```
DEMOS INDEX                  Thu Oct 13 17:33:50 EDT 2005

Demonstration Problem:
----------------------


        Count characters, words, and lines in text.
        Solutions are included with this demonstration
        problem.  The files available are:

        demos/count/Makefile     Commented Makefile.
        demos/count/README       Usage Info.
        demos/count/count.in     Judges input.
        demos/count/count.test   Judges output.
        demos/count/count.txt    Problem statement.
        demos/count/count1.c     Solution in C.
        demos/count/count1.cc    Solution in C++.
        demos/count/count1.java  Solution in JAVA.
        demos/count/count1.lsp   Solution in COMMONLISP.

Java IO Demo:
---- -- ----

        Demo of JAVA IO.  The files available are:

        demos/javaio/javaio.java Demo code.
        demos/javaio/Makefile    Makefile.
        demos/javaio/javaio.in   Test input.
        demos/javaio/javaio.test Test output.
```

```
# Makefile for the ‘count’ Demonstration Problem
#
# File:          Makefile
# Date:          Tue Oct  4 21:27:36 EDT 2005
#
# Exactly ONE of the four files count.c (C), count.cc
# (C++), count.java (Java), or count.lsp (Commonlisp)
# should exist.
#
# UNIX commands supported by this Makefile:
#
#       make            Same as ‘make count.out’.
#
#       make count      Makes the binary program file
#                       ‘count’ by running gcc on
#                       count.c, or g++ on count.cc,
#                       or javac on count.java,
#                       or hpcm_clisp on count.lsp,
#                       depending upon which of count.c,
#                       count.cc, count.java, or
#                       count.lsp exist.  Also makes a
#                       shell script named ‘count’ for
#                       count.java and count.lsp files.
#                       Does nothing if ‘count’ is more
#                       up to date than count.c,
#                       count.cc, count.java, or
#                       count.lsp.
#
#       make count.out  Makes ‘count’ as above and
#                       then runs it with standard
#                       input coming from the file
#                       count.in.  Puts the standard
#                       output in the file count.out,
#                       and then copies that to the
#                       screen.  Does nothing, however,
#                       if count.out is more recent
#                       than both count.in and count.
#
#       make submit     Makes ‘count.out’ just to be
#                       sure that nothing crashes, and
#                       then e-mails count.c, count.cc,
#                       count.java, or count.lsp to the
#                       judges.
#
#       make in-submit  Ditto, but requests that if
#                       the score is ‘Incorrect Output’
#                       or ‘Formatting Error’, the
#                       judge’s input for the failed
#                       test case will be returned in
#                       e-mail to the contestant.
#
#       make inout-submit
#                       Ditto but requests both the
#                       judge’s input and the judge’s
#                       output for the test case.
#
#       make solution-submit
#                       Like ‘make submit’ but requests
#                       that if the score is ‘Completely
#                       Correct’ the judge’s solution
#                       will be returned in e-mail to
#                       the contestant.
#
#       make clean      Removes ‘count’, count.out, and
#                       other intermediate files that
#                       might exist, such as ‘core’,
#                       ‘count.class’, or count.fas.

#
```

```
.SUFFIXES:
.SUFFIXES: .c .cc .java .lsp

default:        count.out

.c:
        rm -f $* core
        gcc -g -o $* $*.c -lm

.cc:
        rm -f $* core
        g++ -g -o $* $*.cc -lm

.java:
        rm -f $* *.class core
        javac -g $*.java
        echo >$* '#!/bin/sh'
        echo >>$* "exec `which java` $*"
        chmod a+r *.class
        chmod a+rx $*

.lsp:
        rm -f $* $*.fas $*.lib core
        hpcm_clisp -c $*.lsp
        echo >$* '#!/bin/sh'
        echo >>$* "exec `hpcm_clisp -which` -I $*.fas"
        chmod a+r $*.fas
        chmod a+rx $*
#
```

