

Metodología de la Programación II

21 Junio 2004.

1. (2.5 puntos) Construir una función recursiva que calcule la descomposición factorial de un número entero y la guarde en un vector de enteros. La cabecera de la función debe ser:

```
void descomposicion (int n, int * factores, int& num_factores)
```

Por ejemplo, para calcular la descomposición factorial de 360 ($360 = 2^3 \times 3^2 \times 5$) se hará:

```
int v[100]; // 100 es un valor suficientemente grande
int n;
...
n = 0;
descomposicion (360, v, n);
```

y el resultado será:

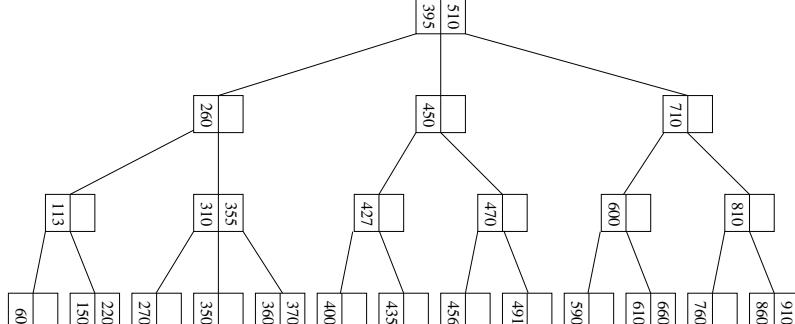
```
n = 6
v = {2,2,2,3,3,5,?,...}
```

2. Dada la siguiente clase para el tipo de dato *Matriz* (declarada en *matriz.h*):

```
class Matriz {
    double * datos;
    int filas, columnas;
public:
    Matriz();
    Matriz(const Matriz & m);
    Matriz & operator = (const Matriz & m);
    void Set (int f, int c, double d);
    double Get (int f, int c) const;
    int Leer (const char * nombre);
    int Escribir (const char * nombre) const;
};
```

- (2 puntos) Escribir la implementación del operador de asignación.
 - (2 puntos) Un objeto *Matriz* se almacena en un fichero, de forma que la primera línea contiene el número de filas y columnas (*n,separador,m,salto de línea*), seguida de los los $n*m$ valores de la matriz, almacenados por filas y en binario. Escribir la implementación de la función “*Leer*”, teniendo en cuenta que devuelve un cero en caso de éxito y un valor distinto de cero en caso de error.
 - (2 puntos) Para cada elemento a_{ij} de la matriz, se define su residuo r_{ij} como:
$$r_{ij} = d_{IJ} + a_{ij} - d_{Ij} - d_{iJ}$$
 donde $d_{IJ} = \frac{\sum_{j=1}^m \sum_{i=1}^n a_{ij}}{n*m}$ $d_{Ij} = \frac{\sum_{i=1}^n a_{ij}}{n}$ $d_{iJ} = \frac{\sum_{j=1}^m a_{ij}}{m}$
Se define el residuo de la matriz como $r_{IJ} = \frac{\sum_{j=1}^m \sum_{i=1}^n \|r_{ij}\|}{n*m}$.
Escribir un programa, que usando la clase *Matriz*, lea una matriz desde un fichero y muestre en la salida estándar el resultado del cálculo del residuo de la matriz.
3. Partiendo del árbol B que indicamos en la figura siguiente, se trata de:
 - (0.5 puntos) Insertar los valores 365, 275, 364, 272, 352.
 - (1 puntos) A partir del árbol resultado anterior, borrar los valores 860, 810 y 600.

indicando cada uno de los pasos que se han seguido.



Duración del examen: 2 horas y media.

Metodología de la Programación II

15 Septiembre 2004.

1. **(1.5 puntos)** Implemente una función recursiva que escriba en la salida estándar un entero, separando cada grupo de 3 cifras con un punto. Por ejemplo, si el entero es -1234567, deberá escribir -1.234.567.
2. Dada la siguiente clase para el tipo de dato *Conjunto*:

```
class Conjunto {  
    int * elementos; // No repetidos, desordenados  
    int n;           // Número de elementos  
public:  
    ...  
};
```

- a) **(1 puntos)** Implementar el constructor de copias.
 - b) **(1.5 puntos)** Implementar el operador de asignación.
 - c) **(1.5 puntos)** Implementar una función *insertar* que reciba un entero, lo inserte en el conjunto, y devuelva si ha tenido éxito (si el conjunto ha crecido).
3. Considere la siguiente clase para el tipo de dato *Polinomio*:

```
class Polinomio {  
    float * coef; // Pos i: Coeficiente grado i  
    int MaxGrado; // Indica el tamaño de coef  
public:  
    ...  
};
```

- a) **(1 puntos)** Sobrecargar el operador << para la salida de un polinomio a un flujo.
 - b) **(1.5 puntos)** Sobrecargar el operador >> para la entrada de un polinomio desde un flujo.
 - c) **(1.5 puntos)** Escriba el archivo cabecera que contiene la declaración de la clase *Polinomio*, incluyendo las cabeceras de las dos funciones anteriores. No es necesario escribir las funciones públicas que no se han mostrado aquí (indíquelas, igualmente, con los 3 puntos suspensivos).
4. Un fichero de descripciones almacena las distintas marcas (cadenas de caracteres) que identifican los distintos tipos de archivos (por ejemplo, un archivo *PGM* tiene en la posición cero los caracteres *P5*). El formato de este archivo es el siguiente:

Entero	Entero	Caracteres (var.)	100 caracteres
Posición inicio marca	Longitud de la marca	Marca	Descripción

y un ejemplo de contenido de este archivo es:

0	2	P5	Imagen PGM
0	2	P6	Imagen PPM
10	8	DAT Cien	Datos científicos

donde podemos ver que si un archivo tiene la cadena “DAT Cien” (de 8 caracteres) a partir de la posición 10 del archivo, se trata de un archivo de datos científicos.

- (1.5 puntos)** Escriba una función (*Insertar*) que recibe el nombre de un archivo de descripciones, junto con una nueva entrada (posición, cadena y descripción), y tiene como efecto que se añade la entrada a dicho archivo (creándolo si es necesario).
- (1.5 puntos)** Implementar una función (*TipoArchivo*) que determine el tipo de un fichero. La función recibe el nombre de este archivo junto con el nombre del archivo con las descripciones. Como resultado devuelve una cadena con la descripción asociada, o “*Tipo desconocido*” si no se ha localizado su tipo.

La cabecera de la función deberá ser:

```
string tipo_fichero(string nombre_fichero, string nombre_descripcion)
```

donde *nombre_fichero* es el nombre del fichero de datos y *nombre_descripcion* es el del fichero que proporciona la información para su reconocimiento.

- (1.5 puntos)** Escribir un programa que, usando la función del apartado anterior, reciba en la línea de comandos el nombre de un archivo y escriba en la salida estándar la descripción del archivo. Tenga en cuenta los posibles casos de error.

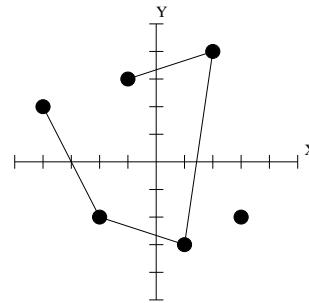
El alumno debe escoger entre las preguntas 2 y 3.
Duración del examen: 2 horas y media.

METODOLOGÍA DE LA PROGRAMACIÓN II (24 Junio 2005)

1. (3 puntos) Vamos a construir un programa para la gestión de gráficos en 2D. Disponemos de una clase **Punto** para trabajar con puntos en 2D y una clase **PoliLinea** que permite representar un trazo en base a una serie de puntos. A continuación tienes la definición de ambas clases (sólo se indica la representación interna de las mismas) y un ejemplo gráfico de un punto en las coordenadas (3,-2) y de una polilínea definida por la lista de puntos (-4,2) - (-2, -2) - (1,-3) - (2,4) - (-1,3):

```
class Punto {
    int x, y;      // Coordenadas de un punto 2D
    ....
};

class PoliLinea {
    Punto *p;      // Vector de puntos
    int num;        // Número de puntos
    ....
};
```



- a) Implementa el constructor de copias de la clase **PoliLinea**.
- b) Implementa la sobrecarga del operador de asignación para la clase **PoliLinea**.
- c) ¿Es necesario implementar el constructor de copias o la sobrecarga del operador de asignación en la clase **Punto** para que todo funcione correctamente? Justifica la respuesta.
- d) Sobrecarga el operador de suma para poder añadir nuevos puntos a una polilínea, de manera que podamos ejecutar operaciones de la forma:
 - 1) punto + polilínea. Se crea una nueva polilínea añadiendo un punto al comienzo de la misma.
 - 2) polilínea + punto. Se crea una nueva polilínea añadiendo un punto al final de la misma.

