CS 240 Summer 2017

Lab 3: Programming using pointers, arrays, and strings (220 pts)

Due: 07/05/2017 (Wed), 11:59 PM

Reading

Read chapters 4 and 5 from K&R (textbook).

Lab 3 Code Base

A tarball of lab3/ is available as

/u/data/u3/park/pub/cs240/lab3.tar

None of the coding problems directly depend on the code in lab3.

Problems [220 pts]

Problem 1 (20 pts)

Write a program that uses uses sizeof() to print the size of all types in Chapter 2 of K & R and their composition with type qualifiers. Deposit main.c in a subdirectory typesize/ created under lab3/.

Problem 2 (50 pts)

Write a program, main() in main.c, that reads from standard input a string and writes another string to standard output. The string that it outputs is the input string whose lower/upper cases are toggled. For example, 'A' becomes 'a' and 'B' is changed to 'b'. Numeric characters '0', '1', ..., '9' are changed to 'A', 'B', ..., 'J', respectively. All other characters are replaced by a space. For example, the string

mY nAMe$Isr?G?R#9

should be translated to

My NamE iSR g r J

Please use getchar() to read from stdin and write your own string processing code (i.e., do not use string

processing libraries at this time). When using getchar(), use '\n' as the terminating symbol which gets generated when the return/enter key is pressed. The object of this exercise to write as compact a code as possible, making use of the ASCII table where upper and lower case letters and their internal number representations are defined. Comments do not count toward code length so you may be liberal with them. Deposit your code in inputstringconv/ under lab3/.

Problem 3 (60 pts)

Write a program, myencrypt, that encrypts a file using a simple substitution cipher. myencrypt reads from stdin three values

N filename1 filename2

where N is an integer (positive or negative), filename1 is the name of the file to be encrypted, and filename2 is the name of the encrypted file. myencrypt opens the file (if opening a file fails an error message should be printed before terminating) and reads its content byte-by-byte using getchar(). Each byte is treated as an ASCII character and N is added to yield a new ASCII symbol. For example, if the input byte is ASCII character 'a' (value 97) and N = 5, it is replaced (i.e., substituted) by 97 + 5 = 102 which is the ASCII symbol 'f'. To decrypt an encrypted file, we run myencrypt with input

-N filename2 filename1

The main issue to consider is that 7-bit ASCII ranges from 0 to 127. For example, if the input character is '}' (value 125) and we add 5, we get 130 which exceeds the 7-bit ASCII range. To map '}' to 0--127, we will perform wrap-around so that 130 gets interpreted as value 2 (ASCII symbol start of text). This is tantamount to performing addition modulo 128. In a similar vein, when subtraction is carried out and the resultant value becomes negative, we have to perform wrap-around to map it back to 7-bit ASCII.

At the top of the main.c file, add a description of how you go about implementing the wrap-around. Put all your functions into main.c. Test using ASCII text files (e.g., C programs) that myencrypt works correctly.

Problem 4 (90 pts)

The dot product of two vectors (x1, x2, ..., xN) and (y1, y2, ..., yN) is defined as

x1*y1 + x2*y2 + ... + xN*yN

We will define the magnitude (or norm) of (x1, x2, ..., xN) as

SQRT(x1*x1 + x2*x2 + ... + xN*xN)

where SQRT() computes the square root of its argument. Write an app that takes two vectors as input on stdin and outputs three quantities: their dot product and magnitude of each vector. The format of the input should be
N
x1 x2 ... xN
y1 y2 ... yN

where the integer in the first line specifies the size of the vectors and the numbers in the following two lines specify the components of the two vectors. Make the type of the vector components float, the calculations should be done as float, and output the results with accuracy 2 digits below the decimal point on a single line. To calculate the square root of a number, use a square root function in math libary by including math.h and linking with the "-lm" option when running gcc.

From the viewpoint of app programming, coding is straightforward. However, as an exercise in C programming, we will put forward constraints that your code must meet.

- Define two arrays of type float,

  float x[MAXSIZE], y[MAXSIZE];

  where MAXSIZE is a constant defined using the C preprocessor command, #define MAXSIZE 50, inside dotheader.h which is included in main.c. x[] and y[] are local to main() and will contain the two vectors read in from stdin. Also, define a local variable vecsize of type int inside main that will hold the size of the vectors read in from stdin.
- Reading the values of the two vectors and their size should be delegated to a function

  int read_vectors(int *, float *, float *);

  called from main() where the first argument passes the pointer to vecsize and the second and third arguments are appropriate pointers so that read_vectors() can update the local variables x[] and y[] in main(). If the input provided by a user is ill-formed (i.e., does not follow the input specification format), read_vectors() returns -1. main() should check for this value and print out a suitable error message before terminating. Use scanf() to read from stdin in read_vectors().
- Calculation of the dot product and vector magnitudes should be done by function

  float calc_dotmag(int , float *, float *, float *, float *);

  where the first argument, vecsize, is passed by value, the second and third arguments representing the vectors are passed by reference, and the last two arguments which represent the magnitude calculations are also passed by reference. The dot product value is conveyed by calc_dotmag() to main() by its return value. Declare three float variables to hold the computed results. Output the values by calling printf() directly from main().
- Put each of the three function in separate files main.c, read_vectors.c, calc_dotmag.c. Create a Makefile that compiles each C source file individually to generate object files which are then linked to generate an executable file named dotmag.exe. Make sure to set up the dependencies correctly. Code, compile, debug, and test that your app works correctly. Testing should include the case where user input is ill-formed. Make sure to comment your code. Deposit your source code (*.c and *.h) files and Makefile in dotmagcode/.

Bonus Problem (20 pts)

Providing adequate comment in C code is important for debugging and long-term software maintenance. Annotate code in Problems 2-4 so that a competent C programmer who reads your code can easily follow what is going on. In future labs, insufficient commenting will lead to penalty points (never mind bonus

points).

Turn-in Instructions

Electronic turn-in instructions:

Follow the same general directions as in lab2.

Rename lab3 to mylab3. Go to the parent directory of your mylab3 directory and type the command

turnin -v -c cs240 -p lab3 mylab3

Please note the assignment submission policy specified in the course home page.

Back to the CS 240 web page