프로그래밍 과제 #2 : 단어 빈도 세기

목표: 주어진 파일 문서에 포함된 단어들의 빈도 수를 출력하는 프로그램 작성

- 이진 탐색 트리(Binary Search Tree (BST))를 이용
- 영어 단어만 처리
- 문서의 각 단어를 차례대로 읽어들여 BST에 삽입
  BST에 해당 단어가 이미 있는 경우, 삽입하지 않고 빈도 수(count)만 증가
- BST 대신 배열 리스트 버전과 성능 비교

입력:

- 파일 이름 입력

출력:

1. 각 단어별 빈도 출력(단어는 알파벳 순으로 출력)
2. 총 단어 수 (노드 수) 출력
3. 트리 높이 출력
4. 수행시간 출력(출력에 소요된 시간 제외)

입출력 예:

파일 이름: sample1.txt
a 1 alice 2 and 2 bank 1 beginning 1 book 2 but 1 by 1 conversations 2 do 1 get 1 had 2 having 1 her 2 in 1 into 1 is 1 it 2 no 1 nothing 1 of 3 on 1 once 1 or 3 peeped 1 pictures 2 reading 1 she 1 sister 2 sitting 1 the 3 thought 1 tired 1 to 2 twice 1 use 1 very 1 was 2 what 1 without 1
노드 개수 = 40
트리 높이 = 16

0.000000 초입니다.

참고:

- 교재 프로그램 **insert_node()를 수정하여, update_BST()**로 작성:
  기존 node가 있는 경우 count 필드값 증가
  기존 node가 없는 경우 새 node 삽입(count = 1)
- 교재 알고리즘 (노드 개수 구하기)
- 교재 알고리즘 (트리 높이 구하기)
- inorder()에서 단어와 빈도수 출력하도록 수정

word-freq-sequential.c (배열 리스트 버전)

```
// 배열 리스트를 사용한 단어 빈도

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <ctype.h>
#include <time.h>

#define MAX_FNAME           100
#define MAX_WORDS           10000
#define MAX_WORD_SIZE       100
#define TRUE                1
#define FALSE               0

// 데이터 형식
typedef struct {
    char    word[MAX_WORD_SIZE];        // 키필드
    int     count;
} element;

// 디스크 파일로부터 한 개 단어를 읽어 배열 word에 저장
// 성공적으로 읽었으면 TRUE, 그렇지 않으면(파일 끝) FALSE 리턴
int getword(FILE *fd, char *word)
{
    char    ch;
    int     i;

    do {
        ch = getc(fd);
        if (ch == EOF)
            return FALSE;
    } while (!isalpha(ch)); // 첫번째 알파벳 문자가 나올때까지 읽는다.
    i = 0;
    do {    // 단어 1개를 읽는다.
        ch = tolower(ch);   // 소문자로 변환
        word[i++] = ch;
        ch = getc(fd);
    } while (isalpha(ch));  // 알파벳 문자이면 계속 읽는다.
    word[i] = '\0';
    return TRUE;
}

// wlist 배열에 단어가 없는 경우 추가, 이미 있는 경우 빈도(count)만 갱신
int update_wlist(int *max_words, element wlist[], char *word)
{
    int     i;

    // word가 wlist에 들어 있는지 찾아서 count 증가
    for (i = 0; i < *max_words; i++) {
        if (strcmp(wlist[i].word, word) == 0) {
            wlist[i].count++;
            return wlist[i].count;
        }
```

