1. 단어 출현 횟수 출력. 이 프로그램은 *The C Programming Language* 책의 6.5절 Self-referntial Structures의 첫번째 예제 프로그램을 교육을 목적으로 자세한 설명을 곁들여 CWEB으로 다시 작성한 것으로 파일을 입력으로 받아서 그 파일에 있는 모든 단어의 출현 횟수를 출력하는 프로그램입니다.

입력 파일에 어떠한 단어들이 들어있는지 미리 알 수 없기때문에 단어들을 알파벳 순으로 나열 할 수는 없어서 파일을 읽어나가는 중간중간에 단어들을 알파벳 순으로 정렬하기 위해서 대표적인 자료 구조인 이진 트리를 이용해서 문제를 해결합니다.

이 프로그램은 놀랄정도로 짧으며, 전체적인 모양새는 다음과 같습니다.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#define MAXWORD 100
⟨ 이진 트리 노드 구조 정의 6 ⟩
⟨ 전역 선언 4 ⟩
⟨ 프로그램에 사용되는 함수 3 ⟩
main( )
{
  struct tnode *root;      /* 단어를 저장할 이진 트리 */
  char word[MAXWORD];      /* 단어, 단어의 최대 길이는 MAXWORD */
  root = Λ;
  while (getword(word, MAXWORD) ≠ EOF)      /* 단어를 읽어들여라 */
    if (isalpha(word[0]))
      root = addtree(root, word);      /* 단어를 트리에 적절하게 저장하라 */
  treeprint(root);      /* 트리에 저장된 단어를 횟수와 함께 출력하라 */
  return 0;
}
```

2. 단어별로 읽어들이기. 먼저 입력 스트림으로부터 단어를 선별하는 함수부터 작성하겠습니다. *getword*( ) 함수는 주어진 입력을 단어별로 다루기 위해서, 입력 스트림으로부터 단어를 빼내는 함수입니다. 여기서 단어란 글자(letter)로 시작하면서 글자와 숫자로 이루어진 문자열이거나 한 자짜리 공백 문자가 아닌 문자(character)입니다. 이 함수의 반환값은 방금 읽어들인 단어의 맨 앞 문자이거나 파일의 끝을 나타내는 `EOF` 혹은 단어가 한 문자로 이루어졌을 때, 그 한 문자 자체입니다.

3. 함수 *getword*( )는 다음과 같습니다: 위의 마디에서 정의했듯이, 글자로 시작한다는 단어의 정의에 따라서 단어의 첫 문자가 글자인지 아닌지를 확인하기 위해서 우선 입력 스트림으로부터 문자 한 개를 읽어들입니다. 즉 단어의 첫 문자가 글자이면, 단어 정의를 만족하므로 계속해서 그 다음 문자를 읽어들일 것이며, 아니면 방금 읽어들인 문자를 반환할 것입니다.

이 때 읽어들인 문자 공백 문자이면 무시하고 공백 문자가 아닐 때까지 문자 한 개씩 계속 읽어들입니다. 그러다가 공백 문자가 아닌 문자를 만났을 때, 그 문자가 파일의 끝을 나타내는 문자, `EOF`인지 확인을 하고, 파일의 끝이면 `EOF`를 반환하고, 아니면 일단 그 문자를 단어 *word*에 저장합니다. 그런데 방금 단어에 저장한 문자가 글자가 아니면, 더 이상 입력을 받아들이지 않고, 방금 읽어들인 글자가 아닌 문자를 반환하고, 다음 단어를 읽어들일 준비를 합니다.

⟨ 프로그램에 사용되는 함수 3 ⟩ ≡

