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

Assignment 1

CSI2120 Programming Paradigms

Winter 2020

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 [5 points]

A simple program to manage ticket sales for a theatre. In the below definition the `type` and the variable `names` are shown in `Courier font`.

1. Create a `struct Play` with the following fields:
   - A `string` for the `name` of the play,
   - A slice of `Ticket` for the `purchased` tickets for the play,
   - A `time.Time` for the date when the show starts `showStart`,
   - A `time.Time` for the date when the show ends `showEnd`.
2. Create `struct Comedy` and `struct Tragedy` with the following field (Note that the default values are given in brackets) and embed a `Play.`
   - A `float32` for the number of `laughs` per minute (0.2 and 0.0),
   - A `int32` for the number of `deaths` in the play (0 and 12),
   - In addition, the fields in the embedded structure `Play` will have different default values depending on the type.
     1. `name` (Tartuffe, Macbeth)
     2. `purchased` (size 0 for both)
     3. `showStart` (Mar. 3rd at 4:00pm, April 16th at 9:30am)
     4. `showEnd` (Mar. 3rd at 5:20pm, April 16th at 12:30pm)
3. Create an interface `Show` implemented such that it can be used for `struct Comedy` and `struct Tragedy` defining methods
   - `getName() (string)` returns the name of the show,
   - `getShowStart() (time.Time)` returns the start time of the show,
   - `getShowEnd() (time.Time)` returns the end time of the show,

- `addPurchase(*Ticket)(bool)` which adds a ticket to the slice of purchased tickets and returns true if adding the ticket worked (It is to fail if ticket for the same seat has already been sold), and
- `isNotPurchased(*Ticket)(bool)` returns true if a ticket for the same seat is not already sold.

4. Create the structure `Seat` with the following fields (Note that the default values are given in brackets)
   - An `int32` for the `number` of the seat (1),
   - An `int32` for the `row` of the seat (1), and
   - A *`Category` as variable `cat` that encodes the category of the seat. (See default values for Category).
5. Create the structure `Category` with the following fields (Note that the default values are given in brackets)
   - A `const` string for the `name` of the category where the name can be `"Prime", "Standard", "Special"` ("Standard"), and
   - A `float32` for the `base` price of the category (25.0).
6. Create the structure `Ticket` with the following fields (Note that the standard go default values are fine)
   - A `string` holding the `customer` name,
   - A `*Seat` to the seat `s` purchased, and
   - A `*Show` to the show `s` the ticket is for.
7. Create the structure `Theatre` with the following fields
   - An slice of `Seat` that contains all the `seats` in the theatre
   - A slice of `Show` that lists all the `shows` coming up.
8. Implement the following global functions
   - `NewSeat` with an arguments seat number and row number both as `int32` and a position as a `const` and a category as a `*Category`
   - `NewTicket` that takes a customer name as `string`, a pointer to a seat as `*Seat` and play as `*Play` as arguments
   - `NewTheatre` that takes an `int32` as input for the number of seats and a slice of `Show` as the shows put on by the theatre.