2. (2 puntos) Considera el siguiente formato para almacenar una polilínea en un fichero de texto:

POLILINEA # Comentario opcional N X1 Y1 X2 Y2 X3 Y3 ... Xn Yn	El fichero siempre comienza por la cadena POLILINEA El comentario es opcional y, en caso de estar: Comienza por el carácter # Sólo ocupa una línea No hay límite de caracteres N es el número de puntos que tiene la polilínea Después de N tenemos la lista de coordenadas de cada punto Cada punto se escribe en una nueva línea y las coordenadas están separadas por un espacio
---	--

- a) Implementa un método para cargar una polilínea desde un fichero a un objeto en memoria:
- ```
void PoliLinea::Leer(const char *nomfich);
```
- b) Implementa un método para escribir una polilínea desde un objeto a un fichero:
- ```
void PoliLinea::Escribir(const char *nomfich, const char *comentario=0);
```

nomfich es el nombre del fichero y comentario es el comentario que hay que escribir (si vale 0 no se escribe nada).

3. (1.5 puntos) Sea a un número positivo y real. Definimos la secuencia de valores reales x_i como:

$$x_i = \begin{cases} 1 & \text{si } i = 0 \\ \frac{1}{2} \left(x_{i-1} + \frac{a}{x_{i-1}} \right) & \text{si } i > 0 \end{cases}$$

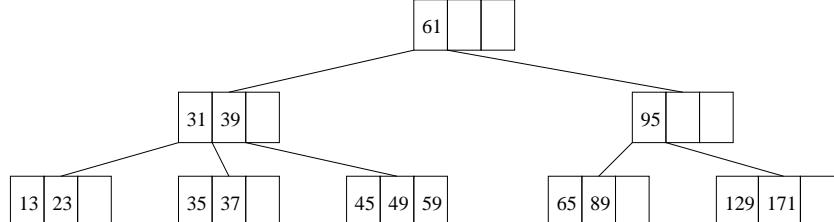
Se demuestra que $x_i \rightarrow \sqrt{a}$ si $i \rightarrow \infty$. Escribe una función recursiva que use este algoritmo para calcular la raíz cuadrada de un número. A esta función debemos darle el número al que le vamos a calcular la raíz junto con el número de términos que deseamos calcular.

4. (2 puntos) Dada la siguiente estructura de celdas enlazadas:

```
struct Celda {
    int num;          // Dato entero
    Celda *sig;       // Puntero a la celda siguiente
};
```

- a) Escribe una función que reciba un puntero a la primera celda de la lista y que devuelva una nueva lista invertida.
- b) Escribe una función que muestre en pantalla los valores de una lista en orden inverso. No se puede hacer una inversión de la lista y después imprimir la lista invertida.

5. (1.5 puntos) Partiendo del árbol B de orden 4 de la siguiente figura, borra (en este orden) los valores: 171, 61, 37, 35, 89 y 129.



Metodología de la Programación II

Examen de teoría. 19 Septiembre 2005.

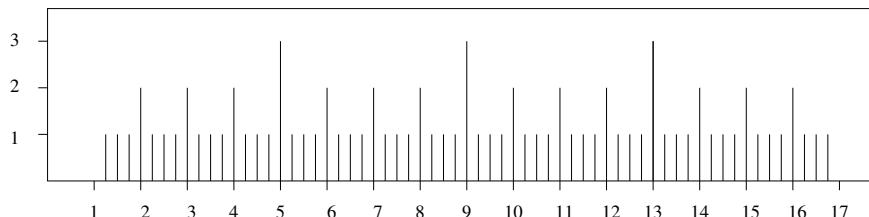
1. (2 puntos) Implemente una función recursiva *Reglar* con la siguiente cabecera:

```
void Reglar (double izq, double der, int min, int max, int n)
```

que dibuja las marcas de una regla. Los parámetros corresponden a:

- *izq,der*: posiciones horizontales donde dibujar las marcas.
- *min,max*: La altura de la marca más pequeña y la más grande.
- *n*: número de marcas de cada altura.

Por ejemplo, si llamamos *Reglar(1,17,1,3,3)* genera el siguiente dibujo:



Observe que el dibujo consiste en realizar tres marcas de altura máxima en el intervalo *[izq,der]* (lo divide en 4 partes) y dibujar una nueva regla con marcas desde altura 2 a 1 en cada uno de los 4 subintervalos. Para realizar el ejercicio, suponga que dispone de la siguiente función:

```
void DibujarMarca (double posicion, int altura);
```

que dibuja una línea vertical de una determinada altura en una posición. Por ejemplo, la línea central del ejemplo anterior se ha dibujado con la llamada *DibujarMarca(9,3)*.

2. (2 puntos) Considere dos secuencias de enteros almacenadas mediante una estructura de celdas enlazadas. Suponiendo que, en ambas, los elementos se encuentran ordenados, implemente una función que mezcle las dos secuencias de elementos en una nueva de forma que las dos originales se queden vacías y la final contiene la mezcla de ellas. Tenga en cuenta que no es necesario, y no se admitirá, ninguna operación de reserva o liberación de memoria.

3. Dada la siguiente clase para el tipo de dato *Conjunto*:

```
struct Celda {  
    int elemento;  
    Celda *sig;  
};  
class Conjunto {  
    Celda * lista; // No repetidos, desordenados  
public:  
    ...  
};
```

- a) (1 punto) Implemente el constructor por defecto y destructor.
- b) (1 punto) Implemente el constructor de copias.
- c) (1 punto) Sobrecargue el operador de asignación.
- d) (1 punto) Sobrecargue el operador “+” para realizar la unión de dos conjuntos.

4. (2 puntos) Considere el tipo *Conjunto* de la pregunta anterior. El formato de un archivo que almacena un conjunto corresponde a un fichero de texto con la siguiente especificación:

- a) La primera línea contiene la cadena mágica CJTMR2 seguida por un salto de línea.
- b) Opcionalmente, puede haber una línea que contiene un carácter '#' seguido por una secuencia de hasta 100 caracteres que finalizan en un salto de línea.
- c) Una línea que contiene un número entero (*N*) indicando el número de elementos del conjunto seguido por un salto de línea.
- d) Una línea que contiene *N* números enteros separados por espacios, y que corresponden a los elementos que contiene el conjunto.

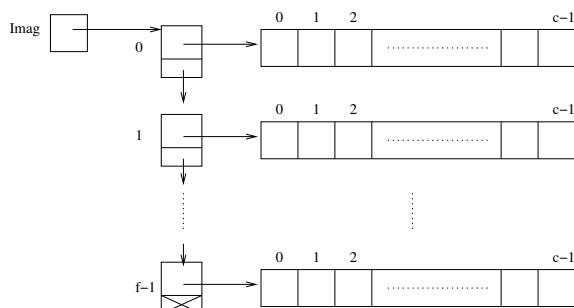
Implemente las funciones miembro “Leer” y “Escribir” para la lectura y escritura de un conjunto en disco. Las funciones reciben como entrada el nombre de un fichero en disco y devuelven un código entero indicando si ha habido errores.

Duración del examen: 2 horas y media.

Metodología de la Programación II

Examen de teoría. 29 Junio 2006.

1. Considere una clase *Imagen*, de f filas por c columnas, que se representa como sigue:



Donde puede observar que la estructura de datos consiste en una lista de f celdas enlazadas en las que se almacenan vectores de c elementos (valores de 0 a 255) para cada una de las filas. Donde f es el número de filas y c el número de columnas.

Existe una celda por cada fila, de forma que toda la estructura cuelga de un único puntero (*Imagen*), que podría ser cero en caso de que la imagen esté vacía.

La última celda contiene un cero en el campo que corresponde al puntero siguiente.

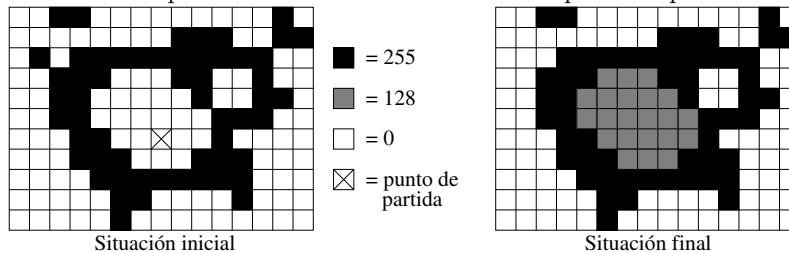
- a) (0.75 punto) Implemente el constructor por defecto (crea una imagen vacía) y el destructor.
b) (0.75 punto) Implemente el constructor de copias.
c) (1 punto) Sobre cargue el operador de asignación.
d) (1 punto) Implemente dos funciones *Set* y *Get* que permitan modificar y acceder al contenido de una posición de la imagen, respectivamente. Las cabeceras deben ser:

```
void Imagen::Set(int f, int c, unsigned char dato);  
unsigned char Imagen::Get(int f, int c) const;
```

- e) (1 punto) Sobre cargue los operadores == y != para poder comprobar la igualdad de dos imágenes.

2. (2 puntos) Considere la parte pública de la clase *Imagen*, en la que se incluyen las funciones *Set* y *Get*. Suponga que disponemos de una imagen en la que únicamente hay puntos con valores 0 y 255. En esa imagen, existe una región vacía (valores 0) rodeada de valores 255. Escriba una función recursiva para que, a partir de una posición cualquiera situada dentro de dicha región vacía, rellene todas las posiciones de dicha región con el valor 128.