```
# hpcm_sandbox below may execute `count' as a special
# unprivileged user named `sandbox', so various files
# must be `a+x' or `a+r'.  `hpcm_clisp -which' returns
# in the judging account the name of a version of the
# hpcm_clisp program that can be run in the sandbox.

count.out:      count count.in
        rm -f count.out core
        chmod a+x . count
        hpcm_sandbox -cputime 60 \
                     -datasize 4m \
                     -stacksize 4m \
                     -filesize 50k \
                     -watch \
                     count \
            <count.in >count.out
        @echo ""
        @echo "========== OUTPUT (count.out)" \
              "=========="
        @cat count.out
        @echo "============== END OUTPUT =============="


submit:         count.out
        hpcm_submit count

in-submit:      count.out
        hpcm_submit -in count

inout-submit:   count.out
        hpcm_submit -inout count

solution-submit:        count.out
        hpcm_submit -solution count

clean:
        rm -f count *.class core count.out \
              count.fas count.lib count.jout
#
```

```
# Author:       walton@deas.harvard.edu
#
# The authors have placed this file in the public
# domain; they make no warranty and accept no liability
# for this file.
#
# RCS Info (may not be true date or author):
#
#   $Author: hc3 $
#   $Date: 2005/10/05 02:11:10 $
#   $RCSfile: Makefile,v $
#   $Revision: 1.26 $
```

```
Count Demo README          Fri Oct  7 08:17:56 EDT 2005

The files in this demo directory are:

    public/count/Makefile       Commented Makefile.
    public/count/README         Usage Info.
    public/count/count.in       Judges input.
    public/count/count.test     Judges output.
    public/count/count.txt      Problem description.
    public/count/count1.c       Solution in C.
    public/count/count1.cc      Solution in C++.
    public/count/count1.java    Solution in JAVA.
    public/count/count1.lsp     Solution in COMMONLISP.

There may be other files used exclusively by the judge,
such as .rc, .jin, and .jtest files.

The Makefile is commented, as opposed to most problem
Makefiles.  For a non-demo problem you are only given
the .txt file and the Makefile.

To run the demo (under UNIX), first

    cp count1.yy count.yy

for exactly ONE of yy = c, cc, java, or lsp.  Then

        make

To check that the output is correct

        diff count.out count.test

Then to submit the demo

        make submit


If you want to edit the solution you chose, you may
first need to

        chmod u+w count.yy

(for the right yy), to make the file writable.

You should try introducing an error in the file and
resubmitting to see the response.  If you are in a
contest that permits 'in-submit' and 'inout-submit',
try

        make in-submit

and then

        make inout-submit

with a source file that has an error which makes it
produce incorrect output.

Read the Makefile for more information.

If you have a non-UNIX system, you can submit the file
count1.yy directly by sending email to the judge with
subject 'submit count.yy' (note there is no '1' here)
and body equal to the file count1.yy (here there is a
'1').  You may run count1.yy using your own system with
count.in as the standard input in order to generate
count.out.

Although in this directory the problem description is
in a .txt file, in other problem directories the problem
description may be in a .html, .htm, or .ps (postscript)
file.
```

```
File:           README
Authors:        walton@deas.harvard.edu
Date:           see above


The authors have placed this file in the public
domain; they make no warranty and accept no liability
for this file.

RCS Info (may not be true date or author):

  $Author: hc3 $
  $Date: 2005/10/07 12:18:37 $
  $RCSfile: README,v $
  $Revision: 1.9 $
```

```
This is a good paragraph to start with.

And to continue in a bit more
complicated
vein,
this is a good paragraph.

But
 the
  ultimate
   in
    poetically
     possible
      paragraphs
       is
        this,
         or
          is
           it!




Oh Well.
```

```
Paragraph 1: 1 lines, 8 words, 39 characters.
Paragraph 2: 4 lines, 14 words, 70 characters.
Paragraph 3: 12 lines, 12 words, 124 characters.
Paragraph 4: 1 lines, 2 words, 8 characters.
```

Paragraph Character/Word/Line Counting.

The Itsy Bitsy Counting Company has a job counting the number of characters, words, and lines in a paragraph.

A paragraph is a sequence of 1 or more non-blank lines.

All the characters of a line count EXCEPT the trailing new line.

