        break;

    }
    return TRUE;
}
```

- List_func.c (리스트 함수 구현)

```c
//링크드리스트 함수 코드 작성
#include "List_func.h"                  //리스트 함수 헤더 포함

ListType* create() {                    //리스트 생성함수
    ListType*plist = (ListType*)malloc(sizeof(ListType));
    plist->head = plist->tail = NULL;       //리스트의 처음과 끝을 나타내는 head, tail값을 NULL
    return plist;
}
void Insert_List_Node(ListType*plist, char* key) {          //리스트의 끝에 삽입하는 함수
    ListNode*temp = (ListNode*)malloc(sizeof(ListNode));
    temp->mean = key;
    temp->link = NULL;
    if (plist->tail == NULL)                            //만약 tail값이 NULL이면 가장 첫 노드에 삽입하므로 temp를 담음
        plist->head = plist->tail = temp;
    else {
        plist->tail->link = temp;                   //tail의 다음 링크가 temp를 가리킴
        plist->tail = temp;                         //tail에 temp값 복사
    }
}
void Print_List(ListType*plist) {                   //리스트 전체를 출력하는 하뭇
    ListNode*p = plist->head;
    while (p->link != NULL) {                       //리스트끝의 전 노드까지 출력
        printf("%s, ", p->mean);                    //리스트 데이터 출력
        p = p->link;
    }
    printf("%s", p->mean);                          //마지막 노드는 ,가 출력되면 안되므로 따로 출력
}
void Delete_List(ListType*plist) {                  //리스트 전체를 삭제하는 함수
    ListNode*p, *q;
    p = q = plist->head;                            //p와 q는 head값부터 시작
    printf("삭제 리스트 : ");
    while (p->link != NULL) {
        printf("%s, ", p->mean);                        //삭제된 데이터 출력
        q = q->link;                                //q를 먼저 다음링크로 옮김
        free(p->mean);                              //동적할당한 의미(mean)문자열 먼저 해제
        free(p);                                    //p노드 해제
        p = q;                                      //q의 위치를 p에 복사

    }
    printf("%s₩n", p->mean);
    free(p->mean);
    free(p);
    free(plist);                                    //리스트 메모리 해제
}
```

- Tree_func.c (트리 함수 구현)

```c
//트리 함수 소스 파일
#include "Tree_func.h"
#include "List_func.h"                  //트리함수에서 리스트 함수를 사용하므로 헤더파일을 포함시켜줌

int compare(char* s1, char *s2) {       //문자열 비교함수
    return strcmp(s1, s2);              //두개 문자열이 같으면 0, 앞문자 크면 양수, 뒷문자 크면 음수를 리턴함
}
int Get_Node_Count(TreeNode*node) {     //노드의 갯수를 계산하는 함수
    int count = 0;
    if (node != NULL)                           //재귀적 서브트리에서 사용하기 편리한 방법 프로그램 사용
        count = 1 + Get_Node_Count(node->left) + Get_Node_Count(node->right); //자식노드 갯수 1을 더하여 리턴

    return count;
}
TreeNode* new_node(Element key) {                       //새로운 트리노드를 생성하는 함수
    TreeNode* temp = (TreeNode*)malloc(sizeof(TreeNode));
    printf("단어 : %s\n", key.word);                    //트리에 저장할 key값 출력
    printf("뜻풀이 : %s\n", key.mean);
    temp->word = key.word;                              //트리노드 단어에 key값 넣기 후
    temp->mean_list = create();                         //링크드 리스트를 생성하고
    insert_List_Node(temp->mean_list, key.mean);        //key값에서 뜻풀이를 넣기전에 삽입하면 안됨
    temp->left = temp->right = NULL;                    //temp의 왼쪽과 오른쪽 값을 NULL로 설정
    return temp;
}
TreeNode* insert_Tree_Node(TreeNode*node, Element key) {        //트리 노드에 삽입하는 함수
    if (node == NULL) return new_node(key);                 //만약 노드가 존재하지 않는다면 새로운 노드 만듦

    if (compare(key.word, node->word) < 0)                  //key가 노드보다 단어상 왼쪽에 가짐
        node->left = insert_Tree_Node(node->left, key);
    else if (compare(key.word, node->word) > 0)             //오른쪽 트리로서 가짐
        node->right = insert_Tree_Node(node->right, key);
    else {                                                  //만약 두개값이 둘 같이 존재한 노드가 있다면
        insert_List_Node(node->mean_list, key.mean);        //링크드리스트 뒤쪽에 뜻풀이를 삽입
        printf("단어 : %s\n", key.word);
        printf("뜻풀이 : %s\n", key.mean);
    }
    return node;                                            //만들은 루트노드를 리턴
}
TreeNode *min_value_node(TreeNode*node) {               //주어진 왼쪽트리에서 최소 키값을 찾음
    TreeNode*current = node;

    while (current->left != NULL)                       //왼쪽 노드가 NULL일때까지 반복
        current = current->left;                        //왼쪽 노드 방향으로 이동
    return current;
}
TreeNode* delete_node(TreeNode*root, char* key, int *check) {   //트리 노드 1개 삭제 함수
    if (root == NULL)return root;

    if (compare(key, root->word) < 0)                   //찾는 key값이 노드보다 더 왼쪽 서브트리(사전상 앞쪽) 방향으로 이동
        root->left = delete_node(root->left, key, check);
    else if (compare(key, root->word) > 0)              //찾는 key값이 노드보다 더 높은 서브트리 오른쪽으로 이동
        root->right = delete_node(root->right, key, check);
    else {                                              //key값과 노드 값이 같을때

        if (root->left == NULL) {                       //삭제하려는 노드가 한개노드일 경우 또는 하나는 서브트리가 있지만 하나만 없을 경우
            TreeNode*temp = root->right;                //temp에 오른쪽 노드 저장
            if (!*check) {
                printf("삭제 단어 : %s\n", root->word); //삭제 단어 출력
                Delete_List(root->mean_list);           //리스트 삭제 함수 호출
                free(root->word);                       //단어 동적할 공간 제거
            }
            free(root);                                 //루트노드 제거
            *check = TRUE;
            return temp;
        }
        else if (root->right == NULL) {                 //찾는 오른쪽 노드가 NULL이라면
            TreeNode*temp = root->left;                 //temp에 왼쪽 노드 저장

