Ejercicio 1.-Adivinar número entre 1 y 1000

Crea un programa que adivine un número entero aleatorio entre 1 y 1000 que será calculado al principio de manera aleatoria utilizando las funciones srand () y rand().

El usuario irá introduciendo números enteros (printf / scanf) hasta que lo adivine. El programa comunicará al usuario si el número introducido es mayor o menor que el número aleatorio.

La introducción de números finalizará cuando el usuario adivine el número. En este momento el programa deberá imprimir en el monitor los números que el usuario ha introducido y el número aleatorio

El programa deberá reservar de manera dinámica memoria para ir almacenando los números que el usuario ha ido introduciendo.

El programa principal debe encargarse de:

- Calcular el número aleatorio
- Reservar memoria para el primer número
- Bucle mientras que el usuario quiera seguir introduciendo números
    - Pedir al usuario número
    - Almacenar el número
    - Comprobar si el número coincide con el aleatorio
- Presentar lista de números y número aleatorio
- Liberar memoria

```
Prototipos:

      void* malloc (size_t n)

      void *realloc(void *ptr, size_t size);

      void free (void *);

Invocaciones:

      puntero = (tipo *) malloc (tamaño en bytes);

      puntero = (tipo *) realloc (puntero, tamaño en bytes);

      free (puntero);
```

Un ejemplo de ejecución puede ser:

```
$ ./01_adivinar_numero.exe
Adivina el numero entre 1 y 1000:
Introduce un numero: 500
El número es mayor.
Introduce un numero: 750
El número es mayor.
Introduce un numero: 825
El número es menor.
Introduce un numero: 800
El número es menor.
Introduce un numero: 790
El número es menor.
Introduce un numero: 780
El número es menor.
Introduce un numero: 777
El número es menor.
Introduce un numero: 760
El número es mayor.
Introduce un numero: 760
El número es mayor.
Introduce un numero: 765
El número es mayor.
Introduce un numero: 770
El número es menor.
Introduce un numero: 767
El número es mayor.
Introduce un numero: 768
El número es mayor.
Introduce un numero: 769

¡Felicidades! Has adivinado el número 769.
Números introducidos:
500 750 825 800 790 780 777 760 760 765 770 767 768 769
```

Ejercicio 2.- Frecuencia de caracteres

Se pide un programa que calcule la frecuencia de repetición de los caracteres recibidos como parámetros de entrada para hacer una clasificación.

El programa deberá:

- Ignorar aquellos valores que sean una cadena de más de un carácter.
- Almacenar los caracteres en una lista de estructuras que contengan el carácter y el número de veces que está repetido (frecuencia).
- Mostrar por pantalla los caracteres repetidos y cuantas veces aparecen.
- Liberar la memoria reservada.

El programa debe incluir las siguientes funciones:

```
// Función para buscar un carácter en la lista de tamanio tam

int buscarCaracter(CharInfo *lista, int tam, char c);

// Función que se le pasa como parámetro una cadena y devuelve
//el número de caracteres que tiene

int tamCadena (char* cadena);
```

Un ejemplo de ejecución puede ser:

```
$ ./clasificar.exe a b a c d a b
Caracteres repetidos y sus frecuencias:

'a' aparece 3 veces
'b' aparece 2 veces
'c' aparece 1 vez
'd' aparece 1 vez
```

Ejercicio 3.- Autopista

Se debe desarrollar un programa que simule el funcionamiento de una autopista.

En una autopista circulan diferentes tipos de vehículos, cada uno con características propias. El programa debe gestionar la creación, movimiento, accidentes y salida de los vehículos, cobrando un peaje a los que completen el recorrido.

Cada vehículo se almacenará en el programa para llevar un seguimiento y tendrá las siguientes características:

- Matrícula
- Tipo de vehículo: moto, coche o camión
- Nombre del conductor
- Carril en el que circula
- Posición en la autopista
- Velocidad (km/iteración)

Cada vehículo se moverá según su velocidad en cada iteración.

