CS 3423 Test 2

1. **[30]** Write a C **function** with prototype:

   ```
   int extractrecs(char *fdb, int num, int number, char *fout, int rsz);
   ```

   This function will extract `number` records starting with record `num` from the file whose pathname is given by `fdb`, writing them to the file whose pathname is given by `fout`. Note that the file `fdb` will shorter by `number` records. You may assume that `fdb` contains the necessary records and the output file, after execution, should only contain the extracted records.

   You may assume that both files consist of an integral number of records and that all records have the same size, **rsz**. A record is merely a consecutive block of bytes of size **rsz**. The record indexing starts at 0 so record 0 consists of bytes 0 through $rsz - 1$, record 1 consists of bytes rsz through $2rsz - 1$, and so on. The function must both open and close the file and reads/writes **must** be record sized. If successful, **extractrecs()** will return a 0 else **extractrecs()** will return -1. The function must use low-level I/O except for printing error messages and you may assume that $rsz \leq 1024$.

2. **[25]** You have a condo in Q. Roo and want to be able to ssh to your wireless router (running openwrt) but the ip address keeps changing. So you simply have a cron job which uses netcat to send the string "condo" to the ssh port on your machine. This creates a log entry in `/var/log/messages` similar to

   ```
   Apr 20 13:15:01 Bad ssh protocol condo from 187.153.61.110
   ```

   You want a `single` program called `getcondo` which will scan the `messages` file and print out the date/time and the ip address of the condo. This can be done with the following command

   ```
   egrep condo /var/log/messages | sed 's/Bad ssh protocol condo from //'
   ```

   Write a single C program which will accomplish the same thing as the above command using the unix programs `egep` and `sed`. Note that the single quotes in the `sed` command are there only to protect the string from the shell and are not seen by `sed`.

3. **[25]** Write a C program which will create a **total** of 33 processes sharing two pipes. Each process must be assigned a unique index 0, 1, ..., 32. The processes will be divided into two groups based on their pid, those with an even pid will be in the Even group and those with an odd pid in the Odd group. Each process in the Even group will write it's index and pid to the pipe of the Odd group while each process in the Odd group will write their index and pid to the pipe of the Even group. All processes will then read as many messages as possible from their group's read pipe printing on stdout, for each message read, a line similar to:

   ```
   Process 35894 index 24 heard from process 35991 index 17
   ```

   Be careful that your code doesn't hang and messages don't get mixed up!

(OVER)

4. **[25]** Write a C `function`, called `xprogs(int fd)`, which will read **all** messages of varying size from the pipe whose file descriptor is passed to the function. A message consists of three parts, the first part is a header which contains the lengths of the second two parts,

```
struct header {
  int flen;  /* length of executable name string including null terminator */
  int alen;  /* length of single argument string including null terminator */
}
```

the second part, of length `flen`, is the null-terminated name of an executable which takes a single argument while the third part, of length `alen`, is a null-terminated argument string. The function will cause the execution of all executables with their single arguments and return -1 on error or 0 on reaching an end-of-file on the pipe.

The strings may be of any length!