            if (!*check) {
                printf("삭제 단어 : %s\n", root->word); //삭제 단어 출력
                Delete_List(root->mean_list);           //리스트 삭제 함수 호출
                free(root->word);                       //단어 동적할 공간 제거
            }
            free(root);
            *check = TRUE;                              //check가 삭제할 노드가 출력이 됐었는지 확인
            return temp;
        }
        else {
            printf("삭제 단어 : %s\n", root->word);     //삭제하려는 노드가 두개의 서브트리를 가지고 있을 경우
            Delete_List(root->mean_list);               //리스트 삭제함수
            free(root->word);                           //단어 삭제
            *check = TRUE;

            TreeNode *temp = min_value_node(root->right);   //오른쪽 서브트리에서 가장 작은 값 찾음
            root->word = temp->word;                    //root노드에 temp의 정보만을 값 복사
            root->mean_list = temp->mean_list
            root->right = delete_node(root->right, temp->word, check);  //후계 노드 삭제
        }
    }
    return root;
}
TreeNode* search(TreeNode*node, char* key) {            //트리 노드 검색 함수(단어)
    TreeNode*p = node;
    while (p != NULL) {                                 //p가 NULL이 아닐때까지 반복
        if (compare(key, p->word) == 0)                 //찾고자 하는값 일치하면 검색 성공
            return p;
        else if (compare(key, p->word) < 0)             //찾는 key값이 노드보다 왼쪽서브트리 방향노드로 이동
            p = p->left;
        else if (compare(key, p->word) > 0)             //찾는 key값이 노드보다 높은서브트리 오른쪽노드로 이동
            p = p->right;
    }
    return p;                                           //찾는 노드를 끝내 못찾은 NULL리턴
}
void Print_Tree(TreeNode *node, int count) {            //트리 노드 출력 함수(후위 순회 방식)
    static int num = 0;                                 //각각 단어를 num으로(노드의 갯수 세기용)
    if (node != NULL) {
        Print_Tree(node->left, count);                  //왼쪽 노드 출력
        Print_Tree(node->right, count);                 //오른쪽 노드 출력
        printf("%s ", node->word);                      //단어 출력 후
        Print_List(node->mean_list);                    //링크드리스트 출력함수를 리스트 단어 출력
        num++;
        if (num % count) printf(", ");                  //한줄당수 num에 찍는 노드 갯수는 각 인풋에서 ,출력
        else if (num == count) num = 0;                 //찾는 노드갯수가 입력받은 num을 넘어 다르로 넘김
    }
}
void Delete_Tree(TreeNode* node) {                      //트리 노드 전체를 삭제하는 함수
    if (node != NULL) {
        Delete_Tree(node->left);                        //왼쪽 트리 삭제
        Delete_Tree(node->right);                       //오른쪽 트리 삭제
        printf("삭제 노드 : %s\n", node->word);
        Delete_List(node->mean_list);                   //먼저 링크드리스트 삭제함수 호출
        free(node->word);                               //단어 동적할 공간노드 제거
        free(node);                                     //노드 제거

    }
}
```

1.3 소스 코드 분석

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    파일 명: 이진트리를 이용한 영어사전 프로그램
*/
//공통으로 사용하는 헤더파일(링크리스트와 트리 선언)
#ifndef COMMON_H                //전처리기 COMMON_H를 define하지 않았다면 endif전까지 포함
#define COMMON_H                //중복 정의를 피하기 위함

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TRUE 1
#define FALSE 0
```

1. 소스코드를 작성한 날짜, 이름, 프로그램명을 작성하고, 필요한 헤더를 포함한다. 이번 코드는 분할컴파일을 이용하여 프로그래밍 했기 때문에 전처리기 조건문을 이용하여 구분하였다.

2. ifndef는 정의되어있지 않으면 발생하는 전처리문으로 COMMON_H가 정의되어있지 않으면 endif전까지 포함한다. 이는 중복으로 헤더파일이 include되지 않기 위해서 사용했다.

3. 참을 나타내는 TRUE와 FALSE를 정의한다.

```
typedef struct ListNode {
    char *mean;
    struct ListNode *link;
}ListNode;
typedef struct ListType {
    ListNode*head;
    ListNode *tail;
}ListType;

typedef struct Element {
    char word[10];
    char mean[10];
}Element;

typedef struct TreeNode {
    char *word;
    ListType *mean_list;
    struct TreeNode *left, *right;
}TreeNode;

#endif
```

4. 링크드 리스트 노드와 리스트를 정의한다.
노드는 의미를 나타내는 mean 문자열 포인터와 다음 노드를 나타내는 link로 구성되고 리스트는 처음을 나타내는 head와 끝을 나타내는 tail포인터로 구성된다.

5. 삽입 연산에서 파일로부터 입력을 받고 저장할 자료형을 선언한다.
Element 구조체 자료형에는 단어 문자열과 의미 문자열이 존재하는데 트리로 키 값을 옮길 때 사용한다.

6. 트리의 각 항을 하나의 노드로 표현한다. 트리의 변수에는 단어 문자열과 링크드리스트 타입으로 의미가 존재한다.
왼쪽 노드와 오른쪽 노드를 가리키는 left, right링크가 있다.

```
//트리 함수 헤더파일

#ifndef TREE_FUNC_H
#define TREE_FUNC_H
#include "common.h"

int compare(char *s1, char *s2);
void Print_Tree(TreeNode *, int);
TreeNode * new_node(Element);
TreeNode * Insert_Tree_Node(TreeNode*, Element);
TreeNode* delete_node(TreeNode*, char* key, int *check);
TreeNode *min_value_node(TreeNode*);
TreeNode* search(TreeNode*, char*);
int Get_Node_Count(TreeNode*);
void Delete_Tree(TreeNode*);

#endif // !TREE_FUNC_H
```

7. 분할컴파일로 작성한 트리의 함수를 선언한 헤더파일이다.
공통 헤더파일인 common.h헤더를 포함한다.

