# Práctica – Parte 2

## 2.1. Título

Tipos abstractos de datos: Pilas, Listas y Colas.

## 2.2. Objetivo Académico

El objetivo es familiarizarse con algunas estructuras de datos: pilas, listas y colas y con la definición y uso de los tipos abstractos de datos (TAD).

A través de un problema específico, una pantalla de '*scroll*' vertical, el alumno profundizará en los conceptos principales del desarrollo y uso de los tipos abstractos de datos. El alumno deberá desarrollar las funciones que envuelven los TAD de Pilas, Listas y Colas.

Adicionalmente, se ampliarán los conceptos relativos a los doble punteros, preparando las estructuras necesarias para los algoritmos de ordenación, búsqueda y árboles.

## 2.3. Enunciado

La Parte 2 de la práctica consiste en la programación de las funciones que van a componer 3 nuevos TAD:

- Pilas
  El TAD de Pilas implementa una estructura de datos tipo LIFO. Los datos se añaden a la Pila (apilar) siempre en la primera posición (cima) y solo es accesible esta posición. Para acceder al resto de datos de la estructura es necesario ir sacando secuencialmente (desapilar) cada dato de la Pila. Es una estructura LIFO, y en consecuencia el primer dato que sale es el último que se añadió.
- Listas
  El TAD de Listas implementa una estructura de datos en una lista. Los datos pueden ser añadidos o borrados por el principio (izquierda) o por el final (derecha) de la lista. El resto de estructuras de datos intermedios no son accesibles.
- Colas
  El TAD de Colas implementa una estructura de datos FIFO. Los datos se añaden a la Cola (encolar) siempre al final de la estructura y se sacan (desencolar) desde el principio de la estructura (cabecera). Es una estructura FIFO, es decir, el primer dato que sale es el primero que se añadió. Excepcionalmente, y por motivos operacionales, este TAD incluye una función atípica para las Colas que es la posibilidad de añadir un elemento a la cabecera de la cola.

Para generalizar el uso de estos TAD, los elementos que se incluyen en las estructuras de datos son siempre punteros no cualificados (void *). De esta forma, con la creación de los tipos de datos adecuados y con el uso de punteros, los TAD pueden usarse para almacenar cualquier unidad informacional.

## 2.4. Definición técnica

En esta práctica y la siguiente usaremos el array donde hemos almacenado las fichas de los discos (DISCOS *Fichas) y un array de punteros (doble puntero) que reordenará las fichas de los discos (DISCOS **Orden). Cada elemento del elemento del array Orden (Orden[i]) contiene el puntero a una ficha de discos (Fichas+j). De forma que, se pueden crear diferentes formas de recorrer el array de las fichas de discos, sin necesidad de copiar todos los datos de las fichas de discos: bastará con mover el puntero a las fichas.

En primer lugar, para entender el uso de los dobles punteros y preparar las funciones de la Parte 3 de la práctica se debe codificar la función '***InitOrden'***. Esta función, copia los punteros a las fichas de discos desde el array '*Fichas'* al array '*Orden',* en el mismo orden que están en el array '*Fichas'*.

A continuación, se deben implementar las funciones de los TAD Pilas, Listas y Colas. Los 3 TAD comparten un tipo de datos común, denominado '*NODO'* que tiene dos campos:

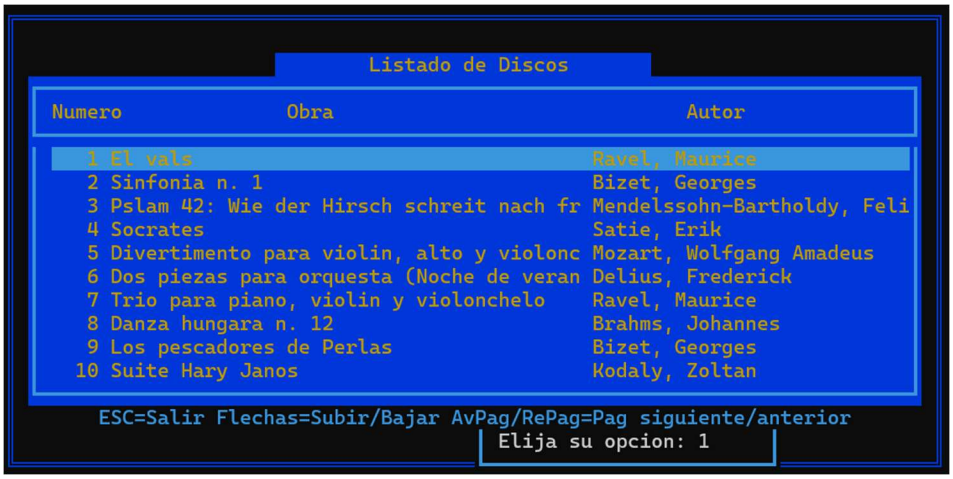
- void *Elemento: Puntero al elemento (datos) del nodo.
- Struct _nodo *Siguiente: Puntero al siguiente *NODO* del TAD.

