**Programming Assignment 2 -- CS429 Fall 2013**

Due Date: 29 Sept 2013

**Purpose**

The purpose of this assignment is mainly to (a) write a simple C program, (b) make sure that you have found all the pieces necessary to write, compile, and execute a C program, and (c) can read a file for input and write output, and (d) understand bit-level operations (shift/mask) in C. You will need to write the program, compile it, and then execute it. You can execute it with several different tests of your own, and when you are satisfied, turn the source in to the TA for grading (use "turnin").

**Motivation**

A common use of the internet is to send files from one computer to another. This is done by passing the file from one computer to another to another to another until it gets to the final computer. Each program along the way may look at the file. Since each computer may have its own character set, the contents of the file may be translated from one character code to another. So an "A" might start out as an "1000001" in ASCII, be translated to "010001" in BCD then "11000001" in EBCDIC then back to "1000001" to end up on the final machine in ASCII. Obviously, "A" is sent and "A" is received.

But not all character codes have the same set of characters. If we start with code 036 in BCD (¬) there is no such character in ASCII. Code 0x84 in EBCDIC (¢) is not in either BCD or ASCII.

And there are other problems. Linux/Unix systems terminate each line of characters with a new-line ('\n'), but Windows systems use both a carriage return ('\r') and a new-line ('\n'). So a byte with the value 0x0A may be replaced by the two byte sequence 0x0D0A.

And some bytes have special meaning. NULL bytes (0x00) are often discarded, as are DEL (0x7F). High order bits may be set to zero (or one) arbitrarily on ASCII systems. Both ASCII and EBCDIC have values which mean "End of Transmission" (EOT) (04 for ASCII and 0x7D for EBCDIC) so all bytes after an EOT can be ignored.

In addition, some files that we want to transmit may be binary, not text -- music files or images. Having bytes changed can mean the file is unreadable upon receipt.

To avoid these problems, we can take any binary file and encode it in just simple safe characters, transmit it, and then decode it on receipt. We can only use the characters that everyone agrees on which have no special meaning. These would be 'A' to 'Z' and '0' to '9'. Lower case letters may be translated to upper case by some machine. This gives us 26 plus 10 or 36 values that we know will be maintained by any machine. We might be able to use a few more -- blank, period, comma -- but it gets risky. A machine might think that one blank is as good as a sequence of blanks.

Since it is easiest to work with bits -- everything in the machine is in bits -- it would be easiest to work with a power of two. That would be 32. So we want to encode everything into 5 bit chunks, each 5 bits defining one character in the range "A-Z" and "0-5". Then we can decode it by converting back from "A-Z" and "0-5" to 5 bit chunks.

**Program Specification**

Your problem is to then write a program to translate from a binary Linux file into just "A-Z" and "0-5" and back. Encode and decode. Your program should take one file name as its input on the command line and produce on standard output a translated file. It will be graded by a script which will feed it files and compare the result with the correct result. If your output matches, your program is correct. Let's call your program 5bit. You should write and turnin "5bit.c".

To be sure that we don't run into problems with long lines, print only 72 characters per line. Why only 72? -- [punched cards](http://en.wikipedia.org/wiki/Punched_card)have 80 columns and the last 8 are for sequencing numbers.

To decode, instead of encoding, pass the option flag "-d". For decoding, ignore any characters that are not "A-Z" or "0-5". So newlines (and carriage returns) and other characters are ignored on input. And ignore the trailing 0 bits that are not enough to make an 8-bit byte.

**Example**

For example, if we have a file that contains "Four score\n" (four.txt) this is the bit stream:

0100011001101111011101010111001000100000011100110110001101101111011100100110010100001010

and breaking it up into 5-bit chunks gives:

01000 11001 10111 10111 01010 11100 10001 00000 01110 01101 10001 10110 11110 11100 10011 00101 00001 010

which is encoded as:

IZXXK2RAONRW42TFBI

Notice how we fill out with zeros to make the last "010" into "01000", which is then "I".

If we have a file (four.5b) that has "IZXXK2RAONRW42TFBI\n", then we would decode it by

% 5bit -d < four.5b

and the output would be "Four score\n".

**Test Cases**

Provided test cases are

* [four.txt](http://cs.utexas.edu/users/peterson/prog2/four.txt)which encodes to [four.5b](http://cs.utexas.edu/users/peterson/prog2/four.5b)
* [gettysburg.txt](http://cs.utexas.edu/users/peterson/prog2/gettysburg.txt) which encodes to [gettysburg.5b](http://cs.utexas.edu/users/peterson/prog2/gettysburg.5b)
* [smile.gif](http://cs.utexas.edu/users/peterson/prog2/smile.gif)which encodes to [smile.5b](http://cs.utexas.edu/users/peterson/prog2/smile.5b)

You can use any file as an input to encode and decoding the output should give you a file which is identical to the input.

**Extensions**

Extensions to consider:

* What programming techniques can make this run faster (or slower). Is speed an issue?
* We are encoding and decoding to make a file "safe" to transmit thru a network that may consider a file a text file and munge the character encodings. But a side effect is that the output file has no obvious relationship to the input -- it is encrypted. But all 5bit programs produce the same encryption, so any other student in the class can decrypt your encrypted file. What would it take to add an option to use a [Caesar cipher](http://en.wikipedia.org/wiki/Caesar_cipher)in addition? How much harder is it for the NSA to decrypt your file if you add a Caesar cipher?
* We limited our output to 5 bits to allow for 6-bit BCD (with no lower-case letters). If we allow lower-case letters and some common punctuation, we can get to 6-bits safe instead of 5-bit. What would you need to change to do 6-bits? (Google for Base64 or uuencode). What would be the benefit of a 6-bit encoding?

Due Date: 29 Sept 2013