8. 필요한 함수를 선언한다.

- compare함수는 두 개의 문자열을 사전식 순서로 비교하고 반환한다.
- Print_Tree함수는 트리를 후위순서로 출력하는 함수이다.
- new_node함수는 새로운 노드를 생성하는 함수이다.
- Insert_Tree_node함수는 입력받은 값을 트리에 삽입하는 함수이다.
- delete_node함수는 문자열에 맞는 트리 노드를 삭제하는 함수이다.
- min_value_node함수는 주어진 이진트리에서 최소 키값을 가진 노드를 반환한다.

- search함수는 주어진 문자열에 맞는 노드를 찾아 노드를 반환한다.
- Get_Node_Count함수는 이진트리의 노드 개수를 찾아 반환한다.
- Delete_Tree함수는 트리 노드 전체를 삭제하는 함수이다.

```
//링크리스트 함수 헤더파일
#ifndef LIST_FUNC_H
#define LIST_FUNC_H
#include "common.h"

void Insert_List_Node(ListType*, char*);
ListType* create();
void Print_List(ListType*);
void Delete_List(ListType*);

#endif // !LIST_FUNC_H
```

9. 분할컴파일로 작성한 링크드리스트의 함수를 선언한 헤더파일이다. 공통 헤더파일인 common.h헤더를 포함한다.

10. 필요한 함수를 선언한다.

- Insert_List_Node함수는 링크리스트에 문자열을 삽입하는 함수이다.
- create함수는 리스트를 새로 생성하는 함수이다.
- Print_List함수는 리스트 전체를 순서대로 출력하는 함수이다.
- Delete_List함수는 리스트를 삭제하는 함수이다.

```
// 메인함수 소스코드
#include "common.h"
#include"List_func.h"
#include"Tree_func.h"
int Interface(FILE*, TreeNode**, char c);

int main() {
    FILE *fp;
    TreeNode *root=NULL;
    char c;

    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open₩n");
        return 0;
    }
    while (!feof(fp)) {
        fscanf(fp, "%c", &c);
        if (!Interface(fp, &root, c))break;
    }
    Delete_Tree(root);
    fclose(fp);
    return 0;
}
```

11. 분할컴파일로 작성한 메인함수 소스코드이다.
메인함수에서는 어떤 헤더가 사용되는지 알기 위하여 3개의 헤더파일을 모두 include하였다. 전처리기로 처리했기 때문에 중복 선언을 피할 수 있다.

12. 필요한 변수들을 선언한다.
- fp는 파일포인터, root는 이진트리를 나타낸다.
- c는 파일로부터 어떤 명령이 내려졌는지 입력받는 변수이다.

13. 파일 포인터를 읽기 형식으로 오픈하고 feof함수를 이용하여 파일 끝까지 입력받고 interface함수에 전달한다. interface함수에는 파일포인터와 이진트리의 주소 값, 입력받은 문자를 전달한다.

14. interface함수는 만약 q가 전달되었다면 FALSE를 리턴한다. q는 프로그램 종료를 의미하므로 interface에서 FALSE가 리턴되면 입력을 멈추고 프로그램을 종료한다.

15. 모든 입력이 끝나면 트리를 삭제하는 Delete_Tree함수를 호출하여 동적할당한 메모리를 모두 해제할 수 있도록 한 후 파일포인터를 닫고 프로그램을 종료한다.

```
int Interface(FILE*fp , TreeNode**root, char c) {
    char word[10];
    int check = FALSE;
    TreeNode *tmp;
    Element element;
    switch (c) {
    case 'q':
        printf("%c₩n", c);
        return FALSE;
    case 'i':
        printf("%c₩n", c);
        fscanf(fp, "%s%s", element.word, element.mean);
        *root = Insert_Tree_Node(*root, element);
        printf("₩n");
        break;
```

16. Interface함수는 입력받은 문자에 따라 맞는 동작을 수행하는 함수이다 변수로는 단어 문자열 배열, 삭제 시 노드가 존재하는지 확인하기 위한 check 변수, 검색한 단어를 전달받을 tmp, 파일로부터 입력받고 삽입할 Element형 element변수가 존재한다.

17. 입력받은 c문자에 따라 switch case문으로 맞는 함수를 호출한다.
만약 q가 입력되었다면 프로그램 종료를 의미하므로 FALSE를 반환한다.

18. 만약 I가 입력되었다면 트리에 삽입을 의미한다.
fscanf로 element변수에 단어와 의미 배열에 이를 입력받은 뒤 트리에 삽입하는 Insert_Tree_Node함수를 호출하여 입력받은 element값을 전달한다.

```
    case 'd':                           //삭제 연산
        check = FALSE;
        printf("%c₩n", c);
        fscanf(fp, "%s", word);
        *root = delete_node(*root, word, &check);
        if (!check) printf("삭제할 단어 %s이(가) 존재하지 않습니다.₩n", word);
        printf("₩n");
        break;
    case 's':                           //검색 연산
        printf("%c₩n", c);
        fscanf(fp, "%s ", word);

        tmp = search(*root, word);
        if (tmp == NULL) {
            printf("%s 단어가 존재하지 않습니다.₩n₩n", word);
            break;
        }
        printf("단어: %s₩n", tmp->word);
        printf("의미: ");
        Print_List(tmp->mean_list);                                    //
        printf("₩n₩n");
        break;
    case 'p':
        printf("%c₩n", c);                                             //
        Print_Tree(*root, Get_Node_Count(*root));                      //
        printf("₩n₩n");
        break;

    }
    return TRUE;
}
```

19. 만약 d가 입력되었다면 트리 노드의 삭제연산을 의미한다.
d 뒤에는 삭제할 단어가 입력되어 있으므로 삭제할 단어를 더 입력받는다.
delete_node함수를 호출하여 입력받은 단어를 전달한다.
만약 check에 FALSE값이 저장되어있다면 삭제할 단어가 존재하지 않는다는 메시지를 출력하고, 존재한다면 삭제를 수행한다.