En la siguiente figura se puede ver un ejemplo con una imagen inicial en la que hay una región vacía delimitada por puntos con valor 255. En la parte derecha vemos el resultado que debe producir el algoritmo.



3. (2 puntos) Se dispone de ficheros de texto que contienen un número indeterminado de líneas, cada una de ellas con los datos correspondientes a una serie de valores reales. Un ejemplo es el siguiente:

```
3 3.1 0.0 2.1  
5 1.0 1.0 1.0 1.0 1.0  
2 5.2 4.7
```

donde puede observar que cada línea contiene un valor entero seguido por tantos elementos como indique éste. Escriba un programa que obtenga en la salida estándar los valores que corresponden a la sumatoria de cada fila. Por ejemplo, en el caso anterior, deberá obtener los valores 5.2, 5 y 9.9. La forma de lanzar el programa desde la línea de órdenes debe permitir:

- a) Llamarlo sin ningún argumento. Los datos con las sumas a realizar se leerán desde la entrada estándar.
b) Llamarlo con un argumento. El argumento corresponde al nombre del archivo con los datos a sumar.

4. Escriba dos programas para transformar ficheros con datos de sumatorias (pregunta anterior), para transformar entre formato binario y texto:

- a) (0.75 punto) Un programa que transforme un fichero de texto a binario.
b) (0.75 punto) Un programa que transforme un fichero de binario a texto.

Para ello, los programas recibirán, en la línea de órdenes, dos argumentos con los nombres del fichero de entrada y salida respectivamente. Tenga en cuenta que un fichero en formato binario contiene todos los datos de forma consecutiva de la siguiente forma.

```
<n1(int)><dato_1(float)>...<dato_n1(float)><n2(int)><dato_1(float)>...<dato_n2(float)>...
```

Finalmente, no olvide que debe optimizar el uso de recursos, y por tanto, no se permiten soluciones en las que se tenga, por ejemplo, todo el fichero en memoria.

Duración del examen: 3 horas.

Metodología de la Programación II

Examen de teoría. Septiembre 2006.

1. **(2 puntos)** Escriba una función recursiva que obtenga el número de repeticiones de cada vocal (5 valores enteros) en una cadena de caracteres.
2. **(2 puntos)** Considere la siguiente clase:

```
class Entero {  
    int *v;  
public:  
    Entero (int i=0) { v= new int; *v=i; }  
    ~Entero () { delete v; }  
    int Set(int i) { *v= i; }  
    int Get() const { return *v; }  
};
```

Realice las modificaciones que crea convenientes sobre la clase *Entero* para que el siguiente código funcione correctamente, describiendo brevemente la razón de dichos cambios.

```
Entero Doble(Entero e)  
{ return e.Get()*2; }  
int main()  
{ Entero x(5), y;  
    y= Doble(x);  
    cout << "Resultado: " << y << endl;  
}
```

3. Se desea crear una clase *Menu* para simplificar el desarrollo de programas que usan menús en modo texto. Para ello, se propone la siguiente representación:

```
class Menu {  
    char *titulo; // Encabezado que damos al menú (0 si no existe)  
    char **opc; // Cadenas que describen cada una de las opciones  
    int nopc; // Número de opciones actualmente en el menú  
public:  
    ...  
};
```

- a) **(1.5 puntos)** Escriba la implementación de las funciones miembro correspondientes al constructor por defecto (crea un menú vacío), destructor, constructor de copias y operador de asignación.
- b) **(1.25 puntos)** Escriba tres funciones:
 - *SetTitulo* que asigne un nuevo valor al título.
 - *GetNumeroOpciones* que devuelva el número de opciones que hay actualmente en el menú.
 - *AddOpcion* que reciba una cadena de caracteres y la añada como una nueva opción después de la última.
- c) **(1 puntos)** Sobrecargue los operadores *<<* y *>>* para poder usar el menú para seleccionar una de las opciones. Concretamente:
 - Sobrecargue el operador *<<* para que podamos imprimir el menú. Un ejemplo de llamada podría ser *cout<<menu*, que imprime en la salida estándar el título del menú seguido por cada una de las opciones (numerándolas, del 1 al número de opciones, en líneas distintas).
 - Sobrecargue el operador *>>* para que podamos escoger una nueva opción. La sintaxis de llamada será *menu>>entero*, y provoca la solicitud de una opción (como un número del 1 al número de opciones desde la entrada estándar) de entre las que incluye el menú.
- d) **(0.25 puntos)** Complete el entorno *Class*, que hemos presentado en la pregunta, con la parte pública correspondiente.

4. **(2 puntos)** Escriba un programa *alreves* que recibe en la línea de órdenes el nombre de un fichero, y escribe en la salida estándar el mismo flujo de caracteres, pero en orden inverso. Por ejemplo, si el archivo *abecedario.txt* contiene los caracteres:

```
abcdefghijklmn  
ñopqrstuvwxyz
```

una posible ejecución del programa obtendría lo que sigue:

```
% alreves abecedario.txt  
zyxwvutsrqpoñ  
nmlkjihgfedcba
```

El programa debe optimizar el uso de la memoria, y dado que el tamaño del archivo puede ser muy grande, no se permite cargar todo el archivo de entrada.

Duración del examen: 2 horas y 30 minutos.



Universidad de
Granada

Metodología de la Programación II

11 Junio 2007

Duración: 3 horas



Dpto. Ciencias de la
Computación e I. A.

Apellidos:

Nombre:

DNI:

1.- (1.5 puntos) Se desea construir una aplicación en la que se necesitan los siguientes módulos:

alumno.h

```
#ifndef __ALUMNO_H__
#define __ALUMNO_H__

class Alumno {
    private:
        Fecha fnacimiento;
    ...
};
```

fecha.h

```
#ifndef __FECHA_H__
#define __FECHA_H__

class Fecha {
    ...
};
```

profesor.h

```
#ifndef __PROFESOR_H__
#define __PROFESOR_H__

class Profesor {
    private:
        Fecha fnacimiento;
    ...
};
```

alumno.cpp

```
// Implementación del módulo
...
```

fecha.cpp

```
// Implementación del módulo
...
```

profesor.cpp

```
// Implementación del módulo
...
```

vectoralumno.h

```
#ifndef __VECTORALUMNO_H__
#define __VECTORALUMNO_H__

class VectorAlumno {
    private:
        Alumno *dat;
    ...
};
```

vectoralumno.cpp

```
// Implementación del módulo
...
```

main.cpp

```
// Se usan todas las clases
int main(int argc, char *argv[]) {
    ...
}
```

a) Escribe los #include necesarios en cada fichero (**hazlo en este mismo folio**).

b) Crea el fichero Makefile para compilar cada uno de los módulos descritos así como para crear un programa ejecutable (main) a partir de ellos.

c) Numera cada una de las reglas que has escrito en tu fichero Makefile y responde a las siguientes preguntas. Tras haber generado todo el proyecto ejecutando make (sin errores), considera que modificamos el contenido de algunos ficheros del proyecto. Di, en cada caso, y de manera independiente, qué reglas se aplicarán (escribe la lista con los números de las reglas que se aplican al lado de cada fichero) tras modificar:

- fecha.cpp
- profesor.h
- profesor.cpp

2.- (3.5 puntos) Suponga que tenemos la siguiente clase para almacenar una serie de alumnos en forma de lista enlazada simple:

```
struct Nodo {  
    Alumno a;          // Información del alumno almacenado en el nodo  
    Nodo *sig;         // Puntero al nodo siguiente de la lista  
};  
class ListaAlumno {  
private:  
    Nodo *primero;    // Comienzo de la lista (0 si está vacía)  
    int num;           // Número de nodos de la lista  
public:  
    ListaAlumno();  
    ListaAlumno(const ListaAlumno &orig);  
    ListaAlumno & operator=(const ListaAlumno &orig);  
    ~ListaAlumno();  
    Alumno Get(int pos) const;           // Devuelve el alumno pos-ésimo de la lista  
    void Set(int pos, const Alumno &alu); // Modifica el alumno pos-ésimo de la  
                                         // lista (NO inserta nuevos nodos)  
};
```

Implemente los métodos indicados en la parte pública de la clase. Si lo necesita, puede implementar nuevos métodos privados en la clase. *Tenga en cuenta que la clase Alumno ya dispone de métodos tales como constructor por defecto, constructor de copia, operator= y destructor.*

3.- (1.5 puntos) Necesitamos sobrecargar dos operadores (<< y +) para realizar dos tareas adicionales con objetos de la clase ListaAlumno:

- Con << deseamos obtener un listado de los elementos de la lista en un flujo de salida. Puedes considerar que << ya está sobrecargado para la clase Alumno.
- Con el operador + deseamos poder obtener una nueva lista de alumnos uniendo una lista ya existente con un nuevo alumno. El alumno se añade al final de la lista.

Implementa ambos operadores para esta clase indicando, en cada caso, si se pueden o deben implementar como miembros o como funciones externas a la misma. Se tendrá en cuenta la eficiencia de estas implementaciones.

4.- (1.5 puntos) Implementa una función recursiva que reciba un número entero en base 10 y lo escriba por consola en una base N cualquiera ($2 \leq N \leq 16$).

