CS 3423.1 Test 2

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

   ```
   int insertrecs(char *fdb, int num, char *fin, int rsz);
   ```

   This function will insert the records from the file **fin** after record **num** in the file whose pathname is given by **fdb**. You may assume that **fdb** contains the necessary records, 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 files and reads/writes **must (NOTE MUST)** be record sized. If successful, **insertrecs()** will return a 0 else **insertrecs()** will return -1. The function must use low-level I/O except for printing error messages and you may assume that $rsz \leq 4096$.

2. **[25]** In the security class that you are teaching you are doing wireless vulnerabilities but unfortunately the students let their replay injections run forever even through they have been disassociated. This means that you cannot connect and hence they can't sucessfully inject and thus have no hope of cracking the WEP key. You need to be able to see to which wireless units you can still connect so that you can correct the situation.

   Since each successful connection logs a line (slightly simplified) in **/var/log/messages** similar to

   ```
   Nov 25 14:07:15 csisl45 dhclient: bound to 10.10.67.187
   ```

   you could use the following command to see your recent connections

   ```
   sed '/bound to 10\.10/!d;s/^.*to //' /var/log/messages | sort -u
   ```

   However you want a `single` program called `wconnect` which will scan the `messages` file and print out the ip addresses of the wireless units to which we can still connect. So write a single C program which will accomplish the same thing as the above command using the unix program `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 the **sed** command.

3. **[25]** Write a C program which will create a **total** of 31 processes. Each process must be assigned a unique index 0, 1, ..., 30. Those processes with an odd **pid** (note, pid not index) will write their pid and index to each of the other processes. Every process then reads the pids sent to it printing out a message 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` with prototype

```
xpmsgs(char *fifo);
```

which will read **all** messages of varying size from the fifo whose pathname 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 fifo.

The executable name in each message is the name only, not a pathname. You may assume that the string lengths are less than 4096.