Overview: You are required to create a system of three communicating concurrent processes which does some relatively lightweight manipulation of text files. These three processes are described below.

The Driver Process: The driver process has two main responsibilities. First, it must create and interconnect the other two processes, i.e., it is the ultimate ancestor process in the system. We will further define this role after specifying the function of the other two processes.

Second, it must open an ASCII file whose filename is passed as a command line argument, and then detect words. A word is defined to be any consecutive sequence of non-whitespace characters, where a whitespace character is a SPACE, a TAB, or a NEWLINE. The driver then transforms words in accordance with two rules:

- all punctuation characters (as defined by the ispunct() library function) are stripped out of the word, and
- all characters in the word are mapped to lower case by the tolower() function.

When the driver detects a word which contains an even number of characters, it sends that word down a pipe to a process called even. It then terminates that word by sending a space down the pipe to even. It performs analogously when a word containing an odd number of characters is detected — the only difference is that a process called odd gets the space-terminated word down its pipe.

On detecting EOF on the standard input, the driver will close its output pipes and enter an infinite loop in which it sleeps for 1 second and then prints a single asterisk on its standard error. When it receives a SIGTERM signal from even, it enters a phase in which it expects to receive the output of even and odd (see below) and to display that information to the user by writing onto its standard output. The final output might appear as:

```
****************************
Words with an even number of letters:
  farp 2
  twit 6
Words with an odd number of letters:
  fubar 1
  zip 8
```

The Even and Odd Processes: The processes even and odd expect to have standard input which appears as

```
word word . . .word word
```

Each constructs a table of (word,count) pairs, updating on incoming words. When even detects

---

1 Modified copy of an original project by Phil Kearns, William and Mary.
EOF on its standard input, it will wait for 10 seconds, signal the driver as specified above, and will then write (to its standard output) a sequence of (word,count) pairs. The process called odd will do exactly the same as even, but it will not signal the driver. In the output generated by even and odd, the words will be null-terminated character sequences, and the counts will be binary integers (i.e., communication will be occurring via pipes). The driver receives and outputs those pairs in human-readable form. It is imperative that the standard output of even and odd be hooked up to the appropriate pipes which feed the driver during its output phase.

You may not make assumptions about the size of the input file. This means that you should dynamically “grow” your data structures as input is processed.

**Multiple Processes:** The driver is responsible for creating the two sub-processes even and odd. The driver will communicate to even via pipes as depicted in the figure below. (From the point of view of even, this communication will be via standard input and output — you must do the right thing to associate the pipes with even’s standard input and output.) Communication between the driver and odd is analogous.

![Diagram of process communication](image.png)

**Specifications and Notes:**
- Be sure to use low level (binary) input output operations when having the processes interact via the pipe(s). Specifically, you should be invoking the kernel entries read and write instead of things like putc and fscanf in order to do pipe i/o. You may communicate with the user (i.e, stdin and stdout of the driver) using whatever i/o constructs you choose (but you would be wise to use the standard i/o library). For the file i/o, the fscanf function does a particularly nice job, or you may use C++ file constructs.

- EOF is never detected on a pipe unless the writer closes its write descriptor for that pipe. Until the input side is closed, a read will hang indefinitely.
- Be sure to terminate this multi-task system gracefully. You should formally exit in the two children of driver, and driver should formally wait for their termination.

- You must submit two source files. The source for driver should be called driver.c. One program evenodd.c should be written to be the even and odd children. Have the process use the value of argv[1] to adapt its personality to be even or odd: the only real difference in behavior is that even will signal the driver while odd will not. In the directory where the binaries of the system will reside, there should be binaries for driver and evenodd.

- Some related man pages of interest:

  fork(2), execl(3), pipe(2), wait(2), exit(3), read(2), write(2), ispunct(3), close(2),
dup2(2), strlen(3), strcat(3), strcmp(3), signal(3), kill(2), sleep(3), getppid(2)
The number in parentheses indicates the specific section of the manual to which the command belongs. So, for example, to read the appropriate man page on wait for this project, you should execute

```
man 2 wait
```

**Write-Up:** For this project, no write-up is required.

**Submission:** Tar or zip all of the files I will need to evaluate your program into an archive with a name that identifies whose work it is and which assignment it is. Drop your archive into my netfiles inbox (lbarnett).

Note the following:
- programs submitted after the deadline will receive no credit;
- programs that do not compile will receive no credit;
- programs that do not execute and/or terminate without crashing will receive no credit;
- programs with no, or minimal, commenting cannot receive an A regardless of correct execution.

Your work on this and all projects in this course is subject to the conditions of the Honor Code as described in the course syllabus.