# Ejercicios resueltos del Tema 5

***Arrays dinámicos, listas, pilas y colas***

## Ejercicio 5-01

Escribe un programa que **genere al azar un array dinámico** de N números enteros (N introducido por el usuario) y luego lo **ordene** por el método que prefieras, mostrándolo por pantalla.

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int *v;
  int i,j,n,aux;

  printf("Fund.Prog. / Tema 5 / Problema 1\n");

  // Reservamos memoria para el vector dinámico
  printf("Introduce el valor de N: ");
  scanf("%i", &n);
  v = (int*)malloc(n * sizeof(int));
  if (v == NULL)
    { printf("No hay memoria suficiente para %i enteros\n", n); exit(1); }

  // Introducimos los valores en el vector dinámico
  for (i=0; i<n; i++)
    v[i] = rand() % 1000 + 1;

  // Ordenamos el vector por el método de la burbuja
  for (i = 0; i < n; i++)
    for (j = n-1; j > i; j--)
    {
      if (v[j] < v[j-1]) {
       aux = v[j];
       v[j] = v[j-1];
       v[j-1] = aux;
      }
    }

  // Mostramos el vector
  for (i=0; i<n; i++)
    printf("%i - ", v[i]);

  free(v);

  return 0;
}
```

## Ejercicio 5-02

scribe un programa que **lea N números enteros** por teclado y los almacene en un array (N también se introducirá por teclado). Luego, se calculará la **media**, el **mínimo** y el **máximo**.

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int *v;
  int i,n;
  int maximo, minimo, suma;
  float media;


  printf("Fund.Prog. / Tema 5 / Problema 2\n");

  // Reservamos memoria para el vector dinámico
  printf("Introduce el valor de N: ");
  scanf("%i", &n);
  v = (int*)malloc(n * sizeof(int));
  if (v == NULL)
    { printf("No hay memoria suficiente para %i enteros\n", n); exit(1); }

  // Introducimos los valores en el vector dinámico
  for (i=0; i<n; i++)
  {
    printf("Introduce el valor del elemento %i: ", i);
    scanf("%i", &v[i]);
  }

  // Calculamos la media, el máximo y el mínimo
  maximo = v[0];
  minimo = v[0];
  suma = v[0];
  for (i=1; i<n; i++)
  {
    if (v[i] > maximo) maximo = v[i];
    if (v[i] < minimo) minimo = v[i];
    suma = suma + v[i];
  }
  media = (float)suma / n;

  // Mostramos el resultado
  printf("Máximo: %i\nMínimo: %i\nMedia: %2.2f\n", maximo, minimo, media);

  free(v);

  return 0;
}
```

# Ejercicio 5-03

.Escribe un programa que, utilizando arrays dinámicos, pregunte al usuario diez números enteros, almacenando los pares en un array y los impares en otro. Después, mostrará en la pantalla el contenido de ambos.

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int v[10], *vpares, *vimpares;
  int i,cont_pares,cont_impares,pos_pares,pos_impares;

  printf("Fund.Prog. / Tema 5 / Problema 3\n");
```

```
  // Pedimos al usuario los diez valores
  cont_pares = 0; cont_impares = 0;

  for (i=0; i<10; i++) {
    printf("Introduce el valor número %i: ", i+1);
    scanf("%i", &v[i]);
    // Llevamos la cuenta de cuántos pares y cuántos impares se han tecleado
    if (v[i] % 2 == 0)
      cont_pares++;       // Contador de pares
    else
      cont_impares++;           // Contador de impares
  }

  // Creamos los dos vectores dinámicos para pares e impares
  vpares = (int*)malloc(cont_pares * sizeof(int));
  vimpares = (int*)malloc(cont_impares * sizeof(int));
  if ((vpares == NULL) || (vimpares == NULL))
    { printf("¡¡Memoria insuficiente!!\n"); exit(1); }

  // Asignamos los números al vector correcto
  pos_pares = 0;    // Índice del vector vpares
  pos_impares = 0;  // Índice del vector vimpares
  for (i=0; i<10; i++)
  {
    if (v[i] % 2 == 0) {
       vpares[pos_pares] = v[i];
       pos_pares++;
    }
    else {
       vimpares[pos_impares] = v[i];
       pos_impares++;
    }
  }

  // Mostramos los vectores
  printf("\nVECTOR DE NUMEROS PARES (%i elementos):\n", cont_pares);
  for (i=0; i<cont_pares; i++)
    printf("%i - ", vpares[i]);

  printf("\nVECTOR DE NUMEROS IMPARES: (%i elementos)\n",cont_impares);
  for (i=0; i<cont_impares; i++)
    printf("%i - ", vimpares[i]);

  printf("\n");
  free(vpares); free(vimpares);

