1. Creación de una biblioteca para el manejo de 1 (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. 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 tamaÃ±o del array  size ***+=*** 10***;***  lista ***=*** realloc***(***lista***,* sizeof*(***Persona***\*) \**** size***);***  ***}***  ***}***  **while*(***preguntarSalir***() !=*** 'S'***);*** |

* + 1. Validación de campos

|  |
| --- |
| 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'***);*** |

|  |
| --- |
| 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 vÃ¡lida"***);***  ***}***  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 vÃ¡lida"***);***  ***}***  ***...***  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. 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 valida"***);***  ***}***  ***...*** |

|  |
| --- |
| 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. 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. 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. 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. 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 vÃ¡lida"***);***  ***}***    persona\_addPersona***(***persona***);***  ***}*while*(***preguntarSalir***() !=*** 'S'***);*** |

1. Creación de una biblioteca para el manejo de mas de una (VARIAS) 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. 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. 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. 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 mÃ¡s memoria  // para hacer un array mÃ¡s 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. 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 vÃ¡lida"***);***  ***}***  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. |

1. Ejercicios y videos

[SOLUCION EJERCICIO 1](https://youtu.be/kKuvJEsxDGo?list=PLZU3OfSJlsrfKiKElmKxjCsyQQF_1ApuV)

[SOLUCION EJERCICIO 2](https://youtu.be/v-j1xhPH2R4?list=PLZU3OfSJlsrfKiKElmKxjCsyQQF_1ApuV)

[SOLUCION EJERCICIO 3](https://youtu.be/3m9wXd8FmR4?list=PLZU3OfSJlsrfKiKElmKxjCsyQQF_1ApuV)

**Ejercicio 1:**

1) Tomar el ejercicio de la clase 22 y crear la biblioteca '**People**' que pueda manejar una lista de '**Personas**' (agregar a

la lista personas)

**Ejercicio 2:**

1) Tomar el ejercicio de la clase 23 y agregar a la biblioteca del ejercicio anterior, la posibilidad de borrar un item.

**Ejercicio 3:**

1) Tomar el ejercicio de la clase 24 y agregar a la biblioteca la posibilidad de crear más de una lista.