Si dos vehículos terminan en la misma posición y carril, chocan. Deben ser eliminados del sistema.

Si un vehículo alcanza el final de la autopista, paga un peaje y es eliminado.

Tarifas del peaje:

- Moto 5€
- Coche 10€
- Camión 20€

La longitud de la autopista debe ser de 300km.

El montante total recaudado debe mostrarse al finalizar el programa junto con la cantidad de vehículos que finalizan el recorrido y el número de accidentes.

Debe crearse un menú que gestione las siguientes funcionalidades:

- Crear vehículo
- Actualizar posición de los vehículos (iteración)
- Finalizar programa

Un ejemplo de ejecución puede ser:

```
$ ./03_autopista.exe

--- Menú de la Autopista ---
1. Crear vehículo
2. Actualizar posiciones
3. Finalizar programa
Seleccione una opción: 1
Ingrese matrícula: 1234ASD
Tipo de vehículo (0=Moto, 1=Coche, 2=Camión): 1
Nombre del conductor: Juan
Carril (1-3): 1
Velocidad (km/iteración): 150


--- Menú de la Autopista ---
1. Crear vehículo
2. Actualizar posiciones
3. Finalizar programa
Seleccione una opción: 1
Ingrese matrícula: 4321DSA
Tipo de vehículo (0=Moto, 1=Coche, 2=Camión): 2
Nombre del conductor: Ana
Carril (1-3): 2
Velocidad (km/iteración): 50


--- Menú de la Autopista ---
1. Crear vehículo
2. Actualizar posiciones
3. Finalizar programa
Seleccione una opción: 2


--- Menú de la Autopista ---
1. Crear vehículo
2. Actualizar posiciones
3. Finalizar programa
Seleccione una opción: 1
Ingrese matrícula: 5678QWE
Tipo de vehículo (0=Moto, 1=Coche, 2=Camión): 0
Nombre del conductor: Alfredo
Carril (1-3): 2
Velocidad (km/iteración): 100


--- Menú de la Autopista ---
1. Crear vehículo
2. Actualizar posiciones
```

```
3. Finalizar programa
Seleccione una opción: 2
Accidente entre 4321DSA y 5678QWE en la posición 100


--- Menú de la Autopista ---
1. Crear vehículo
2. Actualizar posiciones
3. Finalizar programa
Seleccione una opción: 3


--- Estadísticas Finales ---
Vehículos que completaron el recorrido: 1
Total recaudado en peajes: 10€
Número de accidentes: 1
```

Ejercicio 4.- Lista de la compra

En una tienda, los clientes llegan y se forman en una cola para ser atendidos. Una cola es una estructura de datos en la que los elementos se procesan en el orden en que llegan, siguiendo el principio FIFO (First In, First Out). Crea un programa en C que permita gestionar esta cola de clientes utilizando memoria dinámica. El programa debe permitir agregar clientes a la cola, atender al primer cliente de la cola y mostrar el estado actual de la cola. Utiliza enumerados para las opciones del menú y asegúrate de modularizar el código.

Ejemplo del menú:

1. Agregar cliente
2. Atender cliente
3. Mostrar cola
4. Salir

Explicación de la Cola:

Una cola es una estructura de datos lineal en la que los elementos se insertan por un extremo (el final de la cola) y se eliminan por el otro extremo (el frente de la cola). Esto asegura que el primer elemento en entrar sea el primero en salir. Las operaciones básicas de una cola son:

- **Enqueue**: Agregar un elemento al final de la cola.
- **Dequeue**: Eliminar el primer elemento de la cola.
- **IsEmpty**: Verificar si la cola está vacía.
- **IsFull**: Verificar si la cola está llena (en caso de una implementación con capacidad fija).
- **Display**: Mostrar todos los elementos de la cola.

Definiciones de las Funciones:

A continuación se presentan los prototipos de las funciones necesarias para gestionar la cola, junto con sus explicaciones:

*void initQueue(Queue *q, int capacity);*

- **Descripción**: Inicializa la cola con una capacidad inicial.
- **Parámetros**:
    - Queue *q: Puntero a la estructura de la cola.
    - int capacity: Capacidad inicial de la cola.