  return 0;
}
```

# Ejercicio 5-04

Escribe un programa en el que el usuario pueda decidir el tamaño de un array dinámico de números enteros. Luego, se rellenará con números al azar entre 1 y 50000, y se pedirá al usuario que introduzca un número entre esas dos cantidades. Por último, el programa buscará ese número en el array, y devolverá la posición en la que se encuentra.

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
```

```
{
  int *v;
  int n, i, busc, pos;

  printf("Fund.Prog. / Tema 5 / Problema 4\n");

  // Preguntamos la longitud del vector y reservamos memoria para él
  printf("Introduce el valor de n: ");
  scanf("%i", &n);
  v = (int*)malloc(n*sizeof(int));
  if (v == NULL) { printf("Memoria insuficiente\n"); exit(1); }

  // Asignamos valores al azar al vector
  for (i=0; i<n; i++)
    v[i] = rand() % 50000 + 1;

  // Realizamos la búsqueda
  printf("Introduce el número que buscas: ");
  scanf("%i", &busc);

  pos = -1;  // Posición donde se encuentra el número buscado
  for (i=0; i<n && pos == -1; i++)
  {
    if (v[i] == busc)
      pos = i;
  }

  // Mostramos el resultado
  if (pos != -1)
    printf("El número ha sido encontrado en la posición %i\n", pos);
  else
    printf("Ese número no está en el vector\n");

  free(v);

  return 0;
}
```

# Ejercicio 5-05

Una de las técnicas más simples de encriptación (a veces llamada “método César”) consiste en sustituir cada letra del alfabeto por la inmediatamente posterior. Es decir, todas las apariciones de la letra "A" se cambian por la letra "B", la "B" por "C", la "C" por "D", etc. De este modo, un texto como "HOLA" quedará convertido en "IPMB", que resulta ininteligible. Escribe un programa que lea un texto por teclado y lo codifique empleando esta técnica de encriptación. No utilices arrays de caracteres, sino un puntero a carácter y asignación dinámica de memoria.

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  char c[2];
  int memocupada, memreservada, i;
  char *texto;

  printf("Fund.Prog. / Tema 5 / Problema 5\n");

  printf("Escribe un texto pulsando INTRO tras cada carácter (pulsa ESC para terminar)");

  // Vamos a reservar originalmente 10 bytes para el texto
  memreservada = 10;
  // El texto actualmente ocupa 0 bytes
```

```
  memocupada = 0;

  // Reservamos memoria (10 bytes)
  texto = (char*)malloc(memreservada * sizeof(char));
  if (texto == NULL) { printf("Memoria insuficiente!"); exit(1); }

  do
  {
    gets(c);              // Leemos un carácter
    if (c[0] != 27)       // El carácter 27 (ESC) sirve para salir
    {
      texto[memocupada] = c[0];  // Almacenamos el carácter en la cadena dinámica
      memocupada++;
      if (memocupada == memreservada)
      {
       // Hemos sobrepasado la memoria reservada, así que vamos a reservar más
       // Reservaremos otros 10 bytes...
       memreservada = memreservada + 10;
       printf("Realojando %i bytes...\n", memreservada);
       // Realojamos la memoria dinámica con realloc()
       texto = (char*)realloc(texto, memreservada * sizeof(char));
       if (texto == NULL) { printf("Memoria insuficiente!"); exit(1); }
      }
    }
  }
  while (c[0] != 27);     // El carácter 27 (ESC) es la marca para salir

  texto[memocupada] = '\0';      // Colocamos el nulo al final de la cadena

  puts("El texto original es:");
  puts(texto);

  // Encriptamos la cadena
  for (i = 0; i<strlen(texto); i++)
  {
      // Sólo cambiaremos los caracteres alfabéticos (de A a Z)
      if ((texto[i] >= 'A') && (texto[i] <= 'Z') ||
          (texto[i] >= 'a') && (texto[i] <= 'z'))
        {
           if (texto[i] == 'Z') texto[i] = 'A';      // La Z se convierte en A
           else if (texto[i] == 'z') texto[i] = 'z'; // La z se convierte en a
           else texto[i] = texto[i] + 1;             //  El  resto  se  convierte  al  siguiente
carácter en la tabla ASCII
       }
  }

  puts("El texto encriptado es:");
  puts(texto);

  free(texto);

