Université d'Ottawa
Faculté de génie

École de science
d'informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical
Engineering
and Computer Science

Assignemnt 1

CSI2120 Programming Paradigms

Winter 2023

Due on February 24th 2023 at 11:30 through brightspace

8% - 12 points

Instructions :

- Solve the below exercices using GO.
- You must submit a zip file containing your answer to the 3 questions below, each in a separate script.
- This is an individual assignment, not group work.

Question 1. [2+2=4]

Soit le tableau de Points défini dans le code ci-dessous.

Consider the following array of points defined in the code below:

```
package main

import "fmt"

type Point struct {
     x float64
     y float64
}

func main() {

     points := []Point{{8., 1.},{3., 2.},{7., 4.},{6., 3.}}

     fmt.Printf("point= %v\n", points[2])
}
```

a) *Implement function* `MidPoint` *that takes as input two Points defining a line and that computes:*
   a. The coordinates of the mid-point of the line
   b. The length of the line.
   Your function should print the computed mid-point and length (rounded to 2 decimal places) in the following format.

```
Points: (2,4) (6,8)
MidPoint= (4.0, 6.0)
Length= 5.66
```

b) In a `main` function, call the `Midpoint` function for all combination of two Points in the set of Points defined by the array (24 in total). These function calls are done using go functions (therefore executed in distinct threads).
Add to your program all necessary synchronisation mechanisms such that your `main` function will terminate only when all threads are completed.

i. Perform the synchronisation using channels only.

Question 2. [1+1+1+1=4]
The following code shows how to access the rows of a 2D array represented by a slice of slices.

```
package main

import "fmt"

func fct(line []float64) {

     for _,v:= range line {
        fmt.Printf("%f, ", v)
     }
}
```

```
func fct2(matrix [][]float64) {

     matrix[2][0]= 12345.6
}

func main() {
//    array := [][]float64{{7.1, 2.3, 1.1},
//                         {4.3, 5.6, 6.8},
//                         {2.3, 2.7, 3.5},
//                         {4.5, 8.1, 6.6}}

    array := [][]float64{{1.1, 7.3, 3.2, 0.3, 3.1},
                         {4.3, 5.6, 1.8, 5.3, 3.1},
                              {1.3, 2.7, 3.5, 9.3, 1.1},
                              {7.5, 5.1, 0.6, 2.3, 3.9}}

    fct2(array)
    fct(array[2][:])
}
```

a) *Write a function* `sort(tab []float64)` *that sorts a 1D slice of floating point numbers. Use the algorithm of your choice.*

b) *Write a function* `transpose(tab [][]float64)` *that transposes a matrix represented by a slice of slices (rows become columns and vice versa).*

c) *Write a function* `sortRows(tab [][]float64)` *that sorts the rows of a slice of slices. Each row must be sorted in separate thread. The function must return only when all the threads have completed their sort operation.*

d) *Test your functions by writing a* `main` *function that proceeds as follows:*
   a. *Print the input 2D array to console*
   b. *Sort the rows of this array with* `sortRows`
   c. *Transpose the array with* `transpose`
   d. *Sort the rows of this array with* `sortRows`
   e. *Transpose the array with* `transpose`
   f. *Print the output 2D array to console*

   Show your results on the two arrays in the code provided.

Question 3. [1+1+2]

The code below contains a function that creates a thread generating an infinite sequence of random numbers. This function returns the channels through which the numbers are transmitted.

```
import "math/rand"
import "sync"
func RandomGenerator(wg *sync.WaitGroup, stop <-chan bool) <-chan int
{
     intStream := make(chan int)

     go func() {
          defer func() {wg.Done()}()
          defer close(intStream)
          for {
             select {
                case <-stop:
                   return
                case intStream <- rand.Intn(1000000):
             }
          }
     }()

     return intStream
}
```

a) *Modify this function such to generate random numbers that are multiple of* `m`*.*

b) *Write a function* `Multiple(x int, m int) bool` *that checks if a number is a multiple of* `m`*.*

c) *In your* `main` *function, create three generators, one for the multiples of 5, one for the multiples of 13 and one for the multiples of 97. Using the* `select` *statement, read the three channels simultaneously; you must read a total of 100 numbers. For each value received, check if this one is a multiple of 5, 13 and/or 97. At the end of the program, show the total number of generated multiples of 5, 13 and 97.*