Listas abiertas (o listas enlazadas simples)

Programa de ejemplo

Este programa incorpora la mayoría de las operaciones que se pueden realizar sobre una lista abierta. Cada operación está implementada con una función, de manera que son reutilizables en otros programas. La función *main()* sólo contiene unas cuantas *llamadas de prueba* a las funciones de la lista.

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

#define TIPO int      // Cambiando esto podemos modificar el tipo de los datos de la lista

// Definición de la estructura básica de la lista
struct s_nodo
{
   TIPO dato;
   struct s_nodo *siguiente;
};

// Prototipos de funciones
void inicializar(struct s_nodo **primero);
int esta_vacia(struct s_nodo *primero);
int contar_nodos(struct s_nodo *primero);
void insertar(struct s_nodo **primero, TIPO dato);
int buscar_por_contenido(struct s_nodo *primero, TIPO dato);
int buscar_por_posicion(struct s_nodo *primero, int posic, TIPO *result);
void borrar_primero(struct s_nodo **primero);
void borrar_todos(struct s_nodo **primero);
int borrar_por_posicion(struct s_nodo **primero, int posic);
int borrar_por_contenido(struct s_nodo **primero, TIPO dato);
void mostrar_lista(struct s_nodo *primero);

// Programa de prueba. Este programa sólo sirve para comprobar que las funciones son correctas.
// Sustituyendo este programa por cuaquier otro, las funciones deben ser igualmente correctas.
int main(void)
{
   struct s_nodo *lista;
   int n, r, dato;

   printf("\nPROGRAMA DE PRUEBA DE LAS FUNCIONES DE LISTA ENLAZADA\n\n");
   inicializar(&lista);

   printf("1. PRUEBA DE LA INSERCION DE DATOS\n");
   printf("Introduzca números enteros (negativo para terminar)\n");
   do
   {
     scanf("%i", &n);
     if (n>0) insertar(&lista, n);
   }
   while (n>=0);

   printf("2. PRUEBA DE FUNCIONES GENERALES\n");
   printf("El contenido de la lista es:\n");
   mostrar_lista(lista);
   if (esta_vacia(lista))
     printf("La lista está vacía\n");
   else
     printf("La lista tiene %i elementos\n", contar_nodos(lista));

   printf("3. PRUEBA DE LAS FUNCIONES DE BÚSQUEDA\n");
   printf("Introduzca un dato para buscarlo: ");
   scanf("%i", &dato);
   r = buscar_por_contenido(lista, dato);
   if (r == 0)
         printf("No se pudo encontrar el dato %i\n", dato);
   else
         printf("El dato %i está en la posición %i\n", dato, r);

   printf("Introduzca una posición: ");
   scanf("%i", &n);
   r = buscar_por_posicion(lista, n, &dato);
   if (r == 0)
         printf("La posición %i no existe en la lista\n", n);
   else
         printf("En la posición %i se encuenta en número %i\n", n, dato);

   printf("4. PRUEBA DE LAS FUNCIONES DE BORRADO\n");
```

```
   printf("Introduzca un dato para borrarlo: ");
   scanf("%i", &dato);
   r = borrar_por_contenido(&lista, dato);
   if (r == 0)
         printf("No se pudo encontrar el número %i\n", dato);
   else {
         printf("Número borrado. La lista queda así:\n");
         mostrar_lista(lista);
   }

   printf("Introduzca una posición para borrarla: ");
   scanf("%i", &n);
   r = borrar_por_posicion(&lista, n);
   if (r == 0)
         printf("La posición %i no existe en la lista\n", n);
   else {
         printf("Elemento borrado. La lista queda así:\n", n, dato);
         mostrar_lista(lista);
   }

   printf("5. PRUEBA DE BORRADO DE TODA LA LISTA\n");
   printf("Borrando toda la lista...\n");
   borrar_todos(&lista);
   printf("Borrado terminado. La lista queda así:\n");
   printf("Número de elementos: %i\n", contar_nodos(lista));
   mostrar_lista(lista);

   return 0;
}

// ---------------- FUNCIONES DE LA LISTA ABIERTA (Código reutilizable) -------------------------

// ---------------------------------------------------------------------------------------------
// Inicializa la lista, es decir, crea una lista vacía
// ---------------------------------------------------------------------------------------------
void inicializar(struct s_nodo **primero)
{
   *primero = NULL;
}

// ---------------------------------------------------------------------------------------------
// Devuelve 1 si la lista está vacía, 0 en caso contrario
// ---------------------------------------------------------------------------------------------
int esta_vacia(struct s_nodo *primero)
{
   if (primero == NULL) return 1;
   else return 0;
}

// ---------------------------------------------------------------------------------------------
// Devuelve el número de elementos de la lista
// ---------------------------------------------------------------------------------------------
int contar_nodos(struct s_nodo *primero)
{
   int cont = 0;             // Contador del número de nodos
   struct s_nodo *nodo;

   nodo = primero;
   while (nodo != NULL)
   {
      cont++;
      nodo = nodo->siguiente;
   }

   return cont;
}

// ---------------------------------------------------------------------------------------------
// Añade un elemento al principio de la lista
// ---------------------------------------------------------------------------------------------
void insertar(struct s_nodo **primero, TIPO dato)
{
   struct s_nodo *nuevo;

   nuevo = (TIPO*) malloc (sizeof(TIPO));
   nuevo->dato = dato;
   nuevo->siguiente = *primero;
   *primero = nuevo;
}
```

