Creación de una biblioteca para el manejo de una lista dinámica.

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 una lista dinámica

Trataremos de encapsular el manejo de nuestro tipo de dato *Persona* mediante la utilización de un archivo .c y otro archivo .h además del utilizado por la función *main*.

1.1 Creación y utilización de la biblioteca

Crearemos el archivo *Persona.h*, en donde colocaremos por el momento, la definición de la estructura con el tipo de dato *Persona*:

Persona.h

```
struct S_Persona
{
	int edad;
	char nombre[20];
}Persona;
```

En nuestro programa, vemos que es común la generación dinámica de variables del tipo *Persona*. Por lo que escribiremos en el archivo *Persona.c*, una función que se encargue de crear en forma dinámica dicha variable, y devolver un puntero hacia la misma. Llamaremos a esta función ***newPersona***.

Persona.c

```
Persona* persona_newPersona(void)
{
    Persona* persona = (Persona*)malloc(sizeof(Persona));
    return persona;
}
```

Como se observa en la definición, le agregamos al nombre de la función el prefijo "persona_" haremos esto con todas las funciones que se encuentren dentro de este archivo, de modo de agrupar la funcionalidad de "todo lo que tiene que ver con personas" dentro de este archivo, y para una fácil utilización de las funciones, ya que el programador que utilice nuestra biblioteca, sabrá que todas las funciones comienzan con "persona_xxxx"

Agregamos el prototipo de esta función en el archivo *Persona.h*:

Persona.h

```
struct S_Persona
{
	int edad;
	char nombre[20];
}Persona;

Persona* persona_newPersona(void);
```

Ahora estamos en condiciones de modificar nuestro programa para utilizar la función:

```
#include "Persona.h"

int size = 10;
int index=0;
Persona** lista = (Persona**)malloc(sizeof(Persona*)*size);
do {
       Persona* persona = persona_newPersona();
       preguntarNombre(persona->nombre);
       persona->edad = preguntarEdad();

      lista[index] = persona;
      index++;
     if(index>=size)
     {
        // incrementamos el tamaño del array
        size+=10;
        lista = realloc(lista,sizeof(Persona*)*size);
     }
}while(preguntarSalir()!='S');
```

1.1.1 Validación de campos

En muchas ocasiones, no deberíamos dejar que uno o algunos de los campos de nuestra variable *Persona*, se cargue con un valor fuera de rango, por ejemplo, la edad debe ser un número mayor a 0.
Pero en nuestro programa, debido a la asignación directa:

```
persona->edad = preguntarEdad();
```

Si la función *preguntarEdad* no realiza la validación, el valor cargado en el campo podría ser inválido. Una manera de resolver el problema sería:

```
int edadAux = preguntarEdad();

if(edadAux>0)
    persona->edad = edadAux;
else
    printf("La edad no es válida");
```

Trataremos de "migrar" el deber y la tarea de validar, a una función que asigne un valor al campo *edad* de la variable *Persona*. Como esto está muy relacionado con el tipo de dato *Persona*, agregaremos esta función en nuestro archivo *Persona.c*, y la llamaremos ***persona_setEdad***

Escribiremos la función de modo tal que le pasaremos el puntero a la variable *Persona* a la cual queremos asignarle el campo edad, y el posible valor a ser cargado (en el caso de ser correcto)

```
int persona_setEdad(Persona* pPersona, int edad) {
    if(edad>0) {
         pPersona->edad = edad;
         return 0; // OK
    }
    return 1; // error
}
```

No olvidemos colocar el prototipo de esta función en *Persona.h*

Ahora podemos escribir nuestro programa de la siguiente manera:

```
#include "Persona.h"

int size = 10;
int index=0;
Persona** lista = (Persona**)malloc(sizeof(Persona*)*size);
do {
      Persona* persona = persona_newPersona();
      preguntarNombre(persona->nombre);
      int edadAux = preguntarEdad();
      if(persona_setEdad(persona,edadAux))
          printf("La edad no es válida");
      ...
```

De manera similar, podemos validar el campo del nombre, chequeando que posea más de 3 caracteres:

```
int persona_setName(Persona* pPersona, char* pName)
{
      if(strlen(pName)>3) {
            strcpy(pPersona->nombre,pName);
            return 0;
       }
      return 1;
}
```

No olvidemos colocar el prototipo de esta función en *Persona.h*

1.1.2 Implementando la función en nuestro programa

```
#include "Persona.h"

int size = 10;
int index=0;
Persona** lista = (Persona**)malloc(sizeof(Persona*)*size);
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");
      …
```