5.- (2 puntos) Disponemos de un fichero que almacena múltiples listas de números enteros. Cada lista consiste en un valor N seguido de N valores enteros (que son los elementos de la lista). El formato del fichero **binario** que las almacena es el siguiente:

K N1 <N1 números enteros> N2 <N2 números enteros> ... N_K <N_K números enteros>

Por ejemplo, las siguientes listas: (2,5,4), (8,9), (6,5,7,4,5,2) se almacenarían como:

3 3 2 5 4 2 8 9 6 6 5 7 4 5 2

Observa que el número subrayado es el número de listas y los números en negrita indican el número de elementos de cada lista. En el fichero no se almacenan separadores de ningún tipo (espacios, retornos de línea, etc.).

Escribe un programa que reciba por la línea de órdenes un nombre de fichero y un número entero N. Este programa debe abrir el fichero (que tiene el formato explicado) y mostrar la lista N-ésima de números (la numeración comienza en 1) en consola (cout). Además, si el programa recibe un tercer parámetro, lo interpretará como que el usuario desea escribir el resultado en ese fichero en lugar de en cout. En ambos casos la salida es **con formato** (texto ASCII).

Por ejemplo, si el programa se llama **mostrar** y los datos están almacenados en un fichero llamado **datos.bin**, la siguiente ejecución muestra en cout la segunda lista:

```
> mostrar datos.bin 2  
2  
8 9
```

Metodología de la Programación II

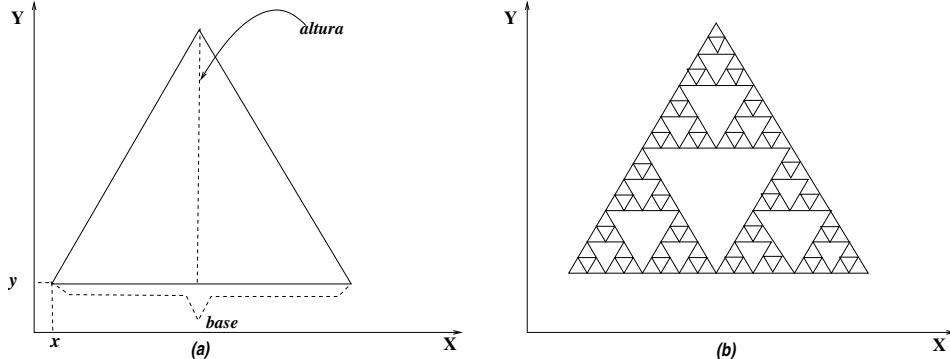
Examen de teoría. 11 de Septiembre de 2007

1. (2 puntos) Implemente una función recursiva Fractal_Triangulo con la siguiente cabecera:

```
void Fractal_Triangulo(double x, double y, double b, double a, int n)
```

que dibuje repetidamente n veces un triángulo de la siguiente forma:

- (a) Dibuja un triángulo cuya base se inicia en una posición (x,y), con longitud **b** y con altura **a**. Ver figura (a).
(b) Dibuja en cada esquina un nuevo triángulo de base **b/2** y altura **a/2** internos al triángulo dibujado en el punto (a).



Este proceso se repite n veces. En la figura (b) se puede observar el resultado tras aplicar el proceso descrito para n igual a 4. Para realizar el ejercicio, suponga que dispone de la siguiente función:

```
void DibujarTriangulo(double x, double y, double b, double a)
```

que dibuja un triángulo de altura a, base b cuya esquina inferior izquierda es (x,y).

2. (5 puntos) Dadas las siguientes clases:

```
class Cadena{  
    char *cad; // termina en '\0'  
public:  
    Cadena();  
    Cadena(const Cadena & C);  
    Cadena & operator=(const Cadena &C);  
    ~Cadena();  
    bool operator ==(const Cadena &C) const;  
    bool operator !=(const Cadena &C) const;  
    ...  
};  
  
class Conjunto{  
    Cadena *Palabras; // desordenadas, sin repetidas  
    int numero; // número de palabras  
public:  
    Conjunto();  
    Conjunto(const Conjunto & D);  
    Conjunto & operator=(const Conjunto &D);  
    ~Conjunto();  
    Conjunto operator+(const Cadena &C) const;  
    ...  
};
```

- (a) (1.5 puntos) Implemente el constructor por defecto, constructor de copias, operador de asignación y destructor para la clase Cadena.
(b) (1 punto) Sobrecargue los operadores relacionales “==” y “!=” para la clase Cadena.
(c) (1.5 puntos) Implemente el constructor por defecto, constructor de copias, operador de asignación y destructor para la clase Conjunto.
(d) (1 punto) Sobrecargue el operador “+” para la clase Conjunto. Este operador crea un nuevo conjunto a partir de un conjunto y una cadena.

3. (3 puntos) Considere los tipos Cadena y Conjunto de la pregunta anterior. El formato de un archivo que almacena un Conjunto corresponde a un fichero **binario** con la siguiente especificación:

- (a) Un entero que indica el número de palabras del conjunto.
(b) Por cada palabra (que son del tipo Cadena) se almacena:
 i. Un entero que indica el número de caracteres de la palabra.
 ii. A continuación tantos caracteres como indica el entero anterior.
• (1.5 puntos) Sobrecargue los operadores “>>” y “<<” para leer y escribir datos de tipo Cadena de/a un flujo de acuerdo a las indicaciones anteriores.
• (1.5 puntos) Implemente las funciones miembro “Leer” y “Escribir” para la lectura y escritura de un Conjunto en un fichero. Se deben usar los operadores “>>” y “<<” implementados para la clase Cadena.

NOTA: Recuerde que el fichero es binario.

Duración del examen: 2 horas y media.

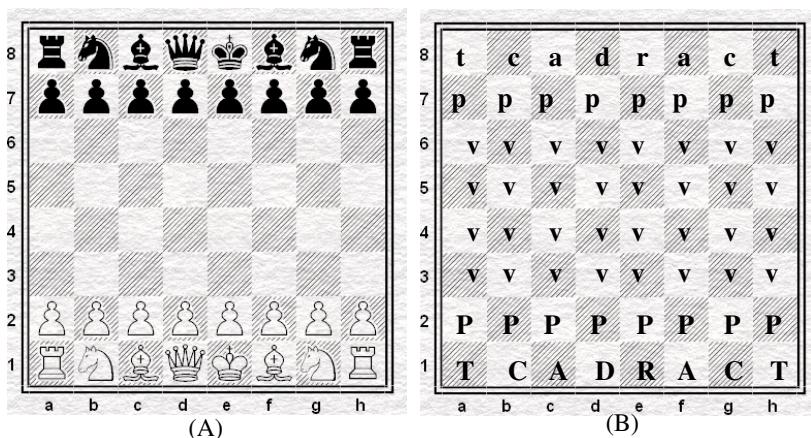
Metodología de la Programación II

Examen de teoría. 16 de Junio de 2008

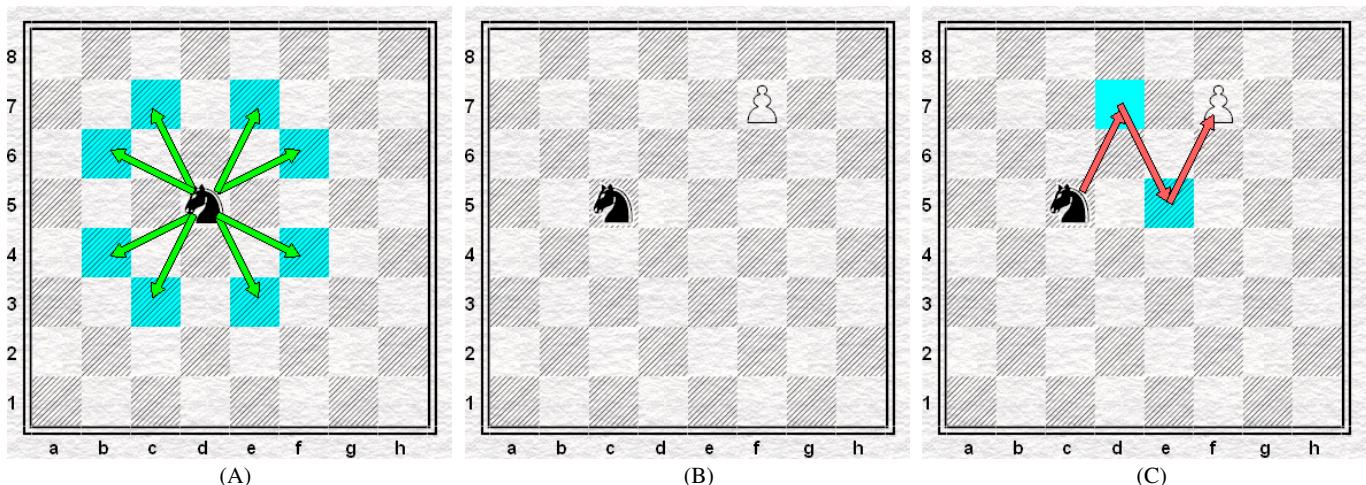
1. Suponga la clase tablero de ajedrez, con la siguiente representación:

```
struct Celda{  
    char pieza;  
    Celda *sig;  
};  
  
class Tablero{  
    char **tablero;//situación del tablero  
    Celda *perdidas;//colección de piezas perdidas, tanto blancas como negras  
    bool turnoblanclas;//true si le corresponde jugar a las blancas, false si le corresponde jugar a las negras.  
....  
};
```

