The Word Count Program

December 14, 2018

Here we have an outline of the program:

```
⟹ Header files to include.
⟹ Preprocessor definitions.
⟹ Global variables.
⟹ Functions.
⟹ The main program.
```

We must include standard I/O definitions, since we want to send formatted output to *stdout* and *stderr*.

Header files to include

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
```

The `status` variable will tell the operating system if the run was successful or not, and `prog_name` is used in case there's an error message to be printed.

Preprocessor definitions

```
#define OK 1                // status code for successful run
#define usage_error 1       // status code for improper syntax
#define cannot_open_file 2  // status code for file access error
```

Global variables

```
int status = OK;  // exit status of command, initially OK
char *prog_name;  // who we are
```

Now we come to the general layout of the `main()` function.

The main program

```
int main(int argc, char **argv)
{
    ⟹ Variables local to main.
    prog_name = argv[0];
    ⟹ Set up option selection.
    ⟹ Process all the files.
    ⟹ Print the grand totals if there were multiple files.
    return status;
}
```

If the first argument begins with a `-`, the user is choosing the desired counts and specifying the order in which they should be displayed. Each selection is given by the initial character (lines, words, or characters). For example, `-cl` would cause just the number of characters and the number of lines to be printed, in that order. The default, if no special argument is given, is `-lwc`.

We do not process this string now; we simply remember where it is. It will be used to control the formatting at output time.

If the `-` is immediately followed by `s`, only summary totals are printed.

Variables local to main

```
int file_count;  // how many files there are
char *which;     // which counts to print
int silent = 0;  // nonzero if the silent option was selected
```

Set up option selection

```
which = "lwc";   // if no option is given, print all three values
if (argc > 1 && *argv[1] == '-') {
    argv[1]++;
```

```
    if (*argv [1] == 's') silent = 1, argv [1]++;
    if (*argv [1]) which = argv [1];
    argc--;
    argv++;
}
file_count = argc - 1;
```

Now we scan the remaining arguments and try to open a file, if possible. The file is processed and its statistics are given. We use a `do ... while` loop because we should read from the standard input if no file name is given.

Process all the files

```
argc--;
do {
    ⟹ If a file is given, try to open @{*(++argv)}; continue if
        unsuccessful.
    ⟹ Initialize pointers and counters.
    ⟹ Scan file.
    ⟹ Write statistics for file.
    ⟹ Close file.
    ⟹ Update grand totals.
} while (--argc > 0);
```

Here's the code to open the file. A special trick allows us to handle input from *stdin* when no name is given. Recall that the file descriptor to *stdin* is 0; that's what we use as the default initial value.

Variables local to main

```
int fd = 0;
```

Preprocessor definitions

```
#define READ_ONLY 0
```

If a file is given, try to open @{}; continue if unsuccessful

```
if (file_count > 0 && (fd = open(@{next file}, READ_ONLY)) < 0) {
    fprintf(stderr, "%s: cannot open file %s\n", prog_name, *argv);
    status |= 2;
    file_count--;
    continue;
}
```

Close file

```
close(fd);
```

We will do some homemade buffering in order to speed things up: Characters will be read into the `buffer` array before we process them. To do this we set up appropriate pointers and counters.

Variables local to main

```
char buffer[BUFSIZ];     // we read the input into this array
register char *ptr;      // the first unprocessed character in
    buffer
register char *buf_end;  // the first unused position in buffer
register int c;          // current character or number of
    characters just read
int in_word;             // are we within a word?
long word_count, line_count, char_count; // number of words, lines,
    and characters found in the file so far
```

Initialize pointers and counters

```
ptr = buf_end = buffer;
line_count = word_count = char_count = 0;
in_word = 0;
```

The grand totals must be initialized to zero at the beginning of the program. If we made these variables local to `main`, we would have to do this

initialization explicitly; however, C's globals are automatically zeroed. (Or rather, "statically zeroed.") (Get it?)

Global variables

```
long tot_word_count, tot_line_count, tot_char_count; // total number
    of words, lines and chars
```

The present section, which does the counting that is `wc`'s *raison d'être*, was actually one of the simplest to write. We look at each character and change state if it begins or ends a word.

Scan file

```
while (1) {
    ⟹ Fill buffer if it is empty; break at end of file.
    c = *ptr++;
    if (c > ' ' && c < 177) {    // visible ASCII codes
        if (!in_word) {
            word_count++;
            in_word = 1;
        }
        continue;
    }
    if (c == '\n') line_count++;
    else if (c ≠ ' ' && c ≠ '\t') continue;
    in_word = 0;  // c is newline, space, or tab
}
```

Buffered I/O allows us to count the number of characters almost for free.

Fill buffer if it is empty; break at end of file

```
if (ptr ≥ buf_end) {
    ptr = buffer;
    c = read(fd, ptr, BUFSIZ);
    if (c ≤ 0) break;
    char_count += c;
    buf_end = buffer + c;
}
```

It's convenient to output the statistics by defining a new function `wc_print()`; then the same function can be used for the totals. Additionally we must decide here if we know the name of the file we have processed or if it was just *stdin*.

Write statistics for file

```
if (!silent) {
    wc_print(which, char_count, word_count, line_count);
    if (file_count) printf(" %s\n", *argv);  // not stdin
    else printf("\n");                       // stdin
}
```

Update grand totals

```
tot_line_count += line_count;
tot_word_count += word_count;
tot_char_count += char_count;
```

We might as well improve a bit on UNIX's 'wc' by displaying the number of files too.

Print the grand totals if there were multiple files

```
if (file_count > 1 || silent) {
    wc_print(which, tot_char_count, tot_word_count, tot_line_count);
    if (!file_count) printf("\n");
    else printf(" total in %d file%s\n", file_count, file_count > 1
        ? "s" : "");
}
```

Here now is the function that prints the values according to the specified options. The calling routine is supposed to supply a newline. If an invalid option character is found we inform the user about proper usage of the command. Counts are printed in 8-digit fields so that they will line up in columns.

```
void wc_print(char *which, long char_count, long word_count, long
    line_count)
{
    while (*which)
        switch (*which++) {
        case 'l': printf("%8ld", line_count);
            break;
        case 'w': printf("%8ld", word_count);
            break;
        case 'c': printf("%8ld", char_count);
            break;
        default:
            if ((status & 1) == 0) {
                fprintf(stderr, "\nUsage: %s [-lwc] [filename ○ ..]\n
                    ", prog_name);
                status |= 1;
            }
        }
}
```