Nuestra Biblioteca "Persona" ahora nos permite crear una variable del tipo *Persona* en forma dinámica, y poder cargar sus campos mediante validaciones de contenido.
Imaginemos que queremos imprimir toda la información de cada variable *Persona* que tenemos, podemos simplemente escribir:

```
printf("Nombre:%s - Edad:%d",persona->nombre,persona->edad);
```

en cualquier parte de nuestro programa, pero como esta funcionalidad también es parte, o tiene relación, del tipo de dato *Persona*, escribiremos una función en nuestro archivo *Persona.c* que se encargue de imprimir por pantalla dicha información, la ventaja de esta manera, es que si en algún momento agregamos o quitamos un campo a la estructura, sabemos que debemos modificar la función que imprime, que estará en el mismo archivo, y no deberemos recorrer todo nuestro programa buscando llamadas a la función *printf*. Llamaremos a la función que imprime los datos de una variable *Persona* ***persona_toString***

```
void persona_toString(Persona* pPersona)
{
   printf("Nombre:%s – Edad:%d",pPersona->nombre,pPersona->edad);
}
```

No olvidemos colocar el prototipo de esta función en *Persona.h*

1.1.3 Lectura de los campos por medio de funciones

Es una buena práctica leer los valores de los campos a través de funciones, en vez de hacerlo directamente:

```
int edadAux = persona->edad;
int edadAux = persona_getEdad(persona);
```

Como se observa en el ejemplo, dejamos de acceder al campo *edad* directamente desde el puntero a la variable *persona*, y escribimos una función, en nuestro archivo *Persona.c* que se encarga de dicha tarea.

```
int persona_getEdad(Persona* pPersona)
{
    return pPersona->edad;
}

char* persona_getNombre(Persona* pPersona)
{
      return pPersona->nombre;
}
```

No olvidemos colocar el prototipo de estas funciones en *Persona.h*

1.1.4 Agrupación de funcionalidad

Si observamos nuestro archivo *Persona.c* y *Persona.h* encontraremos la definición de un tipo de dato y una cierta cantidad de funciones que interaccionan con este tipo de dato.

```
char* persona_getNombre(Persona* pPersona);
int persona_getEdad(Persona* pPersona);
void persona_toString(Persona* pPersona);
int persona_setName(Persona* pPersona, char* pName);
int persona_setEdad(Persona* pPersona, int edad);

Persona* persona_newPersona(void);
```

Si observamos detenidamente las funciones, descubriremos un patrón que se repite en todas excepto en la función que genera la variable Persona: todas las funciones reciben como primer argumento, un puntero a una variable del tipo *Persona*.

Esta característica no es menor, ya que todas estas funciones, tienen la particularidad de realizar una tarea interaccionando con esta variable persona pasada como argumento.

Podemos decir que cada función de este tipo, tiene asignada una tarea con respecto a la variable *Persona* (por ejemplo asignar el campo *edad* de la misma) y que tiene acceso a todos los campos de la variable persona para poder resolver la tarea en cuestión, ya que le pasamos el puntero a la variable, como primer argumento.

Esto también implica que el resultado de estas funciones, dependerá exclusivamente de la variable *Persona* pasada como argumento, y no dependerá de ninguna otra variable, por ejemplo, si tenemos una variable *Persona p1* la cual tiene cargada la edad en 23 y otra variable *p2* que tiene cargada la edad en 33, cuando llamemos a la función *persona_getEdad()* nos devolverá un número que solo depende de la variable *Persona* que le pasamos como argumento:

```
int e;
e = persona_getEdad(p1);
printf("Edad de p1:%d",e);

e = persona_getEdad(p2);
printf("Edad de p2:%d",e);
```

1.1.5 Pasando el uso de la lista a la biblioteca

Para pasar la lista dinámica que tenemos creada en el *main* a la biblioteca, nos basta con cambiar de lugar la definición de las tres variables que utilizamos para manejar la lista:

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

y definirlas dentro del archivo de la biblioteca, como se observa, falta la creación del *array* mediante el uso de *malloc*, por lo que crearemos una función más en la biblioteca que se encargue de la inicialización del *array*:

```
void persona_initLista(void)
{
	size = 10;
	index=0;
	lista = (Persona**)malloc(sizeof(Persona*)*size);
}
```

También cambiaremos de lugar las líneas de código que teníamos dentro del bucle, las cuales se encargaban de copiar el puntero de *Persona* al *array* y luego verificar si el *array* se había quedado sin lugar para hacerlo crecer, por lo que crearemos la función "addPersona" la cual recibirá el puntero a agregar, y hará las tareas mencionadas.

```
void persona_addPersona(Persona* p)
{
    lista[index]=p;
    index++;

    if(index>=size)
    {
      printf("no hay mas lugar, redefinimos el array\r\n");
      size=size+10;
      lista = (Persona**)realloc(lista,sizeof(Persona*)*size);
    }
}
```

No debemos olvidar definir el prototipo de estas dos funciones en el archivo Persona.h

```
void persona_initLista(void);
void persona_addPersona(Persona* p);
```

Con estas dos funciones, hemos migrado el manejo de la lista dinámica a la biblioteca, por lo que nuestro programa en el *main* quedará mucho más simple de leer y escribir.

1.1.6 Programa final utilizando la biblioteca

```
#include "Persona.h"

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