```
int getword(char *word, int lim)
{
  int c, getch(void);
  void ungetch(int);
  char *w = word;
  while (isspace(c = getch( )))      /* 공백 문자는 모두 건너뛰어라 */
    ;
  if (c ≠ EOF)      /* 파일의 끝이 아니면 */
    *w++ = c;      /* 일단 한 문자를 읽어들여서 단어에 저장하라 */
  if (¬isalpha(c)) {      /* 방금 단어 버퍼로 읽어들인 문자가 글자가 아니면 */
    *w = '\0';      /* 그 문자를 이 함수의 반환값으로 하여 반환하라 */
    return c;
  }
  for ( ; --lim > 0; w++)      /* 단어의 첫 문자가 글자이면, 그 다음 글자를 계속 입력 받아라 */
    if (¬isalnum(*w = getch( ))) {      /* 후속 문자가 글자나 숫자가 아니면 */
      ungetch(*w);      /* 방금 읽어들인 문자를 다시 버퍼에 집어 넣어라 */
      break;      /* 한 단어 입력이 끝났다 */
    }
  *w = '\0';
  return word[0];
}
```

5, 7, 9, 10번 마디도 살펴보라.

이 코드는 1번 마디에서 사용된다.

4. 함수 *getword*( )에서 입력 스트림으로부터 한 문자씩 읽어들이기 위해서, 함수 *getch*( )와 *ungetch*( )가 사용되었습니다. 이 프로그램은 내부적으로 *buf* 라는 충분한 길이의 버퍼를 가지고 있습니다.

```
#define BUFSIZE 100
```

⟨ 전역 선언 4 ⟩ ≡

```
char buf[BUFSIZE];      /* ungetch 함수를 위한 버퍼 */
int bufp = 0;           /* buf를 스캔하기 위한 변수 */
```

8번 마디도 살펴보라.

이 코드는 1번 마디에서 사용된다.

5. 함수 *getch*( )는 우선 *buf* 버퍼에 문자가 들어있으면, 그 버퍼에서 문자를 가져오고, 그렇지 않으면, 표준 입출력 함수인 *getchar*( )를 이용해서 입력스트림으로부터 한 문자씩 읽어들입니다. *buf* 와 같은 버퍼가 왜 필요한 것일까요? *getword*( ) 함수는 입력 스트림에서 한 문자씩 읽어들이면서 단어를 결정합니다. 충분한 길이의 문자열을 읽어들인 후에 그것이 단어인지 아니지를 결정합니다. 즉 주어진 단어의 길이보다 한 문자 더 읽어들여야 방금 읽어들인 문자를 보고 현재까지 읽어들인 문자들이 단어를 구성하는지 아닌지를 알 수 있습니다. 따라서 필요보다 한 문자를 더 읽어들였으므로 이 더 읽어들인 문자를 *buf* 버퍼에 저장해 두었다가 다음 문자를 읽어들일 때, 그 버퍼에 저장된 문자 부터 읽어들입니다. 필요 이상으로 읽어들인 문자를 버퍼에 저장할 때 사용하는 함수가 바로 *ungetch*( )입니다.

⟨ 프로그램에 사용되는 함수 3 ⟩ +≡

```
int getch(void)
{
  return (bufp > 0) ? buf[--bufp] : getchar( );
}
void ungetch(int c)
{
  if (bufp ≥ BUFSIZE) printf("ungetch: too many characters\n");
  else buf[bufp++] = c;
}
```

6. 이진 트리. 지금까지 입력 스트림으로부터 단어를 만들어내는 방법에 대해서 알았습니다. 이제부터는 그 단어를 저장하는 이진 트리의 구조와 이진 트리에 단어를 저장하는 방법을 살펴보겠습니다. 트리의 노드 구조는 다음과 같습니다.

⟨ 이진 트리 노드 구조 정의 6 ⟩ ≡
```
struct tnode {
  char *word;      /* 노드에 저장되는 단어를 가리키는 포인터 */
  int count;       /* 노드에 저장된 단어의 출현 횟수 */
  struct tnode *left;      /* 왼쪽 자식 트리를 가리킴 */
  struct tnode *right;     /* 오른쪽 자식 트리를 가리킴 */
};
```
이 코드는 1번 마디에서 사용된다.

7. 함수 *addtree*( )는 읽어들인 단어를 트리에 저장하는 함수입니다. 이 함수는 트리를 다루는 대개의 함수가 그렇듯이 재귀함수입니다. 새로 들어오는 단어가 트리의 루트에 있는 단어보다 단어순으로 작을 때는 왼쪽트리에 클때는 오른쪽 트리에 저장합니다. 그러다가 이미 트리에 있는 단어일 때는 그 단어 노드의 횟수를 하나 증가시킵니다. 이 함수를 잘 보시면 재귀 함수의 아름다움을 느끼실 수 있습니다.

⟨ 프로그램에 사용되는 함수 3 ⟩ +≡
```
struct tnode *addtree(struct tnode *p, char *w)
{
  int cond;
  if (p ≡ Λ) {      /* 트리에 없는 새로운 단어가 들어왔을 때 */
    p = talloc( );      /* 단어를 저장할 새로운 노드를 만들어라 */
    p→word = mystrdup(w);      /* 단어 크기에 해당하는 공간을 만들어라 */
    p→count = 1;
    p→left = p→right = Λ;
  } else if ((cond = strcmp(w, p→word)) ≡ 0) {      /* 단어를 알파벳순으로 비교 */
    p→count++;      /* 이미 있는 단어이면 횟수를 하나 증가시켜라 */
  } else if (cond < 0) {
    p→left = addtree(p→left, w);      /* 왼쪽 자식 트리에 저장하라 */
  } else {
    p→right = addtree(p→right, w);      /* 오른쪽 자식 트리에 저장하라 */
  }
  return p;
}
```

8. 단어를 트리에 저장하기 위해서는 그 단어를 포함하는 트리 노드를 만들어야 합니다. 노드를 생성하는 함수가 *talloc*( )이고, 트리에 포함되는 단어에 해당하는 공간을 만드는 함수가 *mystrdup*( )입니다. 이렇게 생성된 단어는 *p*→*word*가 가리키게됩니다.

⟨ 전역 선언 4 ⟩ +≡
```
struct tnode *talloc(void);
char *mystrdup(char *);
```

9. ⟨ 프로그램에 사용되는 함수 3 ⟩ +≡