A word is a sequence of non-space (non ' ') characters on a line, and is separated from other words on the same line by sequences of space (' ') characters.

The only whitespace characters in the input are space and newline (' ' and '\n'). No line has more than 100 characters in it, not counting the new line at the end.

Paragraphs are separated by one or more blank lines. A blank line may have whitespace characters, but nothing else.

The paragraphs in the input are numbered 1, 2, ... The program reads its standard input, and for each paragraph in that input, prints the paragraph number and the counts, in exactly the following format:

Paragraph #: # lines, # words, # characters.

where each # denotes 1 or more decimal digits.

```
Example Input:
------- -----

This is a good paragraph to start with.

And to continue in a bit more
complicated
vein,
this is a good paragraph.

But
 the
  ultimate
   in
    poetically
     possible
      paragraphs
       is
        this,
         or
          is
           it!




Oh Well.


Example Output:
------- ------


Paragraph 1: 1 lines, 8 words, 39 characters.
Paragraph 2: 4 lines, 14 words, 70 characters.
Paragraph 3: 12 lines, 12 words, 124 characters.
Paragraph 4: 1 lines, 2 words, 8 characters.
```

```c
#include <stdio.h>

main()
{
    int paragraph = 1;

    while ( 1 )
    {
        int characters = 0;
        int words = 0;
        int lines = 0;

        char buffer [102];

        int at_end_of_file = 1;

        while ( fgets ( buffer, sizeof ( buffer),
                        stdin ) )
        {
            char * cp = buffer;

            at_end_of_file = 0;

            while ( * cp == ' ' ) ++ cp;

            if ( * cp == 0 || * cp == '\n' ) break;

            ++ lines;

            do
            {
                ++ words;
                while ( * cp != ' ' &&
                        * cp != '\n' &&
                        * cp != 0 ) ++ cp;
                while ( * cp == ' ' ) ++ cp;
            } while ( * cp != 0 && * cp != '\n' );

            characters += ( cp - buffer );
        }

        if ( at_end_of_file ) break;

        if ( lines > 0  )
        {
            printf ( "Paragraph %d: %d lines, %d words,"
                     " %d characters.\n", paragraph,
                     lines, words, characters );

            ++ paragraph;
        }
    }

    return 1;   /* This line can be omitted.
                 * It is a test that make count.out
                 * works even if count returns an
                 * error code.
                 */
}
```

```cpp
#include <iostream>
using namespace std;

main()
{
    int paragraph = 1;

    while ( ! cin.eof() )
    {
        int characters = 0;
        int words = 0;
        int lines = 0;

        char buffer [101];

        while
          ( cin.getline ( buffer, sizeof ( buffer ) ),
            ! cin.eof() )
        {
            char * cp = buffer;
            while ( * cp == ' ' ) ++ cp;

            if ( * cp == 0 ) break;

            ++ lines;

            do
            {
                ++ words;
                while ( * cp != ' ' && * cp ) ++ cp;
                while ( * cp == ' ' ) ++ cp;
            } while ( * cp );

            characters += ( cp - buffer );
        }

        if ( lines > 0  )
        {
            cout << "Paragraph " << paragraph << ": "
                 << lines << " lines, "
                 << words << " words, "
                 << characters << " characters."
                 << endl;
            ++ paragraph;
        }
    }

    return 1;   // This line can be omitted.
                // It is a test that make count.out
                // works even if count returns an
                // error code.
}
```

```java
// Count Demo Program: JAVA Version
//
// File:     count1.java
// Author:   Bob Walton <walton@deas.harvard.edu>
// Date:     Fri Nov  1 06:37:28 EST 2002
//
// The authors have placed this program in the public
// domain; they make no warranty and accept no liability
// for this program.
//
// RCS Info (may not be true date or author):
//
//   $Author: hc3 $
//   $Date: 2002/11/01 11:38:21 $
//   $RCSfile: count1.java,v $
//   $Revision: 1.4 $

import java.io.*;
import java.util.StringTokenizer;

public class count {

    public static void main (String[] args)
            throws IOException
    {

        BufferedReader reader
           = new BufferedReader
                 ( new InputStreamReader
                       ( System.in ) );

        // Loop through paragraphs.
