Seminario de lenguajes opción GO

Objetivo: Aprender en el lenguaje Go, definición de nuevos tipos, uso de punteros, arreglos, slices, maps y structs. Conocer cómo encapsular código en funciones y poder reusarlas. Parámetros y valores de retorno. Implementar estructuras de datos básicas y ver las que provee el lenguaje. Utilizar y programar funciones anónimas y funciones como métodos. Realizar funciones recursivas. Comparar en la medida que sea posible las características aprendidas con otros lenguajes.

1. Las temperaturas de los pacientes de un hospital se dividen en 3 grupos: alta (mayor de 37.5), normal (entre 36 y 37.5) y baja (menor de 36). Se deben leer 10 temperaturas de pacientes e informar el porcentaje de pacientes de cada grupo. Luego se debe imprimir el promedio entero entre la temperatura máxima y la temperatura mínima. Resolver cargando primero todos los valores usando un arreglo y almacenar los datos en variables escalares como acumuladores y contadores. Probar generar archivos de entrada con los valores y ejecutar, por ejemplo, de la siguiente forma:

```
go run p2-1.go < input2-1.txt
```

   a) Volver a resolver pero usando un arreglo o un *Map* de tres posiciones donde se acumulan los valores de cada grupo.

   b) Modificar la solución para incluir grupo de valores incorrectos, como pueden ser los mayores a 50° y los menores a 20°.

   c) Escribir una función que pasa de grados Celsius a Fahrenheit utilizando nuevos tipos y aplicarla al arreglo de los valores leídos. La conversión se realiza de acuerdo a la siguiente ecuación:

$$F=(C\times 9/5)+32$$

Sub-objetivo: Ver el *tipado* fuerte, usar *casting*. Operaciones y E/S con float. Arreglos. *Maps*. Constantes. Definición de nuevos tipos.

2. Implementar la función factorial de dos formas, una iterativa y otra recursiva. Escribir un programa y compilar de forma que utilice una u otra y la evalúe de 0 a 9. La función factorial se define como:

$$f(n) = n! = \begin{cases} 1 & n = 0 \\ n \times (n-1)! & n > 0 \end{cases}$$

Sub-objetivo: Recursión vs. iteración. Usar funciones.

3. Declare el tipo de datos punto cardinal como un enumerativo. Realizar un programa que lea un punto cardinal del cual viene el viento (N, S, E, O, NO, SE, NE, SO) e imprima hacia cuál se dirige. Encapsule la funcionalidad en una función.
   a. Resolver usando un switch/case.
   b. Resolver usando el orden en que fueron definidos, notar que el contrario es hacia adelante o atrás por su índice par o impar.
   c. Resolver con un *Map* que tiene como índice el tipo punto cardinal y cada elemento es el punto cardinal contrario al índice.
   d. ¿Cómo se declaran los tipos enumerativos definidos por el usuario en otros lenguajes que conoce?
   e. Definir la función que implementa la interfaz Stringer para usar con `fmt.Println` sobre un punto Cardinal.
   f. ¿Qué sucede con las funciones anteriores cuando reciben un valor fuera de rango?

   Sub-objetivo: Declarar tipos enumerativos definidos por el usuario. Usar el operador iota. Uso de *Maps*. Realizar E/S de tipos enumerativos. Funciones sobre tipos enumerativos.

4. Se debe leer tres sucesiones, de *N* (constante), números enteros cada una: $x_1 \ldots x_n$, $y_1 \ldots y_n$, $z_1 \ldots z_n$, almacenarlas en sus respectivos arreglos y calcular luego:

$$R = \left(\sum_{i=1}^{n} 1/x_i - \prod_{i=1}^{n} z_i^3 \right) \times maxmin(y_i)$$

Para la *productoria*, la *sumatoria* y el máximo-mínimo usar funciones. La función *maxmin* retorna el máximo y el mínimo de la serie y luego ambos son multiplicados por el resto de la ecuación.

Sub-objetivo: Resaltar el *tipado* fuerte de Go y usar *casting*. Operaciones con Integer y Float. Arreglos. Funciones que retornan más de un valor.

5. Escribir un programa que defina el tipo vector de flotantes de tamaño fijo, constante, con las operaciones:

```
func Initialize(v Vector ,f float64)
func Sum(v1 , v2 Vector) Vector
func SumInPlace(v1 , v2 Vector)
```

`SumInPlace`, a diferencia de la anterior, guarda el resultado de la suma en el primer vector. Investigar formas que existen para encapsular y separar el código.

Sub-objetivo: Arreglos, funciones y parámetros por referencia.

6. Escribir un programa que implemente dos funciones sobre slices. La primera recibe dos slices de enteros y retorna un tercer slice del tamaño del mínimo entre los dos sumando elemento a elemento. La segunda recibe un slice de enteros y calcula el promedio de sus elementos. Por ejemplo, las definiciones de las funciones pueden ser las siguientes:

```
func Sum(a , b []int) []int
func Avg(a []int) int
```

   a. Re-implemente la función Avg para que retorne un float.