  return 0;
}
```

# Ejercicio extra de arrays dinámicos

Escribe un programa que cargue en memoria el archivo video.dat que usamos en el ejercicio del Videoclub del Tema 4. Como no conocemos en principio el tamaño del archivo, se debe reservar memoria dinámicamente para contener los datos. Después de cargar el archivo, el programa nos permitirá buscar una película por título-

(Observa que, del mismo modo que realizamos la búsqueda de la película, podríamos hacer cualquier otra operación – añadir, eliminar, modificar... – que se nos ocurriese con los datos, aunque estén almacenados en memoria dinámica)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct s_pelicula
{
  char titulo[50];
  char director[20];
  char reparto[200];
  char genero[20];
  char nacionalidad[10];
  int duracion;
  char borrado;
  int codigo;
};


int main(void)
{
  struct s_pelicula *peliculas;
  FILE *f;
  int i, tam, encontrado;
  char busc[100];

  printf("Fund.Prog. / Tema 5 / Ejercicio extra de arrays dinámicos\n");

  // Abrimos el fichero y miramos su longitud
  f = fopen("video.dat", "a+b");
  if (f == NULL) { printf("No se puede abrir el fichero video.dat\n"); exit(1); }
  fseek(f, 0, SEEK_END);  // Saltamos al final del fichero
  tam = ftell(f) / sizeof(struct s_pelicula); // Calculamos el número de registros
  fseek(f, 0, SEEK_SET);  // Nos volvemos a situal al comienzo del fichero

  // Reservamos memoria y leemeos el fichero completo
  peliculas = (struct s_pelicula*)malloc(tam*sizeof(struct s_pelicula));
  if (peliculas == NULL) { printf("Memoria insuficiente\n"); exit(1); }
  fread(peliculas, sizeof(struct s_pelicula), tam, f);
  printf("Archivo video.dat cargado con éxito (%i registros)\n\n", tam);
  fclose(f);

  // Preguntamos por el título que queremos buscar
  printf("Introduzca el título de la película que busca:\n");
  gets(busc);

  // Realizamos la búsqueda
  encontrado = 0;
  for (i = 0; i < tam && encontrado == 0; i++)
  {
    if (strcmp(peliculas[i].titulo, busc) == 0) {
      encontrado = 1;
      i--;
    }
  }

  // Mostramos el resultado
  if (encontrado == 1) {
    printf("Ficha de la película:");
    printf("Código:   %i\n", peliculas[i].codigo);
    printf("Título:   %s\n", peliculas[i].titulo);
    printf("Director: %s\n", peliculas[i].director);
    printf("Reparto:  %s\n", peliculas[i].reparto);
    printf("Género:   %s\n", peliculas[i].genero);
```

```
    printf("País:     %s\n", peliculas[i].nacionalidad);
    printf("Duración: %i min.\n", peliculas[i].duracion);
  }
  else
    printf("Película no encontrada\n");

  free(peliculas);
  return 0;
}
```

# Ejercicio 5-06

> NOTA IMPORTANTE: Los siguientes problemas usan las **funciones propias de las listas abiertas**. Esas funciones se os facilitaron en otro documento para que las tuviérais todas juntas, además de haberlas visto en clase. Por lo tanto, en todos los ejercicios que vienen a continuación supondremos que esas funciones ya están definidas. El alumno/a debería ser capaz tanto de escribir el código de esas funciones propias de las listas abiertas (insertar(), buscar(), borrar(), contar_nodos(), etc) como de escribir programas que utilicen y se aprovechen de esas funciones. Los programas que se muestran a continuación son de este segundo tipo.

El ejercicio 5-06 consiste en escribir un programa que vaya pidiendo números enteros al usuario y los guarde en una lista abierta de números enteros. Cada vez que se teclee un número, éste debe añadirse a la lista, hasta que el usuario introduzca un número negativo. Entonces, se mostrarán en la pantalla todos los números que existen en la lista y el programa terminará, así como el mayor y el menor de todos ellos, y se calculará la media.

```
#include <stdlib.h>
#include <stdio.h>
#include "lista.h"


int main(void)
{
   struct s_nodo *primero = NULL, *nodo, *nodo2;
   int i,aux;

   // Generación de la lista
   for (i=1; i<=100; i++)
      insertar(&primero, rand()%1000 + 1);

   printf("LISTA DESORDENADA:\n");
   mostrar_lista(primero);


   // Ordenación (método de la burbuja)
   for (i=1; i <= 100; i++)
   {
      nodo = primero;
      nodo2 = nodo->siguiente;
      while (nodo2 != NULL)
      {
       if (nodo->dato > nodo2->dato)    // Estan desordenados
       {                                // Vamos a intercambiarlos
             aux = nodo->dato;
           nodo->dato = nodo2->dato;
           nodo2->dato = aux;
       }
       nodo2 = nodo2->siguiente;
       nodo = nodo->siguiente;
      }
   }

   // Mostrar la lista en la pantalla y destruirla
   printf("\n\nLISTA ORDENADA:\n");
```

```
   mostrar_lista(primero);
   borrar_todos(&primero);

   return 0;
}
```

# Ejercicio 5-08

Escribe un sencillo editor de texto. El usuario irá tecleando caracteres, y éstos se irán guardando en una lista de caracteres. Cada carácter introducido por teclado generará un nodo nuevo en la lista. Cuando se introduzca el carácter "#", el programa terminará, guardando todos los caracteres en un archivo de texto llamado "5_08.txt"

(Esta solución utiliza una lista de caracteres, no de cadenas. Recordad que se puede mejorar la recepción de teclas con la función getch() de la librería ncurses, evitando así tener que pulsar Return después de cada carácter)

```
int main(void)
{
   struct s_nodo *lista = NULL;
   FILE *f;
   char c;
   int i;

   // Leemos los caracteres y los insertamos en la lista
   printf("Introduzca un texto (pulse # para terminar)");
   do {
      c = getchar();
      if (c != '#') insertar(&lista, c);
   }
   while (c != '#');

   // Guardamos la lista en un archivo de texto
   f = fopen("5-08.txt", "rw");
   if (f == NULL) { printf("No puedo crear el archivo"); exit(-1); }

   for (i=1; i <= contar_nodos(lista); i++) {
      buscar_por_posicion(lista, i, &c);
      fputc(c, f);
   }
   fclose(f);

   // Destruimos la lista para liberar la memoria
   borrar_todos(&lista);

   return 0;
}
```

