Práctico 6

Listas implementadas con arreglos y memoria dinámica

NOTA: Los ejercicios deberán entregarse completos, siguiendo los criterios aconsejados por la cátedra, con los controles adecuados, modularizado, etc. Especifique los operadores que considere apropiados. Deberá colocar los archivos `nodo.h`, `lista_p.h`, `lista_a.h`, `graph_list.h` y `main.cpp` en un archivo .zip para la entrega.

Cuando nos referimos al tipo de dato abstracto `Lista` en los ejercicios estamos hablando de un alias de `ListaA` o `ListaP` como se ejemplifica en el documento del campus virtual, que se define de la siguiente manera:

```
#include "lista_p.h"
#include "lista_a.h"

#define Lista ListaP
// #define Lista ListaA
```

Recordar que esto es posible solo si no modificamos el segundo parámetro del template (el tamaño).

EJERCICIO 1

Utilizando el tipo de dato abstracto `Lista` provisto por la cátedra, implementar una función de aplicación que produzca el ingreso ordenado de forma ascendente de elementos en una lista de `string` (ya ordenada), es decir, una función que ubique los elementos de forma ordenada a medida que estos sean cargados. Respetar el siguiente prototipo:

```
void insertarOrdenado(Lista<string> &listaOrdenada, string entrada);
```

Importante: La implementación tiene que ser compatible con las dos implementaciones (arreglos y punteros).

EJERCICIO 2

Utilizando el tipo de dato abstracto `Lista` provisto por la cátedra, implementar una función de aplicación que invierta los elementos de la lista.

```
template <class T>
void invertir(Lista<T> &aInvertir);
```

Importante: La implementación tiene que ser compatible con las dos implementaciones (arreglos y punteros).

EJERCICIO 3 (OBLIGATORIO)

Se requiere implementar una funcionalidad para intercalar los elementos de dos listas.

A) Implementar como una **función de aplicación**, garantizar que funcione con ListaA y ListaP. Respetando el siguiente prototipo:

```
template <class T>
Lista<T> mixList (Lista<T> l1 ,Lista<T> l2);
```

Si, `l1=[1,2,3,4,5,6] y l2=[8,9,0] → l3=mixList(l1, l2) → l3=[1,8,2,9,3,0,4,5,6]`

B) Implementar como una **operación básica** de listas (arreglos y punteros). Respetar el siguiente prototipo:

```
template <class T>
Lista<T> Lista<T>::mixList (Lista<T> l);
```

Si, `*this=[1,2,3,4,5,6] y l=[8,9,0] → l3=this->mixList(l) → l3=[1,8,2,9,3,0,4,5,6]`

EJERCICIO 4

Implementar un nuevo TDA llamado GraphList (graph_list.h) para representar grafos dirigidos utilizando lista de adyacencia.
Respetar el siguiente prototipo:

```
class GraphList {
public:
   GraphList();
   ~GraphList();
   GraphList &addNode(T e);
   void addEdge(T f, T t);
   bool isConnected(T f, T t);
   void print();
private:
   std::list<std::list<T>> data;
};
```

EJERCICIO 5

Para cada una de las librerías de listas vistas en la cátedra implementar las siguientes operaciones básicas (para conocer la semántica de cada una de estas operaciones dirigirse a la documentación de std::list de C++) :

Capacidad

```
bool estaVacia();                    //-->empty
```

Acceso a los elementos:

```
T recuperarPrimero();                //-->front
```

```
T recuperarUltimo();                    //-->back
```

Modificadores

```
void eliminarPrimerElemento();          //-->pop_front
void eliminarUltimoElemento();          //-->pop_back
void insertarPrimero(T);                //-->push_front
void insertarUltimo(T);                 //-->push_back
void intercambiar(Lista<T> &);          //-->swap    ListaA o ListaP según
// corresponda
```

Operaciones

```
void juntar(posicion, Lista<T> &);      //-->splice
void eliminarElemento(T);               //-->remove
void unicos();                          //-->unique
```

Complejidad

Compare la complejidad de cada uno de los métodos entre las dos alternativas de implementación (arreglos y punteros). Dejando anotado en el código la complejidad y en un documento de texto la comparación.