Examen Final

1. Buscaminas (1.5 puntos)

¿Quién no ha jugado al Buscaminas? Este entretenido juego acompaña a cierto sistema operativo cuyo nombre no logramos recordar. El objetivo del juego es encontrar todas las minas ubicadas en un campo de dimensiones $M \times N$.
El juego muestra un número en un recuadro que indica la cantidad de minas adyacentes a ese recuadro. Cada recuadro tiene, como mucho, ocho recuadros adyacentes. El campo, de tamaño 4×4, de la izquierda contiene dos minas, cada una de ellas representada por el caracter "*". Si representamos el mismo campo con los números descritos anteriormente, tendremos el campo se muestra debajo:

*...
....
.*..
....

*100
2210
1*10
1110

Entrada
La entrada constará de un número arbitrario de campos. La primera línea de cada campo consta de dos números enteros, n y m ($0 < n; m \leq 100$), que representan, respectivamente, el número de líneas y columnas del campo. Cada una de las siguientes n líneas contiene, exactamente, m caracteres, que describen el campo.

Los recuadros seguros están representados por "." y los recuadros con minas por "*", en ambos casos sin las comillas. La primera línea descriptiva de un campo en la que $n = m = 0$ representa el final de la entrada y no debe procesarse.

Salida
Para cada campo, escribir el mensaje Field #x: en una línea, donde x corresponde al número
del campo, empezando a contar desde 1. Las siguientes n líneas deben contener el campo con los caracteres "." sustituidos por el número de minas adyacentes a ese recuadro. Debe haber una línea en blanco entre los distintos campos mostrados.

Ejemplo de entrada:
4 4
*...
....
.*..
....

3 5
**...
.....
.*...
0 0

Ejemplo de salida:
Field #1:
*100
2210
1*10
1110

Field #2:
**100
33200
1*100

2. El Problema de las Vacas (1.5 puntos)

- Se tienen 2 vacas que deben alimentarse. Se coloca bloques de pasto en fila para alimentarlas.
- Las vacas solo pueden comer el pasto que se encuentra en uno de sus extremos
- La primera de ellas, una vaca inteligente desea aplicar una estrategia que le permita comer más pasto.
- La segunda come el bloque de pasto más grande que se encuentra en uno de los extremos.
- Implemente la estrategia de la 1ra vaca (que empieza a comer) para que logre comer la mayor cantidad de pasto.

Ingreso: La primera línea contendrá el número de test a aplicar. Para cada test, se ingresará el número de bloques de pasto y en la siguiente línea la cantidad de pasto en cada bloque	**Salida:** Mostrará el total de pasto que comió la primera vaca.
3 4 12 15 13 11 8 2 2 1 5 3 8 7 3 6 2 5 3 1 3 1	26 18 8

3. Fragmentación de archivos (1.5 puntos)

Un amigo, bioquímico de profesión, tropezó en su laboratorio mientras llevaba una bandeja llena de archivos de ordenador. Todos los archivos se rompieron al caer al suelo. Él recogió todos los trozos y buscó ayuda para volver a ponerlos juntos.

Por suerte, todos los archivos de la bandeja eran idénticos, y todos se rompieron exactamente en dos trozos. Además, se encontraron todos los trozos. Por desgracia, no todos se rompieron por el mismo sitio, y los trozos se mezclaron unos con otros al caer.

Los fragmentos binarios originales se han traducido a caracteres ASCII 1's y 0's. La tarea consiste en escribir un programa que determine el patrón de bits que contenían los archivos.

Entrada
La entrada comienza con un único entero positivo en una línea, que indica el número de casos de prueba, seguido de una línea en blanco. También habrá una línea en blanco entre cada dos casos consecutivos.

Cada caso constará de una secuencia de ¨fragmentos de archivos", uno por línea, finalizado por un carácter de fin de archivo o una línea en blanco. Cada fragmento se representa como una cadena de 1s y 0s.