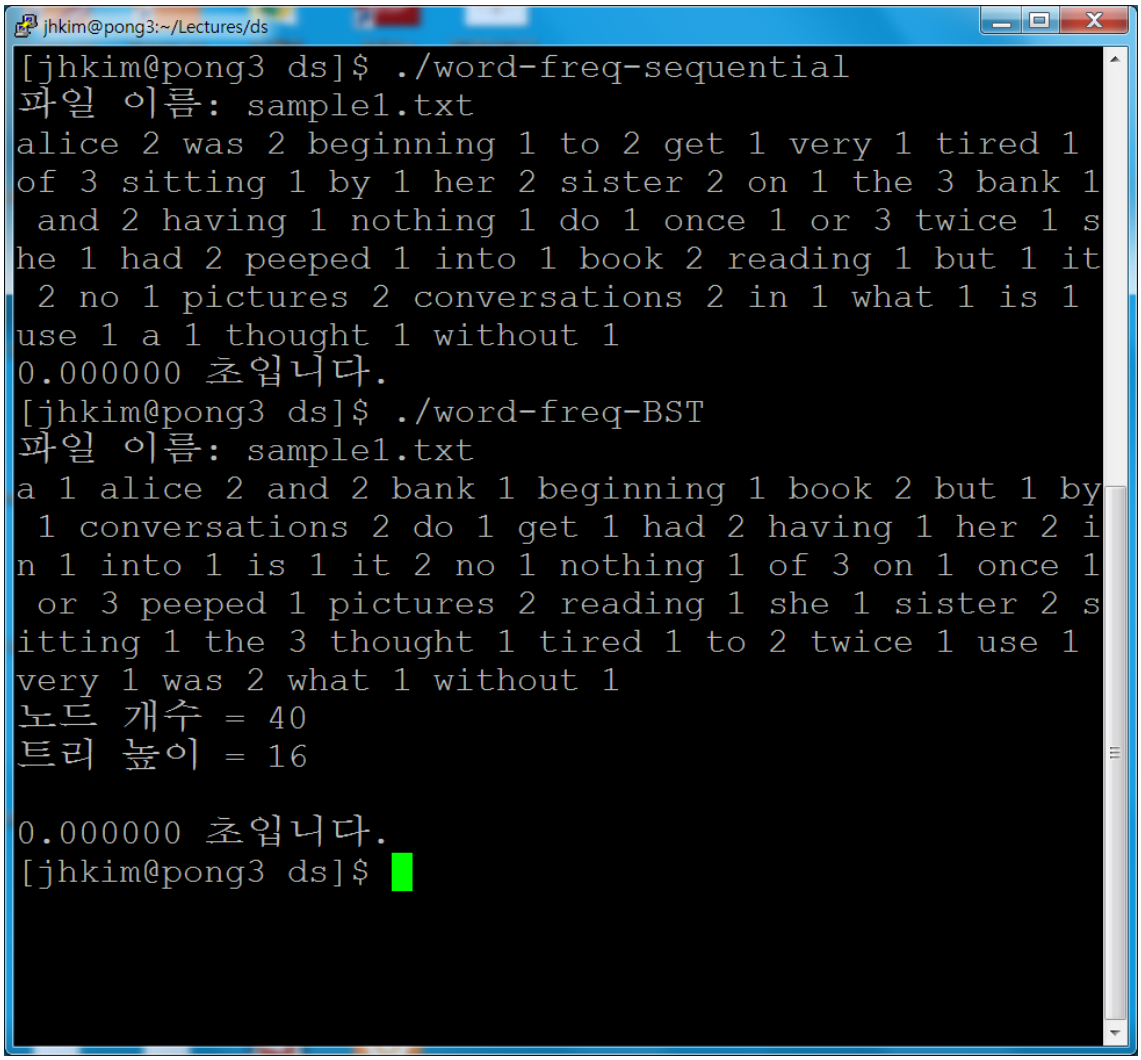
```
    }
    if (i >= MAX_WORDS) {
        fprintf(stderr, "Error: too many words ...\n");
        exit(1);
    }
    // 새 word인 경우 wlist에 추가
    strcpy(wlist[i].word, word);
    wlist[i].count = 1;
    (*max_words)++;
    return wlist[i].count;
}

// 배열 리스트를 사용하는 단어 빈도 계산
void main()
{
     FILE            *fd;
     element         wordlist[MAX_WORDS];
     char            fname[MAX_FNAME];
     char            w[MAX_WORD_SIZE];
     int             flag;
     int             wnum, cnt;
     int             i;
    clock_t         start, finish;
    double          duration;

     printf("파일 이름: ");
     scanf("%s", fname);
     if( (fd = fopen(fname,"r")) == NULL ) {
         fprintf(stderr, "파일을 열수없습니다.\n");
         return;
     }
    wnum = 0;
    cnt = 0;
    start = clock();      // 시간 측정 시작
    do {
        flag = getword(fd, w);   // 단어 1개 읽기
        if (flag == FALSE)
            break;
        cnt = update_wlist(&wnum, wordlist, w); // wordlist update
//          printf(" %s(%d)", w, cnt);
    } while(1);
    finish = clock();     // 시간 측정 종료
    duration = (double)(finish - start) / CLOCKS_PER_SEC;
    for (i = 0; i < wnum; i++) {
        printf("%s %d ", wordlist[i].word, wordlist[i].count);
    }
    printf("\n%.6f 초입니다.\n", duration);
}
```

