TP: Administración de memoría

TP administración de memoria

- El objetivo es simular la administración de memoria dinámica.
- En Go solo se puede pedir memoria pero no tenemos que preocuparnos por liberarla. En este TP vamos a completar el ciclo y vamos a tener que liberarla a mano.
- Vamos a utilizar una librería realizada por la cátedra.
- Se pide completar *Vector.go* y *Persona.go*

Recolector de basura (garbage collector)

- El recolector de basura de un lenguaje nos abstrae del manejo de memoria.
- Libera la memoria que no utilizamos más y garantiza que no accedemos a posiciones de memoria que no sean nuestras (entre otras tareas).
- No todos los lenguajes lo tienen.

administrador_memoria.go

Que es *T*? Es un *tipo*. Puede ser *int, string,* Vector (Struct), etc. Más sobre ***generics*** la próxima clase.

- func PedirMemoria[T any]() *T
- func PedirArreglo[T any](n int) *[]T {
- func LiberarMemoria[T any](dato *T) {
- func LiberarArreglo[T any](datos *[]T)
- func RedimensionarMemoria[T any](datos *[]T, nuevoTam int) *[]T

Ejemplos:

- administrador.RedimensionarMemoria[int](vector.datos, tam_nuevo)
- administrador.PedirArreglo[int](tam)
- administrador.PedirMemoria[Persona]()

Vector

- func CrearVector(tam int) *Vector
- func (vector *Vector) Redimensionar(tam_nuevo int)
- **func (vector *Vector) Destruir()**
- **func (vector Vector) Largo() int**
- **func (vector *Vector) Guardar(pos int, elem int)**
- **func (vector Vector) Obtener(pos int) int**

Vector

```
type Vector struct {
 datos *[]int
}


// CrearVector crea un vector, utilizando el administrador de memoria
func CrearVector(tam int) *Vector {
 vec := administrador.PedirMemoria[Vector]()
 (*vec).datos = administrador.PedirArreglo[int](tam)
 return vec
}
```

Vector

```
// Redimensionar cambia el tamaño del vector
func (vector *Vector) Redimensionar(tam_nuevo int) {
 vector.datos = administrador.RedimensionarMemoria[int](vector.datos, tam_nuevo)
}
```

Persona

- func CrearPersona(nombre string, xadre *Persona) *Persona
- func (per *Persona) Imprimir()
- **func (per *Persona) Destruir()**

Persona

```
// CrearPersona devuelve un puntero a una nueva Persona, con el nombre y el padre/madre indicado (que podría ser nil)
func CrearPersona(nombre string, xadre *Persona) *Persona {
 if xadre != nil && xadre.hij_mayor != nil && xadre.hij_menor != nil {
   panic("En este modelo sólo permitimos hasta 2 hijos")
 }

 per := administrador.PedirMemoria[Persona]()
 (*per).nombre = nombre
 if xadre == nil {
   return per
 }
 (*per).xadre = xadre

 if xadre.hij_mayor == nil {
   xadre.hij_mayor = per
 } else {
   xadre.hij_menor = per
 }

 return per
}
```