# Ejercicio extra de listas enlazadas

Escribe un programa que cargue en una lista enlazada las películas del archivo video.dat que usamos en el Tema 4. Después de cargarlas, las mostrará por la pantalla y pedirá el código de una película para borrarla, y luego volverá a mostrar la lista para comprobar que la película se borró correctamente.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct s_pelicula
{
  char titulo[50];
  char director[20];
  char reparto[200];
  char genero[20];
```

```
  char nacionalidad[10];
  int duracion;
  char borrado;
  int codigo;
};

struct s_nodo
{
  struct s_pelicula peli;
  struct s_nodo* siguiente;
}



/*
   PROTOTIPOS DE FUNCIONES
   Las funciones de lista que se usan en este programa son las mismas que
   las de la lista abierta de números enteros, pero cambiándolas un poco
   para que los nodos contengan películas en lugar de números enteros.
*/

void insertar_peli(struct s_nodo **primero, struct s_pelicula dato);
int borrar_por_contenido(struct s_nodo **primero, int codigo);
void borrar_todos(struct s_nodo **primero);
void mostrar_lista(struct s_nodo *primero);

/*
    PROGRAMA PRINCIPAL
    Observa que este programa se podría ampliar para hacer una gestión
    completa del archivo de datos: buscar, modificar, borrar, etc.
*/

int main(void)
{
  struct s_pelicula pelicula;
  struct s_nodo *primero;
  FILE *f;
  int busc, tam, n;

  printf("Fund.Prog. / Tema 5 / Ejercicio extra de listas abiertas\n");

  // Abrimos el fichero y miramos su longitud
  f = fopen("video.dat", "a+b");
  if (f == NULL) { printf("No se puede abrir el fichero video.dat\n"); exit(1); }
  fseek(f, 0, SEEK_END);  // Saltamos al final del fichero
  tam = ftell(f) / sizeof(struct s_pelicula); // Calculamos el número de registros
  fseek(f, 0, SEEK_SET);  // Nos volvemos a situal al comienzo del fichero

  // Leemos el fichero completo y vamos insertando las películas en la lista
  while (!feof(f))
  {
      n = fread(&pelicula, sizeof(struct s_pelicula), 1, f);
      if (n > 0)
         insertar_peli(&primero, pelicula);
  }

  printf("Archivo video.dat cargado con éxito (%i registros)\n\n", tam);
  fclose(f);

  // Mostramos la lista de películas
  printf("Listado de películas\n----------------------\n");
  mostrar_lista(primero);
```

```
  // Borramos una película
  printf("\n\nIntroduzca el código de la película que quiere borrar:\n");
  scanf("%i", &busc);
  n = borrar_por_codigo(&primero, busc);
  if (n == 0)
      printf("La película no existe\n");
  else
      printf("Película borrada\n");

  // Mostramos de nuevo la lista de películas
  printf("Listado de películas\n----------------------\n");
  mostrar_lista(primero);

  // Liberamos la memoria antes de terminar
  borrar_todo(&primero);

  return 0;
}


/*
    FUNCIONES DE LISTA
    No se incluyen todas las funciones, sólo las que se usan
    en este programa.
*/


// Inserta un nodo al principio de la lista
void insertar_peli(struct s_nodo **primero, struct s_pelicula dato)
{
   struct s_nodo *nuevo;

   nuevo = (struct s_nodo*) malloc (sizeof(struct s_nodo));
   nuevo->peli = dato;
   nuevo->siguiente = *primero;
   *primero = nuevo;
}


// Borra la película cuyo código se pasa como parámetro (si existe)
// Devuelve 0 si la película no existe, o 1 si existe y se ha borrado
int borrar_por_contenido(struct s_nodo **primero, int codigo)
{
   struct s_nodo *nodo, *anterior;
   int encontrado = 0;

   nodo = *primero;
   anterior = *primero;
   while ((nodo != NULL) && (encontrado == 0))
   {
      if (nodo->peli.codigo == codigo)
      {
         if (nodo == *primero)
           *primero = (*primero)->siguiente;
        else
           anterior->siguiente = nodo->siguiente;
        free(nodo);
        encontrado = 1;
      }
      else
      {
```

```
         anterior = nodo;
         nodo = nodo->siguiente;
      }
   }

   return encontrado;

}

// Borra todas las películas y libera la memoria
void borrar_todos(struct s_nodo **primero)
{
   struct s_nodo *nodo, *sig;

   nodo = *primero;
   while (nodo != NULL)
   {
      sig = nodo->siguiente;
      free(nodo);
      nodo = sig;
   }

   *primero = NULL;
}


// Muestra un listado de las películas que hay en la lista
void mostrar_lista(struct s_nodo *primero)
{
  struct s_nodo *nodo;

  nodo = primero;
  printf("Codigo   Titulo\n");
  printf("----------------------------------------\n");
  while (nodo != NULL)
  {
     printf("%i     %s\n", nodo->peli.codigo, nodo->peli.titulo);
     nodo = nodo->siguiente;
  }
}
```

# Ejercicio 5-10

Implementa una pila de números enteros. Utiliza como referencia los apuntes y escribe las funciones necesarias para las operaciones básicas (push y pop). También tienes en los apuntes las declaraciones de tipos de datos y variables que necesitarás. Una vez hecho eso, escribe el programa principal, desde el cual se llame varias veces a push para apilar los siguientes datos: 4, 5, 8, 10, 13. Por último, desapila todos los datos llamando repetidamente a pop (hasta que la pila se quede vacía), mostrando los datos desapilados por la pantalla. Trata de averiguar en qué orden aparecerán los datos antes de ejecutar el programa.

