Exercise 1: Philosopher's Problem

The goal of this exercise is to implement a deadlock. For this purpose you should recall the basic notions of threads and mutexes.

Implement the philosopher's problem in C, C++, C# or Java. The program should read a number n, the number of philosophers, a number $t1$, the maximum "thinking time" of the philosophers, and finally a number $t2$, the maximum eating time (in milliseconds).

For the sake of simplicity, the philosophers obtain an index from 0 to $n-1$. The philosophers sit around a table, i.e., the philosopher 1 is sitting between philosopher 0 and $2@, ... and $n-1$ is sitting between philosopher $n-2$ and 0.

In between each philosopher there is one fork, i.e., two philosophers who sit next to each other share one fork. Between philosopher $n-1$ and 0 is the fork with index 0, ..., and between phil $n-2$ and $n-1$ is fork number $n-1$.

Each philosopher runs in a separate thread. First they think. Once they are done thinking they are hungry and try to eat. For this purpose, they try to take a fork, first the fork on the left (i.e., if the philosopher has index i the fork with index i). If the fork is not available, the philosopher waits until the fork becomes available. (Avoid race conditions here).

After taking the left fork the philosopher tries to take the fork to his right. If this fork is not available, the philosopher waits (but he does not put back the fork in his left hand).

After the philosopher took both forks he eats. When he is done, he puts back the forks (order does not matter).

The following pseudo code shows the life of a philosopher. `take` here is a blocking operation, i.e., the thread waits until this operation can be executed. Additionally we output some status information to know what is actually going on.

```
int index = philosopher index

while(not terminated) {
    thinking_time = random number between 0 and t1

    sleep(thinking_time)

    // now time to eat

    print runtime() ": phil" index "wants to eat now".

    take fork[index]

    print runtime() ": phil " index "took fork " index
    take fork[(index + 1) % n]

    print runtime() ": phil " index "took fork " (index + 1) % n

    int eating_time = random number between 0 and t2

    sleep(eating_time)

    print runtime ": phil" index "is done eating"

    put back forks
}
```

`runtime()` here is a function that outputs the runtime of the program in milliseconds.

Subtask 1: Deadlock

Implement the philosophers and thereby show that you in fact obtain a deadlock. Experiment with different values of n, `thinking_time` and `eaching_time` and explain how these values influence the time until you obtain a deadlock in average (like "are there usually more deadlocks when n is large or small"). Try also to experimentally find the mathematical relation between these values and the deadlock (e.g., does the average time until one reaches a deadlock depend on n, $logn$ or $1/n$?)

It is not necessary to detect a deadlock for this exercise.

Subtask 2: Strategy - one philosopher switches hands.

We can avoid deadlocks by telling philosopher 0 that he should switch the order in which he takes the forks, i.e., first he should take `fork[(index+1)%n]` and only then `fork[index%n]`. The other philosophers stick to the original order (obviously if all philosophers switch the order we again obtain a deadlock).

Implement this strategy and answer the following questions:

Is the strategy fair, i.e., do all philosophers eat the same amount (we assume that total eating time is equivalent to the amount eaten)? If not, which philosophers eats the most and which philosopher eats the least?