CS 240 Summer 2021

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

Due: 07/09/2021 (Fri), 11:59 PM

Reading

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

Lab 3 Code Base

A tarball of lab3/ is available as

/homes/park/pub/cs240/lab3.tar

Unlike lab1 and lab2, none of the coding problems directly depend on the code in lab3/. Instead, code in the subdirectories of lab3/ serve as coding exercises whose lessons and skills will be needed to solve this assignment.

Problems [220 pts]

Problem 1 (70 pts)

Write a program, main() in v51/main.c under lab3/, that reads from standard input a string of characters that may not exceed length 12 excluding the newline character '\n' that is generated when the user presses ENTER (or RETURN) on a keyboard. Store the password as a string in 1-D character array candidatepw[13]. The app, chkformatpw, performs a task similar to new password checkers that determine if a password of a newly created account meets certain requirements. In our case, we will require that the length of a password entered on stdin be between 6 and 12. In addition, the characters must be chosen from lower case letters 'a', 'b', ..., 'z', upper case letters 'A', 'B', ..., 'Z', numeric characters '0', '1', '2', ..., '9', and special characters '#', '$', '%', '&'. There must be at least one character from each of the four sets. Any

password that does not satisfy the requirements will get an error message indicating which requirements were not met. When determining whether a password is sound, utilize the ASCII table to efficiently check that the requirements are met. Use the stdio library function getchar() to read the password entered on stdin character by character. The length constraint should be checked by main() along with checking that the password characters do not fall outside the four sets. The additional four constraints should be checked by calling four checker functions with function prototypes

int checklowercase(char *); // checks if lower case letter exists

int checkuppercase(char *); // checks if lower case letter exists

int checknumerical(char *); // checks if numerical character exists

int checkspecialc(char *); // checks if special character exists

which are coded in their own files checklowercase.c, ..., checkspecialc.c in lab3/v51. Each function expects as argument the password passed as a string. Each function returns 0 if its check succeeds, -1 if it fails. Create a Makefile in v51/ and compile each C source file individually to generate object files which are then linked to generate an executable file named chkformatpw. Test and verify that your implementation works correctly. Do not use string processing library functions in your code.

Problem 2 (70 pts)

Write a program, myfilehide, that encrypts a file -- ASCII or binary -- and saves the encrypted file. myfilehide reads from stdin a string that specifies the file to be encrypted. For example,

% myfilehide
a.out

myfilehide saves the encrypted content in a new file whose name has ".E" added as a suffix. After doing so, the app deletes the original file by calling remove(). In the above example, a.out.E. We restrict file names to be less than 11 characters and spaces are not allowed in a file name. Use scanf() to read the filename from stdin. The encryption app reads the content of the input file using the fgetc() stdio library function one byte at a time until EOF is reached. As with getchar(), fgetc() returns a non-negative integer where a byte read is interpreted as a 8-bit non-negative (i.e., unsigned) integer. Hence a byte containing all 1's, 11111111, is the decimal number

255. Note from our discussion in class that 11111111 is not an ASCII character since the most significant bit is 1. As an example, if a byte read from the input file is 10000001 whose decimal value is 129, the app calculates 255 - 129 = 126. The decimal value 126 has binary byte representation 11111110 which is then written to the encrypted output file using fputc(). This per-byte transformation is applied to every byte of the input file.

The above encryption method is not very secure. However, it has the nice property that running myfilehide on the encrypted file has the effect of decrypting the encrypted file so that the original content is recovered. For example,

% myfilehide
a.out.E

generates a.out.E.E whose content is identical to that of the original file a.out. Since running myfilehide multiple times keeps adding the suffix ".E" which clutters the file names, code myfilehide so that it checks if the input file name has suffix ".E". If so, the app strips the suffix ".E" from the input file name. Hence in the above example, running myfilehide on a.out.E saves the output in file a.out. After doing so, the app remove the original file a.out.E. Code myfilehide as main() in myfilehide.c in v52. Test and verify that your code works correctly. Discuss in lab3ans.pdf under what input your code may break, i.e., crash or otherwise not run as intended. Point out the issues without providing solutions which we will do in later assignments. Do not use string processing library functions in your code.

