Capítulo 6.- Listas enlazadas

Cuestionario

6.1. Una lista enlazada simple es una estructura lineal
a) Verdadero
b) Falso

6.2. Una lista enlazada simple es estática
a) Verdadero
b) Falso

6.3. Para usar una lista, debe definirse su tamaño máximo
a) Verdadero
b) Falso

6.4. Al declarar una lista enlazada simple, es necesario crear un método que detecte si está llena
a) Verdadero
b) Falso

6.5. Para imprimir los elementos de una lista enlazada simple, se implementa un ciclo desde 0 hasta Max
a) Verdadero
b) Falso

6.6. ¿Cómo se detecta si una lista enlazada simple esta vacía?
a) Si `NodoInicial == 0`
b) Si `NodoInicial == -1`
c) Si `NodoInicial == null`
d) Si `NodoInicial == ""`

6.7. Una lista enlazada simple es una estructura …
a) No lineal y estática
b) Lineal y dinámica
c) No lineal y dinámica
d) Lineal y dinámica

6.8. Situación crítica que se presenta cuando se inserta el primer nodo en una lista simple
a) Alta a lista vacía
b) Alta al principio
c) Alta intermedia
d) Alta al final

6.9. ¿Para qué sirve el iterador `GetEnumerator()` de la `ClaseListaSimpleOrdenada`?
a) Para insertar un nodo
b) Para eliminar un nodo
c) Para obtener los nodos contenidos en la lista
d) Para localizar un nodo en particular

6.10. ¿Para qué sirve el método `CompareTo()` de la interfaz `IComparable` en la `ClaseListaSimpleOrdenada`?
a) Para eliminar un nodo
b) Para establecer el criterio de ordenamiento de los nodos
c) Para recorrer todos los nodos de la lista simple

d) Para buscar un nodo en particular

6.11. Una lista enlazada doble es una estructura lineal
a) Verdadero
b) Falso

6.12. Una lista doble es dinámica
a) Verdadero
b) Falso

6.13. Si `NodoInicial = null`, entonces la lista enlazada doble esta vacía
a) Verdadero
b) Falso

6.14. Una lista enlazada doble puede recorrerse en forma ascendente y en forma descendente
a) Verdadero
b) Falso

6.15. El tamaño de una lista doble se define cuando se declara
a) Verdadero
b) Falso

6.16. ¿Cómo se detecta si una lista enlazada doble esta vacía?
a) Si `NodoInicial == 0`
b) Si `NodoInicial == -1`
c) Si `NodoInicial == null`
d) Si `NodoInicial == Max`

6.17. ¿Cómo se detecta si una lista enlazada doble está llena?
a) Si `NodoInicial > NodoFinal`
b) Si `NodoInicial == NodoFinal -1`
c) Si `NodoInicial == NodoFinal`
d) Ninguna de las anteriores

6.18. ¿Cuál es la condición del ciclo para imprimir los nodos de una lista enlazada doble?
a) `while( NodoActual == 0 )`
b) `while( NodoActual != null )`
c) `while( NodoActual == null )`
d) `while (NodoAnterior != null )`

6.19. Cuando se inserta un nodo en la lista doble, ¿Cómo se detecta si se trata del primer nodo insertado?
a) Si `Nodonicial == 0 AND NodoFinal == 0`
b) Si `NodoInicial == -1 OR NodoFinal == -1`
c) Si `NodoInicial == null OR NodoFinal == null`
d) Si `NodoInicial == null AND NodoFinal <> null`

6.20. ¿Cómo se inicializa vacía la lista doble cuando se crea?
a) `NodoInicial = 0 y NodoFinal = 0`
b) `NodoInicial = -1 y NodoFinal = -1`
c) `NodoInicial = Max y NodoFinal = Max`
d) `NodoInicial = null y NodoFinal = null`

Ejercicios

6.1. Agregue un método a la `ClaseListaSimple` de la aplicación de formas de Windows para que devuelva la cantidad de estudiantes ó libros almacenados.

6.2. Modifique el método `InsertarEnListaSimple()` de la aplicación de consola que utiliza la clase genérica *List*, para que ordene los nodos en forma ascendente desde la inserción.

6.3. Desarrolle una aplicación con formas de Windows que administre una lista simple de los proveedores de una empresa y sus productos con los siguientes datos:

- Clave (entero)
- Nombre (cadena)
- Domicilio (cadena)
- Teléfono (cadena)

Cada proveedor debe tener una lista simple de productos suministrados con los siguientes datos:

- Código de barras (cadena)
- Descripción (cadena)
- Unidad de medida (cadena)
- Costo (real)

Implemente un método que prepare un reporte con todos los productos suministrados por cada proveedor. El reporte debe estar ordenado en forma ascendente de acuerdo a la clave del proveedor y sus productos ordenados en forma descendente por el costo.

6.4. Modifique la `ClaseColaLista` para que la cola implementada mediante una lista se pueda recorrer en ambas direcciones y se trate como una cola doble.

6.5. Modifique el método `Push()` de la `ClasePilaOrdenadaLista` para que permita insertar nodos duplicados.

6.6. Agregue un método a la `ClaseColaLista` que reciba un nodo como parámetro y lo elimine de la cola implementada mediante una lista.

6.7. Un restaurante desea almacenar las notas de consumo elaboradas durante el día. Para ello, requiere implementar una pila ordenada mediante una lista simple con los siguientes datos de cada nota:

- Hora (DateTime)
- Alimento ó bebida solicitado (Cadena)
- Precio (Real)

Modifique la `ClaseNodoPilaLista` del Prog. 6.5 (Pila-Ordenada-Lista) para controlar estos datos y los ordene en forma ascendente de acuerdo a la hora.

6.8. Elabore un programa que almacene datos reales en una lista doble ordenada y diseñe un método que devuelva la suma de sus datos.

6.9. Agregue un método a la `ClaseColaLista` que devuelva el valor booleano verdadero (`true`) si la cola está ordenada en forma ascendente ó falso (`false`) en caso contrario.

6.10. Resuelva el ejercicio 6.7 pero con una cola implementada mediante una lista, de tal forma que se pueda obtener un reporte ordenado por hora tanto en forma ascendente como descendente.

6.11. Diseñe un método que invierta el orden de los nodos almacenados en una lista simple.

6.12. Un profesor desea tener un listado de sus alumnos ordenado por calificación tanto en forma ascendente como descendente. Implemente un programa que utilice una lista doble que resuelva este problema y que administre los siguientes datos de los estudiantes:

- Número de matrícula (entero)
- Nombre (cadena)
- Calificación (real)