

Assignment 1

CSI2120 Programming Paradigms

Winter 2019

Due on February 8th before 11:00 pm in Virtual Campus

6 marks

There are [10 points] in this assignment. The assignment is worth 6% of your final mark.

All code must be submitted in go files. Screenshots, files in a format of a word editor, pdfs, handwritten solutions, etc. will not be marked and receive an automatic 0.

Question 1. Structures, Methods and Interfaces [4 points]

Create a simple delivery simulation for a company that is located in Ottawa and ships to Montreal and Toronto.

1. Create a `struct Trip` with the following fields :
 - A `string` for the name of the `destination` for the trip,
 - A `float32` for the `weight` of the load to carry on the trip,
 - A `int` for the `deadline` in hours from for the trip.
2. Create the structures `Truck`, `Pickup` and `TrainCar` with the following fields (Note that the default values are given in brackets):
 - A `string` for the `vehicle` of it (Truck, Pickup and TrainCar),
 - A `string` for the `name` of it (Truck, Pickup and TrainCar),
 - A `string` for the name of the `destination` (“”),
 - A `float32` for the average `speed` (40, 60 and 30),
 - A `float32` for the carrying `capacity` (10, 2 and 30),
 - A `float32` for the `load` the vehicle is assigned to carry (0),
 - In addition, the `Pickup` will have a `bool` field `isPrivate` (true) and the `TrainCar` will have an extra `string` field `railway` (CNR).
 - You must use embedded types in the structures minimizing duplication for full marks.
 - Implement the corresponding global functions
`NewTruck`, `NewPickUp` and `NewTrainCar` returning a structure of the corresponding type with the above initializations.
3. Create an interface `Transporter` with the following two methods

-
- `addLoad` with a `Trip` as argument returning an `error` if the transporter has insufficient capacity to carry the weight, has a different destination or cannot make the destination on time. If the current destination is empty, the destination needs to be updated to the trip's destination.
 - `print` with no argument and no return, printing the transporter to console (see below for an example)
4. Implement the following global functions
 - `NewTorontoTrip` with arguments `weight` as `float32` and `deadline` in hours as `int` returning a pointer to `Trip` with the `destination` field set to "Toronto"
 - `NewMontrealTrip` as above but with the `destination` field set to "Montreal"
 5. Implement methods of the interface `Transporter` for a **pointer** to `Truck`, `PickUp` and `TrainCar`
 6. Supply a main routine that constructs 2 `Truck`, 3 `Pickup` and 1 `TrainCar`. Then go into a loop where you ask a user to create a `Trip` where the user supplies the weight and the deadline in hours from now. You must only create trips that are on time and by transporters that can carry the weights. You are not asked to be efficient, i.e., you may assign the `Trip` to the first `Vehicle` in the list that can make the `Trip`. A trip as a whole must be assigned to one vehicle, but one vehicle can carry multiple trips if the destinations match and there is enough time. Print the list of trips after the loop.

Assume that the distance to Toronto is 400 km while it is 200 km to Montreal for the purpose of determining if a transporter will be on time.

Please see an example input / output on the next page.

Example Input/Output:

```
Destination: (t)oronto, (m)ontreal, else exit? Tor
Weight: 8
Deadline (in hours): 12
Destination: (t)oronto, (m)ontreal, else exit? mo
Weight: 8
Deadline (in hours): 20
Error: Other destination
Destination: (t)oronto, (m)ontreal, else exit? M
Weight: 8
Deadline (in hours): 12
Error: Other destination
Error: Out of capacity
Error: Out of capacity
Error: Out of capacity
Error: Out of capacity
Destination: (t)oronto, (m)ontreal, else exit? q
Not going to TO or Montreal, bye!
Trips: [{Toronto 8 12} {Montreal 8 20} {Montreal 8 12}]
Vehicles:
Truck A to Toronto with 8.000000 tons
Truck B to Montreal with 8.000000 tons
Pickup A to with 0.000000 tons (Private: true)
Pickup B to with 0.000000 tons (Private: true)
Pickup C to with 0.000000 tons (Private: true)
TrainCar A to Montreal with 8.000000 tons (CNR)
```

Question 2. Concurrency [3 points]

Write a program that uses buffered channels to monitor and lock resources. The program will use a “ComputeServer” that will use a maximum of three go routines for the calculation. The program will also use a “DisplayServer” making sure that input and output to the console is completed and interleaved.