```
#include <stdio.h>

struct s_nodo
{
   int dato;
   struct s_nodo *siguiente;
};
typedef struct s_nodo t_nodo;


void push(t_nodo** primero, int v) ;
int pop(t_nodo** primero) ;
```

```
int main(void)
{
   t_nodo* primero;
   int v;

   primero = NULL;
   push(&primero,4);
   push(&primero,5);
   push(&primero,8);
   push(&primero,10);
   push(&primero,13);

   v = pop(&primero);
   printf("Valor = %i\n", v);
   v = pop(&primero);
   printf("Valor = %i\n", v);
   v = pop(&primero);
   printf("Valor = %i\n", v);
   v = pop(&primero);
   printf("Valor = %i\n", v);
   v = pop(&primero);
   printf("Valor = %i\n", v);

   return 0;
}

void push(t_nodo **primero, int v)
{
   t_nodo *nuevo;

   nuevo = (t_nodo*)malloc(sizeof(t_nodo));  // Creamos nodo nuevo
   nuevo->dato = v;                          // Insertamos el dato en el nodo

   nuevo->siguiente = *primero;              // La cima a partir de ahora será "nuevo"
   *primero = nuevo;
}

int pop(t_nodo **primero)
{
   t_nodo *aux;  // Variable auxiliar para manipular el nodo
   int v;        // Variable auxiliar para devolver el valor del dato

   aux = *primero;
   if(aux == NULL)    // Si no hay elementos en la pila devolvemos algún valor especial
      return -1;

   *primero = aux->siguiente;   // La pila empezará ahora a partir del siguiente elemento
   v = aux->dato;               // Este es el dato que ocupaba la cima hasta ahora
   free(aux);                    // Liberamos la memoria ocupada por el nodo que estaba en la
cima
   return v;                   // Devolvemos el dato
}
```

# Ejercicio 5-11

Haz lo mismo que en el ejercicio anterior pero con una cola de números enteros. En esta ocasión, deja que sea el usuario el que vaya tecleando números, que se irán insertando en la cola, hasta que introduzca un número negativo. En ese momento, el proceso de inserción en la cola terminará y se empezarán a extraer los datos de la cola, mostrándolos por la pantalla.

```
#include <stdio.h>

struct s_nodo
```

```
{
   int dato;
   struct s_nodo *siguiente;
};
typedef struct s_nodo t_nodo;

void insertar(t_nodo** primero, int v) ;
int sacar(t_nodo** primero) ;

int main(void)
{
   t_nodo *primero, *ultimo;
   int v;

   primero = NULL;
   insertar(&primero, &ultimo, 4);
   insertar(&primero, &ultimo, 5);
   insertar(&primero, &ultimo, 8);
   insertar(&primero, &ultimo, 10);
   insertar(&primero, &ultimo, 13);

   v = sacar(&primero, &ultimo);
   printf("Valor = %i\n", v);
   v = sacar(&primero, &ultimo);
   printf("Valor = %i\n", v);
   v = sacar(&primero, &ultimo);
   printf("Valor = %i\n", v);
   v = sacar(&primero, &ultimo);
   printf("Valor = %i\n", v);
   v = sacar(&primero, &ultimo);
   printf("Valor = %i\n", v);

   return 0;
}

void insertar(t_nodo **primero, t_nodo **ultimo, int v)
{
   t_nodo* nuevo;

   nuevo = (t_nodo*)malloc(sizeof(t_nodo));   // Creamos el nuevo nodo
   nuevo->dato = v; // Le asignamos el dato
   nuevo->siguiente = NULL;      // El nuevo nodo apuntará a NULL
   if (*ultimo != NULL)   // Si la cola no estaba vacía...
       (*ultimo)->siguiente = nuevo;   // ...enganchamos el nuevo al final
   *ultimo = nuevo; // A partir de ahora, el nuevo será el último
   if (*primero == NULL)  // Si la cola estaba vacía...
      *primero = nuevo;     // ...el último también será el primero
}

int sacar(t_nodo **primero, t_nodo **ultimo)
{
   t_nodo *aux;     // Puntero auxiliar
   int v;    // Para almacenar el valor del dato y devolverlo

   aux = *primero;  // El auxiliar apunta a la cabeza
   if(aux == NULL)  // La cola está vacía: devolver valor especial
      return -1;
   *primero = aux->siguiente;    // El primero apunta al segundo
   v = aux->dato;   // Recoger valor del primero
   free(aux);// Eliminar el nodo primero
   if (*primero==NULL)    // Si la cola se ha quedado vacía...
      *ultimo = NULL;     // ...hacer que el último también apunte a NULL
```

```
   return v; // Devolver el dato que había en el primero
}
```

# Ejercicio 5-12

> NOTA IMPORTANTE: A partir de ahora, supondremos que las funciones pop() y push() ya están implementadas en los problemas que usen pilas, así como insertar() y sacar() en los problemas que usen colas. Esa implementación puedes encontrarla en los dos ejercicios anteriores.

El ejercicio 5-12 consiste en escribir un programa que genere aleatoriamente una pila de 10 números enteros y luego la copie en otra pila vacía.

