INSTITUTO TECNOLÓGICO DE COSTA RICA
Escuela de Computación
Bachillerato en Ingeniería en Computación
IC-2001 Estructuras de Datos
Prof. Mauricio Avilés

Tarea Corta 2 – Listas

Esta tarea consiste en la implementación, utilización y modificación de las siguientes estructuras de datos:

1. Lista con arreglos
2. Lista enlazada

Parte 1

Implementar la clase **ArrayList**. Esta clase debe derivar de la clase abstracta **List** vista en clase. A continuación se detallan los métodos que deben implementarse. Se subrayan los métodos que cuentan con diferencias con respecto de la implementación vista en clase.

1. **Constructor y destructor**
2. **void insert(E pElement)**
   Inserta un elemento nuevo en la posición actual. Si la lista ya alcanzó su tamaño máximo, debe crearse un nuevo arreglo del doble de tamaño y traspasarse todos los elementos al nuevo arreglo. Debe liberarse la memoria del arreglo anterior y actualizar el valor del atributo maxSize.
3. **void append(E pElement)**
   Agrega un elemento nuevo en la última posición de la lista. La posición actual no se modifica. Al igual que el método anterior, debe crearse un arreglo del doble de tamaño si el tamaño máximo ha sido alcanzado.
4. **E remove()**
   Elimina el elemento encontrado en la posición actual. Retorna como resultado el valor de dicho elemento.
5. **void clear()**
   Elimina todos los elementos de la lista y la deja vacía.
6. **E getElement()**
   Retorna el valor del elemento encontrado en la posición actual.
7. **void goToStart()**
   Mueve la posición actual al inicio de la lista.
8. **void goToEnd()**
   Mueve la posición actual al final de la lista.
9. **void goToPos(int pPos)**
   Mueve la posición actual a la posición indicada por parámetro.
10. **void next()**
    Mueve la posición actual al siguiente elemento, si lo hay.
11. **void previous()**
    Mueve la posición actual al elemento anterior, si lo hay.

12. **bool atStart()**
    Retorna verdadero si la posición actual de la lista se encuentra al inicio. De otro modo retorna falso.
13. **bool atEnd()**
    Retorna verdadero si la posición actual de la lista se encuentra al final. De otro modo retorna falso.
14. **int getPos()**
    Retorna un entero con la posición actual dentro de la lista.
15. **int getSize()**
    Retorna el tamaño de la lista.
16. **bool contains(E element)**
    Retorna verdadero si el elemento enviado por parámetro se encuentra dentro de la lista, de otro modo retorna falso.
17. **int indexOf(E element)**
    Retorna la posición de la primera ocurrencia del elemento enviado por parámetro. Si elemento no se encuentra en la lista, se retorna -1 como resultado.
18. **void extend(List L)**
    Agrega al final de la lista actual los elementos de la lista enviada por parámetro, en el mismo orden. La lista enviada por parámetro no debe ser modificada.

Escriba un programa que demuestre extensamente el funcionamiento de todos los métodos de la clase. Utilice cantidades grandes de datos (miles de elementos).

Parte 2

Implementar la clase **LinkedList**. Esta clase debe derivar de la clase abstracta **List** vista en clase. A continuación se detallan los métodos que deben implementarse.

1. **Constructor y destructor**
2. **void insert(E pElement)**
   Inserta un elemento nuevo en la posición actual.
3. **void append(E pElement)**
   Agrega un elemento nuevo en la última posición de la lista.
4. **E remove()**
   Elimina el elemento encontrado en la posición actual. Retorna como resultado el valor de dicho elemento.
5. **void clear()**
   Elimina todos los elementos de la lista y la deja vacía.
6. **E getElement()**
   Retorna el valor del elemento encontrado en la posición actual.
7. **void goToStart()**
   Mueve la posición actual al inicio de la lista.
8. **void goToEnd()**
   Mueve la posición actual al final de la lista.
9. **void goToPos(int pPos)**
   Mueve la posición actual a la posición indicada por parámetro.

10. **void next()**
    Mueve la posición actual al siguiente elemento, si lo hay.
11. **void previous()**
    Mueve la posición actual al elemento anterior, si lo hay.
12. **bool atStart()**
    Retorna verdadero si la posición actual de la lista se encuentra al inicio. De otro modo retorna falso.
13. **bool atEnd()**
    Retorna verdadero si la posición actual de la lista se encuentra al final. De otro modo retorna falso.
14. **int getPos()**
    Retorna un entero con la posición actual dentro de la lista.
15. **int getSize()**
    Retorna el tamaño de la lista.
16. **void reverse()**
    Invierte el orden de los elementos de la lista.
17. **bool equals(List L)**
    Compara los elementos de la lista actual y los de la lista enviada por parámetro. Si los elementos son los mismos y se encuentran en las mismas posiciones, el método retorna verdadero. Si la cantidad de elementos es diferente, si no son iguales, o no están en el mismo orden, retorna falso.

Escriba un programa que demuestre extensamente el funcionamiento de todos los métodos de la clase. Utilice cantidades grandes de datos (miles de elementos).

Parte 3

Basándose en las clases de los ejercicios anteriores, construya dos nuevas clases llamadas **OrderedArrayList** y **OrderedLinkedList** que tienen como objetivo mantener sus elementos ordenados de menor a mayor. Cada una de estas clases se implementa basándose en **ArrayList** o en **LinkedList**, respectivamente.

Cada clase tendrá un atributo de tipo **ArrayList** o **LinkedList** que es donde se van a almacenar los elementos, no debe implementarse nuevamente todos los elementos de dicha clase, si no utilizarlos. Los métodos de esta nueva clase se implementan mediante llamadas a los métodos de la clase en la que está basada. A continuación se enumeran los métodos que debe implementar esta clase:

a. **void insert(E element)**
   Inserta un elemento en la lista. La posición actual debe cambiar automáticamente a la posición adecuada para que los elementos queden ordenados ascendentemente.
b. **E remove()**
c. **void clear()**
d. **E getElement()**
e. **void goToStart()**
f. **void goToEnd()**
g. **void goToPos(int pos)**

h. **void next()**
i. **void previous()**
j. **bool atEnd()**
k. **bool atStart()**
l. **int getPos()**
m. **int getSize()**
n. **int indexOf(E element)**
   En el caso de OrderedArrayList, la búsqueda del elemento debe realizarse por medio de búsqueda binaria.

Escriba un programa que pida al usuario si desea utilizar OrderedArrayList u OrderedLinkedList. Luego, solicite al usuario una serie de números, cada uno se insertará en la lista seleccionada. Luego de insertarlos, imprimir en pantalla los contenidos de cada lista. Finalmente, solicitar al usuario un número para buscar en la lista mediante indexOf, y se imprime en pantalla el resultado.

Entrega: Sección de evaluaciones del TEC-Digital.
Formato: Archivo comprimido ZIP con tres proyectos de Code::Blocks, uno por cada parte de la tarea. Corrobore que se incluyen todos los archivos necesarios para que los proyectos se ejecuten correctamente.