Assignment: Character count with histogram display.

10 points for character display only.

40 points for char and horizontal displays.

60 points for char, horizontal, and vertical solution.

10 extra for GENERALIZED vertical number routine.

(You already know how to do it if you remember the Caesar Block Cipher.)

Read in a stream of text UNTIL YOU ENCOUNTER End of File (EOF).

Keep this simple, don't read and parse whole lines, this isn't Java.

Read character-by-character, until you encounter EOF, and keep a count

of how many of each letter you see in an array of 26 ints called:

int count[26].

read in a character

INPUT LOOP:

count the character if it's a letter.

counts are stored in int count[26]

try to read in another character

CONTINUE WHILE CHARS EXIST TO BE READ

---

Remember that characters are integer values.

So when you are reading in characters, you can

USE THEM AS NUMBERS AS WELL AS DISPLAYING THEM

AS CHARACTERS. This is the fundamental difference

between C programmers and those using less flexible

tools: we actually understand how computers work, and

use that understanding to create efficient programs.

We control the hardware, instead of being fooled by

the interfaces, because C is really Assembly Language,

and it gives us low-level control.

The principle: 'c' - 'a' == 2

'z' - 'a' == 25

'a' - 'a' == 0

SO:

c = cin.get() ;

count[c - 'a']++ ; // this saves you 52 lines of code

Do NOT create a huge switch() or 56 if() statements, that'd

be inefficient.

Once you've finished reading to EOF and counting the characters, display

the result of the counting process by printing out the contents of the

count[] array as data;

count['a'] : 234

count['b'] : 43

.

.

count['z'] : 389

and so on.

There are files of sample data you can run against, so long as

you understand they're just samples, and your program must accurately

count any input streams.

They are:

http://209.129.16.61/~hhaller/data/cisc192/asst6/histo.data

http://209.129.16.61/~hhaller/data/cisc192/asst6/h.dat.

h.dat and histo.data have the same distribution. One is 10 times larger.

Once you've finished reading the text stream and counting

the characters, display the result of the counting process.

Character Output

Start by printing out the contents of the count[] array

as data;

a 320

b 160

c 80

d 40

e 20

f 10

g 5

h 2

i 1

j 2

k 17

l 41

m 67

n 97

o 127

p 157

q 191

r 227

s 257

t 283

u 331

v 367

w 373

x 379

y 383

z 389

Horizontal Graph:

To graph the data, you will print out bars which show the

RELATIVE sizes of the members of count[], SCALED TO FIT

THE COMPUTER DISPLAY.

=================================================================5

Look at the counts in these two graphs, then examine

the lengths of the bars relative to each other. That

is "scaling":

28 a ======================================================

23 b ============================================

16 c ===============================

11 d =====================

7 e =============

3 f =====

0 g

0 h

0 i

0 j

0 k

0 l

0 m

0 n

0 o

0 p

0 q

0 r

0 s

1 t =

11 u =====================

15 v =============================

25 w ================================================

31 x ============================================================

34 y ==================================================================

38 z ==========================================================================

320 a ============================================================

160 b ==============================

80 c ===============

40 d =======

20 e ===

10 f =

5 g

2 h

1 i

2 j

17 k ===

41 l =======

67 m ============

97 n ==================

127 o ========================

157 p =============================

191 q ====================================

227 r ===========================================

257 s ================================================

283 t =====================================================

331 u ==============================================================

367 v =====================================================================

373 w ======================================================================

379 x ========================================================================

383 y ========================================================================

389 z ==========================================================================

1. The bars must be shorter than the screen is wide,

they must NOT "wrap". (Regardless of input, no output

line may extend past the right side of the screen.)

2. The lines must be proportional to each other, starting

in the same column, so control the width of the printed

numbers. Examine the varfield.cpp example program for a

technique relevent to this task.

3. Your program must be able to handle files where

character frequencies are quite large and/or quite small.

That means you will have to "scale" the output, so the

length of the smallest line is in proper proportion to

that of the largest, and all lines in between them are in

proper relationship to each other.

4. In order to scale the lines, cast your mind back to the

ratio and proportion problems you were trained to solve in

Sixth Grade. You will need to know the highest number in

count[] (the most frequently occurring character), and the

number of columns (horizontal method) or lines (vertical

method) you are going to use to display your data.

Once you have it, (say, it's z with a count of 389),

and that your display area on screen for the bars is 75

cols long. Then the 'z' row should have

x 389

-- = ---

75 389

Computing this "Ratio and Proportion" translation is done

using a process I was taught in sixth grade to call "Cross

Multiply and Divide", or "Reducing Fractions":

x = 75 \* 389 / 389 ;

So you would plot 7 '\*' characters to display the

relative size of 34 to 389 on a screen of 75 columns.

Look familiar? "Cross multiply and divide" solves

ratio conversion problems.

Conversely, 'b' with a count of 160 should be:

x 160

-- = ---

75 389

x = 75 \* 160 / 389

So, the 'b' row will have 31 graph characters printed.