La posición inicial figura (A) se codifica como caracteres según la figura (B):



- (a) **(1 punto)** Implemente el constructor por defecto (inicializa el tablero según la figura anterior) y el destructor.
 - (b) **(1.5 puntos)** Implemente el constructor de copia y el operador de asignación.
 - (c) **(1 punto)** Supongamos que tenemos implementada la función `bool Tablero::EsPosible (char co, int fo,char cd,int fd) const`, que devuelve `true` si es posible mover la pieza en posición `co-fo` a `cd-fd` (haya o no captura de una pieza contraria) y `false` en caso contrario. Implemente la función:
`bool Tablero::Mover(char co, int fo,char cd,int fd)`
que dado un movimiento, lo realiza en caso de que sea posible, y devuelve si se ha podido realizar con éxito. **NOTA:** Observe que se pueden modificar todos los campos de la clase. Recuerde que la función `EsPosible` está implementada.
 - (d) **(1 punto)** Sobrecrega el operador `+` tal que dado un Tablero y una cadena de caracteres, devuelva un nuevo tablero con el resultado del movimiento correspondiente o el mismo tablero en caso de error. La cadena de caracteres debe contener cuatro caracteres que indican la posición origen y final del movimiento. Por ejemplo, “a1d4” codifica el movimiento desde la posición de la casilla a1 a la d4. **NOTA:** Se pueden usar las funciones de los apartados anteriores.
2. **(2 puntos)** Implemente una función recursiva tal que, dado un tablero de ajedrez con un caballo y un peón, imprima en pantalla los movimientos que puede hacer el caballo para comerse al peón. Los movimientos posibles del caballo se pueden ver en la imagen (A). Así, se puede observar que, suponiendo que el caballo está en la casilla d5, éste puede moverse a las casillas c3, b4, b6, c7, e7, f6, f4 y e3.



En la figura (B) se puede observar un caballo en la posición c5 y un peón en la posición f7. Una posible secuencia de movimientos para alcanzar el caballo al peón sería f7-e5-d7-c5 (en sentido inverso, es decir c5-d7-e5-f7) como se muestra en la figura (C). La función a implementar tiene como cabecera:

```
bool Movimientos(char **tablero,char columC,int filaC,char columP,int filaP)
```

donde:

- tablero: es una matriz de tipo *char* en la que si la casilla está vacía tendrá un valor de 'v'. Si el caballo ha pasado por una casilla será marcada con 'o'. Inicialmente el tablero tiene un valor 'v' en todas sus casillas.
- columC,filaC: indican la posición del caballo.
- columP, filaP: indican la posición del peón.

La función devuelve true si ha encontrado el camino (además de imprimir en la salida estándar el camino) o false en caso contrario.

3. (2 puntos) Sobrecargue los operadores “>>” y “<<” para la entrada/salida de un objeto de la clase Tablero desde/hacia un flujo. Tenga en cuenta que el formato será el siguiente:

AJEDREZMP2
tcarvact
pppvvppp
vvvvvvvv
vvvvPvvv
vvvvpvvv
vvvvvvvv
PPPvvPPP
TCAvRACT
DdPp
1

El fichero siempre comienza por esta cadena

El estado del tablero en 8 líneas independientes

Lista de piezas perdidas

Turno. 1:blancas 0:negras

4. (1.5 puntos) Escriba un programa que lea desde un fichero -de tipo texto- una secuencia de movimientos, desde el inicio de una partida, y la reproduzca en la salida estándar. El programa leerá los movimientos y, para cada uno, mostrará el estado del tablero después de realizarlo. El programa acaba cuando no hay más movimientos o alguno de ellos no es válido. La llamada será:

reproducir <nombre del archivo>

NOTA: Para resolverlo puede usar las funciones que aparecen en las preguntas anteriores.

Duración del examen: 2 horas y 30 minutos

Metodología de la Programación II

Examen de Teoría. 16 de Septiembre de 2008

1. Suponer la clase **ColeccionCoches** con la siguiente representación:

```
struct Coche {  
    char* Matricula;  
    char* Marca;  
    char* Modelo;  
    float Km;  
    int Precio;  
    char* observaciones;  
}  
Class ColeccionCoches {  
    int NumCoches;  
    Coche* Coches;  
};
```

que mantiene los coches de un negocio de compra-venta de vehículos.

- a) **(1 punto)** Implemente el constructor por defecto y el destructor.
b) **(1.5 puntos)** Implemente el constructor de copia y el operador de asignación.
 2. Diariamente se carga la colección en memoria desde un fichero al inicio de la jornada y se guarda la colección en un fichero al finalizar la jornada. Implemente los métodos:
a) **(2 puntos)** bool LeeColeccion (const char* fichero);
b) **(1.5 puntos)** bool GuardaColeccion (const char* fichero) const;
- para efectuar estas tareas. Devuelven `true` si la lectura y escritura, respectivamente, se realizan correctamente.
- En el fichero (de texto), los datos de cada coche se almacenan de forma consecutiva en el orden que se enumera en la pregunta 1 (un dato por línea, excepto las observaciones, que pueden ocupar cero o más líneas). El número de observaciones por coche es ilimitado (incluso puede no haberlas), aunque cada una de ellas tendrá un límite de 100 caracteres. Al finalizar un bloque con los datos de cada coche se encuentra una línea con cinco caracteres '*'.
3. **(1.5 puntos)** Sobrecargar el operador `-` para que, dada una colección y el nombre de un fichero -que contiene los vehículos vendidos y que tiene el formato comentado anteriormente- devuelva una **nueva** colección, resultante de eliminar los coches cuyos datos están almacenados en el fichero (bastará con comprobar que las matrículas coinciden).
 4. **(2.5 puntos)** Construir una función recursiva que calcule la descomposición factorial de un número entero y la guarde en un vector de enteros. La cabecera de la función debe ser:

```
void descomposicion (int n, int * factores, int & num_factores);
```

Por ejemplo, para calcular la descomposición factorial de 360 ($2^3 \times 3^2 \times 5$) se hará:

```
int v[100];  
int n;  
.....  
n = 0;  
descomposicion (360, v, n);
```

y el resultado será:

```
n= 6;  
v = {2, 2, 2, 3, 3, 5, ?, ?, ...}
```

Metodología de la Programación II

Examen de teoría. 22 Junio 2009.

1. (2 puntos) Desarrolle un programa para pasar un número de base 10 a una nueva base. El programa recibe dos parámetros en la línea de órdenes:

- a) Un número entero que indica la nueva base. Considere que los caracteres a usar para las bases superiores a 10 son las letras 'A', 'B', etc. Podemos suponer que el número no va a ser superior a 16, es decir, tendremos suficientes letras para representar cualquier dígito en la nueva base.
- b) El número entero a transformar, que está escrito en base 10.

Para resolverlo, deberá crear una función **recursiva** para que obtenga en una cadena de caracteres (terminada en '\0') la nueva representación. El programa escribirá en la salida estándar en resultado obtenido en esta cadena.

2. Se desea crear una clase *Menu* para simplificar el desarrollo de programas que usan menús en modo texto. Para ello, se propone la siguiente representación:

```
class Menu {  
    char *titulo; // Encabezado que damos al menú (0 si menú vacío)  
    char **opc;   // Cadenas que describen cada una de las opciones (0 si menú vacío)  
    int nopc;     // Número de opciones actualmente en el menú (0 si menú vacío)  
public:  
    ...  
};
```

- a) (1.5 puntos) Escriba la implementación de las funciones miembro correspondientes al constructor por defecto (crea un menú vacío), destructor, constructor de copias y operador de asignación.
 - b) Escriba tres funciones:
 - (0.25 puntos) *SetTitulo* que asigne un nuevo valor al título.
 - (0.25 puntos) *GetNumeroOpciones* que devuelva el número de opciones que hay actualmente en el menú.
 - (0.75 puntos) Sobrecargue el operador '+' de forma que se pueda añadir una nueva opción. Esta función devolverá, dado un menú y una cadena de caracteres, un nuevo menú que tiene las mismas opciones que el de entrada, más una nueva situada después de la última.
 - c) (0.5 puntos) Sobrecargue el operador << para que podamos imprimir el menú. Un ejemplo de llamada podría ser *cout*<<*menu*, que imprime en la salida estándar el título del menú seguido por cada una de las opciones (numerándolas, del 1 al número de opciones, en líneas distintas).
 - d) (0.75 puntos) Sobrecargue el operador >> para que podamos cargar un menú desde un flujo de entrada. Un ejemplo de llamada será *cin*>>*menu*, que provoca la lectura del menú desde la entrada estándar. Tenga en cuenta que el menú se dará como una línea del título, seguida por una línea con el número de opciones, y finalizando con tantas líneas como indique dicho número (una para cada opción).
 - e) (0.5 puntos) Escriba el fichero de cabecera (*menu.h*) correspondiente a la clase *Menu*. Para ello, incluya todas las funciones que hayan aparecido en los apartados anteriores.
3. (2 puntos) Se dispone de ficheros de texto que contienen un número indeterminado de líneas, cada una de ellas con los datos correspondientes a una serie de valores reales. Un ejemplo es el siguiente:

```
3 3.1 0.0 2.1  
5 1.0 1.0 1.0 1.0 1.0  
2 5.2 4.7
```

donde puede observar que cada línea contiene un valor entero seguido por tantos elementos como indique éste. Escriba un programa que obtenga en la salida estándar los valores que corresponden a la sumatoria de cada fila. Por ejemplo, en el caso anterior, deberá obtener los valores 5.2, 5 y 9.9. La forma de lanzar el programa desde la línea de órdenes debe permitir:

- a) Llamarlo sin ningún argumento. Los datos con las sumas a realizar se leerán desde la entrada estándar.
 - b) Llamarlo con un argumento. El argumento corresponde al nombre del archivo con los datos a sumar.
4. Escriba dos programas para transformar ficheros con datos de sumatorias (pregunta anterior), concretamente:

- a) (0.75 punto) Un programa que transforme un fichero de texto a binario.
- b) (0.75 punto) Un programa que transforme un fichero de binario a texto.