- **Pilas**
  - El TAD Pilas utiliza únicamente el tipo de datos '*NODO'*. '*PILA *'* es igual a '*NODO *'*
  - Una Pila se identifica como un puntero a la cima (NODO *) o NULL si la Pila no está creada o no tiene elementos (es equivalente).
  - Las funciones que envuelven el TAD son:
    - CrearPila(): Devuelve un puntero a una nueva Pila (NULL).
    - PILA *Apilar(PILA *, void *): Inserta un nodo nuevo en la cima con el dato '*Elemento'*. La nueva cima tiene como siguiente nodo la antigua cima. Se devuelve un puntero a la nueva pila.
    - PILA *Desapilar(PILA *,void **): Devuelve en el segundo argumento el elemento que está en la cima y lo elimina de la pila. La nueva cima es el nodo que era el campo *'Siguiente'* de la antigua cima. Se devuelve un puntero a la nueva pila.
    - void *ConsultarCima(PILA *): Devuelve el elemento de la cima. No modifica la pila.
    - bool EsPilaVacia(PILA *): Devuelve '*true'* si la pila está vacía o no está creada o '*false'* si la pila tiene datos.

- **Listas**
  - El TAD Listas utiliza el tipo de datos '*LISTA'* que contiene los siguientes campos:
    - NODO *Primero: Puntero al primer nodo de la lista
    - NODO *Ultimo: Puntero al último nodo de la lista
  - Una Lista se identifica por el puntero a una estructura tipo LISTA.
  - Las funciones que envuelven el TAD son:
    - LISTA *CrearLista(): Devuelve un puntero a una nueva lista
    - int InsertarListaDerecha(LISTA *, void *): Inserta el elemento en un nodo que se coloca como el ultimo nodo de la lista.
    - int InsertarListaIzquierda(LISTA *, void *): Inserta el elemento en un nodo que se coloca como el primer nodo de la lista.
    - void *BorrarListaDerecha(LISTA *): Borra el ultimo nodo de la lista y devuelve un puntero al dato que contenía.
    - void *BorrarListaIzquierda(LISTA *): Borra el primer nodo de la lista y devuelve un puntero al dato que contenía.

- void *ConsultarListaDerecha(LISTA *): Devuelve un puntero al dato del último nodo de la lista. La lista no se modifica.
- void *ConsultarListaIzquierda(LISTA *) : Devuelve un puntero al dato del primer nodo de la lista. La lista no se modifica.
- int LongitudLista(LISTA *): Devuelve el número de nodos de la lista.
- bool EsListaVacia(LISTA *): Devuelve *'true'* si la lista está vacía o no está creada o *'false'* si la lista tiene datos.

- **Colas**
  - El TAD Colas utiliza el tipo de datos 'COLA' que contiene los siguientes campos:
    - NODO *Cabecera: Puntero al primer nodo de la cola
    - NODO *Final: Puntero al último nodo de la cola
  - Una Cola se identifica por el puntero a una estructura tipo COLA.
  - Las funciones que envuelven el TAD son:
    - COLA *CrearCola(): Devuelve un puntero a una nueva Cola
    - int Encolar(COLA *, void *): Inserta el elemento en un nodo que se coloca al final de la cola.
    - void *Desencolar (COLA *): Borra el ultimo nodo de la cola y devuelve un puntero al dato que contenía.
    - void *ConsultarCola(COLA *): Devuelve un puntero al dato del último nodo de la cola. La cola no se modifica.
    - int InsertarCabecera(COLA *, void *): Inserta el elemento en un nodo que se coloca al principio de la cola.
    - bool EsColaVacia(COLA *): Devuelve *'true'* si la cola está vacía o no está creada o *'false'* si la cola tiene datos.