        //
        int paragraph = 1;
        boolean eof_seen = false;
        while ( ! eof_seen )
        {
            int characters = 0;
            int words = 0;
            int lines = 0;

            while ( true )
            {
                String line = reader.readLine();
                if ( line == null )
                {
                    // readLine returns null on EOF.
                    //
                    eof_seen = true;
                    break;
                }

                StringTokenizer tokenizer
                    = new StringTokenizer ( line );

                // Break on blank line.
                //
                if ( ! tokenizer.hasMoreTokens() )
                    break;

                ++ lines;

                // Count words in line.
                //
                while ( tokenizer.hasMoreTokens() )
                {
                    ++ words;
                    tokenizer.nextToken();
                }

                // Count characters in line.
                //
                characters += line.length();
            }

            // Ignore blank ‘paragraphs’.
            //
            if ( lines > 0  )
            {
                // Print paragraph output.
                //
                System.out.println
                    (   "Paragraph " + paragraph + ": "
                      + lines + " lines, "
                      + words + " words, "
                      + characters + " characters."
                    );
```

```
                ++ paragraph;
            }
        }
    }
}
```

```
(defun main () (read-a-paragraph 1))

;; Counts are expressed as a triple:
;;
;;      (line-count word-count character-count)

(defvar blank-line '(1 0 0))
(defvar end-of-file '(0 0 0))

(defun read-a-paragraph (paragraph)
  (let ( (counts (read-a-line)) )
    (cond
      ((equal counts blank-line)
       (read-a-paragraph paragraph))
      ((not (equal counts end-of-file))
       (read-rest-of-paragraph counts paragraph)))))

(defun read-rest-of-paragraph (counts paragraph)
  (let ( (line-counts (read-a-line)))
    (cond ((or (equal line-counts blank-line)
               (equal line-counts end-of-file))
           (format t "Paragraph ~S" paragraph)
           (format t ": ~S lines" (first counts))
           (format t ", ~S words" (second counts))
           (format t ", ~S characters.~%"
                   (third counts))
           (if (equal line-counts blank-line)
               (read-a-paragraph (1+ paragraph))))
          (t
           (read-rest-of-paragraph
             (mapcar #'+ line-counts counts)
             paragraph)))))


(defun read-a-line ()
  (let ( (line (read-line t nil 'eof)) )
    (cond
      ((eq line 'eof) '(0 0 0))
      (t `(1 ,(read-a-word line 0 (length line) 0)
             ,(length line))))))

(defun read-a-word (line index length count)
  (cond
    ((>= index length) count)
    ((char= #\Space (aref line index))
     (read-a-word line (1+ index) length count))
    (t
     (read-rest-of-word line (1+ index) length count))))

(defun read-rest-of-word (line index length count)
  (cond
    ((>= index length) (1+ count))
    ((char= #\Space (aref line index))
     (read-a-word line (1+ index) length (1+ count)))
    (t
     (read-rest-of-word line (1+ index) length count))))
```

```
// JAVA IO Demo
//
// File:     javaio.java
// Author:   Bob Walton <walton@deas.harvard.edu>
// Date:     Thu Feb 12 23:05:12 EST 2004
//
// The authors have placed this program in the public
// domain; they make no warranty and accept no liability
// for this program.
//
// RCS Info (may not be true date or author):
//
//   $Author: hc3 $
//   $Date: 2004/02/13 04:06:10 $
//   $RCSfile: javaio.java,v $
//   $Revision: 1.4 $

import java.io.*;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.Locale;

// This program reads input, parses it into tokens,
// prints info about the tokens, and prints a summary
// at the end.  The program illustrates use of the
// StreamTokenizer and DecimalFormat classes.

public class javaio {

    public static void main (String[] args)
            throws IOException {

        // Set up the StreamTokenizer.