Para ello, los programas recibirán, en la línea de órdenes, dos argumentos con los nombres del fichero de entrada y salida respectivamente. Tenga en cuenta que un fichero en formato binario contiene todos los datos de forma consecutiva de la siguiente forma.

```
<n1(int)><dato_1(float)>...<dato_n1(float)><n2(int)><dato_1(float)>...<dato_n2(float)>...
```

Finalmente, no olvide que debe optimizar el uso de recursos, y por tanto, no se permiten soluciones en las que se tenga, por ejemplo, todo el fichero en memoria.

Duración del examen: 2 horas y 30 minutos.

Metodología de la Programación II

Examen de Teoría. 2 de Septiembre de 2009

1. Suponer la clase **Inventario** con la siguiente representación:

```
struct Libro {  
    char* Titulo;  
    char* Autor;  
    char* Editorial;  
    char* ISBN;  
    int NumPags;  
    float PrecioCompra;  
    float PrecioVenta;  
    char* observaciones;  
}  
  
Class Inventario {  
    int NumLibros;  
    Libro* Libros;  
};
```

que mantiene los libros de un negocio de compra-venta de libros usados.

- a) **(0.5 punto)** Implemente el constructor por defecto y el destructor.
- b) **(1.5 puntos)** Implemente el constructor de copia y el operador de asignación.

2. Diariamente se carga la colección en memoria desde un fichero al inicio de la jornada y se guarda la colección en un fichero al finalizar la jornada. Implemente los métodos:

- a) **(1.5 puntos)** `bool LeeInventario (const char* fichero);`
- b) **(1.5 puntos)** `bool GuardaInventario (const char* fichero) const;`

Ambos métodos devuelven `true` si la lectura y escritura, respectivamente, se realizan correctamente.

En el fichero (de texto), los datos de cada libro se almacenan de forma consecutiva en el orden que se enumera en la pregunta 1 (un dato por línea, excepto las observaciones). En cuanto a las observaciones, su número es *ilimitado* (incluso puede no haberlas), aunque cada una de ellas tendrá un límite de 100 caracteres. Al finalizar un bloque con los datos de cada libro se encuentra una línea con cinco caracteres '*'.

3. **(1 punto)** Sobrecargar el operador `+` para que, dado un inventario y un dato de tipo `Libro` devuelva un nuevo inventario, resultado de añadir el libro al inventario.
4. **(1.5 puntos)** Sobrecargar el operador `-` para que, dado un inventario y el nombre de un fichero -que contiene datos de libros y que tiene el formato comentado anteriormente- devuelva un nuevo inventario, resultante de eliminar los libros cuyos datos están almacenados en el fichero (bastará con comprobar que el ISBN coincide).
5. **(2.5 puntos)** El método de **bisección** usado para calcular el punto de corte de una función con el eje de abscisas se puede enunciar como sigue:

Sea $f(x)$ una función real, estrictamente monótona en $[i,d]$, donde $f(i)$ y $f(d)$ tienen distinto signo.

- 1) Calcular m , el punto medio entre i y d ,
- 2) Si $f(m)=0$, terminar.
- 3) Si $f(m)$ tiene igual signo que $f(i)$, repetir considerando el intervalo $[m,d]$
- 4) Si $f(m)$ tiene igual signo que $f(d)$, repetir considerando el intervalo $[i,m]$

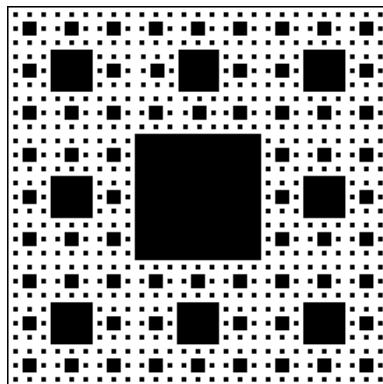
Como es difícil que $f(m)$ sea exactamente cero se considerará que se alcanza una solución válida cuando $f(m)$ sea muy cercano a cero, o sea, si $|f(i) - f(d)| < \varepsilon$, donde ε es una constante.

Implemente una función **recursiva** que reciba como datos de entrada los extremos del intervalo y que devuelva la raíz de la función. Se supone que se dispone de la función $f(x)$ ya implementada en C++.

Metodología de la Programación II

Examen de teoría. 29 de Junio de 2010

1. (2 puntos) Implemente una función recursiva para dibujar, en dos dimensiones, el fractal conocido como “esponja de Menger”. La apariencia de dicho dibujo la podemos ver en esta figura:



El prototipo de la función recursiva debe ser el siguiente:`void Fractal_Menger(double x, double y, double tam)` siendo `(x,y)` las coordenadas de la esquina superior izquierda y `tam` el ancho del cuadrado.

Nota: podemos suponer que disponemos de la función `DibujaCuadrado(x1,y1,tam)`, que dibuja un cuadrado relleno de color negro, cuya esquina superior izquierda viene dada por la coordenada `(x1,y1)` y cuyo ancho es `tam`.

2. (4.5 puntos) Dadas las siguientes clases:

```
class Matriz {  
    int nf, nc; // Número de filas y columnas  
    unsigned char **m;  
    public:  
        ...  
};  
  
class Imagen {  
    Matriz im;  
    public:  
        ...  
};  
  
class Video {  
    int n; // Número de imágenes  
    Imagen *v;  
    public:  
        ...  
};
```

- (a) (1.5 puntos) Implemente el constructor por defecto, el constructor de copia, el operador de asignación y el destructor para la clase `Matriz`.
(b) (0.5 punto) ¿Es necesario implementar el operador de asignación para la clase `Imagen`? Razone la respuesta.
(c) (1 puntos) Implemente el operador de asignación y el constructor de copia para la clase `Video`.
(d) (0.5 punto) Suponga que la clase `Video` no dispone de un constructor de copia implementado. Escriba un ejemplo (código en C++) en el que esta situación llevaría a un error en tiempo de ejecución y razone el motivo de dicho error.
(e) (1 punto) Sobrecargue el operador “+” para la clase `Video`. Este operador crea un nuevo video a partir de un video y una imagen, añadiendo la imagen al final de dicho video.

3. (1.5 puntos) Considere el tipo `Matriz` de la pregunta anterior. Deseamos añadir al módulo funcionalidad para poder escribir en ficheros y leer desde ficheros datos de este tipo. La especificación de dicho fichero es la siguiente:

- (a) Tiene formato binario.
(b) El contenido del fichero es el siguiente:
 i. Dos enteros que indican el número de filas y de columnas de la matriz.
 ii. A continuación vienen todos los elementos de la matriz.

Implemente las funciones miembro “Leer” y “Escribir” para la lectura y escritura de una `Matriz` en un fichero.

4. (2 puntos) Desarrolle un programa para calcular la 8-paridad de un fichero. Definimos 8-paridad al octeto de bits que corresponde al control de paridad de una secuencia de bytes, y se puede calcular con una operación OR exclusivo byte a byte. Es decir, si tenemos un fichero compuesto de los bytes b_1, b_2, \dots, b_n , la 8-paridad se define como el byte resultante de calcular $b_1 \wedge b_2 \wedge \dots \wedge b_n$.

El programa podrá llamarse de dos formas:

- (a) Con un argumento que corresponde al nombre del fichero al que calcular la 8-paridad.
(b) Sin argumentos, para que calcule la 8-paridad de todos los bytes que se lean desde la entrada estándar.

Ejemplos de su ejecución son:

```
prompt> paridad fichero.dat  
00101010  
prompt> paridad  
Lorem ipsum ad his scripta blandit partiendo, eum  
fastidii accumsan euripidis in, eum liber hendrerit an.  
Ctrl+D  
10111001
```

donde podemos ver que hemos obtenido un octeto de bits que corresponde a la representación binaria del byte obtenido como resultado de las operaciones OR exclusivo.

Nota: Consideraremos que un byte corresponde al tamaño de un char.

Duración del examen: 2 horas.