## 2.5. Esfuerzo estimado

El esfuerzo estimado para la 'Parte 2' de la práctica es de 330 LOC (Líneas de código).

## 2.6. Ficheros proporcionados para la Parte 2

En el espacio Canvas de esta asignatura, en el apartado de la 'Parte 2' de la práctica se ha incluido un paquete denominado 'Discoteca2.zip' que contiene los siguientes ficheros:

**a. Ficheros que incluyen la codificación**

- Directorio '.vscode'
  (Hay que modificar los ficheros de este directorio para colocar el directorio donde el alumno haya instalado el compilador MinGW-64)
- Discoteca2.c
- Discoteca.h
- Colas\Colas.h
- Comun\Comun.h
- Comun\Globales.c
- Ficheros\Fichero.c
- Ficheros\Ficheros.h
- Ficheros\strsep.c
- Discos\Discos.c
- Discos\Discos.h
- Discos\LimpiarDisco.c
- Discos\NuevoDisco.c
- Listados\Listado1.c
- Listados\Listado2.c
- Listados\Listados.h
- Listas\Listas.h
- Ordenación\Ordenacion.h
- Pilas\Pilas.h
- Ventanas\Ventanas.h
- Ventanas\DibujarMenu.c
- Ventanas\LeeOpcion.c
- Ventanas\Menu.c
- Ventanas\VentanaError.c
- Ventanas\VentanaSN.c
- Ventanas\VerEstadisticas.c

**b. Ficheros que el alumno debe codificar**

- Colas\CrearCola.c
- Colas\Encolar.c
- Colas\Desencolar.c
- Colas\ConsultarCola.c
- Colas\InsertarCabecera.c
- Colas\EsColaVacia.c
- Listas\CrearLista.c
- Listas\InsertarListaDerecha.c
- Listas\InsertarListaIzquierda.c
- Listas\BorrarListaDerecha.c
- Listas\BorrarListaIzquierda.c
- Listas\ConsultarListaDerecha.c
- Listas\ConsultarListaIzquierda.c
- Listas\EsListaVacia.c
- Listas\LongitudLista.c
- Ordenación\InitOrden.c
- Pilas\CrearPila.c
- Pilas\Apilar.c
- Pilas\Desapilar.c
- Pilas\ConsultarCima.c
- Pilas\EsPilaVacia.c

**c. Objetos para usar si no se consiguió la Parte 0 y 1**

- Comun\DifTiempo.o
- Ficheros\ImportarFichero.o
- Ficheros\ExportarFichero.o
- Ficheros\DescartarFichero.o
- Ventanas\Cuadrado.o
- Ventanas\DibujarEstadisticas.o
- Ventanas\DibujarGestionFichero.o
- Ventanas\DibujarLeerAutor.o
- Ventanas\DibujarDisco.o
- Ventanas\DibujarScroll.o

Estos objetos son el resultado de compilar cada una de las funciones que había que desarrollar en la 'Parte 0' y 'Parte 1' de la práctica. Si el alumno no codificó correctamente alguna de las funciones anteriores o alguna de ellas no tiene el comportamiento esperado, puede sustituir el fichero '.c' por el correspondiente fichero '.o'.

No pueden estar a la vez el mismo fichero '.c' y '.o' porque se produciría un error de compilación.

En el fichero de configuración 'tasks.json' de esta parte de la práctica se ha añadido en 'args' las entrada 'Ventanas*.o', 'Comun*.o' y 'Ficheros*.c' para incluir todos los objetos en el proceso de compilación. Si no se va a utilizar ningún objeto '.o' (porque se van a usar la funciones '.c' desarrolladas por el alumno) hay que comentar estas líneas del fichero 'tasks.json'.

## 2.7. Módulo adicional en C# (solo para alumnos de INF)

Esta tarea se debe realizar en grupo, según los equipos formados para la 'Práctica - Parte 2'.

- Codificar en C#, usando el TAD de Pilas de .NET, un programa que dada una expresión matemática en formato INFIJO, la transforme a formato POSTFIJO, comprobando si la expresión es correcta y evalúe su valor. Debe permitir los operandos +.-,*,/ y ^(potencia). Las variables se definen como una única letra (mayúscula y minúscula). Las constantes son números de un solo dígito. A la hora de evaluar la expresión, el valor de cada variable es igual al código ascii del nombre de la variable menos 'Z' (p.ej. el valor de la variable 'f' es 'f'-'Z').

