Relación de Problemas sobre la STL

1. Implementa una función que, dada una lista l de tipo T y un elemento x de tipo T, elimine el elemento x de todas las posiciones en las que aparezca en l.
2. Implementa una función `elimina_duplicados(list<T> &l)` que elimine los elementos duplicados (sin ordenar previamente la lista).
3. Implementa una función que dada una lista l devuelva una lista que tenga los elementos de l pero en orden inverso.
4. Resuelve el problema anterior pero sobre una única lista pasada por referencia.
5. Implementa una función `list<T> mezclar(const list<T> &l1, const list<T> &l2)` que dadas dos listas ordenadas `l1` y `l2` devuelva una lista ordenada mezclando las dos listas.
6. Implementa una función a la que se le pase una lista de enteros y un entero x de manera que cada vez que aparezca en una posición ese valor, se inserte x-1 en la posición siguiente.
7. Implementa una función `bool contenida(const list<T> &l1, const list<T> &l2)` a la que se le pasen dos listas y compruebe si la lista `l1` está contenida en `l2` (si los elementos aparecen en la otra y en el mismo orden).
8. Tenemos dos listas: l, que contiene n elementos y otra li que contiene una serie de posiciones de la lista anterior (valores de la clase iterador). Construye una función a la que se le pasen esas dos listas y devuelva otra que contenga los elementos de l indicados por las posiciones de la lista li.
9. Un vector disperso es un vector en el que la inmensa mayoría de sus elementos son nulos. Para representar ese tipo de vectores se suele utilizar como representación un lista:

```
template <typename T>
class vdisperso {
  public:
    vdisperso(const vector<T> &v);
    void asignar_coeficiente(int i, const T &x);
    vector<T> convertir() const;
  ...
  private:
    list< pair<int, T> > coefs;
    int n;
}
```

   De esa forma se ahorra una gran cantidad de espacio. Implementa los dos métodos indicados en la parte pública.

10. Una variante del vector disperso es aquella en la que se puede definir cual es el valor nulo. Modifica la clase anterior de forma que se pueda definir cual es el valor nulo al crear un objeto. Implementa también `cambiar_nulo(const T &n)`

11. Una forma eficiente de guardar secuencias de valores iguales consiste en guardar cada uno de los valores seguido del número de veces que aparece en la secuencia. Por ejemplo,

    ```
    <1,1,1,2,2,2,2,2,2,7,7,7,7,7,12,1,1,5,5>
    < (1,3), (2,6), (7,5), (12,1), (1,2), (5,2) >
    ```

    Implementa las funciones:

    ```
    list<pair<T, int> > comprimir(const list<T> &l)+
    list<T> descomprimir(const list<pair<T, int> > &lc)
    ```

    que permitan convertir entre ambas representaciones.

12. Implementa la clase vector dinámico usando como representación el tipo list. ¿Qué orden de eficiencia tiene cada operación?

13. Implementa la función `intercalar(vector<T> &v, vector<pair<int, T> > pos)` que inserta el segundo elemento de cada par de `pos` en `v` en los lugares indicados por el primer elemento de cada par de `pos`

14. Define una clase que permita guardar los datos de los pilotos de F1. Por ejemplo, nombre, apellido (sólo uno), posición en la clasificación, nombre de la escudería. Además debe permitir que se pueda buscar a un piloto por su apellido. ¿Qué podríamos añadir a esa clase para que se pueda buscar también por cualquiera de los restantes campos?.

15. Escribe un programa que permita leer de la entrada estándar una serie de coordenadas (como un par de float) y que contabilice de forma eficiente el número de veces que aparece cada uno de los pares.

16. En un servicio de urgencias de un hospital quieren tener la posibilidad de poder gestionar el orden en que los pacientes se irán atendiendo. Para ello, de cada paciente se guarda nombre, apellidos, dni, gravedad. Al mismo tiempo se quiere poder acceder por dni. Construye una clase adecuada e implementa las operaciones de inserción y borrado.