# *ETEC3702 – Concurrency*
# *Lab 10 – Deadlock and Dining Philosophers*

**Due date: before the final exam.**

## Program 1: Dining Philosophers I

For this program you are going to write a simple simulation of the dining philosophers problem.

When the program is executed it will prompt for a number of philosophers to be entered.

The program will then create a thread for each philosopher and simulate their dining as follows:

Each philosopher will behave as follows:

- Start out with a full plate of food (100 percent)
- While there is food on the plate:
  - Think for between 0.2 and 0.6 seconds. ( use time.sleep() with random.uniform() )
  - Pick up the fork to the left and pause for 0.01 seconds.
  - Pick up the fork to the right and pause for 0.01 seconds.
  - Eat for a random amount of time between 0.2 and 0.6 seconds.
  - After eating that philosophers plate will drop from 5 to 10 whole percentage points.
  - Put the right fork back.
  - Put the left fork back.
- Display a message saying that the philosopher completed the meal.
- Exit the thread.

Notes:

- Display messages that are appropriate for each action.
- The forks for philosopher n would be fork n and fork n+1 with n+1 wrapping back to fork 0 on the last philosopher.
- A fork cannot be held by more than one philosopher at a time.
- The philosopher must pick up both forks in order to eat.

Main program:

- Prompt for and have the user input the number of philosophers / forks: n.
- Create n fork objects and n philosopher threads.
- Start all n philosopher threads.
- Wait for all philosophers to complete.
- Display a message that they've all completed.

**Note: as written this program can DEADLOCK! You might test it and it will complete, but don't be fooled, it CAN happen.**

Test your program at least 10 times.

## Program 2: Dining Philosophers II

Modify your program so that it will always complete and deadlock can never occur.

Test your modified program at least 10 times.

## Questions:

**Q1: Did your program 1 implementation ever deadlock? Describe a sequence of actions that could (or did) cause deadlock.**

**Q2: How did you ensure that your program 2 will never deadlock?**

**Q3: Would you call your method used in Program 2 deadlock prevention, deadlock detection & recovery, or deadlock avoidance? Justify your answer.**

**Q4: On average, will your modified program run more slowly or more quickly than your original? Justify your answer.**

Be sure to submit your code and the complete answers to the questions above.

Here is an example of what your output might look like with 2 philosophers:

```
Enter number of philosophers:2
P0 is thinking...
P1 is thinking...
P0 taking left fork...
P0 taking right fork...
P0 is eating...
P1 taking left fork...
P0 pausing with 90.0 left.
P0 dropping left fork...
P0 dropping right fork...
P0 is thinking...
P1 taking right fork...
P1 is eating...
P0 taking left fork...
P1 pausing with 95.0 left.
P1 dropping left fork...
P1 dropping right fork...
P1 is thinking...
P0 taking right fork...
P0 is eating...
P0 pausing with 83.0 left.

<<<some lines removed to save space>>>

P1 is eating...
P0 taking left fork...
P1 pausing with 15.0 left.
P1 dropping left fork...
P1 dropping right fork...
P1 is thinking...
P0 taking right fork...
P0 is eating...
P0 pausing with 0 left.
P0 dropping left fork...
P0 dropping right fork...
P0 is done!
P1 taking left fork...
P1 taking right fork...
P1 is eating...
P1 pausing with 10.0 left.
P1 dropping left fork...
P1 dropping right fork...
P1 is thinking...
P1 taking left fork...
P1 taking right fork...
P1 is eating...
P1 pausing with 1.0 left.
P1 dropping left fork...
P1 dropping right fork...
P1 is thinking...
P1 taking left fork...
P1 taking right fork...
P1 is eating...
P1 pausing with 0 left.
P1 dropping left fork...
P1 dropping right fork...
P1 is done!
All are done!
```