word-freq-BST.c (이진 탐색 트리 버전)

```
// 이진 탐색 트리를 사용한 단어 빈도

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <ctype.h>
#include <time.h>

#define MAX_FNAME           100
#define MAX_WORDS           10000
#define MAX_WORD_SIZE       100
#define TRUE                1
#define FALSE               0

// 데이터 형식
typedef struct {
    char    word[MAX_WORD_SIZE];                    // 단어
    int     count;
} element;
// 노드의 구조
typedef struct TreeNode {
        element key;
        struct TreeNode *left, *right;
} TreeNode;

// 만약 e1 < e2 -> 음수 반환 (시스템에 따라: -1)
// 만약 e1 == e2 -> 0  반환
// 만약 e1 > e2 -> 양수 반환 (시스템에 따라: 1)
int compare(element e1, element e2)
{
        return strcmp(e1.word, e2.word);
}

// 디스크 파일로부터 한 개 단어를 읽어 배열 word에 저장
// 성공적으로 읽었으면 TRUE, 그렇지 않으면(파일 끝) FALSE 리턴
int getword(FILE *fd, char *word)
{
    char    ch;
    int     i;

    do {
        ch = getc(fd);
        if (ch == EOF)
            return FALSE;
    } while (!isalpha(ch)); // 첫번째 알파벳 문자가 나올때까지 읽는다.
    i = 0;
    do {    // 단어 1개를 읽는다.
        ch = tolower(ch);   // 소문자로 변환
        word[i++] = ch;
        ch = getc(fd);
    } while (isalpha(ch));  // 알파벳 문자이면 계속 읽는다.
    word[i] = '\0';
    return TRUE;
```

```
}

TreeNode * new_node(element elem)
{
//
}

// Binary Search Tree에 삽입하려는 단어가 없는 경우 추가, 이미 있는 경우에는 빈도(count)만 증가
TreeNode * update_BST(TreeNode *root, element key)
{
// 교재 프로그램 insert_node() 참고하여 작성

}

// inorder traversal
void inorder(TreeNode *t)
{
// 방문한 노드의 단어와 count 출력
}

// 노드 개수 세기
int get_node_count(TreeNode *node)
{
//
}

// 최대값 return
int max(int a, int b)
{
      return (a>b)?a:b;
}

// 트리 높이 계산
int get_height(TreeNode *node)
{

}

// Binary Search Tree를 사용하여 단어 빈도 파악
void main()
{
            FILE            *fd;
            element         e;
            TreeNode        *root=NULL;
            TreeNode        *tmp;
            char            fname[MAX_FNAME];       // 파일 이름
            char            w[MAX_WORD_SIZE];     // 읽어들인 단어
            int             flag;                   // 파일 끝이 아닌지 여부
          clock_t         start, finish;
            double          duration;

         // . . .

}
```

입출력 예:



질의:

1. sample2.txt와 sample3.txt 각각에 대해,
   - 다음 단어들의 빈도는?

     would, yes, you, your
   - 단어 종류의 총 개수는?
   - 트리의 높이는?
2. sample3.txt에 대해 배열 리스트 버전과 이진 탐색 트리 버전의 수행 시간을 비교하시오.

제출자료:

1. 소스코드 (실습사이트 [프로그래밍과제]에 소스코드를 온라인으로 제출)
2. 보고서 (실습사이트 [프로그래밍과제]에 문서 파일 제출)

   보고서에 포함되어야 할 내용:

   A. 소스코드 및 설명
      - 소스코드 자체가 주석을 포함하고 있어야 함
      - 주석문과는 별개로 보고서에서 소스코드에 대해 설명

   B. 실행 과정을 보여주는 화면(화면 캡쳐)
      - 다양한 입출력에 대한 실행 화면을 포함해야 함

   C. 동작방법 및 동작 여부 설명

   D. 질의에 대한 답변 및 설명