```
struct tnode *talloc(void)
{
  return (struct tnode *) malloc(sizeof(struct tnode));
}
char *mystrdup(char *s)
{
  char *p;
  p = (char *) malloc(strlen(s) + 1);
  if (p ≠ Λ)
    strcpy(p, s);
  return p;
}
```

10. 이제까지의 과정으로 파일의 모든 단어를 단어의 알파벳 순위에 맞게 이진 트리를 구성하였습니다. 남은 것은 이렇게 완성된 트리를 순회하면서 단어와 그의 출현 횟수를 출력하는 것입니다. 함수 *treeprint*( )가 그 일을 하며, 이 함수 역시 재귀 함수이고, 트리의 순회 방법 중에서 inorder 방법을 택하고 있습니다. 즉 왼쪽 자식 트리를 방문하고, 자신의 노드를 방문하고 마지막으로 오른쪽 자식 트리를 방문합니다. 이 순회 방법을 택함으로써, 단어들을 알파벳 순으로 나열할 수 있습니다.

⟨ 프로그램에 사용되는 함수 3 ⟩ +≡

```
void treeprint(struct tnode *p)
{
  if (p ≠ Λ) {
    treeprint(p→left);
    printf("%4d␣%s\n", p→count, p→word);
    treeprint(p→right);
  }
}
```

11. 프로그램 실행. 명령행에서 "gcc -o wordtree wordtree.c"와 같은 명령으로, wordtree라는 실행 파일을 만듭니다. 그리고 sample.txt가 다음과 같은 내용일 때,

```
Literate programming is a methodology that combines a programming
language with a documentation language, thereby making programs more
robust, more portable, more easily maintained, and arguably more fun
to write than programs that are written only in a high-level
language. The main idea is to treat a program as a piece of
literature, addressed to human beings rather than to a computer. The
program is also viewed as a hypertext document, rather like the World
Wide Web. (Indeed, I used the word WEB for this purpose long before
CERN grabbed it!) This book is an anthology of essays including my
early papers on related topics such as structured programming, as well
as the article in The Computer Journal that launched Literate
Programming itself. The articles have been revised, extended, and
brought up to date.
```

명령 "cat sample.txt |./wordtree"를 실행하면, 아래와 같은 결과를 얻습니다.

```
 1 CERN
 1 Computer
 1 I
 1 Indeed
 1 Journal
 2 Literate
 1 Programming
 4 The
 1 This
 1 WEB
 1 Web
 1 Wide
 1 World
 8 a
 1 addressed
 1 also
 1 an
 2 and
 1 anthology
 1 are
 1 arguably
 1 article
 1 articles
 5 as
 1 been
 1 before
 1 beings
 1 book
 1 brought
 1 combines
 1 computer
 1 date
 1 document
 1 documentation
 1 early
 1 easily
 1 essays
 1 extended
 1 for
 1 fun
 1 grabbed
 1 have
 1 high
 1 human
 1 hypertext
 1 idea
 2 in
 ......
```