Exercise 1

- Write a C program `ex1.c` which creates `n` threads and `m` resource types which are protected by mutexes. It processes a list of resource requests read from an input file `input.txt`. Each request needs to specify the ID of the resource type, and the thread ID. The program should read `m` and `n` as command line arguments (`./ex1 $m $n`).
- The input format is as follows:

```
thread resource
1 2
3 0
1 0
3 2
1 1
3 1
```

Exercise 1

- The program should process the request `(thread=i, resource=j)` as follows:

1. Create the thread i if not existed
2. Print a message that thread i is created
3. Print the ids of mutexes currently locked by the thread i
4. Print a message that thread i tries to lock the mutex j
5. If it cannot lock, then the program should check if there is a deadlock
6. The thread i locks the mutex j for a random number of seconds x (sleep(x)) where $0<x<5$
7. The thread i unlocks the mutex j
8. Print a message that thread i unlocked mutex j
9. If all acquired mutexes are unlocked by thread i then terminate it before returning from start_routine function

Exercise 1

- When a deadlock happens, the program should print a message that "Deadlock is detected" and terminate after printing the status of all threads (which mutexes are locked and requested by which threads).
- If the program finishes processing all requests and no deadlocks occurred then the program should print a message that "No deadlocks".
- The program should process requests in parallel. After a thread is created, the program should not wait till terminating the current thread (using join) but indeed it should process the next request (creating another thread). You can join threads after finishing all requests.
- If the thread i locked a mutex then any threads cannot lock the same mutex before unlocking by thread i. We assume that we have only one instance of each resource type (protected by a mutex).

Exercise 1

- You should solve the problem based on the algorithm presented in slide 9. You can use any data structure or representation you prefer in C to store the nodes and edges of the graph.
- Save the output of your program in a file **ex1.txt** and add to it the answer to the question. Why did the deadlock occur if it is a deadlock? and Why it did not happen if it is not a deadlock?
- Submit **ex1.c**, **ex1.txt** and script **ex1.sh** to run the program.

Exercise 2

- Write a C program `ex2.c` to implement the Banker's Algorithm for deadlock avoidance:
  - Program should read the input from a file (input.txt)
  - For testing purposes consider 5 processes and 3 types of resources. However, your program must be able to process as many processes and resource types, as needed (check next slide for input file structure description)
  - Your program should either say that no deadlock is detected or print out the processes that are deadlocked.
  - Submit **ex2.c**.

Exercise 2

- Data shown on the left will be stored in the input file as shown on the right