```
// ------------------------------------------------------------------------------------------
// Se le pasa un dato y lo busca en la lista. Devuelve la posición que ocupa ese dato,
// o -1 si el dato no existe
// ------------------------------------------------------------------------------------------
int buscar_por_contenido(struct s_nodo *primero, TIPO dato)
{
   struct s_nodo *nodo;
   int pos = 0, encontrado = 0;

   nodo = primero;
   while ((nodo != NULL) && (encontrado == 0))
   {
      pos++;                        // Contador del número de nodos recorridos
      if (nodo->dato == dato)
         encontrado = 1;
      nodo = nodo->siguiente;
   }

   if (encontrado == 0)
      pos = -1;                     // Si no se encontró el dato, devolvemos -1

   return pos;
}

// ------------------------------------------------------------------------------------------
// Se le pasa una posición y devuelve el dato que hay en el elemento que ocupa esa posición
// en la lista. El dato se devuelve en el parámetro "result", que por ese motivo se pasa
// por variable. La función devuelve en el "return" un 1 si ha encontrado el dato o 0 en
// caso contrario.
// ------------------------------------------------------------------------------------------
int buscar_por_posicion(struct s_nodo *primero, int posic, TIPO *result)
{
   struct s_nodo *nodo;
   int i = 0, encontrado = 0;

   nodo = primero;
   while ((nodo != NULL) && (encontrado == 0))
   {
      i++;                          // Contador del número de nodos recorridos
      if (i == posic) {
         encontrado = 1;
        *result = nodo->dato;       // Devolvemos en "result" el dato que hay en este nodo
      }
      nodo = nodo->siguiente;
   }

   return encontrado;
}

// ------------------------------------------------------------------------------------------
// Elimina el primer elemento de la lista (es decir, el último que se insertó)
// ------------------------------------------------------------------------------------------
void borrar_primero(struct s_nodo **primero)
{
   struct s_nodo *segundo;

   segundo = (*primero)->siguiente;
   free(primero);
   *primero = segundo;
}

// ------------------------------------------------------------------------------------------
// Elimina todos los elementos de la lista, dejándola como si se acabase de crear
// ------------------------------------------------------------------------------------------
void borrar_todos(struct s_nodo **primero)
{
   struct s_nodo *nodo, *sig;

   nodo = *primero;
   while (nodo != NULL)
   {
      sig = nodo->siguiente; // Antes de borrar "nodo" guardamos el puntero al siguiente
      free(nodo);
      nodo = sig;            // Menos mal que guardamos el puntero. Si no, no podríamos continuar
   }

   *primero = NULL;
}
```

```
// ------------------------------------------------------------------------------------------
// Elimina el elemento que ocupa la posición "posic". Si se elimina con éxito, la función devuelve
// 1. Si no se puede eliminar (porque la posición no exista), devuelve 0.
// ------------------------------------------------------------------------------------------
int borrar_por_posicion(struct s_nodo **primero, int posic)
{
   struct s_nodo *nodo, *anterior;
   int cont = 0, encontrado = 0;

   nodo = *primero;
   anterior = *primero;
   while ((nodo != NULL) && (encontrado == 0))
   {
      cont++;                        // Contador del número de nodos recorridos
      if (cont == posic)
      {
         if (posic == 1)             // Caso particular: se pretende borrar el primer nodo
           *primero = (*primero)->siguiente;
        else                         // Caso general: se va a borrar un nodo posterior al primero
           anterior->siguiente = nodo->siguiente;
        free(nodo);
        encontrado = 1;
      }
      else
      {
         anterior = nodo;
         nodo = nodo->siguiente;
      }
    }

   return encontrado;
}

// ------------------------------------------------------------------------------------------
// Elimina un dato de la lista. Si el dato aparece varias veces, sólo elimina la primera aparición
// Devuelve 1 si el dato se ha encontrado y eliminado. Devuelve 0 en otro caso.
// ------------------------------------------------------------------------------------------
int borrar_por_contenido(struct s_nodo **primero, TIPO dato)
{
   struct s_nodo *nodo, *anterior;
   int encontrado = 0;

   nodo = *primero;
   anterior = *primero;
   while ((nodo != NULL) && (encontrado == 0))
   {
      if (nodo->dato == dato)
      {
         if (nodo == *primero)       // Caso particular: se va a borrar el primer nodo
           *primero = (*primero)->siguiente;
        else                         // Caso general: se va a borrar un nodo posterior al primero
           anterior->siguiente = nodo->siguiente;
        free(nodo);
        encontrado = 1;
      }
      else
      {
         anterior = nodo;
         nodo = nodo->siguiente;
      }
   }

   return encontrado;
}


// ------------------------------------------------------------------------------------------
// Muestra el contenido de la lista por la pantalla (un elemento en cada línea). Es la única
// función no reutilizable directamente, ya que hay que cambiar el "printf" si se modifica
// el tipo de los datos de la lista
// ------------------------------------------------------------------------------------------
void mostrar_lista(struct s_nodo *primero) {
  struct s_nodo *nodo;
  nodo = primero;
  while (nodo != NULL)
  {
     printf("%i\n", nodo->dato);     // ESTO HAY QUE CAMBIARLO dependiendo del tipo de los datos
     nodo = nodo->siguiente;
  }
}
```