Salida
Por cada caso de prueba en la entrada, se mostrará una única línea de 1s y 0s que indique el patrón de bits de los archivos originales. Si hay 2*N* fragmentos en la entrada, debería ser posible concatenar dichos fragmentos, de manera que se formen *N* copias de la salida. Si no hay una solución única, cualquiera de las posibles será válida.

Nuestro amigo bioquímico está seguro de que no había más de 144 archivos en la bandeja, y que cada uno de ellos tenía menos de 256 bytes de tamaño. Se separará con una línea en blanco la salida de cada dos casos consecutivos.

Ejemplo de entrada
1
011
0111
01110
111
0111
10111
Ejemplo de salida
01110111

4. **Número de Rutas (1.5 puntos)**

Una empresa de transporte tiene agencias en varias ciudades. En cada ciudad se programan salidas a otras ciudades. Dada una ciudad de origen muestre cuántas rutas hay para llegar a la ciudad destino.

Ingreso: La primera línea contendrá el número de test a aplicar. Para cada test, se ingresará línea a línea la matriz de adyacencia indicando con 0 aquellas ciudades no conectadas y con 1 las conectadas	**Salida:** Mostrará el número de caminos de la ciudad origen a la ciudad destino.
2 4 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 3 7 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 4	2 0

5. La propiedad de ventana (2 puntos)

Suponga que se le da una secuencia de símbolos, pero sólo puede ver *k (k ≥ 1)* símbolos consecutivos a la vez cada vez. Entonces decimos que la longitud de la ventana es *k.* Mover esta ventana a lo largo de la secuencia puede darte muchos patrones diferentes. De todas las secuencias posibles de n símbolos diferentes sólo una minoría tiene la propiedad de que las ventanas de longitud *k* muestran sólo *k + 1* patrones diferentes.

Decimos que una secuencia de símbolos tiene la propiedad de ventana si para todo *k* natural el número de patrones diferentes que se pueden ver a través de una ventana de longitud *k* es como máximo *k + 1.*

Ejemplos

ABAABABAB	tiene la propiedad ventana.
ABCABCABC	no tiene la propiedad ventana (compruebe k = 1).
011010	tiene la propiedad ventana.
0110100101	no tiene la propiedad ventana.

En el tercer ejemplo los patrones son:

longitud 1 : 0, 1.
longitud 2 : 01, 11, 10.
longitud 3 : 011, 110, 101, 010.
longitud 4 : 0110, 1101, 1010.
longitud 5 : 01101, 11010.
longitud 6 : 011010.

La secuencia del último ejemplo es una extensión de la secuencia del tercer ejemplo. Así, los primeros 6 símbolos forman una secuencia con la propiedad ventana. El séptimo símbolo añade el patrón '00' al conjunto de patrones de longitud 2 que aparecen en las ventanas anteriores a la ventana que contiene "00". Por tanto, la secuencia formada por los 7 primeros símbolos no tiene la propiedad ventana. En consecuencia, llamamos séptimo símbolo el primer símbolo infractor. Por cierto, contamos de izquierda a derecha como nuestros ordenadores parecen hacer.
El problema consiste en determinar si una secuencia dada tiene la propiedad ventana y, en caso contrario, encontrar la posición del primer símbolo infractor, es decir, el símbolo tal que la secuencia que le precede tiene la propiedad ventana, pero al añadir el símbolo se destruye esta propiedad (el recuento de los símbolos empieza en uno).

Entrada

La entrada son líneas de texto en el que cada línea es una secuencia no vacía de caracteres (ASCII) que se comprueba si tienen la propiedad la propiedad ventana. Ninguna secuencia tendrá más de cien símbolos.

Salida

La salida deben ser varias líneas de texto que contenga para cada línea de la entrada una línea con el resultado de la comprobación de la propiedad ventana de la siguiente forma: **'SI'** (mayúsculas) si la línea goza de la propiedad ventana en caso contrario **'NO:'** (en mayúsculas) seguido de la posición del símbolo infractor. Cada línea debe terminar con un marcador de fin de línea.