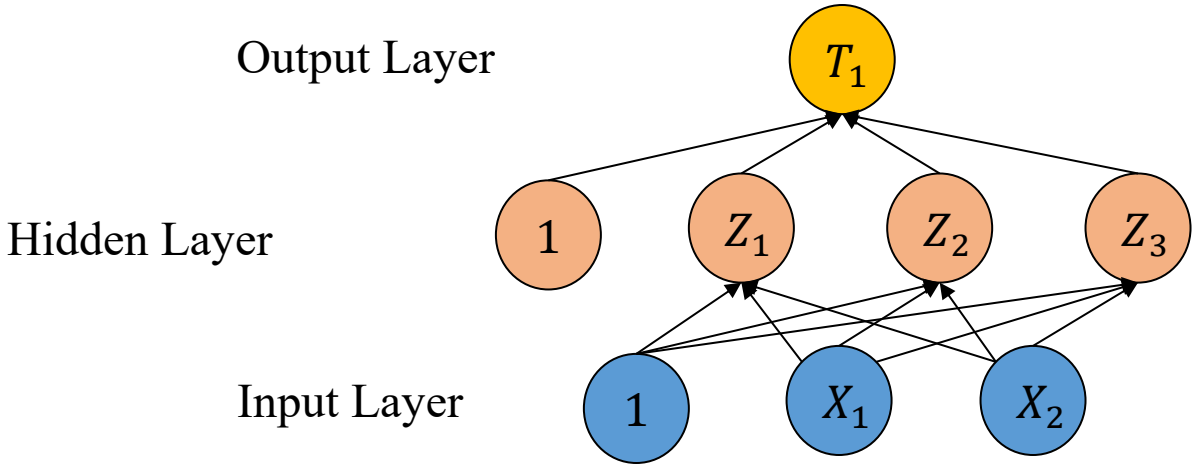
Supply a main routine that constructs a theatre with 25 seats in 5 rows evenly. The seats will have to be numbered from the front to the back of theatre starting at the front left. There are three categories of seats where row 1 is considered prime, rows 2 to 4 are standard and row 5 is special. The theatre will put on one comedy and one tragedy at two different dates of March 3rd, 2020 at 7:30pm to 10:00pm and April 10th, 2020 at 8:00pm to 11:00pm, respectively. The base price of the categories Prime, Standard and Special are 35, 25 and 15, respectively.

Once your program has initialized all the needed structures, your program has to go in a forever loop where an agent can sell tickets in a console application for these two plays. The agent is entering the play and the desired seat. If the seat is already taken but the play is not, the program has to offer an alternative seat in the same category. If the category is sold out, a seat in an alternative category has to be offered, first, in a more expensive category and only if not available a less expensive category. You only need to consider purchases of one ticket at a time.

Please define extra helper functions as needed.

Question 2. Concurrency **[5 points]**

Write a program that simulates a Neural Network where each neuron runs concurrently. We use what is called a multilayer perceptron with 1 hidden layer with an input and output layer. The input is a slice of float values and hence the input layer has as many neurons as the length of the slice, the hidden layer has three neurons and there is a single output in the network. Each neuron calculates the weighted sum of its input (including an offset) and applies the sigmoid function $\sigma(v) = \frac{1}{1+e^{-v}}$ to its output. Consider the Figure below for the example of a slice of size 2.



In the above example the equation for the first hidden neuron $Z_1 = \sigma(\alpha_{01} + \alpha_{11}X_1 + \alpha_{12}X_2)$. The equation for the output neuron is $T_1 = \sigma(\beta_{01} + \beta_{11}Z_1 + \beta_{12}Z_2 + \beta_{13}Z_3)$**.** The inputs X_1 and X_2 simply pass their value to all hidden layers.

Your program must create the network shown above with the following parameters

$$\alpha_{10} = 0.1 \;\; \alpha_{11} = 0.3 \;\; \alpha_{12} = 0.4$$
$$\alpha_{20} = 0.5 \;\; \alpha_{21} = 0.8 \;\; \alpha_{22} = 0.3$$
$$\alpha_{30} = 0.7 \;\; \alpha_{31} = 0.6 \;\; \alpha_{32} = 0.6$$

$$\beta_{10} = 0.5 \;\; \beta_{11} = 0.3 \;\; \beta_{12} = 0.7 \;\; \beta_{13} = 0.1$$

You need to design a Go program to calculate the output of the neural network based on input data. The calculation of each neuron must be done in a go routine; the neuron must therefore wait to have received the output values (communicated via a channel) from its input neurons before calculating the value of its output.

Your program must apply the neural network to a series of input values in a loop and calculate the network output for the input. These input values are $X_{1,k} = \sin\frac{2\pi(k-1)}{N}$ and $X_{2,k} = \sin\frac{2\pi(k-1)}{N}$ where $k = 0 \ldots N-1$ is the value of the current loop variable. You must make sure that the calculation at iteration

k-1 has been completed before calculating the output value at iteration k. The output value of the network is communicated to the main program via a channel. The main program will have to print the value to console. Your Go program must let the user enter the number of inputs N.