20. s가 입력되었으면 트리에서 입력받은 문자열을 검색한다.
search함수는 검색해서 찾은 노드를 리턴하는데 만약 찾은 노드가 존재하지 않는다면 NULL을 리턴한다.
만약 NULL이 리턴되면 단어가 트리에 존재하지 않는다는 메시지를 출력하고 break로 switch문을 빠져나온다.

21. 트리에 문자열이 존재한다면 그 노드에 있는 단어정보와 의미 정보를 출력하는데, 의미는 리스트 형식이기 때문에 리스트를 출력하는 Print_List 함수로 출력을 한다.

22. 만약 p가 입력되었다면 트리 노드 전체를 출력하는데, 트리를 출력하는 함수인 Print_Tree함수를 호출하고 트리 노드의 개수를 알려주는 Get_Node_Count함수를 전달한다. 이것을 전달하는 이유는 트리 개수에 맞춰 마지막 노드는 ,표시가 출력되지 않게 하기 위함이다.

23. q를 제외한 나머지는 프로그램이 계속 수행되도록 해야 하기 때문에 TRUE를 리턴한다.

```
ListType* create( ) {                    //리스트 생성함수
    ListType*plist = (ListType*)malloc(sizeof(ListType));
    plist->head = plist->tail = NULL;
    return plist;
}
void Insert_List_Node(ListType*plist, char* key) {
    ListNode*temp = (ListNode*)malloc(sizeof(ListNode));
//  temp->mean = key;
    temp->mean = (char*)malloc(sizeof(char)*(strlen(key) + 1));
    strcpy(temp->mean, key);
    temp->link = NULL;
    if (plist->tail == NULL)
        plist->head = plist->tail = temp;
    else {
        plist->tail->link = temp;
        plist->tail = temp;
    }
}
```

24. 분할 컴파일로 작성한 링크드 리스트 함수를 구현한 부분이다.
여기서는 링크드 리스트만 사용하기 때문에 List_func헤더만 포함한다.

25. create함수는 리턴타입이 ListType*으로, 리스트를 생성하여 main함수의 리스트에 리턴하는 함수이다.

26. 리스트를 동적할당하여 생성하고, 리스트의 첫 번째 노드를 가리키는 head와 끝을 나타내는 tail을 NULL로 설정한 후 이를 리턴한다.

27. Insert_List_Node함수는 리스트 끝에 노드를 추가하는 함수이다.
먼저 노드를 동적할당한 후 노드의 mean에 전달받은 값을 삽입하고 다음 노드를 나타내는 link에 NULL값을 설정한다.
메인함수에서 동적할당하지 않고 여기서 동적할당을 하는 이유는 나중에 트리 노드를 삭제할 때 같은 주소를 가지고 있어 노드삭제가 제대로 되지 않는 문제가 생겼었기 때문이다.
의미 문자열의 길이 +\0문자열의 길이로 +1의 값으로 동적할당 한다.

28. 만약 연결리스트의 테일 값이 NULL이라면 연결리스트에 노드가 하나도 없는 것이므로 맨 앞을 가리키는 헤드와 끝을 가리키는 테일에 temp노드를 삽입한다.

29. tail은 맨 끝 노드를 가리키므로 tail의 다음노드가 temp를 가리키도록 하고 그러면 temp가 끝 노드가 되므로 tail이 temp를 가리키도록 한다.

```
void Print_List(ListType*plist) {
    ListNode*p = plist->head;
    while (p->link != NULL) {
        printf("%s, ", p->mean);
        p = p->link;
    }
    printf("%s", p->mean);
}
void Delete_List(ListType*plist) {
    ListNode*p, *q;
    p = q = plist->head;
    printf("삭제 리스트 : ");
    while (p != NULL) {
        if(p->link!=NULL)printf("%s, ", p->mean);
        else printf("%s", p->mean);
        q = q->link;
        free(p->mean);
        free(p);
        p = q;

    }
    printf("₩n");
    free(plist);
}
```

30. Print_List는 리스트 안에 있는 mean 값을 출력하는 함수이다.
가장 마지막 노드의 뒤에는 ,표시가 출력되면 안 되므로 가장 마지막 노드의 전까지만 출력한 후 마지막 노드는 따로 출력할 수 있도록 한다.

31. Delete_List함수는 리스트 전체를 삭제하는 함수이다.
먼저 p와 q가 리스트의 헤더를 가리킬 수 있도록 한다.

32. p가 끝까지 돌며 리스트의 노드를 삭제하는데 삭제된 mean값을 출력한다. 여기서도 마지막 노드에는 ,표시가 출력되지 않도록 하기 위해 if문으로 구분하여 출력을 했다.

33. 리스트 노드 p리스트의 head부터 반복하며 노드를 하나씩 삭제한다.
q는 다음 노드를 가리키고 현재 노드인 p를 free를 이용하여 삭제한다.
p를 해제하기 전에는 먼저 동적할당한 mean문자열을 삭제한다.
그리고 삭제한 p에 다음 노드를 나타내는 q을 전달하여 반복문을 돌 수 있도록 한다.

34. 모든 노드의 삭제가 완료되었다면 처음에 create함수를 이용하여 만들었던 리스트 타입까지 동적할당 해제하여 할당을 해제할 수 있도록 한다.

```
 //트리 함수 코드 작성
#include "Tree_func.h"
 #include "List_func.h"

int compare(char* s1, char *s2) {
    return strcmp(s1, s2);
}
int Get_Node_Count(TreeNode*node) {
    int count = 0;
    if (node != NULL)
        count = 1 + Get_Node_Count(node->left) + Get_Node_Count(node->right);

    return count;
}
```

35. 분할컴파일로 작성한 트리 함수를 구현한 부분이다.
트리에서는 트리 함수뿐만 아니라 리스트 함수도 사용하기 때문에 리스트 함수 헤더도 포함시켜준다.