Ejemplo de entrada
ababcababa
0010100100
0010101001

Ejemplo de salida
NO:5
SÍ
SÍ

6. El cargamento solitario (2 puntos)

Los Plusianos creen que la Instalación de Distribución y Embarque de Carga en el Aeropuerto (*ABCDEF* por sus siglas en inglés) en el Aeropuerto Internacional de Cosco (*CIA*) es la más grande del mundo, aunque la segunda más grande en Plusia. Los aviones de carga llegan y salen hacia muchos países alrededor del mundo (Cermán, Q.S.A, Capan, Custralia, etc.). *ABCDEF* se encarga de distribuir, cargar y descargar las cargas transportadas por estos aviones.

Cada carga es una caja cúbica de tamaño fijo y tiene una etiqueta que indica el país de destino. Para conveniencia, a cada país se le asigna un ID único. Por ejemplo, si hay *n* países, cada país recibe un ID único que varía de *1* a *n.*
Cada país *X* tiene su propia estación de carga identificada por su ID de país. Hay dos plataformas (plataforma *A* y plataforma *B*) en cada estación. En la plataforma *A* se colocan las cargas que serán transportadas (por aire) al país X en algún momento conveniente. La plataforma *B* es en realidad una cola de cargas que serán transportadas a países distintos de *X*. Las estaciones de carga tienen una disposición circular (en anillo), es decir, si hay *n* estaciones, cada uno de los siguientes pares de estaciones son adyacentes: *(1, 2), (2, 3), (3, 4), ..., (n− 1, n)* y *(n, 1).*

ABCDEF tiene muchos transportadores de carga terrestres utilizados para transportar carga de estación a estación. Pero estos transportadores tienen un espacio tan estrecho que no se pueden colocar dos cargas una al lado de la otra. Un transportador puede llevar más de una carga (aunque hay un límite en el número máximo de cargas que un transportador puede llevar) solo colocándolas (las cargas) una encima de la otra. Este arreglo en forma de pila tiene el problema de que no se puede remover cualquier carga de la pila a voluntad. Por ejemplo, para remover la tercera carga desde la parte superior, primero se deben remover las dos cargas superiores.
Un transportador de carga se mueve de estación a estación siguiendo estrictamente su disposición en anillo, es decir, de la estación 1 se mueve a la estación 2, luego a la 3, luego a la 4, ..., luego a la *n*, luego a la 1 nuevamente, etc. Requiere exactamente 2 minutos moverse de cualquier estación a su adyacente.
Después de llegar a cualquier estación (digamos, la estación *X*), el transportador de carga primero intenta descargar la carga. Empezando desde la carga superior de su pila, verifica la etiqueta adjunta a la carga. Si encuentra que la carga tiene como destino *X*, entonces la descarga en la plataforma *A*, de lo contrario verifica si la cola en la plataforma *B* tiene alguna posición vacante, y si es así, coloca la carga en la parte trasera de la cola. Este procedimiento de descarga continúa desde la parte superior hasta la inferior de la pila hasta que falle o la pila se vacíe, lo que ocurra primero. Cada intento de descarga exitoso requiere exactamente 1 minuto, es decir, descargar 3 cargas en una estación requerirá exactamente 3 minutos. Después de que la descarga esté completa, el transportador comienza a cargar. El

