COMP 2004
Instructor: Dr Vinicius Prado da Fonseca
Assignment 6

1. (30%) Write a program that uses 2 threads.

    a. One thread accepts user input, a series of positive integers. A negative integer is to be used as a sentinel value to exit.

    b. The second thread will sum the numbers as they are input and on receipt of a sentinel value print the sum, and exit.

    For example:

    ```
    T1: Input a number: 20
    T2: Current Total: 20
    T1: Input a number: 10
    T2: Current Total: 30
    T1: Input a number: -1
    T2: The sum of the input numbers is 30
    ```

    - The 2 threads must be running concurrently (at the same time).
    - The sum must be calculated as each number is input. i.e. The numbers must not be stored and summed at the end.
    - The number of inputs must not be artificially limited. i.e. If 1 000 000 numbers are entered, your program must be capable of performing the action. Note: You are not required to check for integer overflow.
    - Any negative integer can be used as a sentinel value. i.e. Not a specific value, -7, -13, -9000000 are all valid sentinel values.
    - The program must not use busy/wait.
    - The program must not have a race condition.
    - The program must use the synchronization tools discussed in the course.
    - The synchronization examples slides can be used.

2. (70%) A single-lane bridge connects the two NL towns of North Tilt Cove and South Conche. Farmers in the two villages use this bridge to deliver their produce to the neighbouring town. The bridge can become deadlocked if a northbound and a southbound farmer get on the bridge at the same time. (The bridge is narrow, after getting in, one is unable to back up.) Using semaphores and/or mutex locks, implement an algorithm that prevents deadlock. Implement your solution using POSIX synchronization. In particular, represent northbound and southbound farmers as separate threads. Once a farmer is on the bridge, the associated thread will sleep for a random period of time, representing travelling across the bridge. Design your program so that you can create several threads representing the northbound and southbound farmers.

Your program will take 3 arguments. The first is the maximum time in seconds that a farmer will spend on the bridge. The second is the number of northbound farmers, and the third is the number of southbound farmers. The second and third parameters represent the number of threads to create.

The output will be the order that which the farmers access the bridge.

An example run would be:

```
./a.out 5 5 8
5 northbound farmers.
8 southbound farmers.
Farmer 1 south crossed in 3 time.
Farmer 2 south crossed in 1 time.
Farmer 1 north crossed in 3 time.
Farmer 3 south crossed in 3 time.
Farmer 4 south crossed in 3 time.
Farmer 5 south crossed in 4 time.
Farmer 2 north crossed in 3 time.
Farmer 3 north crossed in 0 time.
Farmer 4 north crossed in 1 time.
Farmer 5 north crossed in 4 time.
Farmer 6 north crossed in 2 time.
Farmer 8 north crossed in 4 time.
Farmer 7 north crossed in 1 time.
```

Note: Your program solution will be starvation-free because there are a finite number of farmers crossing in each direction. Therefore it is not necessary to account for starvation in the program.