PRÁCTICA LISTAS

1) Diseñar la clase *ListaEnteros* con los siguientes métodos:
   - Constructor / Destructor
   - Alta al principio
   - Alta al final
   - Alta en posición determinada
   - Borrar primero
   - Borrar último
   - Borrar en determinada posición
   - Borrar primer elemento (se indica qué elemento quiere ser borrado, y se borra su primera aparición)
   - Borrar elemento (se indica qué elemento quiere ser borrado y se borran todas sus apariciones)
   - Obtener primero
   - Obtener último
   - Obtener el elemento de una posición determinada
   - Obtener el tamaño de la lista
   - Listar (muestra todos los elementos de la lista)
   - Listar en forma inversa (muestra desde el último hasta el final)

   Consideraciones:
   - Tener en cuenta que se pueden repetir los elementos.
   - Es importante que cada método tenga explícitas las PRE y POST condiciones.

2) Implementar la clase *ListaEnteros* diseñada en el punto anterior.

   Consideraciones:
   - Alta al principio como alta al final pueden llamar al método Alta en posición determinada pasando un 1 o un *tamaño* como parámetro. Lo mismo sucede con el borrado y el obtener.
   - Hacer el desarrollo de los métodos en forma iterativa.
   - Hacer la declaración de la clase en un archivo punto h y la implementación de los métodos en un archivo punto cpp.
   - La lista es simplemente enlazada.
   - Se pueden llegar a necesitar otros métodos que los indicados, por ejemplo, obtenerSiguiente. Estos métodos serán privados si el usuario no necesita emplearlos.

3) Hacer un proyecto incluyendo la implementación de *ListaEnteros* desarrollada en el punto anterior, con un *main* que la utilice. Deberá tener un menú, donde un usuario pueda ingresar números, borrarlos, obtener alguno en particular, imprimir la lista al derecho y al revés.

4) Probar lo realizado en el punto anterior.

5) Implementar la misma lista del punto 2, con los mismos métodos pero en forma recursiva.

6) Utilizar el mismo proyecto del punto 3 con la nueva implementación. Probar el funcionamiento.

7) Diseñar e implementar una nueva *ListaEnteros* donde el alta será únicamente en orden, por lo que no se indicará la posición. Tener en cuenta que puede haber elementos repetidos.

8) Hacer un proyecto y probar la lista del punto anterior.

9) Diseñar e implementar la *ListaEnteros* similar a la del punto 7 pero sin aceptar elementos repetidos.

10) Utilizando la lista del punto anterior, agregarle los métodos:
    - Recibe una lista por parámetro e indica si ésta está incluida o no (devuelve true o false).
    - Igual que el anterior pero si la lista pasada por parámetro incluye a la lista que llama.
    - Recibe una lista por parámetro y devuelve una tercera lista que es la intersección de ambas.
    - Igual que el anterior pero devuelve la unión.
    - Igual que el anterior pero devuelve la resta (elementos que no están en la lista del parámetro).
    - Retorne la propia lista pero invertida.
    - Recibe una lista e indica si es igual a la propia (mismos elementos).

11) Hacer un proyecto con un menú en donde se puedan utilizar los métodos escritos en el punto anterior.

12) Modificar los métodos que reciben una lista por parámetro, escritos en el punto 10, pasándole un puntero a una lista. Lo mismo los que devuelvan listas, devolverán un puntero a una lista.

13) Probar el funcionamiento de los métodos escritos en el punto anterior.

14) Analizar (escribir) ventajas y desventajas de trabajar con punteros a listas en lugar de las propias listas.

15) Modificar la lista diseñada e implementada en el punto 1 por *ListaChar* (lista de caracteres).

16) Probar la lista escrita en el punto anterior.

17) A la lista escrita en el punto 15 agregarle un método que indique si la lista es palíndroma (capicúa).

18) Probar el uso de la lista del punto anterior.

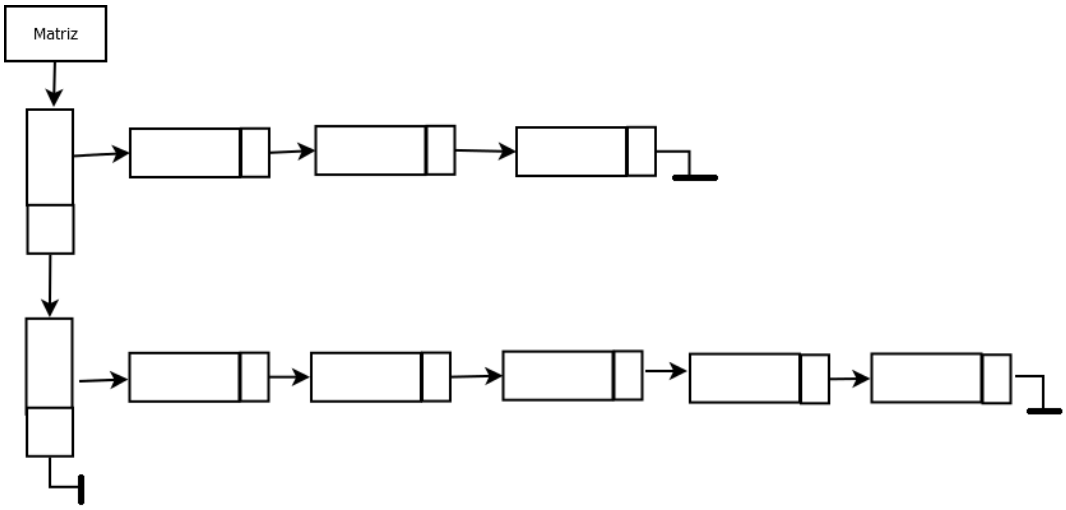
19) A la lista de char agregarle los siguientes constructores (si no estaban):
- Constructor sin parámetro (lista vacía)
- Constructor con un string (arma la lista de char con los caracteres del string)
- Constructor con un const char* (similar al anterior, tener en cuenta que los char* finalizan con un cero)
- Constructor de copia

