Algoritmos y Estructuras de Datos

Importante: Los ejercicios deben entregarse a través de web (**Domjudge y Blackboard**). Cada ejercicio deberá ir en un fichero con nombre:

Nodo.h

Pila.h

Pila.cpp

Hanoi.cpp

Los dos primeros ficheros podrán ser descargados a través de **Blackboard** y no deberán modificados por el alumno. Pila.cpp deberá ser desarrollado completamente y Hanoi.cpp deberá ser modificado.

La fecha de entrega: consultar la página de la actividad en blackboard

(2,5 puntos) Hanoi: Escribe un programa que permita encontrar la solución del juego de las torres de Hanoi para cualquier número de discos. Para resolver este problema se deberá implementar la función recursiva presentada en los apuntes de la asignatura en el fichero Hanoi.ccp. Además para simular el apilamiento de los discos en cada poste se utilizará la estructura de datos Pila que también deberá ser implementada en el fichero Pila.cpp. Esta estructura de datos se compone de los siguientes atributos:

- **Nodo *cima.** Puntero apuntando a la cima de la pila
- **std::string name.** Nombre de la pila que se utilizará para almacenar el nombre del poste

Y los siguientes métodos:

- **Pila(std::string name).** Constructor con parámetros de la estrucutra Pila. Inicializará el puntero a la cima y asignará el nombre indicado a la pila.
- **std::string nombrePila().** Devuelve el nombre de la pila.
- **void apilar(int num).** Recibe un número que representará el tamaño del disco y lo colocará en la cima de la pila. Deberá imprimir por pantalla el movimiento realizado “Apilando disco N en poste P”
- **int desapilar().** Devuelve el número que se encuentra en la cima de la pila que representará al tamaña del disco que se encuentra en la parte superior. Deberá indicar por pantalla el movimiento realizado “Desapilando disco N del poste P”
- **bool estaVacia().** Indica si la pila se encuentra vacía.

El **input** del programa tendrá el siguiente formato. Se recibirá un número entero positivo que indique el número de discos que se encuentran en el poste origen (N).

El **output** representará los movimientos necesarios para mover todos los discos hasta el poste destino.

A continuación se puede observar un ejemplo de la entrada salida del programa.

Input:	Output:
3	Apilando disco 3 en poste A Apilando disco 2 en poste A Apilando disco 1 en poste A Desapilando disco 1 del poste A Apilando disco 1 en poste C Desapilando disco 2 del poste A Apilando disco 2 en poste B Desapilando disco 1 del poste C Apilando disco 1 en poste B Desapilando disco 3 del poste A Apilando disco 3 en poste C Desapilando disco 1 del poste B Apilando disco 1 en poste A Desapilando disco 2 del poste B Apilando disco 2 en poste C Desapilando disco 1 del poste A Apilando disco 1 en poste C Desapilando disco 1 del poste C Desapilando disco 2 del poste C Desapilando disco 3 del poste C

Importante: Los ejercicios deben entregarse a través de web (**Domjudge y Blackboard**). Cada ejercicio deberá ir en un fichero con nombre:

Nodo.h

Cola.h

Supermercado.h

mainSupermercado.cpp

Cola.cpp

Supermercado.cpp

Los cuatro primeros ficheros podrán ser descargados a través de **Blackboard** y no deberán modificados por el alumno. Sin embargo, Cola.cpp y Supermercado.cpp, deberán ser desarrollados completamente por el alumno.

La fecha de entrega: consultar la página de la actividad en blackboard

(3 puntos)Supermercado: Se quiere gestionar el funcionamiento de las cajas de un supermercado para poder saber que clientes se encuentran en cada una de las cajas y en caso de cerrar una de ellas recolocar a todos los clientes en las restantes. Para este propósito se deberán implementar dos clases: la clase Cola que permite almacenar los clientes que se encuentran esperando en una caja y que se van atendiendo según el orden de llegada y la clase Supermercado que permite gestionar varias cajas al mismo tiempo. La clase Cola se compone de los siguientes atributos:

- **Nodo *principio.** Puntero que apunta al primer nodo introducido en la Cola.
- **Nodo *final.** Puntero que apunta al último nodo de la Cola. Este puntero no sería necesario pero nos evita recorrernos toda la cola para encontrar el último elemento

Y será necesario implementar los siguientes métodos:

- **Cola().** Constructor sin parámetros de la estructura Cola. Deberá inicializará los punteros principio y final.
- **void encolar(int num).** Recibe un número que representará el orden de llegada del cliente al supermercado y lo colocará al final de la cola.
- **int desencolar().** Devuelve el número que representa el orden de llegada del cliente que se encuentra en la primera posición de la cola (El primero en llegar).

- **bool estaVacia().** Indica si la cola se encuentra vacía.