36. compare함수는 두 문자열을 사전식 순서로 비교하고, 그 값을 반환하는 함수이다. s1과 s2를 비교하여 만약 사전식 순서로 s1이 더 빨리 나온다면 음수가 반환되고 s1이 더 늦게나온다면 양수가 반환된다.
만약 두 문자열이 같다면 0이 반환된다.

37. Get_Node_Count함수는 이진트리의 노드 개수를 리턴하는 함수이다.
노드의 개수를 세기 위해서는 트리안의 노드들을 전체적으로 순회한다.
각각의 서브트리에 대하여 순환 호출한 다음, 반환되는 값에 1을 더하여 반환한다.

```
TreeNode* new_node(Element key) {
    TreeNode* temp = (TreeNode*)malloc(sizeof(TreeNode));
    printf("단어 : %s₩n", key.word);
    printf("의미 : %s₩n", key.mean);
    temp->word = (char*)malloc(sizeof(char)*(strlen(key.word)+1));
    strcpy(temp->word, key.word);
    temp->mean_list = create( );
    Insert_List_Node(temp->mean_list, key.mean);
    temp->left = temp->right = NULL;
    return temp;
}
TreeNode* Insert_Tree_Node(TreeNode*node, Element key) {
    if (node == NULL) return new_node(key);

    if (compare(key.word, node->word) < 0)
        node->left = Insert_Tree_Node(node->left, key);
    else if (compare(key.word, node->word) > 0)
        node->right = Insert_Tree_Node(node->right, key);
    else {
        Insert_List_Node(node->mean_list, key.mean);
        printf("단어 : %s₩n", key.word);
        printf("의미 : %s₩n", key.mean);
    }
    return node;
}
```

38. new_node함수는 동적으로 메모리를 할당하여 새로운 노드를 성성 후 반환하는 함수이다.
임시 노드 temp에 전달받은 단어를 넣은 후 temp의 단어 문자열 변수를 입력받은 키의 단어 문자열의 크기 +1로 동적할당 한 후 strcpy함수를 통해 이를 복사한다. 그리고 의미를 삽입하기 위해 mean_list를 create함수를 통해 새로 생성한다. 그리고 그 리스트에 의미 문자열을 삽입한다.
그리고 left와 right값을 NULL로 설정한 다음 이를 리턴한다.

39. Insert_Tree_Node함수는 이진트리에 노드를 삽입하는 함수이다. 매개변수로는 트리와 삽입할 key값을 전달받는다. key에는 메인함수에서 동적할당한 단어 포인터와 의미 포인터가 저장되어있다.

40. 만약 트리가 공백이면 새로운 노드를 반환한다. 트리가 공백이 아니라면 순환적으로 트리를 내려가는데, 만약 key문자열과 노드의 문자열을 compare함수로 비교하여 값이 음수가 나온다면 왼쪽으로 내려가고 양수가 나온다면 오른쪽으로 내려간다.
새로운 노드 또는 변경된 루트포인터를 left또는 right에 입력받는다.

41. 만약 중복된 번호를 입력받게 된다면 Insert_List_Node함수를 통해 원래의 노드 리스트에 의미 문자열을 새로 삽입한다.
삽입 연산이 일어날 때마다 어떤 문자열이 삽입되었는지 화면에 출력한다.

```
TreeNode* delete_node(TreeNode*root, char* key, int *check) {
    if (root == NULL)return root;

    if (compare(key, root->word) < 0)
        root->left = delete_node(root->left, key, check);
    else if (compare(key, root->word) > 0)
        root->right = delete_node(root->right, key, check);
```

42. delete_node함수는 트리에서 입력받은 key값을 찾고, 그에 key값을 가지고 있는 노드를 삭제하고 새로운 루트 노드를 반환하는 함수이다.
매개변수로는 트리와 삭제할 단어문자열 key, 문자가 존재하는지 보는 check함수가 있다.

43. 입력받은 단어 key와 노드의 단어를 compare함수를 통해 비교한다.
만약 반환받은 값이 음수가 나온다면 왼쪽으로 노드를 내려가고 양수가 나온다면 오른쪽으로 노드를 내려간다.

```
    else {                                                          //key값과 노드
        if (root->left == NULL) {                                   //삭제려는 노
            TreeNode*temp = root->right;                            //temp에 오른
            if (!*check) {
                printf("삭제 단어 : %s₩n", root->word);                    //삭제
                Delete_List(root->mean_list);                       //리스
                free(root->word);                                           /
            }
            else {
            //  printf("삭제하려는 노드가 두개의 서브트리를 가질때₩n");
            //  printf("아래 노드를 위로 올리고 원래의 아래 노드를 삭제하는 연산₩n");
                printf("삭제 단어 : %s₩n", root->word);                    /
                Delete_List(root->mean_list);
                free(root->word);
            }
            free(root);                                                     /
            *check = TRUE;
            return temp;
        }
```

```
else if (root->right == NULL) {
    TreeNode*temp = root->left;
    if (!*check) {
        printf("삭제 단어 : %s₩n", root->word);
        Delete_List(root->mean_list);
        free(root->word);
    }
    else {
        printf("삭제 단어 : %s₩n", root->word);
        Delete_List(root->mean_list);
        free(root->word);
    }
    free(root);
    *check = TRUE;
    return temp;
```

44. 나머지는 key와 노드 단어 문자열이 같을 때를 의미한다.
위 두 개의 if문은 삭제하려는 노드가 단말 노드이거나 하나의 서브트리만 가지고 있는 경우를 나타낸다.

45. 만약 현재 노드의 왼쪽이 NULL이라면 temp에 오른쪽 노드를 저장하고 오른쪽 노드가 NULL이라면 temp에 왼쪽 노드를 저장한다.

