Use this syntax for `otp_enc_d`:

```
otp_enc_d listening_port
```

listening_port is the port that `otp_enc_d` should listen on. You will always start `otp_enc_d` in the background, as follows (the port 57171 is just an example; yours should be able to use any port):

```
$ otp_enc_d 57171 &
```

In all error situations, this program must output errors as appropriate (see grading script below for details), but should not crash or otherwise exit, unless the errors happen when the program is starting up (i.e. are part of the networking start up protocols like bind()). That is, if given bad input, once running, `otp_enc_d` should recognize the bad input, report an error to the screen, and continue to run. All error text must be output to *stderr*.

This program, and the other 3 network programs, should use "localhost" as the target IP address/host. This makes them use the actual computer they all share as the target for the networking connections.

`otp_enc`: This program connects to `otp_enc_d`, and asks it to perform a one-time pad style encryption as detailed above. By itself, `otp_enc` doesn't do the encryption - `otp_end_d`does. The syntax of `otp_enc` is as follows:

```
otp_enc plaintext key port
```

In this syntax, *`plaintext`* is the name of a file in the current directory that contains the plaintext you wish to encrypt. Similarly, *`key`* contains the encryption key you wish to use to encrypt the text. Finally, *`port`* is the port that `otp_enc` should attempt to connect to `otp_enc_d` on.

When `otp_enc` receives the ciphertext back from `otp_enc_d`, it should output it to *stdout*. Thus, `otp_enc` can be launched in any of the following methods, and should send its output appropriately:

```
$ otp_enc myplaintext mykey 57171
$ otp_enc myplaintext mykey 57171 > myciphertext
$ otp_enc myplaintext mykey 57171 > myciphertext &
```

If `otp_enc` receives key or plaintext files with bad characters in them, or the key file is shorter than the plaintext, it should exit with an error, and set the exit value to 1. If `otp_enc`cannot find the port given, it should report this error to the screen (not into the plaintext or ciphertext files) with the bad port, and set the exit value to 2. Otherwise, on successfully running, `otp_enc` should set the exit value to 0.

`otp_enc` should NOT be able to connect to `otp_dec_d`, even if it tries to connect on the correct port - you'll need to have the programs reject each other. If this happens, `otp_enc`should report the rejection and then terminate itself.

All error text must be output to *stderr*.

`otp_dec_d`: This program performs exactly like `otp_enc_d`, in syntax and usage. In this case, however, `otp_dec_d` will decrypt ciphertext it is given, using the passed-in ciphertext and key. Thus, it returns plaintext again to `otp_dec`.

`otp_dec`: Similarly, this program will connect to `otp_dec_d`and will ask it to decrypt ciphertext using a passed-in ciphertext and key, and otherwise performs exactly like `otp_enc`, and must be runnable

in the same three ways. `otp_dec` should NOT be able to connect to `otp_enc_d`, even if it tries to connect on the correct port - you'll need to have the programs reject each other, as described in `otp_enc`.

`keygen`: This program creates a key file of specified length. The characters in the file generated will be any of the 27 allowed characters, generated using the standard UNIX randomization methods. Do not create spaces every five characters, as has been historically done. Note that you specifically do not have to do any fancy random number generation: we're not looking for cryptographically secure random number generation! `rand()` is just fine. The last character keygen outputs should be a newline. All error text must be output to *stderr*, if any.

The syntax for `keygen` is as follows:

```
keygen keylength
```

Where *`keylength`* is the length of the key file in characters. `keygen` outputs to stdout. Here is an example run, which redirects stdout to a key file of 256 characters called "mykey" (note that mykey is 257 characters long because of the newline):

```
$ keygen 256 > mykey
```

Files and Scripts

You are provided with 5 plaintext files to use (one, two, three, four, five). The grading will use these specific files; do not feel like you have to create others.

You are also provided with a grading script ("p4gradingscript") that you can run to test your software. If it passes the tests in the script, and has sufficient commenting, it will receive full points (see below). EVERY TIME you run this script, change the port numbers you use! Otherwise, because UNIX may not let go of your ports immediately, your successive runs may fail!

Finally, you will be required to write a compilation script (see below) that compiles all five of your programs, allowing you to use whatever C code and methods you desire. This will ease grading. Note that only C will be allowed, no C++ or any other language (Python, Perl, awk, etc.).

Example

Here is an example of usage, if you were testing your code from the command line:

```
$ cat plaintext1
THE RED GOOSE FLIES AT MIDNIGHT
$ otp_enc_d 57171 &
$ otp_dec_d 57172 &
$ keygen 10 > myshortkey
$ otp_enc plaintext1 myshortkey 57171 > ciphertext1
Error: key 'myshortkey' is too short
$ echo $?
1
$ keygen 1024 > mykey
$ otp_enc plaintext1 mykey 57171 > ciphertext1
$ cat ciphertext1
GU WIRGEWOMGRIFOENBYIWUG T WOFL
$ keygen 1024 > mykey2
$ otp_dec ciphertext1 mykey 57172 > plaintext1_a
$ otp_dec ciphertext1 mykey2 57172 > plaintext1_b
$ cat plaintext1_a
THE RED GOOSE FLIES AT MIDNIGHT
$ cat plaintext1_b
WVIOWBTUEIOBC  FVTROIROUXA JBWE
$ cmp plaintext1 plaintext1_a
```

```
$ echo $?
0
$ cmp plaintext1 plaintext1_b
plaintext1 plaintext1_b differ: byte 1, line 1
$ echo $?
1
$ otp_enc plaintext5 mykey 57171
otp_enc error: input contains bad characters
$ otp_enc plaintext3 mykey 57172
Error: could not contact otp_enc_d on port 57172
$ echo $?
2
$
```