## 2.8. Módulo adicional en C (solo para alumnos del doble grado (BA-INF)

Esta tarea se debe realizar en grupo, según los equipos formados para la 'Práctica - Parte 2'.

- Codificar con Pilas y recursividad la resolución del juego de las "Torres de Hanoi", indicando el número total de movimientos que han sido necesarios.

En el siguiente enlace de la Wikipedia, se puede encontrar toda la información necesaria para resolverlo:

https://es.wikipedia.org/wiki/Torres_de_Han%C3%B3i

## 2.9. Normas Particulares

La Parte 2 de la práctica es en grupo y deberá utilizarse obligatoriamente las plantillas proporcionadas. Se puntuarán con 0 puntos aquellos ejercicios que no utilicen las plantillas.

## 2.10. Contenido de la Entrega

La entrega consiste en los siguientes ficheros/documentos comprimidos en un único fichero zip con la siguiente nomenclatura: '**Nombre_Grupo.zip**', siendo '**Nombre_Grupo**' el nombre del grupo de 'Práctica Discoteca *' al que pertenecen los alumnos que han realizado la práctica, sin acentos ni caracteres nacionales.

a. Lista de alumnos que han realizado la práctica, indicando el grupo en el que están matriculados y el porcentaje de participación de cada uno (.pdf).
b. Declaración sobre el uso de herramientas e IA (.pdf)
c. Fuentes C (.c)
d. Ficheros de cabecera (.h)
e. Ejecutable de la aplicación (Biblioteca2.exe. Hay que escanearlo para comprobar que no contenga virus u otro software malicioso)
f. Módulos adicionales
    i. Alumnos de INF
        1. Fuente C# desarrollado (.cs)
        2. Captura de pantalla con los datos de salida de la ejecución.
    ii. Alumnos de BA-INF
        1. Fuentes C desarrollados (.c y .h)
        2. Capturas de pantalla con los datos de salida de la ejecución de los diferentes escenarios.

### 2.11. Rúbrica y Pruebas Funcionales

Como mínimo la Parte 2 de la práctica debe pasar las siguientes pruebas:

- ***Pruebas técnicas***
  - El paquete enviado está nombrado adecuadamente
  - El paquete enviado contiene todos los elementos que se solicitan y en especial la declaración sobre el uso de IA
  - El programa compila sin errores ni warnings
  - Se ejecuta y muestra el menú principal
  - No se ha modificado el fuente 'Discoteca2.c'
  - No se han modificado las plantillas de las funciones a desarrollar
  - Las funciones desarrolladas por el alumno están correctamente comentadas
  - Se han desarrollado correctamente las 21 funciones solicitadas
  - Se muestra correctamente la pantalla de listado.
  - El módulo adicional compila sin errores ni warnings
  - Se han incluido las capturas de pantalla de la ejecución del módulo adicional
- ***Pruebas funcionales***

  Inicie la ejecución del programa y siga la siguiente navegación comprobando que se producen los resultados descritos.
  - Teclea '2' + 'Enter' + '1' + 'Enter'
  - Introduce 'Discoteca.csv' + 'Enter'. Se deben importar los discos del fichero
  - Se han cargado 1370 discos y se han descartado 2 discos
  - Teclea '3' + 'Enter' + '1' + 'Enter'. Se muestran los 10 primeros discos
  - Pulsa 'PgDn' continuamente hasta que se muestren los últimos discos y se llegue al final de las fichas
  - El último disco es el 'Concierto grosso n. 1' de 'Corelli, Arcangelo'
  - Pulsa 'ESC' + '3' + 'Enter' + '1' + 'Enter'. Se muestran los 10 primeros discos
  - Pulsa 'PgDn' continuamente hasta que se muestren los últimos discos y se llegue al final de las fichas
  - El último disco es el 'Concierto grosso n. 1' de 'Corelli, Arcangelo'
  - Pulsa 'PgUp' continuamente hasta que se muestren los primeros discos y se llegue al principio de las fichas
  - El primer disco es el 'El vals' de 'Ravel, Maurice'