File “count.c”

```
1 #include <stdio.h>
2
3 main (argc, argv)
4     int argc;
5     char *argv[];
6 {
7  int      c, i, inword;
8  FILE     *fp;
9  long     linect, wordct, charct;
10 long     tlinect = 1, twordct = 1, tcharct = 1;
11
12  i = 1;
13  do {
14    if (argc > 1 && (fp=fopen(argv[i], "r")) == NULL) {
15      fprintf (stdout, "can't open %s\n", argv[i]);
16      exit (1);
17   }
18    linect = wordct = charct = 0;
19    inword = 1;
20    while ((c = getc(fp)) != EOF) {
21      ++charct;
22      if (c == '\n')
23          ++linect;
24      if (c == ' ' || c == '\t' || c == '\n')
25          inword = 0;
26      else if (inword == 0) {
27          inword = 1;
28          ++wordct;
29      }
30    }
31    printf("%7ld %7ld %7ld", linect, wordct, charct);
32    if (argc > 1)
33     printf(" %s\n", *argv);
34    else
35      printf("\n");
36    fclose(fp);
37    tlinect += linect;
38    twordct += wordct;
39    tcharct += charct;
40  } while (++i < argc);
41  if (argc > 1)
42   printf("%7ld %7ld %7ld total\n", linect, twordct, tcharct);
43   exit(0);
44 }
```

Specification for program “count”

Name

count – count lines, words, and characters

Usage

count filename [filename...]

Description

count counts the number of lines, words, and characters in the named files. Words are sequences of characters that are separated by one or more spaces, tabs, or line breaks (carriage return).
If a file supplied as argument does not exist, a corresponding error message is printed and processing of any other files continues. If no file is supplied as an argument, **count** reads from the standard input.
The computed values are given for each file (including the name of the file) as well as the sum of all values. If only a single file or if the standard input is processed, then no sum is printed. The output is printed in the order lines, words, characters, and either the file name or the word “total” for the sum. If the standard input is read, the fourth value (name) is not printed.

Options

None.

Example

```
% count datei
```

84 462 3621 datei

Sample Solution for Training Exercise "count" with Mill's functional notation

The following lists line numbers and abstractions (specifications) of the source code from those lines.

Line(s)	Abstraction
21-29	Sequence
	The meaning of the components:
21	charct := charct + 1
22-23	*c* = *newline* → linect := linect + 1 | ( )
24-29	(*c*=*whitespace* ∨ c=tab ∨ c=newline) → inword:= 0 | (inword = 0 → inword, wordct := 1, wordct + 1 | ( ))
	Because there are 4 paths through this sequence, the following results: *c* = *newline* → charct, linect, inword := charct + 1, linect + 1, 0 | *c* ≠ *newline* ∧ (c=whitespace ∨ c=tab) → charct, inword := charct + 1, 0 | *c* ≠ *newline* ∧ (c≠whitespace ∧ c≠tab ∧ c≠newline) ∧ inword = 0 → charct, wordct, inword := charct + 1, wordct + 1, 1 | *c* ≠ *newline* ∧ (c≠whitespace ∧ c≠tab ∧ c≠newline) ∧ inword ≠ 0 → charct := charct + 1
14-39	Sequence
	Once again, first the meaning of the components:
14-17	argc > 1 → (file-open(argv[i]) = *failure→ err-msg and halt* | fp *:= stream*)
18-30	To determine the meaning of this portion sensibly, the lines 18-19 are included and the tasks of the variables are investigated: 1. Variable "charct" is incremented in all 4 cases; i.e., counts every character; 2. Variable "linect" is only incremented if a *newline* is read; i.e., counts lines; 3. Variable "inword" is a switch that takes on values 0 and 1. If whitespace is seen, the switch is set to 0. If other characters are seen, it is switched to 1 and at the same time "wordct" is incremented; i.e., counts words. *c* ≠ EOF → charct, wordct, linect := character-count(stdin), word-count(stdin), line-count(stdin). Note: because "inword" is initialized to 1, the first word is not counted if it comes at the beginning of a stream not preceded by whitespace.