BECAUSE 160 / 389 is the SAME PROPORTION (RATIO) as 31 / 75

(Note: the exposition above is an EXAMPLE. You will

create a GENERALIZED PROGRAM which can process ANY INPUT

STREAM, not just the single example file I provided you.

YOU WILL NOT HARD-CODE THE DATA INTO THE PROGRAM. It's

sad that I have to say this, but lately students have done

exactly that.

5. It makes sense to store the 'count' data in an array.

Remember that you can record the counts in the array

thusly:

(after you read in a character into c with

c = cin.get() ;

and forced it to be lowercase with

c = tolower(c) ;

and confirmed that isalpha(c):

you can increment the count for which ever

character c contains with:

count[c - 'a']++ ;

(Do NOT create a 56-line switch() or multiple if() statements.)

You can have your program ask the Linux system for the

number of lines and columns the user has in hir window

with:

int lines = atoi(getenv("LINES") ;

int cols = atoi(getenv("COLUMNS") ;

Like so many tools, this won't work on Windows.

PART III

Building on the code from the previous assignments which

counted text:

This is one of the oldest of all computer report

algorithms, yet the logic is still used in programming

"Progress Bars" and "Bar Charts" in GUIs. Some things

never change, and basic tools rarely get discarded.

Count the characters in an input stream, graph the counts

using a bar graph format. Scale the bars so they display

neatly on the screen with no wrapping nor scrolling.

---

Your program must be able to handle files where character

frequencies are quite large and/or quite small, and/or

exhibit great variance between the large and small counts.

That means you will have to scale the output, so the length

of the smallest line is in proper proportion to the that

of the largest, and the lengths of all lines in between

them are in proper relationship to each other.

Doing the "horizontal histogram" is not much of an

intellectual challenge, and while we programmers prefer

this kind of display because we can sort it, business

managers don't like it as much as the vertical format.

VERTICAL METHOD: The Manager's Choice.

You will need to do the same scaling, then pass through

the array of totals for each row of the vertical graph,

determining whether or not, for each position in each row,

the letter corresponding to that column merits an asterisk.

(If this kind of reasoning is strange for you, just think

about it. If you have 25 lines, what percentage of Max is

25? (It's 100% of 25, otherwise expressed as 25/25.) So,

the top line's threshold is Max or Nearly to get a star.

The second line is 24/25 of Max. Calculate that number,

and the members of the array who are greater than or equal

to it get a star, otherwise a space. And so on....

First print letter frequencies, then the graph:

a 320

b 160

c 80

d 40

e 20

f 10

g 5

h 2

i 1

j 2

k 17

l 41

m 67

n 97

o 127

p 157

q 191

r 227

s 257

t 283

u 331

v 367

w 373

x 379

y 383

z 389

----------------------------------------------------

\*

\* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

a b c d e f g h i j k l m n o p q r s t u v w x y z

This isn't hard, if you think about it.

After you read in all the input data and fill the count[]

array, you then process, not the array letter by letter

as you did with the horizontal method, but instead,

you process the DISPLAY SCREEN line by line using a

for() loop (the "outer" loop.) For each line in, say,

a 60-line screen, you start with the top line, (line 60)

and calculate what is the minimum or THRESHOLD VALUE which

each element in count[] must have to get an asterisk on

that line. (What is 60/60 of Max?)

Once you've calculated this threshold, you set up a

loop (the "inner" loop), which looks at each member of

count[], and if it meets the threshold value, prints a

'\*', otherwise prints a ' '.

After you've processed the whole array, that completes one

display line, so you print a linefeed ('\n') and start

the next iteration of the "outer" loop: calculate a new

threshold based on it now being line 59 instead of 60,

(this threshold value will be lower than the previous one),

then scan through the count[] array, printing '\*' and '

' for the appropriate elements.

Obviously, the "outer" loop will be a downward-counting

loop, something like:

for (line = LINES ; line >=0 ; line--)

To print out the numeric totals vertically beneath

the columns they correspond to, come up with a general

purpose function for formatting numbers of arbitrary size

vertically on several lines.

\*

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

----------------------------------------------------

a b c d e f g h i j k l m n o p q r s t u v w x y z

3 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 2 2 2 3 3 3 3 3 3

2 6 8 4 2 1 0 0 0 0 1 4 6 9 2 5 9 2 5 8 3 6 7 7 8 8

0 0 0 0 0 0 5 2 1 2 7 1 7 7 7 7 1 7 7 3 1 7 3 9 3 9

--------------------------------------

UNIX NOTE: If you're writing this so it senses the size,

in columns and rows, of the user's terminal, you may

need to explicitly do this at the command line:

$ export LINES

$ export COLUMNS

If you encounter "segfault" errors, this may be the problem.

(I set your .bashrc files to do this, it ought to be OK.)

This document is copyright (C) 200 T. E. Harrisburg,

M.S., All Rights Reserved. It may not be stored on any

information storage and retrieval device, web page, etc.,

without the written consent of the owner.