transportador continúa tomando la carga en la parte delantera de la cola en la plataforma B y colocándola en la parte superior de su pila hasta que la cola esté vacía o la pila esté llena, lo que ocurra primero. Cada intento de carga exitoso también requiere exactamente 1 minuto, es decir, cargar 4 cargas desde una estación requerirá exactamente 4 minutos. Después de que la carga esté completa, el transportador se mueve a la siguiente estación en el anillo.
De esta manera, durante muchos años, los transportadores de carga han estado realizando la tarea de mover las cargas desde la plataforma B a la plataforma A de las estaciones correspondientes, desde donde los aviones de carga las transportan a sus países de destino. Pero, después de un conflicto con la administración sobre la escala salarial, los empleados de *ABCDEF* han iniciado una huelga indefinida desde el domingo pasado. Los aviones están llegando y saliendo, pero no hay nadie para cargar y descargar la carga. No hay nadie para distribuir las cargas en cola a sus estaciones de destino. Toda la instalación ahora está paralizada.
Pero tú, como siempre conocido y odiado por tus colegas como *el hombre del jefe*, has decidido romper la huelga y salvar el *CIA*. Vas a comenzar a trabajar mañana por la mañana y distribuir las cargas en cola a sus estaciones correspondientes utilizando un transportador de carga. Inicialmente tu transportador estará vacío y comenzarás tu viaje desde la estación 1 y continuarás moviéndote alrededor del anillo hasta que todas las cargas hayan sido distribuidas a sus estaciones de destino. Pero antes de comenzar el trabajo, has decidido escribir un programa para determinar exactamente cuánto tiempo tomará completarlo.

Entrada
La entrada contiene varios conjuntos de datos. La primera línea del archivo de entrada contiene un entero SET, que indica cuántos conjuntos de datos hay. Luego sigue *SET* conjuntos de datos. En nuestro ejemplo de entrada el valor de *SET* es 2.

La primera línea de la entrada contiene tres enteros: *N, S y Q*. N ($2 \le N \le 100$) es el número de estaciones en el anillo. S ($1 \le S \le 100$) es la capacidad de tu transportador de carga, es decir, el número máximo de cargas que tu transportador puede llevar. Q ($1 \le Q \le 100$) es el número máximo de cargas que la cola en la plataforma B puede acomodar. Se asume que todas las colas en el sistema tienen la misma capacidad.

Luego siguen N líneas. Asumiendo que estas líneas están numeradas de 1 a N, para $1 \le i \le$ N, la línea i contiene un entero Q_i ($0 \le Q_i \le Q$) que indica el número de cargas en cola en la estación i seguido de Q_i enteros que indican las estaciones de destino de las cargas en cola de adelante hacia atrás. Puedes asumir que ninguna de estas Q_i cargas tendrá la estación i como destino.

Salida
Para cada conjunto de datos, muestra el número de minutos que tomará completar el trabajo en una línea separada.

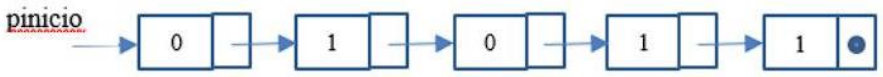
Ejemplo de entrada

2
5 2 3
3 4 5 2
2 1 3
0
3 3 5 1
1 4
6 3 4
4 4 5 2 6
2 1 3
0
3 3 5 1
2 4 6
2 1 5

Ejemplo de salida

72
112

7. Se desea implementar números binarios utilizando listas enlazadas. Cada nodo de la lista almacenará el dígito binario (un 0 o un 1). El dígito menos significativo ocupará la primera posición de la lista. Por ejemplo, si el número fuera 11010, se almacenaría como:



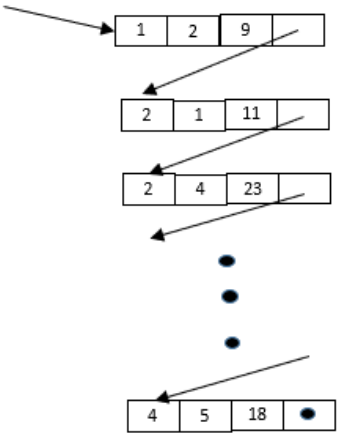
Escriba un programa en C++ que permita ingresar dos números binarios desde teclado, almacenarlos en una lista enlazada de la forma indicada y luego obtener una tercera lista que sea la suma de los números almacenados. Se debe mostrar en pantalla las tres listas (debe implementar la lista con punteros, no puede usar los contenedores stl como stack, queue, etc). **(1.5 puntos)**