        //
        Reader reader
            = new BufferedReader
                  ( new InputStreamReader
                        ( System.in ) );
        StreamTokenizer tokenizer
            = new StreamTokenizer ( reader );

        // Set to read any string of non-whitespace
        // characters as a word.
        //
        tokenizer.resetSyntax();
        tokenizer.wordChars ( '!', '\u00FF' );
        tokenizer.whitespaceChars ( '\u0000', ' ' );
        //
        // You must not set the same character to be
        // both a word character and a whitespace
        // character.

        // Set to read end of line as a token.
        // If this function is not called, end of
        // line is treated as a simple space character.
        //
        tokenizer.eolIsSignificant ( true );

        // Read numbers as tokens.  If not called,
        // numbers are not handled specially.
        //
        // WARNING: This makes isolated '.'s input as
        // the the number 0, while `-'s may input as
        // a separator.
        //
        tokenizer.parseNumbers();

        // Parse certain characters as 1-character
        // tokens.
        //
        tokenizer.ordinaryChar ( ',' );
        tokenizer.ordinaryChar ( '(' );
        tokenizer.ordinaryChar ( ')' );

        // Set up number formatter.  Note that it is
        // important in ACM programming contests to
        // insist on an ENGLISH formatter.
        //
        // Also, do NOT put commas in the output.
        //
        DecimalFormat formatter = (DecimalFormat)
            NumberFormat.getInstance ( Locale.ENGLISH );
        formatter.applyPattern ( "#0.00" );


        // Process a paragraph.  Paragraphs are
        // separated by blank lines.
        //
```

```
        int paragraph = 1;
        boolean eof_seen = false;
        while ( ! eof_seen )
        {
            int numbers = 0;
            int words = 0;
            int separators = 0;
            int lines = 0;

            boolean eop_seen = false;
            boolean line_is_blank = true;

            while ( ! eop_seen && ! eof_seen )
            {
                tokenizer.nextToken();
                switch ( tokenizer.ttype )
                {

                case StreamTokenizer.TT_EOF:

                    if ( line_is_blank )
                    {
                        eof_seen = true;
                        break;
                    } else
                        throw new RuntimeException
                            ( "EOF in bad place" );

                case StreamTokenizer.TT_EOL:

                    if ( ! line_is_blank )
                        ++ lines;
                    else if ( lines != 0 )
                        eop_seen = true;
                    line_is_blank = true;
                    break;

                case StreamTokenizer.TT_NUMBER:

                    System.out.print ( "NUMBER ");
                    System.out.print ( tokenizer.nval );
                    System.out.print ( " = ");
                    System.out.print
                        ( formatter.format
                            ( tokenizer.nval ) );
                    System.out.println();
                    line_is_blank = false;
                    ++ numbers;
                    break;

                case StreamTokenizer.TT_WORD:
                    System.out.print ( "WORD ");
                    System.out.print ( tokenizer.sval );
                    System.out.println();
                    line_is_blank = false;
                    ++ words;
                    break;

                case '(':
                case ')':
                case ',':
                case '-':
                    System.out.print ( "SEPARATOR ");
                    System.out.print
                        ( (char) tokenizer.ttype );
                    System.out.println();
                    line_is_blank = false;
                    ++ separators;
                    break;

                default:
                    throw new RuntimeException
                        ( "Bad token type "
                          + tokenizer.ttype );
                }
            }

            if ( lines > 0  )
            {
                System.out.println
                    ( "Paragraph " + paragraph + ":" );

                System.out.println
                    ( "    " + lines + " lines, "
                           + words + " words, "
                           + numbers + " numbers, "
                           + separators
                           + " separators." );
```

```
                double m =
                    ( (double) 100.0 )
                    / ( words + numbers + separators );

                System.out.println
                    ( "    "
                      + formatter.format
                            ( m * words )
                      + "% words, "
                      + formatter.format
                            ( m * numbers )
                      + "% numbers, "
                      + formatter.format
                            ( m * separators )
                      + "% separators." );

                ++ paragraph;
            }
        }
    }
}
```

