# CSCI 3020U

# Lab #5: Banker's Algorithm

**Alexandar Mihaylov 100536396**

**Jeremy Kwok - 100341977**

**Taylor Smith - 100372402**

**Elias Amal - 100494613**

**Luisa Rojas - 100518772**

## *Note: Enter must be pressed after each iteration(intended)

### Objective

The overall objective of this lab is to implement Banker's Algorithm by using threads to represent different processes/customers that make requests to the processor/banker. These requests can potentially put the system in a deadlock and as a result should be filtered to only allow requests that lead to a safe state of the system. This method allows a program to avoid deadlocking by essentially always keeping the system in a safe state.

### Program Flow

*Initialization:*
The user first runs the program by giving it the number of resources, for example by running `$ ./banker 3 5 4`.
There are customer# threads created and are initiated with a mutex lock. The max number of resources for each customer are then created with a random number generator, the need matrix is then also set to the resource_max array and the allocation matrix are all initialized to 0. This means that at the start all the customers are assumed to have no resources allocated to them. The threads/customers are then all run until completion, at which point they are joined in the main. The individual run of each thread/customer is described below.

*Thread Instance:*
Each customer then creates random requests that are created with a random number generator, and the thread is run until completion. The customer then takes the mutex lock and checks if the request it made can be granted by checking to see if the system will be left in a safe state. The request will either be granted or denied, and then the mutex will be released. If the request was **denied** then another randomly generated request will be made until it is granted. Once a request is **granted** the number of resources that are allocated to that customer will be updated. Check if the customer has all the required resources by comparing the allocated amount to the maximum amount of resources for that customer. The customer then waits for the mutex and issues the release of the resources and can finally be marked as **complete**.

*Completion:* Once each thread/customer has reached completion then the program will terminate completely.