Creación de una biblioteca para el manejo de muchas listas dinámicas.

Programación I – Laboratorio I.
Tecnicatura Superior en Programación.
UTN-FRA

Autores: *Ing. Ernesto Gigliotti*

Revisores: *Lic. Mauricio Dávila*

Versión : 1

Índice de contenido

1 Creación de una biblioteca para el manejo de muchas listas dinámicas

Modificaremos nuestra biblioteca para poder manejar más de una lista, es decir, la biblioteca tendrá una función que me permitirá crear una lista en particular, y luego usar las funciones existentes hasta el momento, sobre la lista generada, dando siempre la posibilidad de crear más de una lista e interactuar con cualquiera de ellas.

1.1 Definiendo la estructura que representa a la lista

Hasta el momento la única estructura que utilizamos representa a un ítem de la lista, es decir una "Persona", definiremos otra estructura que nos representa a la lista, dentro de esta estructura, definiremos los campos que necesitamos para que la lista funcione, que son las 3 variables que por el momento declaramos globales en nuestra biblioteca:

```
int size;
int index;
Persona** lista;
```

Borramos estas tres variables globales, y definimos una *struct* que representa a la lista con estas tres variables como campos:

```
struct S_PeopleList
{
    int size;
    Persona** lista;
    int index;
};
typedef struct S_PeopleList PeopleList;
```

Cada vez que creamos una variable de este tipo, tendremos dentro los campos necesarios para manejar el *array* en forma dinámica utilizando las funciones que ya escribimos (deberemos realizarles unos cambios menores que detallaremos a continuación)

1.2 Creación de una lista

Modificaremos la función *persona_initLista*, ya que ahora esta función no inicializará las variables globales (porque ya no existen) sino que creará una estructura del tipo *PeopleList* e inicilizará sus campos:

```
PeopleList* persona_initLista(void)
{
    PeopleList* pl = (PeopleList*)malloc(sizeof(PeopleList));

    pl->index=0;
    pl->size=2;
    pl->lista = (Persona**)malloc(sizeof(Persona*)*pl->size);

    return pl;
}
```

Como se observa en el código, ahora en la inicialización, lo que hacemos es crear una estructura *PeopleList* en forma dinámica e inicializamos sus campos de la misma forma que antes inicializabamos las variables globales.

1.3 Agregar ítems a una lista en particular

A continuación, modificaremos la función que agrega una persona a la lista, ya que como no existen más las tres variables globales que conformaban a "la lista", ahora la lista debe ser pasada como argumento, por lo que la función quedaría:

```
void persona_addPersona(PeopleList* pl, Persona* p)
{
    pl->lista[pl->index]=p;
    pl->index++;

    // si no hay mas lugar, pedimos más memoria para hacer un array más grande
    if(pl->index>=pl->size)
    {
      printf("no hay mas lugar, redefinimos el array\r\n");
      pl->size+=10;
      pl->lista = (Persona**)realloc(pl->lista,sizeof(Persona*)*pl->size);
    }
}
```

Como se observa en el código, el primer argumento de la función es la lista (*PeopleList*) a la que vamos a agregar un ítem, y el segundo argumento el ítem a agregar, esto nos deja ver que ahora la función es "multi-lista" y no trabaja con una sola lista global como antes, sino que la lista con la que va a operar, se pasa como argumento.

Dentro de la función seguimos haciendo lo mismo que antes, pero ahora en los lugares donde se usaban algunas de las tres variables globales, usaremos el campo de la *struct PeopleList* correspondiente.

1.4 Programa final

La única modificación que haremos sobre el programa, es que ahora la función *persona_initLista* nos devolveró un puntero a la estructura que representa a la lista, y dicho puntero deberemos pasarlo como argumento en la función *persona_addPersona*.

```
#include "Persona.h"

PeopleList* pl = persona_initLista();

do {
      Persona* persona = persona_newPersona();

      char nombreAux[20];
      preguntarNombre(nombreAux);
      if(persona_setName(persona,nombreAux))
            printf("El nombre no es valido");

      int edadAux = preguntarEdad();
      if(persona_setEdad(persona,edadAux))
            printf("La edad no es válida");

       persona_addPersona(pl,persona);
}while(preguntarSalir()!='S');
```

NOTA: Es importante aclarar que todos los espacios de memoria reservados deben ser liberados, el código se omitió en este material para simplificar la explicación.