Universidad de
Granada

Metodología de la Programación II

15 Septiembre 2010

Duración: 2:15 horas



Dpto. Ciencias de la
Computación e I. A.

1.- (3.5 puntos) Deseamos crear un software que nos ayude a gestionar un parking de vehículos. Para ello debemos implementar los siguientes módulos:

struct Fecha { unsigned char dia, mes; unsigned short int anyo; unsigned char hor, min, seg; };	class Coche { private: char * matricula; Fecha hora_entrada; };	class Parking { private: Coche *coc; int ncoches; int nplazas; };
---	---	--

En la clase **Parking**, **ncoches** contiene el número de coches que hay en el interior del parking (initialmente cero). El atributo **nplazas** indica el número de plazas libres que hay en el parking.

Debes implementar los siguientes métodos básicos:

- Constructor con parámetros para la clase **Coche** que permita crear un objeto de ese tipo a partir de una matrícula (char*) y una hora de llegada al parking (Fecha).
- Destructor, constructor de copia y operador de asignación para la clase **Coche**.
- Constructor con parámetros para la clase **Parking** que permita crear un objeto de ese tipo dándole el número de plazas totales del parking.
- Destructor, constructor de copia y operador de asignación para la clase **Parking**.

2.- (1.25 puntos) Cuando llega un nuevo coche al parking, una cámara reconoce su matrícula y, a continuación, se procede a su registro. Para ello debes implementar el siguiente método:

```
bool Parking::EntraCoche(const char * matricula, const Fecha &horaentrada);
```

Dicho método debe almacenar los datos del nuevo coche en el objeto de tipo **Parking** siempre y cuando haya plazas disponibles. Si la operación ha tenido éxito devolverá **true** y si no (por ejemplo, en el caso de que no haya plazas) devolverá **false**.

3.- (1.25 puntos) Al salir un coche del **Parking**, debemos calcular el tiempo que ha pasado dentro y borrarlo de nuestro listado. Para ello debes implementar el siguiente método:

```
int Parking::SalirCoche(const char * matricula, const Fecha &horasalida);
```

Este método tiene como dato de entrada la matrícula del coche que abandona el parking (que ha sido reconocida mediante una cámara) y la hora y devuelve un entero que se corresponde con el número de segundos que ha estado en el interior del parking. Si la matrícula no ha sido reconocida o si existe algún tipo de error, el método devolverá un número negativo.

Nota: Puedes suponer que ya disponemos de una función que recibe como parámetros dos fechas y devuelve la diferencia en segundos.

4.- (2.5 puntos) El software necesita hacer copias de seguridad periódicas del estado del parking. También necesitamos un método que permita restaurar los datos del parking desde una copia de seguridad. Implementa los siguientes métodos:

- void Parking::GuardarEnFichero(const char *fich) const;
- void Parking::LeerDesdeFichero(const char *fich);

Los datos se almacenan en un fichero binario. Dicho fichero comienza con dos enteros que indican el número de coches y el número de plazas. A continuación se almacenan (también en binario) los datos de cada uno de los coches. Observa que la matrícula es un dato de longitud variable.

5.- (1.5 puntos) Implementa una función que permita calcular el término n -ésimo de la serie de los números de Catalán de acuerdo a la siguiente expresión recursiva:

$$C_n = \begin{cases} 1 & \text{si } n=0 \\ \frac{2(2n-1)}{n+1} C_{n-1} & \text{si } n>0 \end{cases}$$

Nota: Los primeros términos de la serie, desde el 0-ésimo, son 1, 1, 2, 5, 14, 42, 132, ...

Metodología de la Programación

Examen de teoría. Grado en Ingeniería Informática. Julio 2011.

1. (1.5 puntos) Supongamos que en la parte privada de la clase `VectorDinamico` tenemos definido:

```
int NumMaxElems; // Núm. de casillas reservadas en v
int NumOcupados; // Núm. de casillas ocupadas
int * v;          // Acceso a los datos
```

de manera que mediante `v` se accede a los datos almacenados en el vector dinámico. Construir el método privado:

```
bool redimensionar (int nuevotam);
```

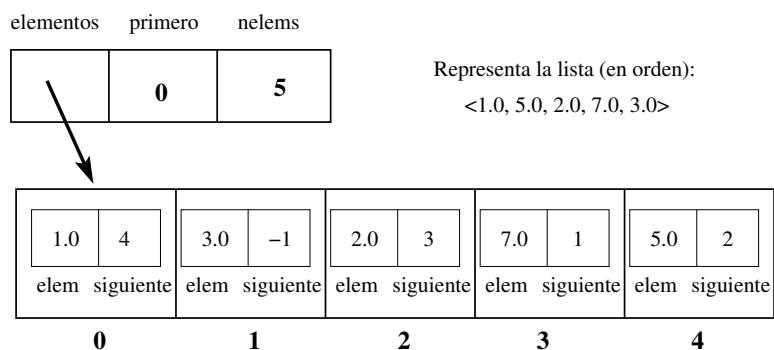
que cambia el número de casillas reservadas (`NumMaxElems`) al valor dado por `nuevotam`. El nuevo tamaño puede ser mayor o menor que el que tenía, y el vector debe conservar todos los elementos que sean posibles al cambiar el tamaño. Devuelve un valor booleano indicando si se han conservado todos los elementos “ocupados”.

2. Supongamos que tenemos una clase que denominamos `Lista`.

El código que declara la estructura de datos, así como un ejemplo con 5 elementos son:

```
struct dato {
    float elem;
    int siguiente
};

class Lista {
private:
    dato *elementos;
    int primero;
    int nelems;
public:
    .....
    .....
};
```



donde el campo `elem` es el elemento de cada posición de una lista de `float`, y `siguiente` es el índice en el vector `elementos` donde se encuentra el siguiente elemento (-1 si no hay más elementos).

Para este nuevo tipo de dato que almacena listas de elementos `float`, implemente los siguientes métodos:

- (1 punto) El constructor de copia.
 - (1 punto) El operador de asignación.
 - (1.5 puntos) El operador lógico de igualdad `==`. Dos listas son iguales si tienen el mismo número de elementos y éstos están dispuestos en el mismo orden *lógico* (esto es, si al recorrerlas se obtiene la misma secuencia de valores).
 - (1 punto) Un método que recibe un nombre de archivo (a través de un parámetro de tipo cadena estilo C) y almacena la `Lista` en dicho fichero y en formato **binario**. El fichero está compuesto por un `int` que corresponde al número de elementos de la lista (sea n este número), seguido de otro `int` que indica el índice en el vector del primer elemento de la lista, y seguido finalmente por una secuencia de n parejas `float-int`.
 - (1.5 puntos) Un método que, dado el nombre de un archivo con el formato indicado en el punto anterior, cargue el contenido de la lista almacenada en dicho fichero.
3. (2.5 puntos) Escribir un programa similar a `grep` que busque una palabra en una serie de ficheros de texto. La palabra a buscar y los ficheros en los que buscar se proporcionan en la línea de órdenes. Por ejemplo:

```
busca examen fich1 fich2 fich3
```

busca la palabra `examen` en los ficheros `fich1`, `fich2` y `fich3`.

Cada vez que encuentre la cadena buscada, debe indicar el fichero en el que se ha localizado, el número de línea, y la línea completa que la contiene. Un ejemplo de salida de este programa es:

```
fich1 (línea 33): El examen ha sido fácil
fich3 (línea 2): ya te dije ayer que hoy era el examen
fich3 (línea 242): finalmente, el examen tiene tres preguntas
```

Las restricciones que se imponen, y que se deben cumplir en la resolución son:

- El número de ficheros que se pueden proporcionar es ilimitado.
- Cada uno de los ficheros sólo puede ser leído una única vez, y no pueden copiarse completos en memoria.
- Se desconoce a priori el número de líneas de los ficheros.
- Las líneas de los ficheros tienen una longitud indeterminada, aunque nunca mayor de 500.

Duración del examen: 3 horas.

Metodología de la Programación

Examen de teoría. Grado en Ingeniería Informática. Septiembre de 2011.

1. Considere que tenemos almacenada una secuencia **ordenada** de números enteros en una lista de celdas enlazadas definidas con la siguiente estructura:

```
struct Celda {  
    int dato;      // Dato en la celda actual  
    Celda *sig;   // Puntero al siguiente elemento de la lista  
};
```

- a) (0.75 puntos) Defina una función que recibe un entero y una lista y modifica dicha lista ordenada insertando el entero en la posición correspondiente.
b) (0.75 puntos) Defina una función que recibe un entero y una lista y modifica dicha lista eliminando la primera aparición de ese entero en la lista.

Nota: Tenga en cuenta que en ambas funciones se puede dar el caso de que la lista que se pasa esté vacía.

