Práctica 7

Importante: Los ficheros deben entregarse a través de web (por **Domjudge** y **Blackboard**).

Para el ejercicio de lista contigua se deberán entregar 3 **ficheros** con nombres:

ListaContigua.h, ListaContigua.cpp, assertdomjudge.h y mainContigua.cpp

Para el ejercicio de lista enlazada se deberán entregar 4 **ficheros** con nombres:

Nodo.h, ListaEnlazada.h, ListaEnlazada.cpp, assertdomjudge.h y mainEnlazada.cpp

Estos ficheros serán proporcionados a través de Blackboard y algunos deberán ser modificados.

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

ListaContigua (3,6 puntos): Desarrollar la clase "ListaContigua" que representa una lista de números enteros y que además permite realizar operaciones sobre ellos.

La clase **ListaContigua** tiene los siguientes atributos:

- **int n**: atributo que almacena de forma privada el número de elementos almacenados actualmente en la lista contigua.
- **int capacidad**: atributo que indica el máximo número de elementos que se pueden almacenar en la lista.
- **int incremento**: atributo que indica el número de posiciones que se incrementa/decrementa la capacidad de la ListaContigua cuando es necesario.
- **int *vector.** Puntero a un array de enteros que permitirá almacenar **capacidad** elementos de tipo int. Este array deberá ser reservada de forma dinámica en sus constructores y ser liberada en su destructor.

Para esta clase deberán implementarse los siguientes métodos públicos:

- **(0,1 Puntos) Constructor por parámetros. ListaContigua(int incremento)** Inicializará los atributos n, capacidad e incremento así como el puntero al vector de enteros.
- **(0,1 Puntos) Destructor.** Se encargará de liberar la memoria que fue reservada de forma dinámica para almacenar el vector.
- **(0,1 Puntos) int getValor(int pos).** Devuelve el elemento de la lista contigua que se encuentra en la posición **pos.**
- **(0,1 Puntos) int setValor(int pos, int val).** Modifica el elemento de la lista contigua que se encuentra en la posición **pos** por el valor **val**. **OJO**: Este elemento tenía que haberse insertado anteriormente
- **(0,1 Puntos) int getN().** Devuelve el tamaño actual de la lista contigua.
- **(0,1 Puntos) int getCapacidad().** Devuelve la capacidad de la lista contigua (Máximo número de elementos que puede albergar).

- **(0,75 Puntos) void insertar(int pos, int val).** Inserta un nuevo elemento en la posición **pos** de la lista con valor **val**, dejando primero un hueco para introducirlo (desplazando los elementos a la derecha con **memmove**). En el caso de que al insertar se alcance la máxima capacidad del vector deberá incrementarse esta en la cantidad **incremento** usando **realloc**.
- **(0,75 Puntos) void eliminar(int pos).** Elimina el elemento que se encuentra en la posición **pos** y por tanto deberá desplazar a la izquierda todos los elementos que se encuentren a su derecha mediante **memmove**. Si al eliminar este elemento el número de elementos del vector es menor o igual que (**capacidad-2xincremento**), se deberá reducir la capacidad en incremento elementos (**capacidad-incremento**).
- **(0,75 Puntos) void concatenar(ListaContigua *lista).** Concatena la lista indicada como parámetro al final de nuestra lista (almacenada en vector).
- **(0,75 Puntos) int buscar(int num).** Busca un elemento en la lista contigua con valor igual a **num** y retorna su posición o -1 si no se ha podido encontrar.

ListaEnlazada(4,4 puntos): Desarrollar la clase "ListaEnlazada" que representa una lista de números enteros y que además permite realizar operaciones sobre ellos.

La clase **ListaContigua** tiene los siguientes atributos:

- **int n**: atributo que almacena de forma privada el número de elementos almacenados actualmente en la lista contigua.
- **Nodo *lista.** Puntero a un objeto de tipo struct Nodo que está definido en el fichero Nodo.h. Esta estructura tiene dos campos: elemento de tipo entero donde se guardará los números y un puntero a Nodo para apuntar al siguiente elemento. Cada uno de estos elementos deberá ser reservado de forma dinámica en sus constructores y ser liberada en su destructor.

Para esta clase deberán implementarse los siguientes métodos públicos:

- **(0,1 puntos) Constructor por parámetros. ListaEnlazada().** Crea una lista de tamaño 0 y para ello inicializará los atributos n así como el puntero al primer nodo **Lista**.
- **(0,25 puntos) Destructor.** Se encargará de liberar la memoria que fue reservada de forma dinámica para almacenar el vector.
- **(0,75 puntos) Nodo * getNodo (int pos).** Método privado que permite obtener el nodo de la lista que se encuentra en la posición **pos.**
- **(0,1 puntos) int getValor(int pos).** Devuelve el elemento de la lista contigua que se encuentra en la posición **pos (**Utiliza internamente getNodo).
- **(0,1 puntos) int setValor(int pos, int val).** Modifica el elemento de la lista contigua que se encuentra en la posición **pos** por el valor **val (**Utiliza internamente getNodo) **OJO**: Este elemento tenía que haberse insertado anteriormente
- **(0,1 puntos) int getN().** Devuelve el tamaño actual de la lista contigua.

- **(0,75 puntos) void insertar(int pos, int val).** Inserta un nuevo elemento en la posición **pos** de la lista con valor **val**, cambiando los punteros correspondientes. **(**Utiliza internamente getNodo)
- **(0,75 puntos) void eliminar(int pos).** Elimina el elemento que se encuentra en la posición **pos** cambiando los punteros correspondientes **(**Utiliza internamente getNodo).
- **(0,75 puntos) void concatenar(ListaContigua *lista).** Concatena la lista indicada como parámetro al final de nuestra lista. **(No** utiliza internamente getNodo)
- **(0,75 puntos) int buscar(int num).** Busca un elemento en la lista contigua con valor igual a **num** y retorna su posición o -1 si no se ha podido encontrar. **(No** utiliza internamente getNodo)

Utilizar los ficheros mainContigua.cpp y mainEnlazada.cpp proporcionados a través de Blackboard para realizar las pruebas necesarias y para enviar al corrector automático. Este programa permite realizar las siguientes operaciones:

- N: permite crear un nueva lista indicando el incremento.
- I: permite insertar un valor en la lista en una posición.
- E: permite eliminar un elemento de la lista.
- V: permite ver el valor de un elemento de la lista.
- T: permite ver todos los valores de la lista.
- S: permite modificar un valor de la lista.
- L: informa sobre la longitud actual de la lista.
- M: informa sobre la capacidad máxima de la lista.
- C: permite concatenar nuestra lista con otra con n elementos.
- B: permite buscar un valor en la lista.
- F: termina.

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