DES Encryption

You'll find a source file, encrypt.c, along with a few sample inputs on the course homepage. Under the following curl commands, you'll be able to downlaod these same files, along with expected outputs for each input and an executable program named decrypt, which should be able to decrypt the files created by your encrypt program. The decrypt program has been compiled for the common platform systems, so it should be able to run on any of the university Linux machines. The expected output files are binary, so don't copy-and-paste them.

```
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise21/encrypt.c
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise21/input-1.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise21/input-2.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise21/input-3.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise21/expected-1.bin
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise21/expected-2.bin
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise21/expected-3.bin
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise21/decrypt
```

When you complete the encrypt.c program, it will perform DES encryption on an arbitrary text file. On the command line, you'll give it an encryption key and the name of a text input file and a binary output file for the encrypted output. The program will read the input file in 8-byte blocks, encrypt each block with the given key and write each encrypted block to the output file. When your program is complete, your should be able to run it as follows to create an output file that matches each of the expected outputs.

```
# output.bin should match expected-1.bin
$ ./encrypt abcd1234 input-1.txt output.bin

# output.bin should match expected-2.bin
$ ./encrypt passw0rd input-2.txt output.bin

# output.bin should match expected-3.bin
$ ./encrypt Lucy input-3.txt output.bin
```

If you get output that matches the expected output files, then your program is working. If you want to see decryption working, you can try out the decrypt executable provided with the exercise, and make sure you get a file back that matches the original input. You'll probably need to mark it as executable first:

```
$ chmod +x decrypt
```

Then, you can try encrypting and decrypting:

```
# After decrypting, output.txt should match input-2.txt
$ ./encrypt passw0rd input-2.txt output.bin
$ ./decrypt passw0rd output.bin output.txt
```

The partial implementation of encrypt.c already checks the command-line arguments, opens the input and output files and reads the input file a block at a time. To complete it, you'll need to do the following:

- DES requires an 8-byte key. Fill in the code to copy the first command-line argument (argv[1]) into the key array. If the given key is too short, set the remaining bytes to zero to get an 8-byte key. If the given key is too long, just use the first 8 characters.
  DES also requires a parity bit in each byte of the key. The des_setparity() function fills in the value of the parity bits, and there's already a call to this function, after you've copied the key from argv[] to the key array.
- There's already code to read input blocks, 8 bytes at a time. Add code to encrypt each block with the 8-byte key (using ecb_crypt()), and write each block to the output file.
- When you get near the end of the file, the last block might not be a full 8 bytes. For a short block like this, the return value from fread() will tell you how many bytes you successfully read. Fill in the

remaining bytes with zero before you encrypt. That way, the decrypt program will be able to tell that these bytes weren’t part of the original text input file (this technique will work for a text file, but if we were encrypting a binary file, we’d need some other way to tell that these zeros added to the end weren’t part of the original file)

Submitting Your Work

When you're done, submit the source, encrypt.c using the exercise_21 assignment on Moodle.