이진 트리 – 영어사전 만들기

실습

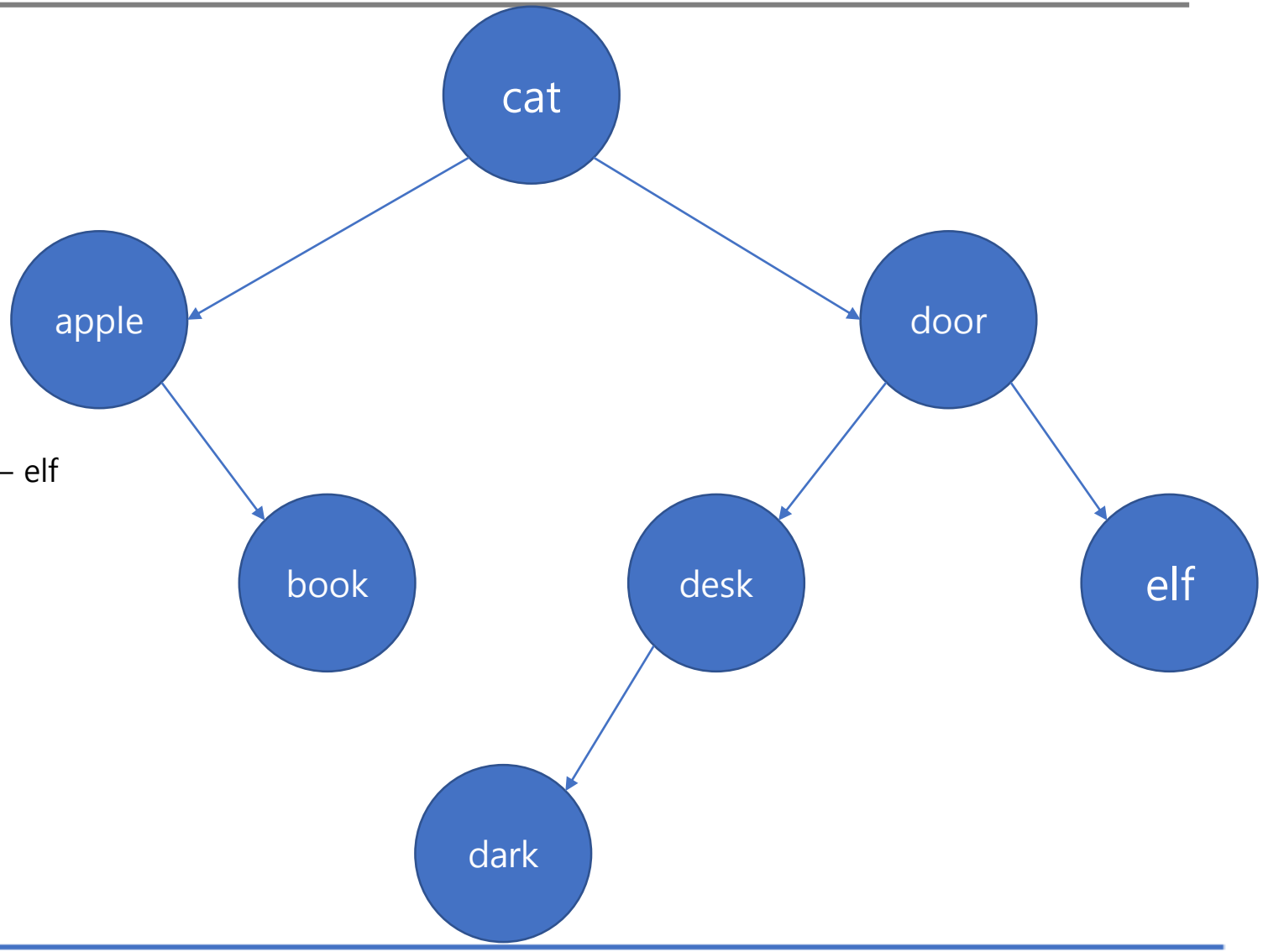
트리(Tree)

- 영어사전 만들기

* 규칙

- 이진 트리 자료구조를 만족함.
- 알파벳 순서대로 트리 구조의 각각의 노드에 영어 단어를 삽입함.
- 각각의 노드의 left에는 더 빠른 영어 단어가, right에는 더 느린 영어 단어가 위치함.
- 각각의 노드에는 영어 단어와 한글 뜻 2개의 정보가 들어 있음.
- main문에서는 단축키를 입력 받아 삽입, 삭제, 출력, 탐색이 모두 가능해야함.

