Ejercicios del Tema 8

1. Implementa, con ayuda de una pila, un procedimiento no recursivo que reciba como parámetro un número entero n >= 0 y escriba por pantalla sus dígitos en orden lexicográfico y su suma. Por ejemplo, ante n = 64323 escribirá:

6 + 4 + 3 + 2 + 3 = 18

2. C++ utiliza llaves, "{" y "}", para delimitar grupos de instrucciones. Si tratas un programa C++ como una cadena de caracteres puedes verificar que el programa tiene las llaves equilibradas usando una pila (equilibrada = hay tantas llaves abiertas como cerradas y cada vez que aparece una cerrada la llave anterior que haya aparecido debe ser una abierta). Implementa un subprograma que lo haga.

3. Como extensión del ejercicio anterior, implementa un subprograma que reciba una cadena de caracteres que contiene, entre otros símbolos, llaves, paréntesis y corchetes abiertos y cerrados y devuelva un valor indicando si está equilibrada. La secuencia está equilibrada respecto a los tres tipos de símbolos mencionados si para cada uno de ellos hay tantos abiertos como cerrados, y si cada vez que aparece uno cerrado el último que apareció fue su correspondiente abierto.

4. Una expresión aritmética construida con los operadores binarios +, -, * y / y operandos (números de un único dígito) se dice que está en forma posfija si es o bien un sólo operando o dos expresiones en forma postfija, una tras otra, seguidas inmediatamente de un operador. A continuación puedes ver un ejemplo de una expresión escrita en notación infija habitual, junto con su forma posfija:

Forma infija: (8 / (2 - 1)) * (8 + 2)

Forma postfija: 821-/82+*

Implementa un algoritmo iterativo que devuelva el valor de evaluar una expresión dada en forma posfija. La expresión se recibe en una cadena de caracteres y se puede asumir que es correcta. Para ahorrar aspectos que compliquen el algoritmo se supone que la expresión no contiene operadores unarios ni potencias y que los operandos son de una única cifra.

5. Implementa un subprograma que reciba una pila y escriba todos sus elementos, desde la base hasta la cima, separados por espacios. Haz dos versiones, una versión recursiva y otra iterativa.

Haz una tercera versión, implementando la funcionalidad como parte del TAD Pila soportado por lista enlazada.

6. Extiende las tres implementaciones de las pilas con una operación nueva, numElems que devuelva el número de elementos almacenados en ella. Hazlo de forma que su complejidad sea O(1).

7. Implementa un subprograma que reciba una pila y un número positivo *n* e invierta los *n* valores almacenados en la parte superior de la pila.

8. Realiza una implementación del TAD Cola usando array estático circular.

9. Extiende con un operador de igualdad (==) la implementación de las colas mediante array estático circular proporcionada en clase.

10. Realiza una implementación de las colas usando array dinámico circular.

11. Para evitar los casos especiales cuando la cola está vacía (en la operación `pon`) o se queda vacía (en la operación `quita`), en la implementación enlazada de la cola podemos usar una variante de la lista enlazada simple: la lista enlazada con nodo cabecera. El coste de las operaciones del TAD no cambia pero la implementación es más sencilla. Realiza tú una implementación con esa estructura de datos como soporte.

12. Extiende la implementación de la cola basada en lista enlazada con una nueva operación, numElems, que devuelva el número de elementos almacenados en ella. Hazlo de forma que su complejidad sea O(1).

13. Se dice que una frase es palíndroma si la sucesión de caracteres obtenida al recorrerla de izquierda a derecha es la misma que si se recorre de derecha a izquierda (asumiendo que la sucesión no tendrá tildes y todo serán o bien letras o bien espacios). Esto sucede, por ejemplo, con la socorrida frase *dábale arroz a la zorra el abad* si ignoramos la tilde de la primera palabra y los espacios en blanco. Implementa un subprograma iterativo de coste lineal en tiempo que decida si una frase leída de teclado es o no palíndroma.

14. El/La profesor/a de EDA ha decidido sacar a un alumno a hacer un ejercicio a la pizarra. Para seleccionar al "afortunado" ha numerado a cada uno de los *n* alumnos con un número del 1 al *n* y los ha colocado a todos en círculo. Empezando por el número 1, va "salvando" al segundo de cada dos (es decir, "salva" al 2, luego al 4, luego al 6, etc.), teniendo en cuenta que al ser circular, cuando llega al final sigue por los que quedan sin salvar. ¿Qué número tendrá el alumno "afortunado" que finalmente queda sin salvar y sale a la pizarra?

Implementa una función `int selecciona(int n)` que devuelva el número de alumno seleccionado si tenemos *n* alumnos ($n \geq 1$).

Generaliza la función anterior para el caso en el que el/la profesor/a salve a uno de cada *m* en lugar de a uno de cada 2 ($m \geq 2$).

15. Dado un número natural N≥2, se llaman números supervivientes a los que resultan de ejecutar el siguiente proceso: a partir de la secuencia 1,2,...,N se elimina el primer número de cada dos (es decir, se eliminarían los números 1, 3, 5, etc. y sobrevivirían los pares); de la secuencia resultante (la formada por todos los pares de la secuencia inicial) se elimina el primer número de cada 3; y

así sucesivamente. El proceso termina cuando se va a eliminar el primer número de cada *m* pero sobreviven menos de *m* números. Implementa un subprograma que reciba N como parámetro y muestre por pantalla la secuencia de números supervivientes resultante.

16. Imagina que usas una lista de enteros para representar un número con un nº de dígitos indeterminado; cada elemento de la lista es un dígito del número, siendo el primer elemento de la lista el dígito más significativo del número. Implementa un subprograma que, dados dos números enteros representados de la manera descrita, devuelva una lista con la suma de ambos números. Hazlo usando operaciones del TAD lista.