```
int main(void)
{
   t_nodo *pila = NULL, *copia = NULL, *aux = NULL;
   int v, i;

   printf("Generando la pila aleatoriamente...\n");
   for (i=1; i<=10; i++) {
      v = rand() % 100;          // Elige números al azar entre 0 y 99
      push(&pila, v);
   }

   printf("Copiando en pila auxiliar\n");
   for (i=1; i<=10; i++) {
      v = pop(&pila);
      push(&aux, v);             // Esta pila quedará AL REVÉS que la original
   }

   printf("Creando pila definitiva\n");
   for (i=1; i<=10; i++) {
      v = pop(&aux);
      push(&copia, v);           // Esta pila será una copia exacta de la original
   }

   return 0;
}
```

# Ejercicio 5-13

Escribe un programa que elimine el elemento N-ésimo de una pila de números enteros. La pila se generará automáticamente, insertándole datos aleatorios. El número de elementos de la pila también será aleatorio. El usuario introducirá por teclado qué elemento de la pila desea borrar. Puedes utilizar una segunda pila (auxiliar) para realizar el proceso

```
int main(void)
{
   t_nodo *pila = NULL, *copia = NULL, *aux = NULL;
   int v, i, elem, tamano;

   printf("Generando la pila aleatoriamente...\n");
   tamano = rand() % 90 + 11;    // El tamaño de la pila estará entre 10 y 100 elementos
   for (i=1; i<=tamano; i++) {
      v = rand() % 100;          // Elige números al azar entre 0 y 99
      push(&pila, v);
   }

   // Preguntamos al usuario qué elemento quiere eliminar
   do {
       printf("¿Qué elemento desea eliminar de la pila (1 - %i)\n", tamano);
      scanf("%i", &elem);
   }
   while ((elem < 1) || (elem > tamano));
```

```
   // Copiamos toda la pila en la auxiliar, salvo el elemento que vamos a eliminar (si existe)
   while (pila != NULL) {
      v = pop(&pila);
      if (v != elem)
         push(&aux, v);            // Esta pila quedará AL REVÉS que la original
   }

   // Creamos la pila definitiva a partir de la pila auxiliar
   while (aux != NULL) {
      v = pop(&aux);
      push(&copia, v);             // Esta pila será una copia exacta de la original
   }

    // Al final, la pila "copia" tendrá todos los elementos de la pila original menos el
queremos
   // eliminar. Ahora se podría procesar la pila "copia" para hacer otras cosas con ella.
   return 0;
}
```

# Ejercicio 5-14

Escribe un programa que calcule el mayor, el menor y la media de los elementos guardados en una pila de 100 números enteros. La pila se generará aleatoriamente con números comprendidos entre –500 y 500

```
int main(void)
{
   t_nodo *pila = NULL;
   int v, i, mayor, menor, suma;
   float media;

   printf("Generando la pila aleatoriamente...\n");
   for (i=1; i<=100; i++) {
      v = (rand() % 1001) - 500;       // Elige números al azar entre -500 y 500
      push(&pila, v);
   }

   // Buscamos el mayor, el menor y la media
   mayor = -9999;
   menor = 9999;
   suma = 0;
   while (pila != NULL) {
      v = pop(&pila);
      suma = suma + v;
      if (v > mayor)
         mayor = v;                    // Es el elemento más grande encontrado hasta ahora
      if (v < menor)
         menor = v;                    // Es el elemento más grande encontrado hasta ahora
   }
   media = (float)suma / 100;

   // Mostramos los resultados
   printf("Mayor: %i, Menor: %i, Media: %2.2f\n", mayor, menor, media)

   return 0;
}
```

# Ejercicio 5-15

Escribe una función que reciba como parámetro un puntero a la cabeza de una cola y calcule el número de elementos que contiene

```
// Esta función cuenta el número de elementos de una cola, pero al hacerlo destruiremos
// su contenido. Se podrían contar los elementos sin destruirlos, recorriendo los nodos
// de la lista (ver implementación de las listas abiertas), pero entonces ya no estaríamos
// tratando con una cola en el sentido estricto de la palabra, sino con una lista abierta.

int contar_elementos(t_nodo **primero)
{
   int cont = 0;

   while (*primero != NULL) {
      cont++;
      sacar(&primero);
   }

   return cont;
}
```

# Ejercicio 5-16

Escribe un programa que cree dos colas de 4 números enteros cada una (los números se introducirán por teclado). Después, las comparará y dirá si son iguales

```
int main(void)
{
   t_nodo *primero_A = NULL, *ultimo_A = NULL;
   t_nodo *primero_B = NULL, *ultimo_B = NULL;
   int i, v, dato_A, dato_B, iguales;

   printf("Introduzca los datos de la cola A\n");
   for (i=1; i<=4; i++) {
      scanf("%i", &v);
      insertar(&primero_A, &ultimo_A, v);
   }

   printf("Introduzca los datos de la cola B\n");
   for (i=1; i<=4; i++) {
      scanf("%i", &v);
      insertar(&primero_A, &ultimo_A, v);
   }