트리(Tree)



- 입력 순서 : cat – apple – book – door – desk – dark – elf

- 출력 : apple – book – cat – dark – desk – door - elf

트리(Tree)

- Tree.h

```
#define MAX_WORD_SIZE 100
#define MAX_MEANING_SIZE 100

typedef struct {
    char word[MAX_WORD_SIZE];    // 영어 단어
    char meaning[MAX_MEANING_SIZE];    // 한글 뜻
}element;

typedef struct TreeNode {
    element key;                // 영어단어 + 한글 뜻
    struct TreeNode *left, *right;
}TreeNode;                  // 트리 구조 내의 각각의 노드
```

트리(Tree)

- **Tree.cpp**

```
// 선택할 메뉴 출력
void help()
{
    printf("**************₩n");
    printf("i: 입력₩n");
    printf("d: 삭제₩n");
    printf("s: 탐색₩n");
    printf("p: 출력₩n");
    printf("q: 종료₩n");
    printf("**************₩n");
}
```

```
// 영어단어의 알파벳 순서 비교
int compare(element e1, element e2)
{
    return strcmp(e1.word, e2.word);
}
```

트리(Tree)

- main.cpp

```
void main()
{
    char command;
    element e;
    TreeNode *root = NULL;
    TreeNode *tmp;

    do {
        help();// 선택할 메뉴 출력
        command = getchar();
        while (getchar() != '₩n' );  // 버퍼 비우기 대용

        switch (command) {
        case 'i' :
            printf("단어:");
            gets_s(e.word, sizeof(e.word));
            printf("의미:");
            gets_s(e.meaning, sizeof(e.meaning));
            insert_node(&root, e);
            break;
        case 'd' :
            printf("단어:");
            gets_s(e.word, sizeof(e.word));
            delete_node(&root, e);
            break;
        case 'p' :
            display(root);
            printf("₩n");
            break;
        case 's' :
            printf("단어:");
            gets_s(e.word, sizeof(e.word));
            tmp = search(root, e);

            if (tmp != NULL)
                printf("의미:%s₩n", tmp->key.meaning);
            break;
            }
        } while (command != 'q');
}
```

영어 단어 추가 ->

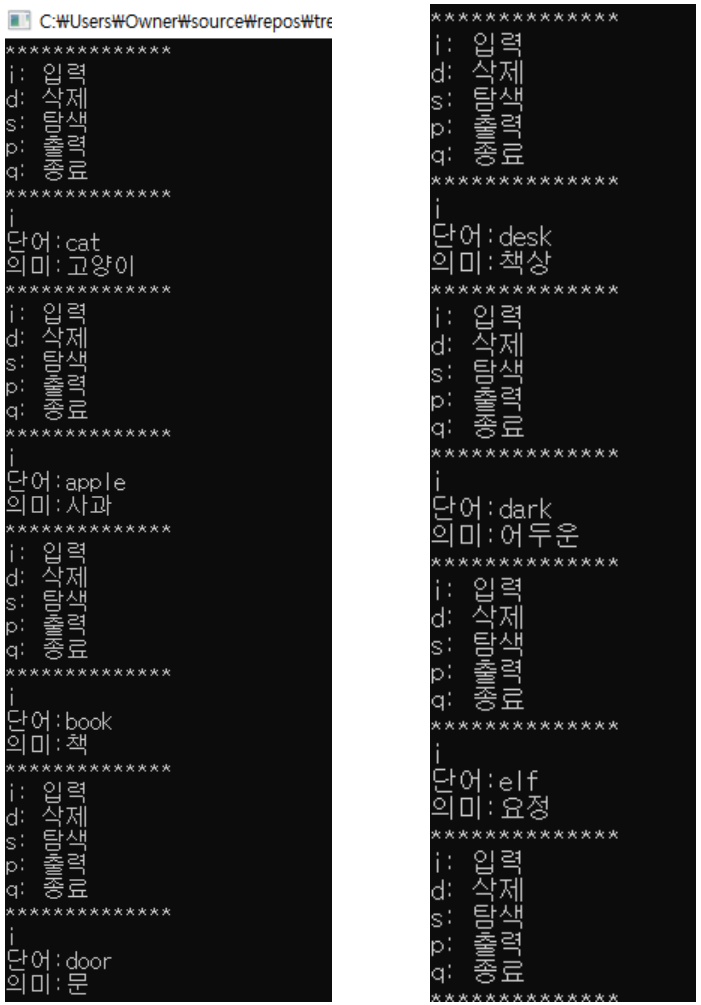
삭제 ->

모든 영어 단어 출력 ->

탐색
(한글 뜻 찾기)->

트리(Tree)