```
# Makefile for JAVA IO Demo
#
# File:         Makefile
# Date:         Fri Nov  1 02:34:42 EST 2002
#
# See demonstration Makefile for documentation.
#
# The program for this problem is named:

P = javaio

.SUFFIXES:
.SUFFIXES: .c .cc .java .p .lsp

default:        $P.out

.c:
	rm -f $* core
	gcc -g -o $* $*.c -lm

.cc:
	rm -f $* core
	g++ -g -o $* $*.cc -lm

.p:
	rm -f $* $*.p2c core
	p2c -o $*.p2c $*.p
	gcc -g -o $* -x c $*.p2c -lp2c -lm

.java:
	rm -f $* *.class core
	javac -g $*.java
	echo >$* '#!/bin/sh'
	echo >>$* "exec `which java` $*"
	chmod a+r *.class
	chmod a+rx $*

#

.lsp:
	rm -f $* $*.fas $*.lib core
	hpcm_clisp -c $*.lsp
	echo >$* '#!/bin/sh'
	echo >>$* "exec `hpcm_clisp -which` -I $*.fas"
	chmod a+r $*.fas
	chmod a+rx $*

$P.out: $P $P.in
	rm -f $P.out core
	chmod a+x . $P
	hpcm_sandbox -cputime 60 \
	             -datasize 4m \
	             -stacksize 4m \
	             -filesize 50k \
	             -watch \
	             $P \
	    <$P.in >$P.out
	@echo ""
	@echo "========== OUTPUT ($P.out)" \
	      "=========="
	@cat $P.out
	@echo "============== END OUTPUT =============="

submit:         $P.out
	hpcm_submit $P

clean:
	rm -f $P *.class $P.p2c core $P.out \
	      $P.fas $P.lib


#
```

```
# Author:       walton@deas.harvard.edu
#
# The authors have placed this file in the public
# domain; they make no warranty and accept no liability
# for this file.
#
# RCS Info (may not be true date or author):
#
#   $Author: hc3 $
#   $Date: 2002/11/01 09:12:45 $
#   $RCSfile: Makefile,v $
#   $Revision: 1.1 $
```

```
This is a nice sentence.
And another.

These are some numbers:
   1 2 3 4 5 6 7 8 9 10
   8.4 123456789

These are some strange cases:
   . - a-b -a -3.0a

How about some separators, (a good thought).
Well, not everything that should be is a separator.
```

```
WORD This
WORD is
WORD a
WORD nice
WORD sentence.
WORD And
WORD another.
Paragraph 1:
    2 lines, 7 words, 0 numbers, 0 separators.
    100.00% words, 0.00% numbers, 0.00% separators.
WORD These
WORD are
WORD some
WORD numbers:
NUMBER 1.0 = 1.00
NUMBER 2.0 = 2.00
NUMBER 3.0 = 3.00
NUMBER 4.0 = 4.00
NUMBER 5.0 = 5.00
NUMBER 6.0 = 6.00
NUMBER 7.0 = 7.00
NUMBER 8.0 = 8.00
NUMBER 9.0 = 9.00
NUMBER 10.0 = 10.00
NUMBER 8.4 = 8.40
NUMBER 1.23456789E8 = 123456789.00
Paragraph 2:
    3 lines, 4 words, 12 numbers, 0 separators.
    25.00% words, 75.00% numbers, 0.00% separators.
WORD These
WORD are
WORD some
WORD strange
WORD cases:
NUMBER 0.0 = 0.00
SEPARATOR -
WORD a-b
SEPARATOR -
WORD a
NUMBER -3.0 = -3.00
WORD a
Paragraph 3:
    2 lines, 8 words, 2 numbers, 2 separators.
    66.67% words, 16.67% numbers, 16.67% separators.
WORD How
WORD about
WORD some
WORD separators
SEPARATOR ,
SEPARATOR (
WORD a
WORD good
WORD thought
SEPARATOR )
NUMBER 0.0 = 0.00
WORD Well
SEPARATOR ,
WORD not
WORD everything
WORD that
WORD should
WORD be
WORD is
WORD a
WORD separator.
Paragraph 4:
    2 lines, 16 words, 1 numbers, 4 separators.
    76.19% words, 4.76% numbers, 19.05% separators.
```