20) Agregarle los siguientes métodos:
- longitud (devuelve la longitud de la lista)
- Concatenar (se le pasa una lista y la concatena en la propia lista)

21) Hacer un proyecto para realizar pruebas con la lista desarrollada.

22) El método concatenar desarrollado en el punto 20 reemplazarlo por la sobrecarga del operador +: el método será *operador+*

23) Diseñar e implementar la clase *ListaMatrizEnteros.* Será una lista de listas de enteros. Gráficamente:



24) Probar el uso de la matriz implementada en el punto anterior (carga y muestra, solamente).

25) ¿Qué ventajas y desventajas tiene la implementación anterior comparando con el uso de matrices visto?

26) Agregarle métodos para sumar y restar matrices. Testearlos.

27) Diseñar e implementar la clase *PilaEnteros*.

28) Hacer un proyecto para probar el uso de la clase desarrollada en el punto anterior.

29) Hacer una función factorial iterativa, simulando la recursividad con la pila.

30) Diseñar e implementar la clase *ColaEnteros.*

31) Hacer un proyecto para probar la clase desarrollada en el punto anterior.

32) Utilizando la función rand() generar números aleatorios para simular el funcionamiento de la cola. Por ejemplo, ejecutar n ciclos, por cada ciclo "tirar" un número aleatorio entre 0 y 99. Si es mayor a 49 ingresa un nuevo número aleatorio en la cola, si es menor sale un elemento de la cola.
Probarlo cambiando el 49 por otros números (30, 20, 60 y 70).
La función rand() devuelve un entero entre 0 y RAND_MAX, para lograr que sea entre 0 y 99 se hace módulo 100. Se debe incluir la librería cstdlib.
Para que en cada corrida no repita números se debe cambiar la semilla con que comienza las cuentas, esto se hace colocando la instrucción
srand (número);
al principio del programa. número puede tomarse del reloj de la siguiente manera:
srand (time (NULL));
En este caso se debe incluir ctime.
Ver http://www.cplusplus.com/reference/cstdlib/rand/

33) Diseñar e implementar una Lista como la desarrollada en los puntos 1 y 2 pero doblemente enlazada (puntero al siguiente y puntero al anterior).

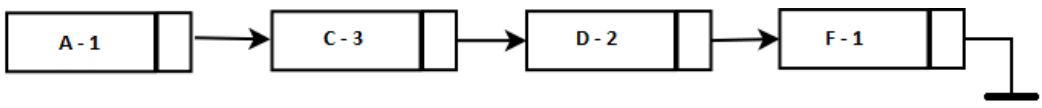
34) Probar la lista desarrollada en el punto anterior.

35) Diseñar e implementar una lista de caracteres, simplemente enlazada y circular (el último nodo apunta al primero).

36) Probar la lista desarrollada en el punto anterior.

37) Diseñar e implementar una lista de caracteres, los cuales pueden estar repetidos pero el ingreso es siempre en orden.

38) Modificar la lista del punto anterior para que, en cada nodo, guarde el elemento (carácter) y su frecuencia (cantidad de apariciones). Por ejemplo, si a la lista se ingresan los siguientes caracteres: C, A, D, C, F, C, D. Deberá quedar de la siguiente forma:



39) Diseñar e implementar la lista de enteros como en el punto 1, utilizando cursor.

40) Diseñar e implementar la lista desarrollada en el punto anterior para elementos genéricos, utilizando templates.

Ejercicios tomados en parciales

41) Considerar como implementada la clase Alimento a partir de la siguiente interfaz:

```
class Alimento
{
public:
        // Crea un alimento con su nombre, la cantidad de calorías y una lista
        // de los ingredientes que lo conforman
        Alimento (string nombre, unsigned int calorias, Lista<string>*
        Ingredientes);
        string getNombre();                 // devuelve el nombre del alimento
        unsigned int getCalorias ();        // devuelve la cantidad de calorías
        Lista<string>* getIngredientes (); // devuelve la lista de ingredientes
};
```

