데이터구조 레포트 4

학년	2학년	학번	20211379
이름	장원영	제출일	2024.06.19

일체의 부정 행위 없이 스스로의 노력으로 작성한 레포트임을 확인합니다.

문제 1.

코드 및 주석

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 이진 탐색 트리 노드 구조체 정의
typedef struct TreeNode {
        char* word; // 단어를 저장할 문자열 포인터
        int count; // 단어의 등장 횟수
        int* lines; // 단어가 등장한 줄 번호를 저장할 배열
        int line_count; // 단어가 등장한 줄 번호의 개수
        struct TreeNode* left; // 왼쪽 자식 노드
        struct TreeNode* right; // 오른쪽 자식 노드
} TreeNode;

// 새로운 노드 생성 함수
TreeNode* create_node(const char* word, int line) {
        TreeNode* new_node = (TreeNode*)malloc(sizeof(TreeNode)); // 새로운 노드 메모리 할당
        new_node->word = strdup(word); // 단어를 복사하여 저장
        new_node->count = 1; // 등장 횟수를 1로 초기화
        new_node->lines = (int*)malloc(sizeof(int)); // 줄 번호 배열 메모리 할당
        new_node->lines[0] = line; // 첫 번째 줄 번호 저장
        new_node->line_count = 1; // 줄 번호 개수를 1로 초기화
        new_node->left = new_node->right = NULL; // 자식 노드들을 NULL로 초기화
        return new_node;
}

// 트리에 단어 삽입 함수
TreeNode* insert(TreeNode* root, const char* word, int line) {
        if (root == NULL) {
                return create_node(word, line); // 루트 노드가 없으면 새로운 노드 반환
        }
        int cmp = strcmp(word, root->word);
        if (cmp == 0) {
                root->count++;
                root->lines = (int*)realloc(root->lines, (root->line_count + 1) * sizeof(int)); // 줄 번호 배열 크기 재할당
                root->lines[root->line_count++] = line; // 새로운 줄 번호 저장
        } else if (cmp < 0) {
                root->left = insert(root->left, word, line); // 왼쪽 서브트리에 삽입
        } else {
                root->right = insert(root->right, word, line); // 오른쪽 서브트리에 삽입
        }
        return root;
}

// 트리를 중위 순회하며 출력 함수
void inorder_print(TreeNode* root) {
        if (root != NULL) {
                inorder_print(root->left); // 왼쪽 서브트리 순회
```

```
                printf("단어: %s, 횟수: %d, 줄: ", root->word, root->count); // 단어, 횟수 출력
                for (int i = 0; i < root->line_count; i++) {
                        printf("%d ", root->lines[i]); // 줄 번호 출력
                }
                printf("\n");
                inorder_print(root->right); // 오른쪽 서브트리 순회
        }
}

// 트리 메모리 해제 함수
void free_tree(TreeNode* root) {
        if (root != NULL) {
                free_tree(root->left); // 왼쪽 서브트리 메모리 해제
                free_tree(root->right); // 오른쪽 서브트리 메모리 해제
                free(root->word); // 단어 메모리 해제
                free(root->lines); // 줄 번호 배열 메모리 해제
                free(root); // 노드 메모리 해제
        }
}

// 메인 함수
int main() {
        FILE* fp = fopen("sample.txt", "r"); // 파일을 읽기 모드로 엽니다.
        if (fp == NULL) { // 파일 포인터가 비어있다면
                printf("파일을 찾을 수 없습니다\n"); // 파일을 찾을 수 없다는 경고문 표시
                exit(1); // 프로그램 종료
        }

        TreeNode* root = NULL; // 이진 탐색 트리의 루트 노드 초기화
        char input_string[100]; // 충분히 큰 문자열 버퍼를 선언합니다.
        int line_num = 1; // 현재 줄 번호
        int count = 0; // 총 단어 개수를 저장할 count 변수

        while (fgets(input_string, sizeof(input_string), fp) != NULL) { // 파일에서 한 줄씩 읽기
                char* token = strtok(input_string, " \t\n\r?");
                while (token != NULL) { // 현재 줄에서 더 이상 분리할 토큰이 없을 때까지 반복
                        root = insert(root, token, line_num); // 단어를 트리에 삽입
                        token = strtok(NULL, " \t\n\r?"); // strtok로 문자열 분리 기준 문자를 구분합니다. 탭, 개행, 캐리지 리턴 ?를
문자열 토큰 기준으로 사용합니다.
                }
                line_num++; // 다음 줄 번호로 증가, 각 단어가 어느 줄에서 나왔는지 기록하기 위함
        }

        inorder_print(root); // 트리를 중위 순회하며 출력
        printf("계 : %d", count); // 총합 출력
        free_tree(root); // 트리 메모리 해제