* 입력


C:₩Users₩Owner₩source₩repos₩tre
****
i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료
****
i
단어:cat
의미:고양이
****
i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료
****
i
단어:apple
의미:사과
****
i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료
****
i
단어:book
의미:책
****
i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료
****
i
단어:door
의미:문

****
i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료
****
i
단어:desk
의미:책상
****
i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료
****
i
단어:dark
의미:어두운
****
i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료
****
i
단어:elf
의미:요정
****
i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료
****

* 출력


****
p
apple - book - cat - dark - desk - door - elf -
****

트리(Tree)

* 삭제

```
**************
d
단어:dark
**************
i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료
**************
p
apple - book - cat - desk - door - elf -
**************
i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료
**************
d
단어:book
**************
i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료
**************
p
apple - cat - desk - door - elf -
```

* 탐색

```
****************
s
단어:apple
의미:사과
****************
i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료
****************
s
단어:book
의미:책
****************
```

트리(Tree)

- 소스 코드 링크

https://github.com/ahnsangjae/MY_Workspace/tree/master/%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0/Tree%20(%EC%98%81%EC%96%B4%EC%82%AC%EC%A0%84)

링크에 들어가서 Tree.cpp 에 있는 함수의 정의부를 완성해야함.

void display(TreeNode *p) : 트리에 있는 모든 영어단어를 알파벳 순서대로 출력함.
TreeNode *search(TreeNode *root, element key) : 입력한 영어단어에 맞는 노드를 탐색해서 해당 노드의 주소를 반환함.
void insert_node(TreeNode **root, element key) : 입력한 영어단어를 알파벳 순서에 맞는 위치에 삽입함.
void delete_node(TreeNode **root, element key) : 트리 구조를 유지하면서 입력한 영어단어에 맞는 노드를 삭제함.

트리(Tree)

- **Tree.cpp**

```
void display(TreeNode *p)
{
    if (p != NULL)
    {
        display(p->left);
        printf("%s - ", p->key.word);
        display(p->right);
    }
}
```

```
TreeNode *search(TreeNode *root, element key)
{
    TreeNode *p = root;

    while (p != NULL)
    {
        switch (compare(key, p->key))
        {
        case -1:
            p = p->left;
            break;
        case 0:
            return p;
        case 1:
            p = p->right;
            break;
        }
    }
    return p;
}
```

트리(Tree)

- **Tree.cpp**

```
void insert_node(TreeNode **root, element key)
{
    TreeNode *p, *t;
    TreeNode *n;

    t = *root;
    p = NULL;

    while (t != NULL)
    {
        if (compare(key, t->key) == 0)
            return;

        p = t;

        if (compare(key, t->key) < 0)
            t = t->left;
        else
            t = t->right;
    }

    n = (TreeNode *)malloc(sizeof(TreeNode));

    if (n == NULL)
        return;

    n->key = key;
    n->left = n->right = NULL;

    if (p != NULL)
        if (compare(key, p->key) < 0)
            p->left = n;
        else
            p->right = n;
    else
        *root = n;
}
```

트리(Tree)

- Tree.cpp

```
void delete_node(TreeNode **root, element key)
{
    TreeNode *p, *child, *succ, *succ_p, *t;

    p = NULL;
    t = *root;

    while (t != NULL && compare(t->key, key) != 0)
    {
        p = t;
        t = (compare(key, t->key) < 0) ? t->left : t->right;
    }

    if (t == NULL)
    {
        printf("key is not in the tree");
        return;
    }
```

```
// 단말 노드인 경우
if ((t->left == NULL) && (t->right == NULL))
{
    if (p != NULL)
    {
        if (p->left == t)
            p->left = NULL;
        else
            p->right = NULL;
    }
    else
        *root = NULL;
}
```

트리(Tree)

- Tree.cpp

```
// 하나의 자식만 가지는 경우
else if ((t->left == NULL) || (t->right == NULL))
{
    child = (t->left != NULL) ? t->left : t->right;

    if (p != NULL)
    {
        if (p->left == t)
            p->left = child;
        else
            p->right = child;
    }
    else
        *root = child;
}
```

```
else
{
    succ_p = t;
    succ = t->right;

    while (succ->left != NULL)
    {
        succ_p = succ;
        succ = succ->left;
    }

    if (succ_p->left == succ)
        succ_p->left = succ->right;
    else
        succ_p->right = succ->right;

    t->key = succ->key;
    t = succ;
}

free(t);
}
```