Implementar el método comidasParaCeliacos de la clase BuscadorDeComidas:

```
class BuscadorDeComidas
{
public:
        // Post: busca en “comidas” aquellas que tienen algún ingrediente de la
        // lista “permitidos” y ninguno de la lista “noPermitidos” y tienen una
        // caloría menor a “caloriaMaxima”.
        // Devuelve una lista con los alimentos que cumplen con estas
        // características.
        Lista<Alimento *>* comidasParaCeliacos (Lista<Alimento *>* comidas,
        Lista<string>* permitidos,
        Lista<string>* noPermitidos, unsigned int caloriaMaxima);
};
```

42) Una empresa argentina tiene sucursales en distintas provincias.
Cada sucursal tiene un número y pertenece a una provincia. Puede haber más de una sucursal por provincia. Cada sucursal desarrolla distintos proyectos. Cada proyecto tiene un identificador y un determinado presupuesto.
Se pide:
a) Indicar cuales son los TDA que intervienen en el planteo de esta situación, describiendo brevemente el objetivo de cada TDA.
b) El diseño básico de la implementación de las clases correspondientes a los TDA. Se debe indicar obligatoriamente pre y post condiciones de cada método, argumentos recibidos y dato retornado, así como una descripción breve de lo que hace el método.
c) Un método que reciba como argumento una provincia y un número real x correspondiente a un monto de dinero y retorne una lista de todos los proyectos pertenecientes a sucursales de esa provincia que no superen el monto x. Los

proyectos de la lista que se retorna deben estar ordenados de manera creciente por monto.

43) Una empresa Discográfica tiene contratados a distintos Artistas. De cada Artista se conoce su nombre y el grupo de Actividades que ha llevado adelante. La Actividad de un Artista se identifica por un nombre (vocalista, guitarrista, compositor, etc.) y la cantidad de años que ejerció esa actividad.
Se pide:
a) Indicar cuáles son los TDA que intervienen en el planteo de esta situación, describiendo brevemente el objetivo de cada TDA.
b) Especificar la interfaz de las clases correspondientes a los TDA. Se debe indicar obligatoriamente pre y post condiciones de cada método, argumentos recibidos y tipo de retorno.
c) Implementar la clase Discográfica a partir de la siguiente especificación de su interfaz:

```
class Discografica {
public:
// post: la Discográfica queda creada sin ningún Artista contratado.
Discografica();
// post: destruye todos los Artistas contratados.
~Discografica();
// post: devuelve el grupo de Artistas contratados por la Discográfica.
Lista<Artista*>* getArtistasContratados();
// post: devuelve una nueva Lista con los Artistas contratados que hayan realizado
// alguna Actividad de las indicadas en nombresActividades, por al menos
// experienciaMinima años.
Lista<Artista*>* buscarArtistasConExperienciaEn(int experienciaMinima,
Lista<String>* nombresActividades);
};
```

44) Un Blog está compuesto por un conjunto de Artículos. Cada Artículo posee un título, un texto y una lista de palabras clave. Todo Artículo puede tener asociado un conjunto de Comentarios. Cada Comentario tiene una descripción y una calificación. La calificación es un valor comprendido entre 1 y 10.
Se pide:
a) Indicar cuales son los TDA que intervienen en el planteo de esta situación, describiendo brevemente el objetivo de cada TDA.
b) Especificar la interfaz de las clases correspondientes a los TDA. Se debe indicar obligatoriamente pre y post condiciones de cada método, argumentos recibidos y tipo de retorno.
c) Implementar la clase Blog a partir de la siguiente especificación de su interfaz:

```
class Blog {
public:
// post: el Blog queda creado sin Artículos.
Blog();
// post: destruye todos los Artículos que componen el Blog.
~Blog();
```

```
    // post: devuelve los Artículos que componen el Blog.
    Lista<Articulo*>* getArticulos();
    // post: devuelve una nueva Lista con los Artículos que tengan entre sus palabras
    // clave palabraClave  y que su calificación promedio (tomada de sus comentarios)
    // esté comprendida entre calificadaDesde y calificadaHasta.
    Lista<Articulo*>* buscarArticulos(string palabraClave, int calificadaDesde, int
            calificadaHasta);
};
```

45) Implementar el método seleccionarVideo de la clase Editor a partir de las siguientes especificaciones:

```
classEditor {
public:
/* post: selecciona de videos aquel que tenga por lo menos tantos Comentarios
como los indicados y  el promedio de calificaciones sea máximo. Ignora los
Comentarios sin calificación.
*/
Video*seleccionarVideo(Lista<Video*>* vides, intcantidadDeComentarios);

};

classVideo {
public:
// post: inicializa el Video con el título y URL.
Video(string titulo, string url);
// post: devuelve el título del Video.
string getTitulo();
// post: devuelve la URL del Video.
string getUrl();
// post: devuelve los comentarios asociadas al Video.
Lista<Comentario*>* getComentarios();
~Video();
};

classComentario {
public:
// post: inicializa el Comentario con el contenido y calificación 0.
Comentario(string contenido);
string getContenido();
// post: devuelve la calificación [1 a 10] asociada, o 0 si el Comentario no tiene
// calificación.
intgetCalificacion();
// pre : calificacion está comprendido entre 1 y 10
// post: cambia la calificación del Comentario.
voidcalificar(intcalificacion);
};
```