        fclose(fp); // 파일 닫기

        return 0; // 프로그램 종료
}
```

출력 화면 및 설명

```
단어: fantasy, 횟수: 1, 줄: 2
단어: just, 횟수: 1, 줄: 2
단어: life, 횟수: 1, 줄: 1
단어: real, 횟수: 1, 줄: 1
단어: the, 횟수: 1, 줄: 1
단어: this, 횟수: 2, 줄: 1 2
계 : 9%
```

이진 탐색 트리 노드 구조체 정의 (TreeNode)

이진 탐색 트리 노드 구조체는 단어를 저장할 문자열 포인터, 단어의 등장 횟수, 단어가 등장한 줄 번호를 저장할 배열, 단어가 등장한 줄 번호의 개수, 그리고 왼쪽과 오른쪽 자식 노드를 포함합니다. 이 구조체를 사용하여 단어와 그 등장 위치를 트리에 저장합니다.

char* word: 단어를 저장할 문자열 포인터입니다. 동적 메모리 할당을 통해 단어를 저장하고, strdup 함수를 사용하여 문자열의 복사본을 저장합니다. 이를 통해 입력 문자열이 변경되어도 트리에 저장된 단어는 변하지 않습니다.

int count: 단어의 등장 횟수를 저장합니다. 처음 노드를 생성할 때 1로 초기화되고, 단어가 다시 등장할 때마다 증가시킵니다.

int* lines: 단어가 등장한 줄 번호를 저장할 정수 배열입니다. 이 배열은 동적 메모리 할당을 통해 관리되며, 새로운 줄 번호가 추가될 때마다 배열의 크기를 재할당합니다.

int line_count: 단어가 등장한 줄 번호의 개수를 저장합니다. 줄 번호 배열의 크기를 관리하는 데 사용됩니다.

struct TreeNode* left: 왼쪽 자식 노드에 대한 포인터입니다. 이진 탐색 트리의 왼쪽 하위 트리를 가리킵니다.

struct TreeNode* right: 오른쪽 자식 노드에 대한 포인터입니다. 이진 탐색 트리의 오른쪽 하위 트리를 가리킵니다.

새로운 노드 생성 함수 (create_node)

TreeNode *create_node(const char* word, int line) 함수는 새로운 노드를 생성하고 초기화합니다.

먼저, TreeNode *new_node = (TreeNode*)malloc(sizeof(TreeNode))를 통해 새로운 노드의 메모리를 동적으로 할당합니다. 이 할당된 메모리 블록은 TreeNode 구조체 크기만큼입니다.

new_node->word = strdup(word)를 통해 입력된 단어를 복사하여 new_node의 word 필드에 저장합니다. strdup 함수는 문자열을 복사하고 새로운 메모리를 할당하여 반환합니다.

new_node->count = 1로 단어의 등장 횟수를 1로 초기화합니다. 이는 새로운 단어가 처음 등장했음을 나타냅니다.

new_node->lines = (int*)malloc(sizeof(int))로 줄 번호를 저장할 정수 배열을 동적으로 할당합니다. 이 배열은 한 개의 정수를 저장할 수 있습니다.

new_node->lines[0] = line을 통해 첫 번째 줄 번호를 배열에 저장합니다.

new_node->line_count = 1로 줄 번호 개수를 1로 초기화합니다.

new_node->left = new_node->right = NULL로 자식 노드들을 NULL로 초기화합니다. 이는 새로 생성된 노드가 자식 노드가 없는 리프 노드임을 나타냅니다.

트리에 단어 삽입 함수 (insert)

TreeNode *insert(TreeNode* root, const char* word, int line) 함수는 주어진 단어와 줄 번호를 트리에 삽입합니다.

먼저, if (root == NULL) 조건문을 통해 루트 노드가 NULL인지 확인합니다. 루트 노드가 NULL이면 트리가 비어있음을 의미하므로, create_node 함수를 호출하여 새로운 노드를 생성하고 반환합니다.

int cmp = strcmp(word, root->word)를 통해 주어진 단어와 현재 노드의 단어를 비교합니다. strcmp 함수는 두 문자열을 사전 순서로 비교하여 같으면 0, 앞서면 음수, 뒤면 양수를 반환합니다.

if (cmp == 0) 조건문을 통해 두 단어가 같은지 확인합니다. 같으면 현재 노드의 단어가 이미 트리에 존재함을 의미하므로, root->count++로 등장 횟수를 증가시키고, root->lines 배열을 realloc 함수로 재할당하여 줄 번호를 추가합니다.

else if (cmp < 0) 조건문을 통해 주어진 단어가 현재 노드의 단어보다 앞서는지 확인합니다. 앞서면 왼쪽 서브트리에 삽입해야 하므로, root->left = insert(root->left, word, line)을 호출하여 왼쪽 서브트리에 재귀적으로 삽입합니다.

else 조건문을 통해 주어진 단어가 현재 노드의 단어보다 뒤서면 오른쪽 서브트리에 삽입해야 하므로, root->right = insert(root->right, word, line)을 호출하여 오른쪽 서브트리에 재귀적으로 삽입합니다.