17. Extiende la implementación del TAD Lista con una nueva operación `concatena` que reciba dos listas l1 y l2 y añada todos los elementos de l2 a l1. Si l1=[3,4,5] y l2=[5,6,7], `l1.concatena(l2)` hará que l1=[3,4,5,5,6,7]. ¿Qué complejidad tiene la operación? ¿Podríamos conseguir complejidad O(1) de alguna forma?

18. **[Examen 2º parcial, junio 2014]** Extiende el TAD Lista visto en clase con una nueva operación cuya cabecera en C++ es:

```
void insertarPorPosicion(const T &elem, int pos);
```

que inserta el elemento `elem` de acuerdo con la posición `pos` (operación `pon_ppio`) de forma que cuando `pos` es 0 se añade al principio de la lista, mientras que cuando es igual al número de elementos de la lista se insertará detrás del último (operación `pon_final`). Indica la complejidad de la operación. Si llamas a otros métodos, debes implementarlos también.

19. **[Examen 2º parcial, junio 2015]** Extiende el TAD Lista visto en clase con una nueva operación que modifique la lista intercalando sus nodos de la siguiente forma: supongamos que los nodos de la lista enlazada de izquierda a derecha son n_1; n_2; n_3; n_4; ... ; n_{k-3}; n_{k-2}; n_{k-1}; n_k , al finalizar la ejecución del método los nodos estarán colocados como n_1; n_k; n_2; n_{k-1}; n_3; n_{k-2}; n_4; n_{k-3}; n_5; ....

Indica la complejidad de tu implementación. Esta debe ser lo más eficiente posible. Para ello, se debe evitar liberar y reservar memoria, y hacer copias de los campos. Si haces uso de métodos auxiliares, impleméntalos también.

20. Implementa un subprograma genérico que reciba una lista e imprima todos sus elementos por pantalla.

21. Implementa un subprograma que reciba una lista de enteros y cuente cuántas posiciones hay en ella tales que el elemento que hay en esa posición es igual a la suma de todos sus precedentes.

22. Implementa un subprograma que reciba una lista de enteros y multiplique por dos todos sus elementos.

23. Implementa un subprograma que duplique los elementos de la lista dada. P.e., si la lista es [1,2,3] la transforma en [1,1,2,2,3,3]

24. Implementa un subprograma que reciba dos listas de enteros ordenados crecientemente y devuelva una nueva lista ordenada con la unión de los enteros de las dos listas iniciales.

25. Implementa un subprograma que elimine de una lista de enteros dada todos los elementos que sean números pares.

26. Extiende el TAD Lista implementando :

I. Dos operaciones nuevas `rcbegin()` y `rcend()` que devuelvan sendos objetos `rConstIterator` (iteradores constantes inversos) al final y al principio de la lista, de forma que permitan recorrer la lista del final al principio.

II. Dos operaciones nuevas `rbegin()` y `rend()` que devuelvan sendos objetos `rIterator` (iteradores inversos) al final y al principio de la lista, de forma que permitan recorrer la lista del final al principio y cambiar los valores de sus elementos.

Y úsalos para:

III. Implementar un subprograma que muestre la lista del final al principio.

IV. Volver a implementar un subprograma que reciba una lista de enteros y multiplique por dos todos sus elementos.

27. Haciendo uso del TAD Lista extendido en el ejercicio anterior implementa una solución, usando iteradores, para el ejercicio 16.

28. El agente 0069 ha inventado un nuevo método de codificación de mensajes secretos. El mensaje original X se codifica en dos etapas:

a) X se transforma en X' reemplazando cada sucesión de caracteres consecutivos que no sean vocales por la sucesión inversa.

b) X' se transforma en el mensaje X'' obtenido cogiendo sucesivamente el primer carácter de X', luego el último, luego el segundo, luego el penúltimo, etc.

Por ejemplo, si el mensaje original X = *Bond, James Bond*, los resultados de las dos etapas de codificación son: X' = *BoJ ,dnameB sodn* y X'' = *BnodJo s, dBneam*. Suponiendo que el mensaje se almacena en una lista de caracteres, implementa un subprograma que codifique el mensaje que recibe y otro que decodifique el mensaje codificado.

29. Implementa:

I. Un subprograma que busque un elemento en una lista genérica y devuelva un iterador mutable al primer elemento encontrado (o a `end()` si no hay ninguno).

II. Reimplementa el subprograma anterior para que, en lugar de recibir la lista, reciba dos iteradores que marcan el principio y el final del intervalo [principio, final) en el que hay que buscar el elemento.

III. Implementa un subprograma que reciba una lista genérica Lista<T>, dos iteradores que marcan el principio y el final de un intervalo [principio, final) de la lista y un valor de tipo T, y elimine todos los elementos que se encuentren en ese intervalo de la lista que sean iguales al valor dado.

30. [Examen 2º parcial, sept 2017]Extiende el TAD Cola implementado con lista enlazada simple visto en clase con una nueva operación cuya cabecera en C++ es

```
void llevarAlPrincipio(unsigned int pos);
```

que lleva el elemento que está en la posición *pos* de la cola a la primera posición de la misma. Ten en cuenta que pos = 1 corresponde al primer elemento de la cola; pos = 2 corresponde al segundo; y así sucesivamente.

Si la posición *pos* no existe, deberá señalarse un error _Posicion inexistente_.

Aparte de implementar esta operación, debes indicar la complejidad de la misma.

Para resolver este ejercicio no se puede crear ni destruir memoria dinámica ni tampoco modificar los valores almacenados en la cola.