Por otro lado la clase Supermercado tiene los siguientes atributos:

- **Cola *cajas.** Array de punteros a objetos de tipo Cola que representan cada una de las cajas..

- **int n_cajas.** Número de cajas que hay en el supermercado

Y será necesario implementar los siguientes métodos::

- **Supermercado(int n)**. Constructor que se encarga de reservar memoria para las n cajas que tiene el supermercado e incializa el atributo n_cajas.

- **void nuevoUsuario(int n,int id).** Encola el usuario con el id indicado en la caja que se encuentra en la posición n del array de cajas.

- **void cerrarCaja(int n).** Esta función simula el cierre de la caja n y el reparto de los usuarios en las cajas restantes. Para ello será necesario desencolar todos los usuarios que se encuentran en la caja n en el orden de llegada, e ir encolándolos en las cajas restantes que no estén vacías. Debido a que los usuarios se deberán repartir de forma equitativa en las cajas restantes, iremos recorriendo las cajas por orden de 0 a N-1 e introduciremos un único usuario por cada caja que no esté vacía. Cuando hayamos introducido un usuario en cada una de las cajas volveremos a empezar desde la caja 0 hasta que no queden más usuarios.

- **int atenderUsuario(int n)**; Atiende al usuario que se encuentra en la caja n y por tanto lo desencola de la cola que representa dicha caja. Esta función devolverá el id del usuario atendido.

- **bool cajaVacia(int n).** Indica si la caja n tiene o no tiene usuarios esperando.

Utilizar el fichero mainSupermercado.cpp proporcionado a través de Blackboard para realizar las pruebas necesarias y para enviar al corrector automático. Este programa simula el funcionamiento de un supermercado con las siguientes opciones:

- N: Crea un objeto de tipo supermercado que permitirá controlar varias cajas.
- U: Encola un usuario en una de las cajas.
- C: Simula el cierre de una de las cajas repartiendo los usuarios entre las cajas restantes
- A: Atiende al primer usuario que se encuentre en la caja indicada
- S: Termina.

Importante: Los ejercicios deben entregarse a través de web (**Domjudge y Blackboard**). Cada ejercicio deberá ir en un fichero con nombre:

Nodo.h

ListaCircular.h

RuletaRusa.cpp

ListaCircular.cpp

Los tres primeros ficheros podrán ser descargados a través de **Blackboard** y no deberán modificados por el alumno.

La fecha de entrega: consultar la página de la actividad en blackboard

(2,5 puntos)RuletaRusa: Escribe un programa que simule el juego de la ruleta rusa pero sin heridos ya que utilizaremos balas de fogueo. Para jugar a la ruleta rusa lo primero que hay que hacer es poner una única bala en el tambor de una pistola que tiene N huecos en la recámara. Entonces en el turno de cada jugador se gira el tambor de forma aleatoria y a continuación disparamos la pistola. Si suena “BANG” habremos perdido y por tanto acabará la partida, en otro caso sonará “CLACK” para indicar que no había ninguna bala y que se deberá seguir jugando. Para simular este juego utilizaremos una lista enlazada circular que permite almacenar en cada elemento un string (para almacenar “BANG” o “CLACK”) y se deberán implementar los siguientes métodos:

-Constructor sin parámetros

-void insertar(int pos, string s). Introduce el string s en la posición pos de la lista.

-void eliminar(int pos). Elimina el elemento que se encuentra en la posición pos en la lista.

-string getElemento(int pos). Devuelve el string que se encuentra en la posición pos de la lista.

-void girar(int g). Gira todos los elementos de la lista hacia la izquierda o la derecha como indique su valor. Si el valor es positivo todos los elementos de la lista se desplazarán tantas posiciones a la derecha como indique el valor. Si el valor es negativo el desplazamiento será hacia la izquierda.

El **input** del programa tendrá el siguiente formato. En primer lugar, se recibirá un número entero positivo que indique el número de balas (N) que pueden entrar en el tambor. En segundo lugar, se indicará la posición (0 a N-1) en la que se colocará la bala

dentro del tambor. A continuación se podrá encontrar una lista de enteros que indicará los giros que deben hacerse antes de cada jugada.

El **output** representará el resultado del disparo de cada jugador pudiendo ser únicamente “BANG” or “CLACK”. Si el resultado fue “BANG” el programa deberá terminar.

A continuación se puede observar un ejemplo de la entrada salida del programa.

Input:	Output:
6 0 1 1 0 -1 -1	CLACK CLACK CLACK CLACK BANG

NOTA: Para cada una de las funciones implementados se deberá incluir una pequeña descripción de su funcionamiento, sus precondiciones mediante excepciones si las hubiera y el análisis de su complejidad temporal y espacial. Esta información deberá incluirse en la cabecera de cada función.