Ejemplo:

Número binario	Lista
10110	0 1 1 0 1
1101	1 0 1 1
Resultado 100011	1 1 0 0 0 1

8. Cierto problema requiere utilizar matrices de gran tamaño, pero de pocas celdas utilizadas. Para evitar desperdiciar memoria el programador ha diseñado una lista dinámica con registros que representan a cada celda utilizada. Ejemplo si la celda es la que corresponde a la fila 24 y columna 96 y tiene el valor 235, entonces el registro de la lista contiene los tres datos (24, 96 y 235). Diseñe un programa que reciba los valores de n celdas y los almacenes en la lista indicada. Luego determine la suma de los elementos de cada fila e indique que fila o filas tiene la suma mayor. Asuma que si una fila no tiene elementos, la suma es cero. Ejemplo, si la matriz fuera:

	0	1	2	3	4	5	6
0							
1			9				
2		11			23		
3	45						
4				2		18	



Debe implementar la lista con punteros, no puede usar los contenedores stl como stack, queue, etc). **(1.5 puntos)**

9. Desarrolle un programa que haga uso de una Pila para evaluar expresiones booleanas en notación posfija (también conocida como notación polaca inversa). Considere expresiones booleanas formadas por los siguientes elementos:

Operadores	**Operandos**
Operador Binario AND (&)	Verdadero (1)
Operador Binario OR (|)	Falso (0)
Operador Binario XOR (^)	
Operador Unario NOT (!)	

Su programa debe permitir el ingreso de expresiones a través de consola y debe mostrar el resultado o indicar que la expresión es inválida. **(1.5 puntos)**

Ejemplos	
`1 0 &` → 0	`1 1 ^` → 0
`0 !` → 1	`0 0 ^` → 0
`1 0 | !` → 0	`1 1 ^ 1 ^` → 1
`1` → 1	`0 1 ^ 0 ^` → 1
`0 ! ! !` → 1	`0 1 ^ 0 ^ !` → 0
`0 1 & ! 0 ^ 1 1 | ^ !` → 1	`0 ! 1 ^ 0 ^ !` → 1
`0 1 1 0 & | ^ ! ! ! !` → 1	`0 1 | ! 1 1 & &` → 0
`0 1 & 0 !` → Expresión Inválida. Insuficientes operadores.	
`0 1 & &` → Expresión Inválida. Insuficientes operandos.	
`2 4 e f` → Expresión Inválida.	
`susti` → Expresión Inválida.	

10. Dada una clase genérica *Vector*, cuya definición aparece a continuación **(1.5 puntos)**:

```
#include "ExcepcionVectorRango.h"


template <typename T, long tam=10>
class Vector {
    T array[tam];
    int nelem;
public:
// Se omite la implementación de los métodos
    Vector() : nelem(0) {}
    ~Vector() { nelem=0; }
    bool addElement(const T& elem) {
        if (nelem<tam) {
            array[nelem++]=elem; return true;
        } else return false;
    }

    T operator[](int n) const {
        if (n<nelem) return array[n];
        else throw ExcepcionVectorRango(n, nelem);
    }
    int getNumElementos() const { return nelem; }
    int getCapacidad() const { return tam; }
};
```

Se pide:

(a) Implementa un método miembro de la clase llamado *buscar* que, dado un objeto de tipo T, devuelva cierto si el elemento esta almacenado en el vector o falso en caso contrario.

Como ejemplo, el siguiente código debería funcionar sin errores:

```
int main() {
    Vector<double> vd;

    if (vd.buscar(0.0)) {...}
}
```

(b) Indica que requisitos debe cumplir el tipo T para que se pueda usar con la clase *Vector* (incluyendo el método *buscar*).

(c) Implementa un método miembro llamado eliminar (T elem) que, dado un objeto de tipo T, elimine la primera ocurrencia de dicho elemento en el vector, desplazando los elementos restantes.