Line(s)	Abstraction
31	stdout := “linect, wordct, charct”
32-35	argc > 1 → stdout := *argv (*i.e., program name*) *and newline* | stdout := *newline*
36	*close stream*
37-39	tlinect, twordct, tcharct := tlinect + linect, twordct + wordct, tcharct + charct
	Because there are 3 paths through this sequence, the following results: argc > 1 ∧ open-file(argv[i]) = failure → stdout := err msg *and halt* | argc > 1 ∨ open-file(argv[i]) = success → tcharct, tlinect, twordct, stdout := tcharct + character-count(*stream*), twordct + word-count (*stream*), tlinect + line-count(*stream*), “line-count(*stream*), word-count(*stream*), character-count(*stream*), pgm-name” argc ≤ 1 → tcharct, tlinect, twordct, stdout := tcharct + character-count (<nil>), twordct + word-count(<nil>), tlinect + character-count(<nil>), “line count(<nil>), word-count(<nil>), character-count(<nil>)” Note: If argc is ≤ 1, fp is not initialized; i.e., the program reads from an undefined stream (label <nil> above).
3-44	Sequence
	Once again, first the meaning of the components:
10	tlinect, twordct, tcharct := 1, 1, 1
12-40	for all indexes of command-line arguments from 1...argc – 1 do [14-39]
41-42	argc > 1 → stdout := “linect, twordct, tcharct” | ( )
43	Halt

Sample Solution for Training Exercise "count" with informal notation

The following lists line numbers and abstractions (specifications) of the source code from those lines.

Line(s)	Abstraction
22-23	If (c = '\n') then linect ← linect + 1;

24-29	If (c = ' ', or c = '\t' or c = '\n') then
	  inword ← 0;
	else if (inword = 0) then
	  inword ←1;
	  wordct ← wordct +1;

20-30	c ← getc(fp)
	while (c ≠ EOF) do
	  charct ← charct + 1;
	  22-23;
	  24-29;
	  c ← getc(fp);

14-17	if argc > 1 then
	  fp ← fopen(argv[i], "r")
	  if fp = NULL then
	  print(stdout, "can´t open", argv[i]);
	  exit;

32-35	If (argc > 1) then
	  print(*argv);
	else
	  print('\n');

13-40	do
	  14-17;
	  linect ← 0; wordct ←0; charct← 0; inword ←1;
	  20-30;
	  print(linect, wordct, charct);
	  32-35
	  fclose(fp);
	  tlinect ← tlinect +linect;
	  twordct ← twordct +wordct;
	  tcharct ← tcharct +charct;
	while ((i←i+1) < argc)

10-44	tlinect ← 1; twordct ←1; tcharct← 1;
	i ← 1;
	13-40;
	if (argc > 1) then
	  print(linect, twordct, tcharct);
	  exit;

Faults in program “count”

The classification of each fault is given in braces.

1. Fault in line 10: The variables are initialized with 1, should be with 0.
   `{Commission, Initialization}`
   Causes failure: The sums are incorrect (off by one).

2. Fault in line 14: The variables “fp” is not initialized in the case that the input should be taken from “stdin”.
   `{Omission, Initialization}`
   Causes failure: The program cannot read from stdin.

3. Fault in line 15: The invocation of fprintf uses “stdout” instead of “stderr”.
   `{Commission, Interface}`
   Causes failure: Error messages appear on the standard output (stdout) instead of the standard-error output (stderr).

4. Fault in line 16: Component is terminated with “exit (1)”, where “continue” should have been used.
   `{Commission, Control}`
   Causes failure: If a file is not found, the program stops there instead of continuing on to other files, also, no sum is printed.

5. Fault in line 19: The variable “inword” is initialized with 1 instead of 0.
   `{Commission, Initialization}`
   Causes failure: Depending on whether the first symbol in a file is whitespace, the program reports that files with **n** words have either **n** or (**n – 1**) words.

6. Fault in line 33: *argv is used instead of argv[i].
   `{Commission, Data}`
   Causes failure: The program prints its own name instead of the file name when reporting the counts.

7. Fault in line 41: Argc is compared with 1, but should be compared with 2.
   `{Commission, Computation}`
   Causes failure: The program prints out sums even when only a single file was processed.

8. Fault in line 42: Instead of “tlinect” the variable “linect” is used.
   `{Commission, Data}`
   Causes failure: The sums are not computed correctly. Example:

```
% . /count file2 file2 file2
1  2     14    ./count
1  2     14    ./count
1  2     14    ./count
1  7     43    total
```

Worksheet for faults from Code Reading

Subject identifier: _____

Isolated Faults				
I. Nr.	Line Nr.	O/C	Type	Description
1	10	C	Initialization	The variables are initialized with 1, should be with 0
2	14	O	Initialization	The variable “fp” is not initialized in the case that the input should be taken from “stdin”
3	15	C	Interface	The invocation of fprintf uses “stdout” instead of “stderr”
4	16	C	Control	Component is termined with “exit(1)”, where “continue” should be used
5	19	C	Initialization	The variable “inword” is initlized with 1 instead of 0
6	33	C	Data	“*argv” is used instead of “argv[1]”
7	41	C	Computation	“argc” is compared with 1, but should be compared with 2
8	42	C	Data	Instead of “tlinect” the variable “linect”