마지막으로, 현재 노드 root를 반환합니다. 이를 통해 재귀 호출에서 트리의 루트 노드가 유지됩니다.

트리를 중위 순회하며 출력 함수 (inorder_print)

void inorder_print(TreeNode* root) 함수는 이진 탐색 트리를 중위 순회하며 각 노드의 단어, 등장 횟수, 줄 번호를 출력합니다.

if (root != NULL) 조건문을 통해 현재 노드가 NULL이 아닌지 확인합니다. NULL이면 재귀 호출을 종료합니다.

inorder_print(root->left)를 호출하여 왼쪽 서브트리를 중위 순회합니다. 중위 순회는 왼쪽 자식 -> 현재 노드 -> 오른쪽 자식 순서로 순회합니다.

printf("단어: %s, 횟수: %d, 줄: ", root->word, root->count)를 통해 현재 노드의 단어와 등장 횟수를 출력합니다.

for (int i = 0; i < root->line_count; i++) 루프를 통해 현재 노드의 줄 번호 배열을 순회하며 줄 번호를 출력합니다.

printf("\n")을 통해 줄 바꿈을 추가하여 가독성을 높입니다.

inorder_print(root->right)를 호출하여 오른쪽 서브트리를 중위 순회합니다.

트리 메모리 해제 함수 (free_tree)

void free_tree(TreeNode* root) 함수는 이진 탐색 트리의 모든 노드를 해제하여 동적 할당된 메모리를 반환합니다.

if (root != NULL) 조건문을 통해 현재 노드가 NULL이 아닌지 확인합니다. NULL이면 재귀 호출을 종료합니다.

free_tree(root->left)를 호출하여 왼쪽 서브트리를 재귀적으로 해제합니다.

free_tree(root->right)를 호출하여 오른쪽 서브트리를 재귀적으로 해제합니다.

free(root->word)를 통해 현재 노드의 단어 문자열에 할당된 메모리를 해제합니다.

free(root->lines)를 통해 현재 노드의 줄 번호 배열에 할당된 메모리를 해제합니다.

free(root)를 통해 현재 노드의 구조체에 할당된 메모리를 해제합니다.

메인 함수 (main)

FILE* fp = fopen("sample.txt", "r")를 통해 파일을 읽기 모드로 엽니다. 파일 포인터 fp는 파일의 시작 위치를 가리킵니다.

if (fp == NULL) 조건문을 통해 파일 포인터가 NULL인지 확인합니다. 파일이 열리지 않으면 printf("파일을 찾을 수 없습니다\n")을 출력하고 exit(1)로 프로그램을 종료합니다.

TreeNode* root = NULL을 통해 이진 탐색 트리의 루트 노드를 NULL로 초기화합니다. 이는 빈 트리를 나타냅니다.

char input_string[100]을 선언하여 파일에서 읽어올 한 줄의 문자열을 저장할 버퍼를 만듭니다. 이 버퍼는 충분히 큰 크기로 설정합니다.

int line_num = 1을 통해 현재 줄 번호를 1로 초기화합니다. 이는 파일의 첫 번째 줄을 나타냅니다.

int count = 0을 선언하여 단어 개수를 저장할 변수를 초기화합니다.

while (fgets(input_string, sizeof(input_string), fp) != NULL) 루프를 통해 파일에서 한 줄씩 읽어옵니다. 파일의 끝에 도달할 때까지 이 루프가 반복됩니다.

char* token = strtok(input_string, " \t\n\r?")를 통해 현재 줄을 구분 문자(공백, 탭, 개행, 캐리지 리턴, 물음표)를 기준으로 분리하여 첫 번째 단어를 추출합니다.

while (token != NULL) 루프를 통해 현재 줄에서 더 이상 분리할 토큰이 없을 때까지 반복합니다. 각 단어를 이진 탐색 트리에 삽입합니다.

root = insert(root, token, line_num)를 호출하여 단어와 줄 번호를 트리에 삽입합니다.

token = strtok(NULL, " \t\n\r?")를 호출하여 다음 단어를 추출합니다.

count++; 를 통해 insert 한 만큼 카운트를 증가함으로서 문제에서 요구하는 단어의 총합 카운트 로직을 수행합니다.

line_num++을 통해 줄 번호를 증가시킵니다. 이는 각 단어가 어느 줄에서 나왔는지를 기록하기 위함입니다.

inorder_print(root)를 호출하여 트리를 중위 순회하며 각 단어와 그 등장 위치를 출력합니다.

printf("계 : %d", count); 를 통해 총합 카운트를 출력합니다.

free_tree(root)를 호출하여 트리의 모든 노드를 해제하고 동적 할당된 메모리를 반환합니다.

fclose(fp)를 호출하여 파일을 닫습니다. 이는 파일에 대한 모든 작업이 완료되었음을 나타냅니다.

return 0을 통해 프로그램을 정상 종료합니다. 0은 일반적으로 성공적인 실행을 나타냅니다.