HW2: Handling Strings and Arrays of Characters

Assigned: 4 June 2018 **Due:** 12 June 2018

Make a subdirectory "hw2" of your cs240 folder for this assignment. Copy all files from /courses/cs240/sum18/ramin/GROUP/hw2 to your hw2 subdirectory. These are test files that you will use later in this assignment.

(1) Do Exercise 1-18 in K&R:

Write pseudo code for the program trim.c before trying to write the code itself. Then write the program "trim.c", include the pseudo code in your comments at the beginning of the file, and compile it to be the executable program "trim".

Trailing blanks and tabs are defined as blanks and tabs that precede a '\n' character or an EOF to terminate a line, without any intervening characters of other kinds. You should copy a getline() function of the kind you see in the text to read in lines from stdin into an array, trim the line, and then write out the trimmed line to stdout, continuing until you encounter an EOF.

NOTE: If there are 4 non-blank lines in the input to trim, there should be 4 lines in the output, but lines consisting of only blanks, and tabs, should NOT be output. The trim function should NOT affect any blanks or tabs that do not come at the end of the line.

(2) Do Exercise 1-19 in K&R:

Write pseudo code for the program reverse before trying to write the code itself. Then write the program "reverse.c", include the pseudo code in your comments at the beginning of the file, and compile it to be the executable program "reverse". This program should read lines from stdin and write the reversed lines to stdout until it encounters an EOF.

(3) Visible Typewriter:

This will be explained further in class. Write pseudo code and code for a program vt.c which reads characters from stdin and writes to stdout. For each character input, it should print on a single line: (a) the ASCII name of the character, and (b) the hexadecimal value of the character. The program should stop when it encounters an EOF condition on stdin (which can be entered from the keyboard by typing <CTRL-D>).

To hold "names" for the ASCII characters, define an array asciiname of the form shown in /courses/cs240/sum18/ramin/GROUP/hw2/visitype.c:

char asciiname[] = "NUL\0" "SOH\0" "STX\0" . . .. . .";

You can find a list of ASCII symbol names in the ASCII Code Chart. You can also type out ASCII symbol names to the terminal screen by typing "help ascii". (To get out of help, type CRs and CTRL-Cs.) There are 128 such names taking 3 chars and a \0 each. In each character string used to define an element in the asciiname[] array, you will need to put in 1 or 2 spaces for names which are only 2 or 1

char long, so that each name takes up exactly 4 characters (the '\0' at the end of each counts as one of the 4). Note that each quoted string, as it runs out of space on one line, is ended (close quote) and continued on the next line (open quote again). The C compiler will concatenate all these quoted strings on successive lines in setting the asciiname[] array contents.

Some of the chars in the string must be quoted using the "\" quote char, for example: . . .\0 !\0 \"\0 #\0. . . , so they don't have their normal effect (for example, the " character must be written as \" so it doesn't have the effect of ending the character string). Note that this \" counts as only a SINGLE character in the string initialization, since the \ character doesn't actually get put in the string. Find the characters that need to be quoted in a table in K&R which I've mentioned in class and in the lecture notes.

Using this array and reading in a character c (as an int value), your code can print out the name of the character you've read using:

```
printf("%s", &asciiname[4*c]);
```

The 4*c brings you to the right position in the asciiname[] array to start the name of the character c (for example NUL for c == 0), and when printf encounters the \0 it will see that the string has ended. Note that the "%s" format means "string" form. That format expects a POINTER to the string, which is supplied by the "&" in front of asciiname[4*c]. This is just magic for now (I am telling you it will work), but you will understand it in a few weeks when we cover it.

(4) Generate a histogram:

Modify vt.c to give a histogram of all characters encountered before EOF, instead of a line-by-line output. The histogram should have the following form:

DEC	HEX	ASCII NAME	COUNT
0	0	NUL	3
65	41	A	1
97	61	a	2
125	7D	~	1

... and so on. There should be no lines for zero counts. Call this program hist.c, and compile it as hist.

(5) TESTING YOUR CODE

You will find three test files in your directory, trim.in, reverse.in, and vt.in. To turn in your assignment, start a typescript. Print out each of the program files above (call them trim.c, reverse.c, vt.c, and hist.c). Compile these programs (use gcc, and give the executable programs names trim, reverse, vt, and hist). For some of these files which contain non-printable characters, you will need to use the command od –x (octal dump in hexadecimal format) to see the contents of these files.

```
od  -x  trim.in
./trim  <trim.in  >trim.out
od  -x  trim.out
./reverse  <reverse.in
```

```
od  -x  vt.in
./vt  <vt.in  >vt.out
od  -x  vt.out
```

Your programs should perform the specified tasks for these .in files. The test for hist should be performed on all three files: trim.in, reverse.in and vt.in.

Include in your typescript file all tests that demonstrate how your programs work properly. As a reference, a sample `output` for hw2 is included. Print the typescript file out and turn it in, as always. The typescript should have all the items as listed in hw1.pdf. Also leave the typescript file in your hw2 directory along with the source files and executables.

(6) Grading Rubric

Your programs will be graded based on the rubric.