Sub-objetivo: Slices, funciones y parámetros.

7. Se debe leer una secuencia de caracteres que finaliza con *CR* e informar la cantidad de letras, números y caracteres especiales leídos.
   a. Modificar el programa anterior para que cuente de forma separada mayúsculas de minúsculas.
   b. Modificar para que, además, cuente de forma separada las ocurrencias de cada dígito decimal. Utilice la estructura de datos Map.

      Sub-objetivo: Operaciones sobre caracteres (runas) y estructuras de control.

8. Escriba un programa que implemente la función:
   func Convert (v **int** , b **int**) **string**

   a. La función debe convertir el entero v, en un string en su representación en base b. El string será el valor de retorno. Por ejemplo si se invoca:

      s = **Convert** (23 , 2)

      En s se almacenaría el valor “10111”.

      La base debe ser mayor que 1 y menor que 37 dado que iría de base-2 hasta base-36, que usaría los dígitos:

"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"

b. Re-implemente la función considerando que v puede ser negativo, y se empleará el mismo símbolo (-) para su representación en base-b.

9. Usando memoria dinámica con punteros escribir una programa que implemente y use una lista enlazada de enteros.
   a. Definir el tipo, operaciones para agregar elementos adelante, atrás, poder iterar, etc. Las operaciones pueden ser:

```
func New() List
func IsEmpty(self List) bool
func Len(self List) int
func FrontElement(self List) int
func Next(self List) List
func ToString(self List) string
func PushFront(self List, elem int)
func PushBack(self List, elem int)
func Remove(self List) int
func Iterate(self List, f func(int) int)
```

   b. Generar el programa que utiliza las operaciones programadas.
   c. Investigar y usar el paquete: "container/list". Ver las diferencias y similitudes con su implementación. Pensar de qué forma se podría hacer de tipos genéricos.
   d. Ver de mejorar la interfaz de las funciones, por ejemplo usando métodos y códigos de error.

   Sub-objetivo: Uso de memoria dinámica, structs, funciones anónimas. Estudiar biblioteca estándar ofrecida por el lenguaje. Pensar cómo encapsular código, orientar al alumno a pensar en packages. Métodos para mejorar la interfaz y ver la posibilidad de retornar más de un valor con código de errores en los casos que sea necesario.

10. Implementar usando slices un programa que use una pila de enteros.
    a. Definir el tipo, operaciones para agregar elementos adelante, atrás, poder iterar, etc. Las operaciones pueden ser:

```
func New() Stack
func IsEmpty(s Stack) bool
func Len(s Stack) int
func ToString(s Stack) string
func FrontElement(s Stack) int
```

```
func Push(s Stack , element int)
func Pop(s  Stack) int
func Iterate(s Stack , f func (int) int)
```

b. Re-implementar usando la lista enlazada.

   Sub-objetivo: Uso de slices, structs, funciones anónimas. Estudiar usar código de terceros. Pensar cómo encapsular código usando packages. Ver de cómo reusar código. Pensar en definir con métodos.

11. Implementar el tipo de datos Ingresante y la funcionalidad solicitada.
    a. Definir el tipo de datos Ingresante del cual se tiene la siguiente información:
       - Apellido.
       - Nombre.
       - Ciudad de origen.
       - Fecha de nacimiento (día, mes, año).
       - Si presentó el título del colegio secundario
       - Código de la carrera en la que se inscribe (APU, LI, LS).
       - Definir la función que implementa la interfaz Stringer para usar con `fmt.Println`.

         ```
         func (i Ingresante) String() string
         ```

    b. Definir las funciones de comparación entre ingresantes por edad y por orden alfabético/lexicográfico.
    c. Cargar varios datos en un slice de ingresantes y ordenar primero por edad y luego por apellido y nombre. Investigar el package ‘‘sort’’.

       Sub-objetivo: Uso de arreglos, slices, structs, funciones de comparación. Interfaz *Stringer*. Pensar cómo encapsular código, orientar al alumno a pensar en packages. Métodos para mejorar la interfaz.

12. Un banco dispone de un listado en donde almacena la información de aquellos clientes que vienen a pagar impuestos. De cada cliente conoce: DNI, Nombre, Apellido, código de impuesto a pagar (A, B, C o D) y el monto a pagar. Se pide:

    a. Realizar la atención de los clientes hasta que se recauden al menos 10.000 pesos o hasta que se terminen los clientes.
    b. Al finalizar la atención informar el código de impuesto que más veces se pagó por los clientes que fueron atendidos.
    c. Al finalizar la atención informar, en caso de que hayan quedado, la cantidad de clientes sin atender.