46. 삭제하려는 노드가 단말 노드일 경우는 아래 더 이상의 노드가 존재하지 않으므로 단말 노드만 지우면 된다. 먼저 삭제할 단어를 출력하고 Delete_List함수로 노드의 리스트를 모두 지워준 후 단어도 삭제한다.

47. 여기서 밑의 else연산은 check변수가 TRUE일 때 즉, 두 개의 서브트리를 가졌을 때의 조건으로 위의 부모 노드를 삭제하고 아래노드를 부모노드로 복사한 후 원래 아래노드를 삭제할 때의 연산이다.

48. 삭제하려는 노드가 하나의 서브트리만 가지고 있다면 자기 노드를 삭제 후 서브 트리를 부모 노드에 붙여준다. 노드의 삭제를 한 후 이를 알리는 check변수를 TRUE로 바꾸고 temp를 리턴한다.

```
        else {
            printf("삭제 단어 : %s₩n", root->word);
            Delete_List(root->mean_list);
            free(root->word);
            *check = TRUE;

            TreeNode *temp = min_value_node(root->right);
            ListNode *p = temp->mean_list->head;
            root->word = (char*)malloc(sizeof(char)*(strlen(temp->word) + 1));
            strcpy(root->word, temp->word);
            root->mean_list = create();
            while (p != NULL) {
                Insert_List_Node(root->mean_list, p->mean);
                p = p->link;
            }
        //  root->mean_list = temp->mean_list;
            root->right = delete_node(root->right, temp->word, check);  //후계
        }
    }
    return root;
}
```

49. 나머지는 삭제하려는 노드가 두 개의 서브트리를 가지고 있는 경우를 나타낸다. 먼저 현재 노드를 삭제하고 check변수를 TRUE로 바꾼다.

50. temp는 min_value_node함수를 이용하여 오른쪽 노드의 가장 최소 key값을 리턴받고 이를 root노드에 단어와 리스트 값을 복사한다.

50. 여기서 복사를 수행할 때 주소 값을 바로 복사하는 것이 아니라 새로 동적할당을 하는 이유는 같은 복사한 주소와 원래의 주소가 같은 값을 가리키기 때문에 free를 하면 복사한 주소도 다음과 같이 메모리 해제가 되는 문제가 생긴다.

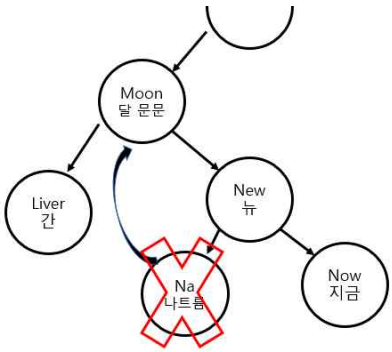
```
d
삭제 단어 : liver
삭제 리스트 : 간

d
삭제 단어 : new
삭제 리스트 : 뉴, 새로운

p
now 지금, na 硼硼硼硼硼?

삭제 트리 : now
삭제 리스트 : 지금
삭제 트리 : na
삭제 리스트 : 硼硼硼硼硼
```

이것이 메인함수 에서 문자열 동적할당을 하지 않고 리스트 삽입 함수에서 동적할당으로 삽입한 이유이다.

51. 삭제한 root의 단어를 temp(오른쪽 노드의 최소 값)의 단어의 길이로 동적할당한 후 복사한다. root의 리스트 또한 새로 생성한 후 temp 의미 리스트의 헤더부터 끝까지 반복하며 이를 삽입한다.

```
d
삭제 단어 : moon
삭제 리스트 : 달, 문문
삭제하려는 노드가 두개의 서브트리를 가질때
아래 노드를 위로 올리고 원래의 아래 노드를 삭제
삭제 단어 : na
삭제 리스트 : 나트륨, 나

p
liver 간, now 지금, new 뉴, na 나트륨, 나, yg
```

52. moon은 밑에 두 개의 서브트리를 가지고 있다. 이를 가지고 테스트 해보면 먼저 moon단어가 제거되고 그 자리에 na단어가 복사된 뒤 아래에 있던 na단어와 의미리스트가 삭제된 것을 볼 수 있다.




```c
TreeNode *min_value_node(TreeNode*node) {
    TreeNode*current = node;

    while (current->left != NULL)
        current = current->left;
    return current;
}
TreeNode* search(TreeNode*node, char* key) {
    TreeNode*p = node;
    while (p != NULL) {
        if (compare(key, p->word) == 0)
            return p;
        else if (compare(key, p->word) < 0)
            p = p->left;
        else if (compare(key, p->word) > 0)
            p = p->right;
    }
    return p;
}
```

53. min_value_node함수는 주어진 이진 탐색 트리에서 최소 키 값을 가지는 노드를 찾아 반환하는 함수이다.
현재 노드를 current노드에 복사를 한 뒤 current의 left가 NULL일때까지 반복한다. 그러면 current에는 가장 작은 키 값을 가지는 노드가 있다.
그리고 current노드를 반환한다.

54. search함수는 이진 탐색트리를 반복을 통해 탐색하는 함수이다.
노드가 NULL이 아닐 때까지 반복을 하게 되는데, 만약 key값과 같은 노드 단어를 비교하여 같은 값을 찾게 되면 그 노드를 반환한다.
만약 key값이 노드의 단어보다 사전 순서로 더 빨리 나온다면 음수가 반환되므로 왼쪽으로 내려간다. 반대로 양수가 나온다면 오른쪽으로 내려간다.