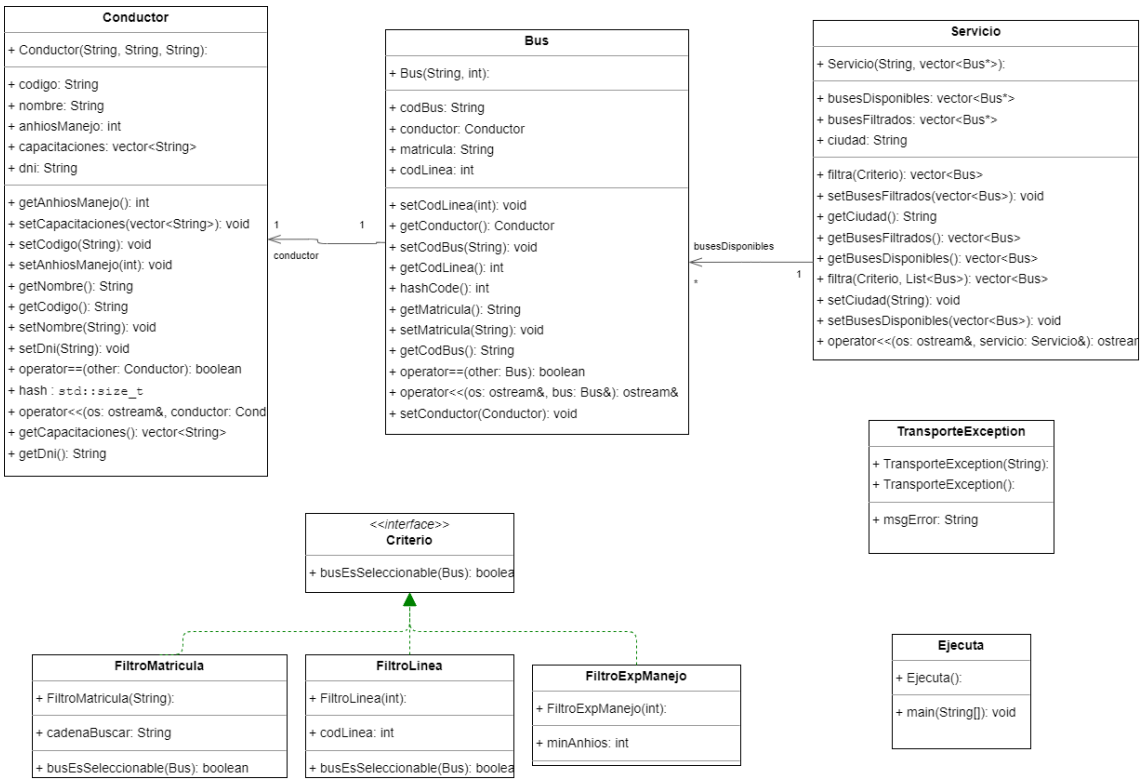
(d) Sobrecarga el operador == para la clase Vector de tal forma que compare si dos vectores tienen el mismo número de elementos y los mismos valores almacenados.

(d) Define e implementa la clase *ExcepcionVectorRango*, de manera que el siguiente código compile y funcione sin errores:

```
int main() {
    Vector<string> vs;
    while (vs.addElement("HOLA"));
    try {
        for (int i=vs.getNumElementos(); i>=0; i++)
            std::cout << vs[i] << endl;
    } catch (ExcepcionVectorRango &ex) {
        std::cerr << ex.what() << "posicion: " << ex.getPos()
                  << ", num. elementos: " << ex.getNumElem() << endl;
    }
}
```

11. Caso: Gestión de líneas de autobuses. (4 puntos)

La empresa “Viaje Feliz” es una empresa de buses que ha crecido bastante en los últimos 20 años (hace servicios en todo el Perú), y desea gestionar su flota de buses. Para eso ha pedido a Julio (un estudiante del curso) que prepare un programa que les ayude en esa gestión. A continuación, se muestra el diagrama de clases realizado por el alumno:



Todas las clases que se van a desarrollar deben estar en el paquete viajefeliz.com.pe y se debe tener en cuenta las siguientes consideraciones:

a) La clase **TransporteException** es una excepción que se utilizará para tratar las distintas situaciones inesperadas o adversas.

b) Información de autobús
   - La clase **Bus** contiene la información correspondiente a un autobús de la línea como son: el código de autobús (String codbus), la línea a la que pertenece (int codLinea), la matrícula del autobús (String matricula) y un conductor (Conductor conductor).
     - El constructor de la clase recibirá el código del bus y Código de línea. Tenga en cuenta que si se recibe un código de línea negativo o un código de bus nulo se lanzará una excepción del tipo **TransporteException**.
     - Se debe encapsular los atributos.
     - Dos autobuses serán considerados iguales si coinciden en el código del autobús y su matrícula.
     - La representación del autobús (Sobrecarga del operador << para `std::ostream`) debe mostrar el código del bus, la línea y la matrícula en el siguiente formato:
       - **Bus(bus126, 25, DHLIQ126)**
   - La clase **Conductor** contiene la información de los conductores de los autobuses, como son: el código del conductor (String codigo), el dni (String dni), el nombre (String nombre), la cantidad de años de manejo (int anhiosManejo) y un listado de todas las capacitaciones a las que ha asistido (vector<String> capacitaciones).
     - El constructor recibirá el código, dni y nombre del conductor. Si alguno de estos valores es nulo se lanzará una excepción del tipo **TransporteException**.
     - Se debe encapsular los atributos.
     - Dos conductores serán considerados iguales si coinciden en el código y el dni.
     - La representación del conductor (Sobrecarga del operador << para `std::ostream`) debe mostrar el código, el dni y el nombre en el siguiente formato:
       - **Conductor(con126, 45444726, Carlos Carlin)**

c) La clase **Servicio** contiene la relación de autobuses de la empresa que se encuentran en la ciudad. Contiene un String que indica el nombre de la ciudad (String ciudad), la relación de autobuses disponibles (vector<Bus> busesDisponibles) y la relación de autobuses que serán seleccionados por haber pasado todos los filtros necesarios (vector<Bus> busesFiltrados).
   - El constructor recibirá el nombre de la ciudad del servicio y la relación de buses disponibles.
   - Esta clase posee el método sobrecargado **filtra** que se comporta de la siguiente manera:
     - **public vector<Bus> filtra(Criterio criterio),** que filtra los buses disponibles (**busesDisponibles**) en función al criterio indicado.
     - **public vector<Bus> filtra(Criterio criterio, List<Bus> busesAFiltrar)** que filtra la relación de buses pasada como segundo argumento (**busesAFiltrar**), en función al criterio indicado.
     - Se debe encapsular los atributos.
     - Se debe validar que los parámetros ingresados criterio y busesAFiltrar (cuando corresponda) no sean nulos, arrojando **TransporteException** cuando sea necesario.
   - Adapte la Sobrecarga del operador "<<"para que devuelva el nombre de la ciudad y la relación de buses disponibles.

d) La interfaz "**Criterio"** especifica un método **boolean busEsSeleccionable(Bus bus)** que sirve para conocer si el bus **b** cumple con las restricciones establecidas por la empresa para dar servicio.

e) Se va a implementar la lógica de la interfaz Criterio:

- La clase **FiltroLinea** implementa la interfaz **Criterio**. Tiene una variable de instancia llamada **codLinea (int)**. Dispondrá de los métodos siguientes:
  - Un constructor al que se le pasará como argumento un valor entero con el código de la línea al que deben pertenecer las líneas seleccionadas. Tenga en cuenta que si se recibe un código negativo se lanzará una excepción del tipo **TransporteException.**
  - El método **boolean busEsSeleccionable(Bus bus)** que valide que el bus seleccionable sea aquél cuyo código de línea coincida con el entero pasado en el constructor.
  - Redefine la Sobrecarga del operador "<<" para que muestre:
    Autobuses de la línea [xxx]