   // Comparamos las dos colas
   iguales = 1;
   for (i=1; i<=4; i++) {
      dato_A = sacar(&primero_A, &ultimo_A);  // Sacamos un dato de la cola A
      dato_B = sacar(&primero_B, &ultimo_B);  // Sacamos un dato de la cola B
      if (dato_A != dato_B)
         iguales = 0;
   }

   if (iguales == 1)
      printf("Las colas eran iguales");
   else
      printf("Las colas eran diferentes");

   return 0;
}
```

# Ejercicio 5-17

Escribe un programa que genere una cola de cadenas de caracteres. Cada cadena tendrá un máximo de 20 caracteres. El usuario irá tecleando cadenas, que se almacenarán en la cola, hasta que introduzca la cadena "FIN". Entonces, el programa mostrará todas las cadenas de la cola por la pantalla dejando un espacio en blanco entre cada una y la siguiente.

```
struct s_nodo {                  // Tenemos que cambiar la definición del nodo
   char dato[21];
   struct s_nodo *siguiente;
}

int main(void)
{
   struct s_nodo *primero = NULL, *ultimo = NULL;
   int terminar;
   char cad[21];

   printf("Introduzca varias cadenas (teclee FIN para terminar)\n");
   terminar = 0;
   do {
      gets(cad);
      if (strcmp(cad, “FIN”) == 0)
        terminar = 1;
      else
         insertar(&primero, &ultimo, cad);
   }
   while (terminar == 0);

   // Sacamos las cadenas de la cola y las mostramos por la pantalla separadas por un espacio
   while (primero != NULL) {
      strcpy(cad, sacar(&primero, &ultimo));
      printf(“%s “, cad);
   }

   return 0;
}
```

# Ejercicio 5-18

Repite el programa anterior pero utilizando pilas en lugar de colas. Observa el orden en el que se muestran las palabras en la pantalla y compáralo con el de las colas.

```
struct s_nodo {                  // Tenemos que cambiar la definición del nodo
   char dato[21];
   struct s_nodo *siguiente;
}

int main(void)
{
   struct s_nodo *primero = NULL;
   int terminar;
   char cad[21];

   printf("Introduzca varias cadenas (teclee FIN para terminar)\n");
   terminar = 0;
   do {
      gets(cad);
      if (strcmp(cad, “FIN”) == 0)
        terminar = 1;
      else
         push(&primero, cad);
   }
   while (terminar == 0);

   // Sacamos las cadenas de la cola y las mostramos por la pantalla separadas por un espacio
   while (primero != NULL) {
      strcpy(cad, pop(&primero));
      printf(“%s “, cad);
   }
```

```
   return 0;
}
```

# Ejercicio 5-19

Recuerda que un palíndromo es una frase que se lee igual de izquierda a derecha que de derecha a izquierda. Escribe un programa que lea un texto introducido por teclado y guarde los caracteres en una cola, y luego determine si el texto forma un palíndromo o no.

```
struct s_nodo {                  // Tenemos que cambiar la definición del nodo
   char dato;
   struct s_nodo *siguiente;
}

int main(void)
{
   struct s_nodo *primero = NULL, *ultimo = NULL;
   struct s_nodo *pila = NULL;
   int i, es_palindromo;
   char cad[100];
   char c1, c2;

   printf("Introduzca una frase\n");
   gets(cad);

   // Metemos todos los caracteres en una cola y en una pila
   for (i=0; i<strlen(cad); i++) {
       insertar(&primero, &ultimo, cad[i]);
        push(&pila, cad[i]);
   }

   // Sacamos los caracteres de la cola y de la pila. En una saldrán ordenados del primero
   // al último, y en la otra, al revés. Si la frase era un palíndromo todos los caracteres
   // deben coincidir al sacarlos
   es_palindromo = 1;
   while (primero != NULL) {
       c1 = sacar(&primero, &ultimo);
       c2 = pop(&pila);
       if (c1 != c2)
         es_palindromo = 0;
   }

   if (es_palindromo == 1)
       printf("La cadena %s es un palíndromo\n", cad);
   else
       printf("La cadena no era un palíndromo\n");

   return 0;
}
```

# Ejercicio 5-20

Escribe un programa que lea por teclado una expresión matemática con paréntesis, del tipo de: ((8 + 3) * (2-5)) + (4*3). Utiliza una pila para guardar los caracteres. El programa debe determinar si los paréntesis son correctos, es decir, si se cierran todos los paréntesis que se abren.