55. 만약 찾는 값이 없다면 p에는 NULL이 저장되어 있기 때문에 NULL이 반환될 것이다.

```
void Print_Tree(TreeNode *node, int count) {
    static int num = 0;
    if (node != NULL) {
        Print_Tree(node->left, count);
        Print_Tree(node->right, count);
        printf("%s ", node->word);
        Print_List(node->mean_list);
        num++;
        if (num < count) printf(", ");
        else if (num == count) num = 0;
    }
}
void Delete_Tree(TreeNode* node) {
    if (node != NULL) {
        Delete_Tree(node->left);
        Delete_Tree(node->right);
        printf("삭제 트리 : %s₩n", node->word);
        Delete_List(node->mean_list);
        free(node->word);
        free(node);

    }
}
```

54. Print_Tree함수는 트리를 후위순회하며 모든 노드를 출력하는 함수이다 매개변수로는 이진트리와 노드의 개수를 전달받는다.
먼저 단어를 출력한 후 Print_List함수를 호출하여 그 노드에 있는 의미 리스트를 출력할 수 있도록 한다.

55. static 정적변수로 num을 선언하고 마지막 노드는, 표시가 되면 안 되므로 노드의 개수보다 작을 때만 ,가 표시되도록 한다. 만약 num과 노드 개수가 같아진다면 다시 num을 0으로 설정한다.

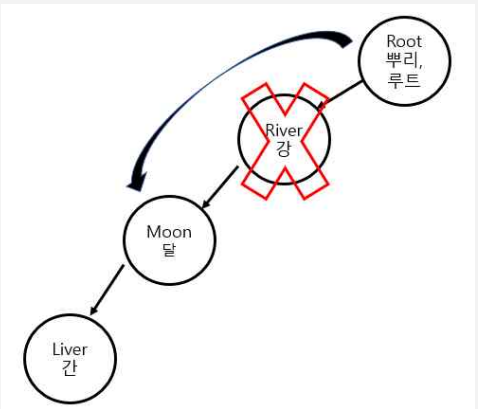
56. Delete_Tree함수는 트리 전체를 순환하며 노드를 삭제하는 함수이다.
삭제 전 어떤 노드를 삭제하는지 단어를 출력한다.

57. 부모 노드를 먼저 삭제하게 되면 그 밑의 노드들은 접근할 수 없기 때문에 가장 아래에 있는 단말 노드부터 삭제한다. 따라서 왼쪽 자식노드, 오른쪽 자식노드를 삭제하고 부모 노드를 삭제한다.
삭제를 할 때는 동적할당한 단어 문자열뿐만 아니라 Delete_List를 통해 의미 링크리스트도 삭제할 수 있도록 한다.

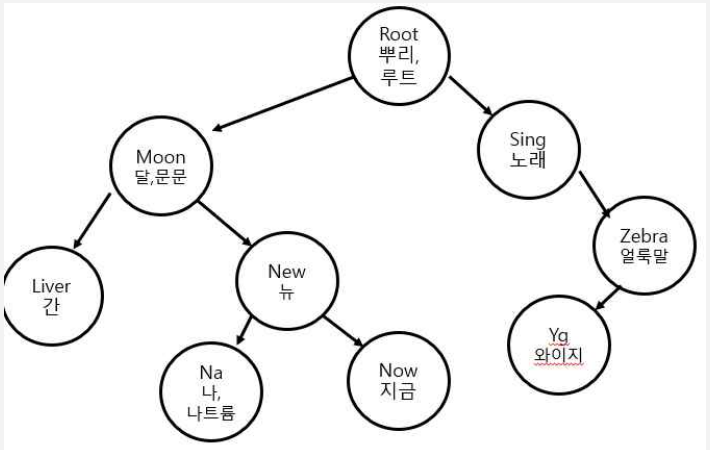
58. 후위순환의 알고리즘과 똑같이 출력했기 때문에, 결과적으로 루트노드가 마지막으로 삭제될 것이고 모든 노드가 메모리 해제될 것이다.

1.4 실행 창

```
d
삭제할 단어 hi이(가) 존재하지 않습니다.

s
hello 단어가 존재하지 않습니다.

i
단어 : root
의미 : 뿌리

i
단어 : river
의미 : 강

i
단어 : moon
의미 : 달

i
단어 : root
의미 : 루트

i
단어 : liver
의미 : 간

d
삭제 단어 : river
삭제 리스트 : 강
```



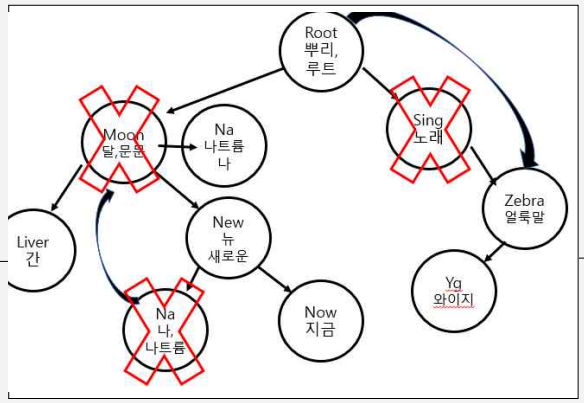

```
p
liver 간, moon 달, root 뿌리, 루트

s
단어: root
의미: 뿌리, 루트

s
단어: liver
의미: 간

p
liver 간, moon 달, root 뿌리, 루트

i
단어 : new
의미 : 뉴

i
단어 : moon
의미 : 문문

i
단어 : now
의미 : 지금

p
liver 간, now 지금, new 뉴, moon 달, 문문, root 뿌리, 루트

i
단어 : sing
의미 : 노래

i
단어 : zebra
의미 : 얼룩말

i
단어 : yg
의미 : 와이지

i
단어 : na
의미 : 나트륨

p
liver 간, na 나트륨, now 지금, new 뉴, moon 달, 문문, yg 와이지, zebra 얼룩말, sing 노래, root 뿌리, 루트

i
단어 : na
의미 : 나
```