Use two global buffered channels as semaphores and two wait groups to wait for all routines to finish before exiting.

```
const (
    NumRoutines = 3
    NumRequests = 1000
)

// global semaphore monitoring the number of routines
var semRout = make(chan int, NumRoutines)
// global semaphore monitoring console
var semDisp = make(chan int, 1)

// Waitgroups to ensure that main does not exit until all done
var wgRout sync.WaitGroup
var wgDisp sync.WaitGroup
```

Use a structure for the compute tasks:

```
type Task struct {
    a, b float32
    disp chan float32
}
```

Implement the following functions:

```
func solve(t *Task) A function that sleeps for a random time between 1 and 15 seconds, adds
the numbers a and b and sends the result on the display channel.
func handleReq(t *Task) A function that acts as intermediary between ComputeServer and
solve.
func ComputeServer() (chan *Task) A function that uses the channel factory pattern
(lambda) and listens for requests on the created channel for tasks. It calls the handleReq function.
func DisplayServer() (chan float32) A function that uses the channel factory pattern
(lambda) and listens for requests on the created channel for results to print to the console.
```

The draft main routine (to be completed) is given as follows:

```
func main() {
    dispChan := DisplayServer()
```

```
    reqChan := ComputeServer()
    for {
        var a, b float32
        // make sure to use semDisp
        // ...
        fmt.Print("Enter two numbers: ")
        fmt.Scanf("%f %f \n", &a, &b)
        fmt.Printf("%f %f \n", a, b)
        if a == 0 && b == 0 {
            break
        }
        // Create task and send to ComputeServer
        // ...
        time.Sleep( 1e9 )
    }
    // Don't exit until all is done
}
```

Example run:

```
Enter two numbers: 2.4 3
2.400000 3.000000
Enter two numbers: 8.0 1.5
8.000000 1.500000
-----
Result: 5.400000
-----
Enter two numbers: 0 0
0.000000 0.000000
-----
Result: 9.500000
-----
```

Question 3. Shared Resources [3 points]

In this question, you need to process an array of triangles. These triangles are represented by 3 points in the two-dimensional plane.

```
type Point struct {
    x float64
    y float64
}

type Triangle struct {
    A Point
    B Point
    C Point
}
```

These triangles will be stored in an array. For the purpose of this question, you must use the following initialization function for this array.

```
func triangles10000() (result [10000]Triangle) {
    rand.Seed(2120)
    for i := 0; i < 10000; i++ {
        result[i].A= Point{rand.Float64()*100.,rand.Float64()*100.}
        result[i].B= Point{rand.Float64()*100.,rand.Float64()*100.}
        result[i].C= Point{rand.Float64()*100.,rand.Float64()*100.}
    }
    return
}
```

Create two methods to calculate the area and perimeter (boundary length) of a triangle:

```
func (t Triangle) Perimeter() float64

func (t Triangle) Area() float64
```

The area of a triangle can be easily found via the determinand:

$$A_T = \frac{1}{2} \begin{vmatrix} b_x - a_x & c_x - a_x \\ b_y - a_y & c_y - a_y \end{vmatrix}$$

Create a Go function to partition a slice of `Triangle` according to the triangles' perimeter / area ratio. (Aside: This will find "slivers", important for meshing applications). The `classifyTriangle` function accepts two stacks. Triangles with a ratio greater than 1.0 will have to be put on the `highRatio Stack` while those with a ratio is less than or equal to 1.0 have to put on the `lowRatio Stack`. (see next page).

```
func classifyTriangles(highRatio *Stack, lowRatio *Stack,  
                      ratioThreshold float64, triangles []Triangle,  
                      // ? )
```

Replace the question mark with one or more arguments of your choice such as channels or other mechanisms for competitive or non-competitive synchronization (if needed).

To make this sorting more efficient, you are asked to subdivide the array of triangles into 10 slices of size 1000. You will then call the `classifyTriangles` function 10 times to perform a concurrent sort.

You must also create the `Stack` type which must be protected against simultaneous access but can be specific for type `Triangle`.

Once the processing is complete, your main function should display the number of triangles in each of the two stacks and also show the item on top of each stack.