- La clase **FiltroMatricula** implementa la interfaz **Criterio**. Tiene una variable de instancia llamada **cadenaBuscar (String)**. Dispondrá de los métodos siguientes:
  - Un constructor al que se le pasará como argumento una cadena de caracteres (String). Tenga en cuenta que, si se recibe cadena nula, se lanzará una excepción del tipo **TransporteException.**
  - El método **boolean busEsSeleccionable(Bus bus)** que valide que el bus seleccionable sea aquél cuya matrícula contenga la cadena brindada en el constructor.
  - Redefine la Sobrecarga del operador "<<" para que muestre:
    Autobuses cuya matricula contiene [xxx]

- La clase **FiltroExpManejo** implementa la interfaz **Criterio**. Tiene una variable de instancia llamada **minAnhios (int).** Dispondrá de los métodos siguientes:
  - Un constructor al que se le pasará como argumento un valor entero que indica la cantidad mínima de años que debe tener un conductor de autobús para ser considerado apto. Tenga en cuenta que si se recibe un valor negativo se lanzará una excepción del tipo **TransporteException.**
  - El método **boolean busEsSeleccionable(Bus bus)** que valide *(1)* que el conductor del autobús supera la cantidad mínima de años requeridos, o *(2)* que si no supera la cantidad mínima requerida, que la cantidad de años faltantes sea menor que la cantidad de capacitaciones recibidas (cada capacitación es equivalente a un año de experiencia).

f) Ahora, cree una clase ejecutable (**Ejecuta**) que haga lo siguiente:

- Genere la programación de un servicio (**Servicio**) al que se le pase el nombre de la ciudad y una relación de "N" autobuses disponibles (N > 5). Cada autobús debe contener sus datos básicos, así como la información de sus conductores (información básica y la relación de capacitaciones realizadas). Estos valores serán seteados en duro (no es necesario pedirlos por teclado)
- Realice el filtrado de los buses disponibles del servicio en función a los 3 criterios implementados (los **busesFiltrados** serán aquellos que pasen los 3 criterios a la vez**).** A continuación, imprima toda la relación de buses filtrado: bus, conductor y las capacitaciones del conductor ordenadas alfabéticamente.
- La salida será parecida a la que se muestra a continuación:

imprimiendo los datos del servicio

Servicio{ciudad='Lima', buses=[Bus(bus121, 25, LIAQP121), Bus(bus122, 25, MOBPJ122), Bus(bus123, 25, DHLIQ123), Bus(bus124, 27, DHQLI124), Bus(bus125, 25, DHEAQ123), Bus(bus126, 25, DHLIQ126)]}

imprimiendo los buses filtrados, sus conductores y sus capacitaciones

Bus

Bus(bus121, 25, LIAQP121)

Conductor

Conductor(con121, 45444721, Julio Perez)

capacitaciones

[Actualizacion de Reglas, Concientizacion]

Bus

Bus(bus126, 25, DHLIQ126)

Conductor

Conductor(con126, 45444726, Carlos Carlin)

capacitaciones

[Actualizacion de Reglas, Civismo, Concientizacion, Manejo de ira, Mecanica basica, Respeto de normas]

Consideraciones:

• La tarea es grupal (grupo de máximo 5 personas)

• La fecha límite de entrega es el sábado 21/12/2024 a las 23:59 horas.

• Se debe subir un archivo zip que contenga:

- La solución a cada problema en c++ (sólo los archivos .cpp, no son necesarios los .exe). Cada pregunta debe seguir la nomenclatura: **probleman.cpp**, es decir, el primer programa será **problema1.cpp**, el segundo será **problema2.cpp**, etc. (para los problemas del 1 al 9)
- Puede usar un directorio con varios archivos para los problemas 10 y 11.
- Acompañe un informe en Word explicando las preguntas teóricas que se hacen o para complementar la explicación de las soluciones que implementó.