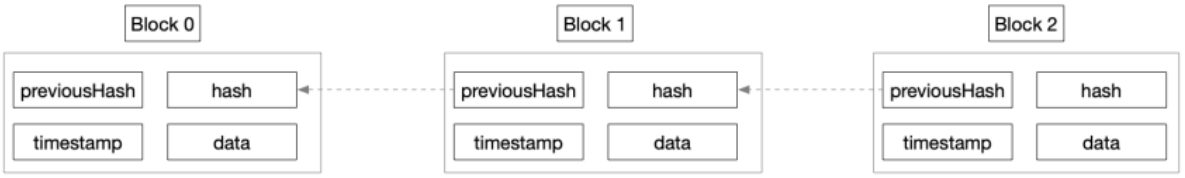
Ejercicios obligatorios

1) Usando la estructura de datos definida en el ejercicio 10 resolver el siguiente problema. Se dispone de una lista con la información de los ingresantes a la Facultad del año anterior. De cada ingresante se conoce: apellido, nombre, ciudad de origen, fecha de nacimiento (día, mes, año), si presentó el título del colegio secundario y el código de la carrera en la que se inscribe (APU, LI, LS). Con esta información de los ingresantes se pide que **recorra el listado una vez** para:
    a) Informar el nombre y apellido de los ingresantes cuya ciudad origen es "*Bariloche*".
    b) Calcular e informar el año en que más ingresantes nacieron.
    c) Informar la carrera con la mayor cantidad de inscriptos.
    d) Eliminar de la lista aquellos ingresantes que no presentaron el título.

2) Implemente una *blockchain* para que sea soporte de una *cryptomoneda* que incluya la creación de billeteras para los clientes. Una blockchain, o cadena de bloques, es un sistema digital distribuído que funciona como un libro de contabilidad público e inmutable. Almacena información sobre transacciones de forma segura y descentralizada, sin necesidad de intermediarios. Cada transacción se agrupa en un bloque, que se enlaza con el bloque anterior, creando una cadena irrompible.

    Utilice *structs* para representar la transacción (con el monto, el identificador de quien envía dinero, el identificador de quien recibe ese dinero y la fecha/hora de la transacción - timestamp -), el bloque (que tienen el hash, el hash previo, la transacción (data) y la fecha/hora de creación de dicho bloque), la cadena de bloques y la billetera (con el identificador, nombre y apellido del cliente).

    Diagrama de la cadena:



    Tip: puede utilizar la librería **crypto/sha256** para crear el hash del bloque anterior.

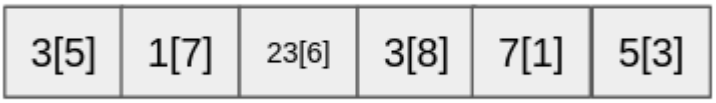
    a) Defina todos los tipos de datos que va a utilizar.
    b) Programe funciones para:
        i) Crear una billetera
        ii) Enviar una transacción

iii) Insertar un bloque en la cadena
iv) Obtener el saldo de un usuario (recorriendo toda la cadena)
v) Realizar una función que valide que la cadena sea consistente recorriéndola y verificando que el hash almacenado del bloque anterior es válido
vi) Si utilizó un *slice* para la estructura de la cadena de bloques cambie la implementación con una lista enlazada. Puede reutilizar la implementación del ejercicio 9. ¿Cuál fue el impacto que tuvo en su programa?
i) ¿Cómo validar que la transacción solo se pueda hacer si la billetera que va a enviar los fondos tiene saldo suficiente?

3) Implemente una serie de funciones para manejar *slices* de enteros que estadísticamente tienen muchas rachas de números repetidos. Utilice una estructura (que con el objetivo de ahorrar memoria) tenga en cada elemento el número entero y la cantidad de ocurrencias. Implemente:

```
func New(s slice) OptimumSlice
func IsEmpty(o OptimumSlice) bool
func Len(o OptimumSlice) int
func FrontElement(o OptimumSlice) int
func LastElement(o OptimumSlice) int
func Insert(o OptimumSlice, element int, position int) int
func SliceArray(o OptimumSlice) []int
```

Por ejemplo, si se invoca Insert con o =

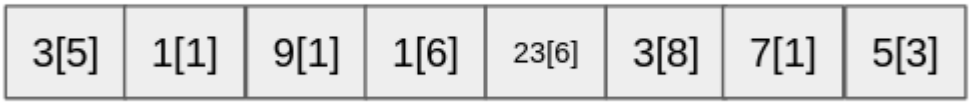


que sería la representación del arreglo:

```
{3,3,3,3,3,7,7,7,7,7,7,7,23,23,23,23,23,23,3,3,3,3,3,3,3,3,7,5,5,5}
```

y donde X[Y], X es el elemento e Y es la cantidad de ocurrencias consecutivas
element = 9
position = 6

el resultado sería:



Nota: no se permite realizar el Insert convirtiendo el OptimunSlice a un slice, insertar y luego volver a convertirlo.