1. Since I set the size of the array, if the user input the filename which is far bigger than the size, `*** stack smashing detected ***` happens.
2. Since some ascii codes cannot be printed through the terminal, in the file, the text could be printed as some weird characters.
3. If there is `\n` in the file, then the program can sometimes cause the error.

Problem 3 (80 pts)

Given two binary vectors X = (x1, x2, ..., xN) and Y = (y1, y2, ..., yN), each a 1-D array of N binary numbers, the number of positions where corresponding bit values of the two vectors are different is called the Hamming distance of the two vectors. For example, if X = (1,1,0,0) and Y = (1,0,0,1) then their Hamming distance is 2 since X and Y differ in their second and fourth positions counting from left to right. The Hamming distance is a useful tool in various subfields of computer science including communication networks where it captures how much two binary vectors differ. In the above example, X = (1,1,0,0) may comprise four bits transmitted by a sender whereas Y = (1,0,0,1) are the actual bits received by a receiver. The Hamming distance,

d(X,Y) = 2, indicates that two bits of X flipped -- i.e., changed their value from 0 to 1, or 1 to 0 -- while traveling from sender to receiver. This is common when sending bits wirelessly using a smartphone or laptop over cellular and WiFi interfaces.

Write an app, calchamming, that takes as input two vectors whose components are binary from stdin, calculates their Hamming distance, and outputs the value to stdout. The format of the input should be

N
x1 x2 ... xN
y1 y2 ... yN

where N is an integer specifying the size of the 1-D arrays (i.e., dimension of 1-D vector), x1 x2 ... xN are the N bit values of the first vector, and y1 y2 ... yN are the bit values of the second vector. The individual bit values x1 x2 ... xN are separated by a single space. The same holds for y1 y2 ... yN. Assume that N cannot be greater than 15. Declare N, X[15], Y[15] to be local variables of main() of type int.

Perform reading of the input from a function

int readinput(int *, int *, int *);

where the first argument is a pointer to N, the second and third arguments point to the two 1-D arrays X and Y. readinput() returns 0 if successful, -1 if there is an error. For example, if N exceeds 15 then readinput() returns -1. Consider other cases that readinput() should consider as invalid input and return -1 to its caller main(). main() checks the return value of readinput() and terminates the app by calling exit(1) if it is -1. Perform calculation of the dot product by calling function

int calchamm(int, int *, int *);

where the first argument is the value of N, and the second and third arguments are pointers to the two arrays. calchamm() computes the Hamming distance of the binary vectors and returns the value to the caller main(). Since readinput() is tasked with checking that the input is valid, calchamm() can focus on performing the Hamming distance calculation. Print the Hamming distance to stdout by calling

void writeoutput(int);

where the int argument is the distance value.

Place the code of readinput() in its own file readput.c in v53/. Analogously for calchamm() and writeoutput() with filenames calchamm.c and writeoutput.c, respectively. Create a Makefile to streamline compilation as in Problem 2. Test and verify that your app works correctly.

Turn-in instructions

Electronic turn-in instructions:

i) For problems that require answering/explaining questions, submit a write-up as a pdf file called lab3ans.pdf. Place lab3ans.pdf in your directory lab3/. You can use your favorite editor subject to that it is able to export pdf files which many freeware editors do. Files submitted in any other format will not be graded. The TAs need to spend their time evaluating the content of your submission, not switching between editors or hunting down obscure document formats which wastes time and is in no one's interest.

ii) We will use turnin to manage lab assignment submissions. In the parent directory of lab3, run the command

turnin -c cs240 -p lab3 lab3

You can verify/list your submission by running: turnin -c cs240 -p lab3 -v. Please double-check that you submitted what you intended to submit.

Back to the CS 240 web page