Lab 1.2

Out: February 4 & 7, 2005

What you'll learn

In this lab, you'll become more familiar with C arrays and how to manipulate them. You've already used C arrays to hold characters—in this lab we'll try other types of arrays, including two-dimensional arrays (arrays of arrays).

How you'll do it

In the first part of this lab, you'll try two different ways to calculate the number of bytes that your system uses to store different data types.

The second part of this lab will build on Lab 1.1. We will once again read in some characters, but instead of sorting them, we will generate charts of character bigram frequencies and print them out. A "character bigram frequency" will be explained later.

1 Task 1: one dimensional arrays and `sizeof`

The goal of this task is to figure out how many bytes C uses to store integers and characters and to get a taste of what we can do with this information. In class, you learned about the `sizeof` function[1]. We'll use it to verify our results.

An array is just a list of items of a certain type. Each item is stored in a place in memory, and the array indicates the memory address of the first item in the array. The bracket operator lets you access the items in the array. First, you'll print out (in hexadecimal) all the memory addresses of the items in the array without using `sizeof`. You can then subtract two adjacent addresses to see how much space each item takes up. To verify that the sizes that you have are correct, use `sizeof` and print out the resulting integer.

[1]Actually, `sizeof` is not a function, and you won't find it mentioned in `stdlib.h` or anything. It's a built-in compiler feature that looks like a function, and it is evaluated at compile-time, as opposed to a function which would be evaluated at run-time.

Task:

- Declare two arrays, one of integers and one of characters. Make them both 10 items long. It is not necessary to initialize the arrays.
- Print out the address of each item in both arrays, 10 addresses per array.
- Subtract two adjacent addresses to calculate the size of an integer and the size of a character, then verify using `sizeof`.

2 Task 2: two dimensional arrays

A character bigram frequency chart counts how often each *pair* of characters occurs in a given chunk of text. For example, the line of text "All for one, and one for all!" has the following character bigram frequencies (row, col):

	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
a	0	0	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
e	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
j	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	0	0	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	2	0	0	0	0	0	0	0	0
p	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

To generate such a chart, it's necessary to process the line of text and and count all the occurrences of each pair of characters. There are 26 letters in the English alphabet, so there are 26*26 lowercase character pairs in English. Thus in counting the character pairs, you'll need a data structure that can store 26*26 different numbers. We would like you to use a two-dimensional array (an array of integer arrays) to store your counts.

You may find some of the code that you wrote in Lab 1.1 to be helpful here—youshould screen out any characters that you want to ignore, like punctuation, or spaces, and turn any capital letters into their lowercase versions. This means that in the example above, "All for one, and one for all!" produces the same chart as "allforoneandoneforall".

Task: Write a program that reads in a string of any length (terminated by a newline character) and generates a character bigram frequency chart for the string. Any uppercase characters should be converted to lowercase, and any other characters should be treated as if they were not in the text being processed.