*int isFull(Queue *q);*

- **Descripción**: Verifica si la cola está llena.
- **Parámetros**:
    - Queue *q: Puntero a la estructura de la cola.
- **Retorno**: 1 si la cola está llena, 0 en caso contrario.

*int isEmpty(Queue *q);*

- **Descripción**: Verifica si la cola está vacía.
- **Parámetros**:

    - Queue *q: Puntero a la estructura de la cola.
- **Retorno**: 1 si la cola está vacía, 0 en caso contrario.

*void resizeQueue(Queue *q);*

- **Descripción**: Redimensiona la cola cuando está llena.
- **Parámetros**:
    - Queue *q: Puntero a la estructura de la cola.

*void enqueue(Queue *q, char *name);*

- **Descripción**: Agrega un cliente al final de la cola. Verifica si la cola está llena para redimensionar.
- **Parámetros**:
    - Queue *q: Puntero a la estructura de la cola.
    - char *name: Nombre del cliente.

*void dequeue(Queue *q);*

- **Descripción**: Atiende al primer cliente de la cola. Si está vacía hay que indicar que está vacía. Ej: "La cola esta vacia."
- **Parámetros**:
    - Queue *q: Puntero a la estructura de la cola.

*void displayQueue(Queue *q);*

- **Descripción**: Muestra todos los clientes en la cola.
- **Parámetros**:
    - Queue *q: Puntero a la estructura de la cola.

*void freeQueue(Queue *q);*

- **Descripción**: Libera la memoria asignada a la cola.
- **Parámetros**:
    - Queue *q: Puntero a la estructura de la cola.

*void menu(Queue *q);*

- **Descripción**: Maneja la interacción con el usuario utilizando enumerados.
- **Parámetros**:
    - Queue *q: Puntero a la estructura de la cola.

Funciones adicionales que se pueden emplear:

Librería <string.h>:

1. **strcpy**:
    - **Prototipo**: char *strcpy(char *dest, const char *src);
    - **Descripción**: Copia una cadena de caracteres a otra.
    - **Parámetros**: char *dest - Destino de la copia, const char *src - Fuente de la copia.
2. **strcmp**:
    - **Prototipo**: int strcmp(const char *str1, const char *str2);

- **Descripción**: Compara dos cadenas de caracteres.
- **Parámetros**: const char *str1 - Primera cadena, const char *str2 - Segunda cadena.

Ejemplo:

$ ./ColaDinamica.exe

```
1. Agregar cliente
2. Atender cliente
3. Mostrar cola
4. Salir
Elige una opcion: 1
Nombre del cliente: Pepe


1. Agregar cliente
2. Atender cliente
3. Mostrar cola
4. Salir
Elige una opcion: 1
Nombre del cliente: Marcos


1. Agregar cliente
2. Atender cliente
3. Mostrar cola
4. Salir
Elige una opcion: 1
Nombre del cliente: Pablo


1. Agregar cliente
2. Atender cliente
3. Mostrar cola
4. Salir
Elige una opcion: 2
Atendiendo a Pepe


1. Agregar cliente
2. Atender cliente
3. Mostrar cola
4. Salir
Elige una opcion: 3
Clientes en la cola:
Marcos
Pablo
```

```
1. Agregar cliente
2. Atender cliente
3. Mostrar cola
4. Salir
Elige una opcion: 2
Atendiendo a Marcos


1. Agregar cliente
2. Atender cliente
3. Mostrar cola
4. Salir
Elige una opcion: 2
Atendiendo a Pablo


1. Agregar cliente
2. Atender cliente
3. Mostrar cola
4. Salir
Elige una opcion: 2
La cola esta vacia.


1. Agregar cliente
2. Atender cliente
3. Mostrar cola
4. Salir
Elige una opcion: 3
La cola esta vacia.


1. Agregar cliente
2. Atender cliente
3. Mostrar cola
4. Salir
Elige una opcion: 4
Saliendo...
```