Práctica 1

Guía de Listas

Ejercicio 1.1. Diseñar e implementar la clase PilaEnteros.

Ejercicio 1.2. Hacer un proyecto para probar el uso de la clase desarrollada en el punto anterior.

Ejercicio 1.3. Hacer una función factorial iterativa, simulando la recursividad con la pila.

Ejercicio 1.4. Diseñar e implementar la clase ColaEnteros.

Ejercicio 1.5. Hacer un proyecto para probar la clase desarrollada en el punto anterior.

Ejercicio 1.6. 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 (numero);
```

al principio del programa. *numero* 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/

Ejercicio 1.7. 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 máximo
- Obtener el mínimo
- Obtener el tamaño de la lista
- Obtener el promedio
- 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.

Ejercicio 1.8. 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 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.

Ejercicio 1.9. 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.

Ejercicio 1.10. Probar lo realizado en el punto anterior.

Ejercicio 1.11. 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.

Ejercicio 1.12. Hacer un proyecto y probar la lista del punto anterior.

Ejercicio 1.13. Diseñar e implementar la ListaEnteros similar a la del punto 11 pero sin aceptar elementos repetidos.

Ejercicio 1.14. Utilizando la lista del punto anterior, agregarle los métodos:

- Incluida: Recibe una lista por parámetro e indica si ésta está incluida o no (devuelve true o false).
- Incluye: 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).

Ejercicio 1.15. Hacer un proyecto con un menú en donde se puedan utilizar los métodos escritos en el punto anterior.

Ejercicio 1.16. Modificar los métodos que reciben una lista por parámetro, escritos en el punto 14, pasándole un puntero a una lista. Lo mismo los que devuelvan listas, devolverán un puntero a una lista.

Ejercicio 1.17. Probar el funcionamiento de los métodos escritos en el punto anterior.

Ejercicio 1.18. Analizar (escribir) ventajas y desventajas de trabajar con punteros a listas en lugar de las propias listas.

Ejercicio 1.19. Modificar la lista diseñada e implementada en el punto 7 por ListaChar (lista de caracteres).

Ejercicio 1.20. Probar la lista escrita en el punto anterior.

Ejercicio 1.21. A la lista escrita en el punto 19 agregarle un método que indique si la lista es palíndroma (capicúa).

Ejercicio 1.22. Probar el uso de la lista del punto anterior.

Ejercicio 1.23. 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.

Además, agregarle los siguientes métodos:

- longitud (devuelve la longitud de la lista)
- Concatenar (se le pasa una lista y la concatena en la propia lista)

Ejercicio 1.24. Hacer un proyecto para realizar pruebas con la lista desarrollada en el punto anterior.

Ejercicio 1.25. El método concatenar desarrollado en el punto 23 reemplazarlo por la sobrecarga del *operador +*: el método será operador+