Create a compressed tgz archive file with this syntax:
Lab <LAB_number>_<Your_Id_number>_<YourLastname>. Tgz
Example:
`Lab01_123456_Smith.tgz`
which includes all the source programs, and if necessary the input data, for each assignment.
For example: **`Lab1-123456_Smith.tgz`** must include **`Lab1.1.c, Lab1.2.c,`** and any other useful file.
Drop your **`tgz file`** in the folder **Elaborati** of the SDP site of Portale della didattica.

Lab 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 numbers between **`10-100`**,
- fills **`v2`** with **`n2`** random numbers between **`20-100`**,
- 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`**, and **`fv3`**.

Lab 1.2 (signal and kill system calls)

Write a C program that generate a child. The parent opens a file given as argument in the command line, then **loops forever**,
1) reading each line,
2) printing the line number and the line content
3) rewinding the file.

- The child process sends a **`SIGUSR1`** signal to the parent at random intervals between **`1`** and **`10`** seconds.

The first received signal must force the parent to skip the print statement.

- The next received **`SIGUSR1`** signal must force the parent to restart printing.
- This behaviour is repeated for all the received **`SIGUSR1`** signals. After **`60`** seconds, the child must send a **`SIGUSR2`** signal to the parent, and then terminates. Receiving this signal, also the parent will terminate.

Lab 1.3 (Synchronization with semaphores)

Implement a concurrent program in C language, using Pthreads, which creates two client threads, then it acts as a server.
A client thread loops reading the next number from the binary file (f**`v1.b`** and f**`v2.b`**, respectively), and storing it in a global variable. Then, it signals to the server that the variable is ready, and it waits on a semaphore a signal from the server indicating that the number has been processed (simply multiplied by 2), finally, it prints the result and its identifier.
The server loops waiting the signals of the clients, doing the multiplication, storing the results on the same global variable, and signalling to the client that the string can be printed.
The main thread waits the end on the threads, and prints the total number of served requests.