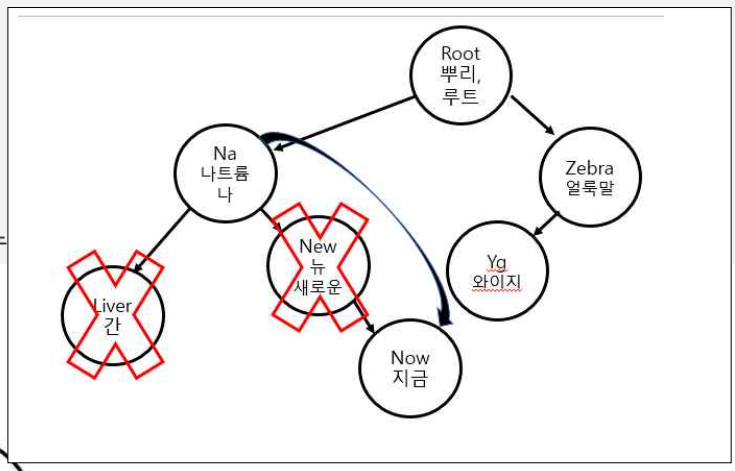


데이터 파일

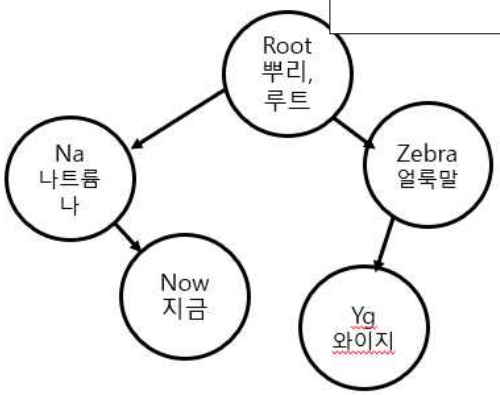
```
d hi
s hello
i root 뿌리
i river 강
i moon 달
i root 루트
i liver 간
d river
p
s root
s liver
p
i new 뉴
i moon 문문
i now 지금
p
i sing 노래
i zebra 얼룩말
i yg 와이지
i na 나트륨
p
i na 나
d moon
p
d sing
i new 새로운
d none
s new
d liver
d new
p
```

```
d
삭제 단어 : moon
삭제 리스트 : 달, 문문
삭제 단어 : na
삭제 리스트 : 나트륨, 나

p
liver 간, now 지금, new 뉴, na 나트륨, 나, yg 와이지, zebra 얼룩말, sing 노래, root 뿌리, 루트

d
삭제 단어 : sing
삭제 리스트 : 노래

i
단어 : new
의미 : 새로운

d
삭제할 단어 none이(가) 존재하지 않습니다.

s
단어: new
의미: 뉴, 새로운

d
삭제 단어 : liver
삭제 리스트 : 간

d
삭제 단어 : new
삭제 리스트 : 뉴, 새로운

p
now 지금, na 나트륨, 나, yg 와이지, zebra 얼룩말, root 뿌리, 루트

삭제 트리 : now
삭제 리스트 : 지금
삭제 트리 : na
삭제 리스트 : 나트륨, 나
삭제 트리 : yg
삭제 리스트 : 와이지
삭제 트리 : zebra
삭제 리스트 : 얼룩말
삭제 트리 : root
삭제 리스트 : 뿌리, 루트

C:₩Users₩dmk46₩OneDrive₩바탕
 종료되었습니다.
이 창을 닫으려면 아무 키나 누
```





최종 트리



1.5 느낀 점

이번 문제는 지금까지 배웠던 링크리스트와 트리를 전부 사용하여 활용하는 문제였습니다. 몇 개월 전부터 계속 해오던 것이기 때문에 빨리 끝낼 수 있을 것이라고 생각했지만 중간에 걸리는 부분이 많아 생각보다 빨리 끝내지 못했던 것 같습니다.

특히 가장 어려웠다고 생각했던 부분은 트리의 삭제 연산이었던 것 같습니다. 처음에는 책에 있는 코드를 응용하여 프로그래밍을 했었는데, 두 개의 서브트리를 가지는 부분에서 부모노드는 삭제가 되었는데, 원래 밑에 있던 노드를 삭제하면 위의 노드도 삭제가 되는 문제가 생겼었습니다. 그래서 원래는 Element자료형도 문자형 포인터였는데 두 번 동적할당하는 것을 막기 위해 이를 배열로 바꾸고 동적할당을 삽입하는 과정에서 하도록 바꾸고 아래로부터 복사를 할 때 원래의 주소 값을 받아오는 것이 아니라 새로 동적할당하여 받을 수 있도록 하였습니다. 이렇게 바꾸니 아래에 있던 노드도 삭제가 되어서 낭비되는 메모리가 없도록 할 수 있었습니다.
이번 코드를 다 작성하니 리스트 함수, 트리 함수, 메인에서 사용할 함수가 선언과 구현부분이 다 섞여서 프로그램을 작성한 저도 읽기가 힘든 코드가 되었었습니다. 이 부분을 해결하기 위해 교수님께 분할컴파일로 작성해도 되는지 여쭈어보았고 가능하다고 해주셔서 분할 컴파일을 이용하여 코드를 짰습니다. 각각 함수의 선언부분과 구현부분을 나누고 공통으로 사용하는 헤더는 또 따로 선언하여 필요한 헤더가 있을 때마다 불러올 수 있도록 프로그래밍을 하였습니다. 분할컴파일을 할 때 그에 대한 정보를 조금 공부해보았고 ifndef와 endif로 헤더가 중복 선언이 되지 않게 할 수 있었습니다.

이번 과제를 프로그래밍하며 제가 포인터에 대해 어느 정도 알고 있다고 생각했었는데 많은 오류가 나올 때는 아직 많이 부족하다는 것을 깨닫게 되었습니다. 특히 삭제함수를 구현할 때는 삭제를 하지 않아도 제대로 출력이 되었기 때문에 그냥 아래노드를 삭제하지 말고 해버릴까 하는 생각도 했습니다. 하지만 계속 여러 방법으로 코드를 짜보았고 마침내 성공할 수 있었습니다. 저는 이 경험을 계속 기억하고 앞으로 항상 자만하지 않고 배운 것을 복습하고 앞으로 배울 내용도 예습하며 수업시간을 잘 따라갈 수 있도록 노력하겠습니다.