```
// La estructura s_nodo debe adaptarse para que el dato sea de tipo "char"

int main(void)
{
   struct s_nodo *pila = NULL;
   int i, correcto;
   char cad[100], c;
```

```
   printf("Introduzca una expresión matemática con paréntesis\n");
   gets(cad);

   // Recorremos la cadena. Cada paréntesis abierto que encontremos se insertará en la pila.
   // Cada paréntesis cerrado hará que se elimine un paréntesis abierto de la pila.
   correcto = 1;
   for (i=0; i<strlen(cad); i++) {
       if (cad[i] == '(')
           push(&pila, cad[i]);
       if (cad[i] == ')')
       {
           c = pop(&pila);
          if (c == -1)            // Es lo que devuelve pop() si la pila está vacía
              correcto = 0;             // Se ha cerrado un paréntesis que no se había abierto
       }
   }

   // Si la pila no está vacía ahora, es que se han abierto más paréntesis de los que se
   // han cerrado
   if (pila != NULL)
       correcto = 0;

   // Mostramos el resultado
   if (correcto == 1)
       printf("Los paréntesis de la expresión son correctos\n"));
   else
       printf("Hay un error en los paréntesis de la expresión");

   return 0;
}
```

**Nota: este programa se podría mejorar para decir qué tipo de error se ha producido (si se han cerrado más paréntesis de los que se han abierto o al revés) e incluso para marcar en qué lugar de la expresión está el error**

# Ejercicio 5-21

*NOTA: Este ejercicio se propone únicamente como curiosidad para quien quiera profundizar en la utilización de estructuras dinámicas y recursividad (v. tema 6)*

Escribe un programa que resuelva el problema de las Torres de Hanoi, mostrando en la pantalla los movimientos que hay que hacer para alcanzar la solución (por ejemplo: mover anillo 1 a barra 2, mover anillo 2 a barra 3, etc). Pista: puedes utilizar 3 pilas estructuras de tipo pila, donde irán apilándose los anillos. El programa debe funcionar para cualquier número de anillos. El número de anillos se introducirá por teclado.

Numeraremos los discos según su tamaño: el disco 1 será el de menor tamaño, el disco 2 el siguiente, el 3 el siguiente, etc. Así, cada barra puede ser representada con una pila de números enteros, en la que el número de cada elemento representa el tamaño del disco.

Trata primero de resolver manualmente el problema con sólo 3 discos. Te darás cuenta de que, para mover 3 discos de la barra 1 a la barra 3, primero hay que mover 2 discos de la barra 1 a la 2, y luego moverlos de la barra 2 a la 3.
A su vez, para mover 2 discos de la barra 1 a la 2, hay que mover antes 1 disco de la barra 1 a la 3, y de ahí a la 2.

En general, para mover n discos de la barra i a la barra j, hay que mover antes n-1 discos de la barra i a la barra que queda libre (que se puede calcular como 6-i-j), y, de ahí, moverlos hasta la barra j.
Tenemos, pues, un **procedimiento recursivo**:
-Problema: mover m discos dela barra i a la barra j
-Caso base: si (m=0) no hay que hacer nada
-Caso general: si (m>0) hay que mover m-1 discos de la barra i a la barra 6-i-j, y luego mover los m-1 discos de la barra 6-i-j a la barra j

```
int main(void)
{
    struct s_nodo *pila1 = NULL, *pila2 = NULL, *pila3 = NULL;
   int v, n, i;
```

```
    printf("¿Cuántos discos desea utilizar? (un número muy alto requerirá mucho tiempo)\n");
    scanf("%i", &n);

    // Insertamos n discos en la pila 1 en orden de mayor a menor
    for (i=0; i<=n-1; i++)
       push(&pila1, n-i);

    // Iniciamos el proceso recursivo que colocará los discos en la tercera pila
    hanoi(&pila1, &pila2, &pila3, n, 1, 3);   // Mover n discos de la pila 1 a la pila 3

    // Mostramos el resultado para asegurarnos
    printf("Contenido de la pila 3 después de la ejecución del algoritmo:\n");
    while (pila3 != NULL) {
         v = pop(&pila3);
        printf("%i\n", v);
    }

    return 0;
}


void hanoi(struct s_nodo **pila1, struct s_nodo **pila2, struct s_nodo **pila3,
          int num_discos, int pila_orig, int pila_dest)
{
   int v;

   // Caso base: num_discos == 0 (no hacer nada)

   if (num_discos > 0)           // Caso general
   {
      // Primera llamada recursiva
      hanoi(pila1, pila2, pila3, num_discos - 1, pila_orig, 6 - pila_orig - pila_dest);
      printf("%i --> %i\n", pila_orig, pila_dest);  // Mostramos el movimiento
      switch(pila_orig) {                     // Sacamos un disco de la pila de origen
         case 1: v = pop(pila1); break;
         case 2: v = pop(pila2); break;
         case 3: v = pop(pila3); break;
      }
      switch(pila_dest) {                     // Introducimos el disco en la pila de destino
         case 1: push(pila1, v); break;
         case 2: push(pila2, v); break;
         case 3: push(pila3, v); break;
      }
      // Segunda llamada recursiva
      hanoi(pila1, pila2, pila3, num_discos - 1, 6 - pila_orig - pila_dest, pila_dest);
   }

}
```

La complejidad de esta solución es $O(2^n)$. Es decir, el número de cálculos necesarios para hallar una solución es aproximadamente $2^n$, siendo n el tamaño del problema (el número de discos) Por ejemplo, con 3 discos se necesitan 7 movimientos (casi $2^3$), con 4 se necesitan 15 (casi $2^4$), con 5 se necesitan 31 ($2^5$), con 6, 63 ($2^6$), etc.

Esto quiere decir que, si ejecutamos el programa con un número de discos muy grande, el tiempo de ejecución pronto será excesivo. Por ejemplo, con 64 discos se necesitan $2^{64}$ movimientos, es decir, alrededor de 18 trillones.