Implementación en listas

1) Método que, dadas dos posiciones, intercambia sus datos.
2) Método que une la lista original con otra que se pasa por parámetro, modificando a la lista que llama. La lista pasada por parámetro va al final.
3) Método que une la lista original con otra que se pasa por parámetro y devuelve un puntero a la nueva lista. La lista pasada por parámetro va al final.
4) Método que revierte la lista, modificándola (el último elemento pasa al primer lugar, etc).
5) Método que revierte la lista en una nueva lista, y devuelve un puntero de la misma.
6) Método que recibe una lista por parámetro, ambas (la original y la del parámetro) están ordenadas, se hace un merge entre las dos, modificando la original, que tiene que quedar ordenada y sin elementos repetidos. Para comparar se usa un método de Dato (comparar), ejemplo a.comparar_con(b). Este método devuelve -1 si a es menor que b; 1, si a es mayor que b; y 0 si son iguales.
7) Método que elimina un dato: se pasa un dato por parámetro y se elimina la primera ocurrencia. Usar el método comparar. Si no lo encuentra no hace nada.
8) Método que elimina todas las ocurrencias del dato.
9) Método que recibe una lista por parámetro y devuelve un puntero a una nueva lista que es A – B (los elementos de A que no están en B), donde A es la lista original y B es la del parámetro.

Uso de listas

Para estos ejercicios se suponen implementados los siguientes métodos de Lista, además de los que figuran en el .h, teniendo en cuenta que hay un atributo más que es *actual*.

```
// reinicia el puntero actual a la primera posición (o nulo si la lista es vacía).
// PRE:
// POS: pone el puntero a la primera posición o apuntando a NULL
void reiniciar ( );

// consulta si hay un elemento siguiente (si el actual no apunta a NULL)
// PRE:
// POS: devuelve true si el actual no apunta a NULL, false de lo contrario
bool hay_siguiente ( );

// devuelve el siguiente elemento (el elemento que apunta actual)
// PRE: hay_siguiente tiene que haber devuelto true previamente
// POS: devuelve el elemento actual y avanza
Dato siguiente ( );
```

1) Considerar como implementada la clase Alimento a partir de la siguiente interfaz:
   class Alimento

```
{
public:
        // Crea un alimento con su nombre, la cantidad de calorías y una lista
        // de los ingredientes que lo conforman
        Alimento (string nombre, unsigned int calorias, Lista<string>* ingredientes);
        string obtener_nombre(); // devuelve el nombre del alimento
        unsigned int obtener_calorias (); // devuelve la cantidad de calorías
        Lista<string>* obtener_ingredientes (); // devuelve un ptr la lista de ingredientes
};
```

Implementar el método comidas_para_celiacos de la clase Buscador_de_comidas:

```
class Buscador_de_comidas
{
        public:
        // Post: busca en "comidas" aquellas que tienen algún ingrediente de la
        // lista "ingredientes_permitidos" y ninguno de la lista
"ingredientes_no_permitidos"
        // y tienen una caloría menor a "caloría_maxima".
        // Devuelve una lista con los alimentos que cumplen con estas características.
        Lista<Alimento *>* comidas_para_celiacos (Lista<Alimento *>* comidas,
                Lista<string>* ingredientes_permitidos, Lista<string>*
        ingredientes_no_permitidos,
                unsigned int caloría_maxima);
};
```

2) Dada la clase implementada según la interfaz:

```
class Universidad
{
public:
        // Crea una universidad con su nombre, el ranking y una lista de carreras
        Universidad (string nombre, int ranking, Lista<string>* carreras);
        string obtener_nombre(); // devuelve el nombre de la universidad
        unsigned int obtener_ranking (); // devuelve el ranking
        Lista<string>* obtener_carreras (); // devuelve ptr a lista de carreras
};
```

Implementar el método recomendar_universidades de la clase Buscador_universidades:

```
class Buscador_universidades
{
        public:
        // Post: busca en "universidades" aquellas que tienen alguna carrera de la
        // lista "vocaciones" y un ranking mayor o igual a ranking_minimo.
        // Devuelve una lista con las universidades que cumplen con estas características.
        Lista<Universidad *>* recomendar_universidades (Lista<Universidad *>*
                        universidades, Lista<string>* vocaciones, int ranking_minimo);
};
```

3) Dada la clase implementada según la interfaz:

```
class Restaurante
{
public:
        // Crea un restaurante con su nombre, el precio promedio por cubierto y
        // una lista de platos que sirve
        Restaurante (string nombre, int precio_promedio, Lista<string>* platos);
        string obtener_nombre(); // devuelve el nombre del restaurante
        int obtener_precio_promedio (); // devuelve el precio promedio
        Lista<string>* obtener_platos (); // devuelve ptr a la lista de platos
};
```

Implementar el método recomendar_restaurantes de la clase Buscador_restaurantes:

```
class Buscador_restaurantes
{
        public:
        // Post: busca en "restaurantes" aquellos que tienen por lo menos dos platos de la
        // lista "platos_deseados" y un precio promedio menor o igual a precio_maximo.
        // Devuelve una lista con los restaurantes que cumplen con estas características.
        Lista<Restaurante *>* recomendar_restaurantes (Lista<Restaurante *>*
                        restaurantes, Lista<string>* platos_deseados, int precio_maximo);
};
```