
Computer Programming: Skills & Concepts (CP1)

Case Study 5: Counting Words

John Longley

17th November, 2008



Task

List the words occurring in a piece of text, along with their frequencies. E.g.

da da da dum

3 da

1 dum

Subtasks include:

- Reading a word from input
- Checking if word already seen, and updating count
- Adding new words
- Writing out word list, when done reading

Some useful functions

```
int ReadWord () ;
    /* reads a word from input, reports success/failure */
int InWordList () ;
    /* returns i if current word appears at index i
       in table, or -1 if absent */
void IncrementCount (int i) ;
    /* increments frequency count of word at index i */
void AddToList () ;
    /* adds current word to wordlist */
void WriteList() ;
    /* prints the results */
```

The main function

```
int main(void) {  
    int index;  
    while (ReadWord()) {  
        index = InWordList();  
        if (index != -1)  
            IncrementCount(index);  
        else  
            AddToList();  
    }  
    WriteList();  
}
```

Assumes the current word and the frequency table are stored in *global variables*.

Some useful types and global variables

```
#define MAX_WORDS 2000          /* max number of words */
#define MAX_WORD_LENGTH 30     /* max length of a word */

typedef char Word_t[MAX_WORD_LENGTH + 1];

typedef struct {
    Word_t word;
    int count;
} ListEntry_t;

ListEntry_t entries[MAX_WORDS]; /* a table of words/counts */
int wordCount = 0;             /* current number of words */
Word_t currentWord;           /* word being processed */
```

Subtask: checking if a word is already listed

```
int InWordList() {
    int i;
    for (i = 0; i < wordCount; ++i) {
        if (strcmp(entries[i].word, currentWord) == 0) {
            return i; /* found */
        }
    }
    return -1; /* not found */
}
```

Subtask: incrementing frequency count

```
void IncrementCount (int i) {  
    entries[i].count += 1;  
}
```

(Might seem hardly worth it, but aids clarity and readability.)

Subtask: adding a new word

```
void AddToList() {
    if (wordCount == MAX_WORDS) {
        printf("Word list is full!\n");
        return;
    }
    if (strlen (currentWord) >= MAX_WORD_LENGTH) {
        printf("Word too long!\n");
        return;
    }
    strcpy(entries[wordCount].word, currentWord); /* backwards! */
    entries[wordCount].count = 1;
    wordCount++;
}
```

Subtask: displaying the results

```
void WriteList() {
    int i;
    printf("%d words found\n", wordCount);
    printf("*****\n");
    for (i = 0; i < wordCount; ++i) {
        printf("%3d %s\n", entries[i].count, entries[i].word);
    }
}
```

Subtask: reading a word

```
int ReadWord() {
    int i=0, c=SkipWhiteSpace() ;
    if (c==EOF) return 0 ;
    while (c != EOF && !isspace(c) && i < MAX_WORD_LENGTH) {
        currentWord[i] = c;
        ++i;
        c = getchar();
    }
    if (i==MAX_WORD_LENGTH) return 0 ;
    currentWord[i] = '\0' ;
    return 1 ;
}
```

One little sub-subtask remains...

```
char SkipWhiteSpace() {  
    int ch ;  
    do {ch = getchar() ;}  
    while (ch != EOF && isspace(ch)) ;  
    return ch ;  
}
```

N.B. A well-structured program is not only much more pleasant to read, but easier to modify.

(E.g. try modifying our code to read from a file rather than keyboard input.)