1.1 (C programming)

Implement a C program that

- takes from the command line two integer numbers **`n1`**, **`n2,`**
- allocates two vectors **`v1`** and **`v2`**, of dimensions **`n1`** and **`n2`**, respectively,
- fills **`v1`** with **`n1`** random even integer numbers between **`10-100`**,
- fills **`v2`** with **`n2`** random odd integer numbers between **`21-101`**,
- sort **`v1`** and **`v2`** (increasing values),
- save the content of vectors **`v1`** and **`v2`** in two text files **`fv1.txt`** and **`fv2.txt`**, respectively,
- save the content of vectors **`v1`** and **`v2`** in two **binary** files **`fv1.b`** and **`fv2.b`**, respectively,

Use command **`od`** for verifying the content of files **`v1`**, **`v2`** (**`man od`** for help)

Using standard ANSI C

```
FILE *fpw;          // file pointer
if ((fpw = fopen (filename, "wb")) == NULL){  // w for write,
                                             // b for binary
    fprinf(stderr," error open %s\n", filename);
    return(1);
}
fwrite(buffer,sizeof(buffer),1, fpw); // write sizeof(buffer)
bytes from buffer
```

Using Unix system calls

```
#include <unistd.h>
#include <fcntl.h>

int fdo;            // file descriptor
if ((fdo = open(filename, O_CREAT | O_WRONLY, 0777)) < 0){
    fprinf(stderr," error open %s\n", filename);
    return 1;
}
write(fdo, buffer, sizeof(buffer));
```

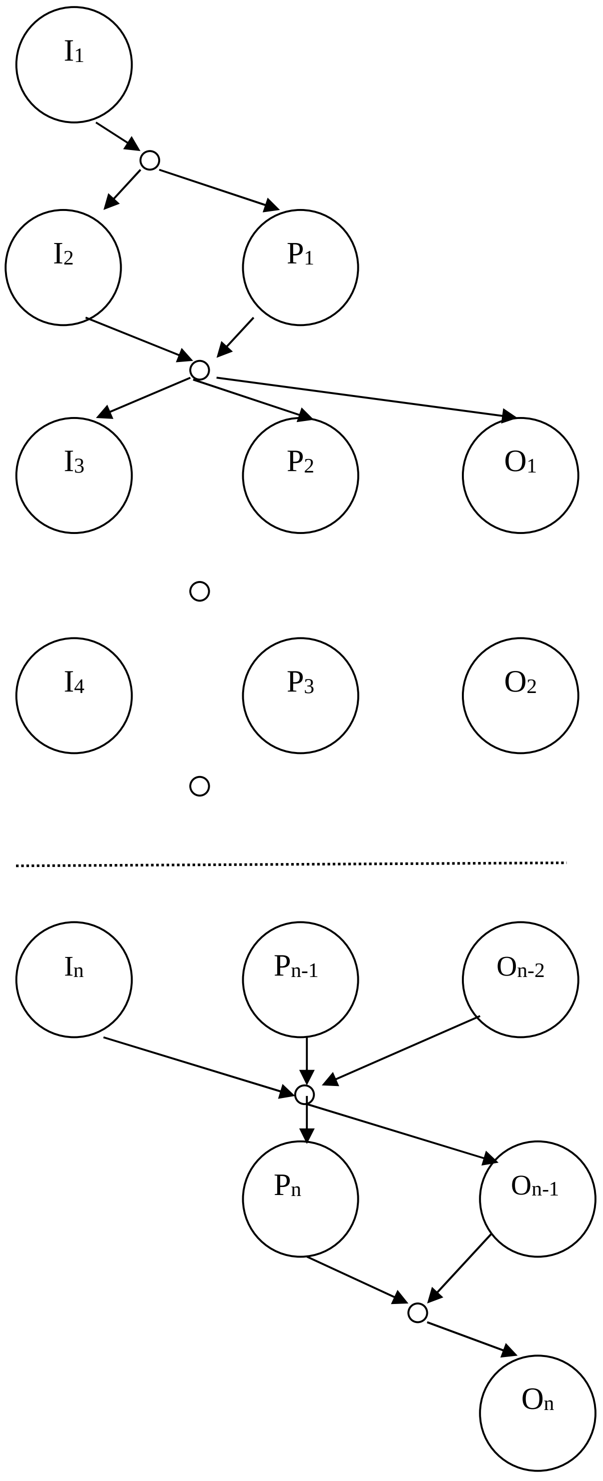
1.2 Parallel processes

Implement a concurrent C program that performs the same task of exercise **1.1**, but in concurrency.
The main process, creates two children, and waits their termination, collecting their exit status.
Each child creates a file with filename (**`v1`** or **`v2)`** corresponding to its identity, and returns its identity (**`1`** or **`2`**).

1.3 Concurrent processing with threads

Write a concurrent program in C language, using Pthreads, which takes a filename as an argument in the command line, and implements the precedence graph drawn in the next page. It represents the sequence of operations that a main thread performs for processing the content of a text file.
Each node of the graph represents a thread.
All threads are created by the main thread. There are three types of threads
The Input thread (the main thread), which gets the next character from the file in a global variable **`next.`**
Each Processing thread transforms to upper case the content of a global variable **`this.`**
Each Output thread prints the content of a global variable **`last`**.

The small circles in the graph represent synchronization points, where the main thread appropriately updates the content of the global variables.
The first small circle, for example, corresponds to the main thread executing the statement **`this = next`** so that it can create, after this statement, the thread P_1 and get in next the next character in concurrency.
The other small circles correspond to similar sequential statements.


I_1
I_2
P_1
I_3
P_2
O_1
I_4
P_3
O_2
I_n
P_{n-1}
O_{n-2}
P_n
O_{n-1}
O_n