2. Se define una **matriz bilineal simétrica** como una matriz de $n \times n$ enteros en la que todos los elementos significativos (distintos de un valor por defecto) están situados en las 2 diagonales principales y tal que al recorrer ambas diagonales en orden creciente de filas, presentan los mismos elementos. Un ejemplo de este tipo de matrices es la matriz del ejemplo. Es una matriz 6×6 de valores enteros y con el 0 como valor por defecto.

$$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 2 \\ 0 & 4 & 0 & 0 & 4 & 0 \\ 0 & 0 & 7 & 7 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 9 & 0 & 0 & 9 & 0 \\ 6 & 0 & 0 & 0 & 0 & 6 \end{pmatrix}$$

Se quiere construir la clase **MatrizBS**. Resuelva los siguientes problemas:

- a) (0.75 puntos) Definir la parte privada de la clase. Debe minimizarse el uso de memoria, guardando lo estrictamente necesario, para lo que será necesario usar memoria dinámica. Nótese que no se deben guardar los $n \times n$ valores de la matriz, ya que serían valores redundantes.
b) (0.75 puntos) Implementar el constructor por defecto y el destructor. El constructor por defecto creará una matriz de tamaño 4×4 , en la que los elementos de las dos diagonales principales serán todos 1 y el valor por defecto será 0.
c) (0.75 puntos) Implementar un constructor que reciba tres valores: n (el número de filas y columnas), un vector de enteros que contiene n elementos correspondientes a los valores en las diagonales y, finalmente, el valor que corresponde a las posiciones fuera de las diagonales. Este último será un parámetro opcional cuyo valor por defecto será cero.
d) (0.75 puntos) Implemente la sobrecarga del operador de asignación de la clase **MatrizBS**.
3. (1.5 puntos) Escribir un programa que reciba como parámetros -en la línea de órdenes- tres nombres de ficheros de texto. Los dos primeros ficheros contienen números reales ordenados en orden creciente y separados por espacios en blanco. El programa tomará los datos de esos ficheros y los irá copiando ordenadamente (de forma creciente) en el tercer fichero, de forma que al finalizar también esté ordenado.

El tiempo para realizar la parte teórica del examen es de 2 horas

Examen de Prácticas. Grado en Ingeniería Informática. Septiembre de 2011.

Definimos una permutación de tamaño n como una secuencia de los n enteros desde el 0 al $n - 1$ en un determinado orden. Se desea realizar un programa que genere una permutación de forma aleatoria y la imprima en la salida estándar. Para ello, se propone la creación de la clase **Permutacion** -que se diseña para contener una de estas secuencias- y un programa principal que la use para imprimirla. Concretamente, la solución estará compuesta por los siguientes archivos:

1. **Makefile**. Contendrá las reglas necesarias para crear el ejecutable escribiendo **make**, y para eliminar los archivos intermedios escribiendo **make clean**.
 2. **permutacion.h**. Contiene la clase **Permutacion** y las cabeceras de las funciones asociadas al uso de permutaciones, junto con una breve especificación sobre lo que hace cada función.
 3. **permutacion.cpp**. Contiene la definición de las funciones del archivo **permutacion.h**.
 4. **generar.cpp**. Contiene el programa que hace uso de la clase **Permutacion**. Este programa lee un número entero (n) desde la entrada estándar, genera una permutación aleatoria, y la imprime en la salida estándar.
- (3 puntos) Implemente los archivos **Makefile**, **permutacion.h**, **generar.cpp** y **permutacion.cpp**.

El tiempo para realizar la parte práctica del examen es de 1 hora



Metodología de la Programación

Convocatoria de Junio. Curso 2011/2012

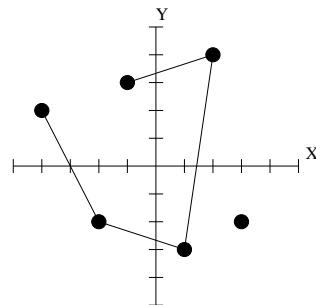
15 de Junio de 2012

Para poder gestionar figuras planas en una aplicación de gráficos 2D deseamos crear una estructura de datos capaz de representar adecuadamente esas figuras. La estructura escogida será la de una línea poligonal construida en base a una serie de puntos. Dispondremos de las clases **Punto** y **PoliLinea**

A continuación mostramos la declaración de ambas clases (sólo la representación interna) y un ejemplo de un punto en las coordenadas (3, -2) y de una polilínea definida por los puntos (-4, 2), (-2, -2), (1, -3), (2, 4) y (-1, 3)

```
class Punto {
    int x, y;      // Coordenadas de un punto 2D
    ...
};

class PoliLinea {
    Punto *p;      // Vector de puntos
    int num;        // Número de puntos
    ...
};
```



1. Implemente los siguientes métodos para la clase **PoliLinea**:

- (0.5 punto) El constructor sin parámetros (crea una línea poligonal vacía) y el destructor.
- (1.5 puntos) El constructor de copia y el operador de asignación.
- (1 puntos) El operador de acceso `[]`, que permite acceder (tanto para lectura como para escritura) a un dato de tipo **Punto** en una **PoliLinea**
- (2 puntos) Los operadores lógicos de igualdad `==` y desigualdad `!=`. Dos datos **PoliLinea** son iguales si tienen el mismo número de puntos, éstos son iguales y están dispuestos en el mismo orden o en orden inverso (en definitiva, son iguales cuando al representarse gráficamente se obtiene la misma figura).
- (2 puntos) Sobrecargar el operador `+` para poder añadir un punto a una línea poligonal de manera que podamos ejecutar operaciones de la forma:
 - polilínea `+` punto. Se crea una **nueva** polilínea añadiendo un punto al final de la misma.
 - punto `+` polilínea. Se crea una **nueva** polilínea añadiendo un punto al inicio de la misma.

En ambos casos, la **PoliLinea** inicial **no** se modifica.

Si fuera preciso implementar algún método para la clase **Punto**, deberá hacerlo.

Escribir el código correctamente modularizado en ficheros `.cpp` y `.h`.

2. Considere el siguiente formato que permite almacenar una **PoliLinea** en un fichero de texto:

POLILINEA
Comentario opcional
N
X1 Y1
X2 Y2
X3 Y3
...
Xn Yn

El fichero siempre comienza por la cadena **POLILINEA**
 El comentario es opcional y, en caso de estar:
 Comienza por el carácter **#**
 Sólo ocupa una línea
 No hay límite de caracteres
 N es el número de puntos que tiene la polilínea
 Despues de N tenemos la lista de coordenadas de cada punto
 Cada punto se escribe en una nueva línea y las coordenadas están separadas por un espacio

- (2 puntos) Implementar un método para cargar en memoria una **PoliLinea** desde un fichero:
`void LeerPolilinea (const char *nombre);`
- (1 puntos) Implementar un método para escribir en un fichero una **PoliLinea**:
`void EscribirPolilinea (const char *nombre, const char *comentario=0);`

Duración del examen: 2 horas y media.

Metodología de la Programación

Convocatoria de Septiembre. Curso 2011/2012

11 de Septiembre de 2012

Considere el tipo de dato *Matriz*, diseñado para almacenar una estructura bidimensional de elementos *double*, con la siguiente representación:

```
class Matriz {
    int nfilas;           // Número de filas de la matriz (ejemplo: 4)
    int *ncolumnas;       // Número de columnas de cada fila (ejemplo: 3,2,5,2)
    double **datos;       // Datos de la matriz
public:
    .....
};
```

	0	1	2	3	4
0	5.0	6.2	1.1		
1	1.4	3.2			
2	1.0	2.3	3.2	4.7	5.0
3	1.6	5.9			

Observe que este tipo de dato **difiere de las matrices 2D clásicas** en que el número de columnas puede ser distinto para cada fila. Para este tipo de dato, implemente los siguientes métodos de la clase:

- (0.5 puntos) El constructor sin parámetros (crea una matriz vacía) y el destructor.
- (1.25 punto) El constructor de copia y el operador de asignación.
- (0.75 punto) Sobrecargue los operadores de E/S, es decir, >> y << para la entrada y salida de una matriz completa en formato de texto.
- (0.5 puntos) Dos métodos **Get** y **Set**, para acceder o modificar el contenido de una posición, respectivamente.
- (0.75 punto) Método **Vflip** que devuelva una **nueva** matriz *intercambiando* las filas (primera por la última, segunda por la penúltima, ...)
- (0.5 puntos) Método **Max** que devuelva la posición (*fila* y *columna*) donde se encuentra el mayor valor de la matriz.
- (0.75 puntos) Método **Escribir** para salvar un objeto de tipo *Matriz* en un fichero. Este fichero debe tener la siguiente estructura:

MP
Comentario
4
<contenido en binario>

Cadena mágica: Dos caracteres seguidos por un separador (salto de línea)
 OPCIONAL: Comienza por el carácter '#', hasta salto de línea
 Un entero en texto con el número de filas, seguido por un separador
 Número de columnas más datos en binario (consulte detalles a continuación)

Contenido Binario	int	int	int	int
	3	5.0	6.2	1.1
	double	double	double
	2	1.4	3.2	5
	double	double	double
	2	1.6	5.9	
	double	double	double

Observe que después de las tres primeras líneas de texto (dos si no hay comentario), aparece una lista de números en binario. Corresponde a las filas, desde la primera a la última, cada una con un *int* -que indica el número de columnas- seguido de los correspondientes datos *double*.

- (1 puntos) Método **Leer** que carga una matriz desde un fichero que tiene la estructura indicada en el apartado anterior.

Duración del examen: 2 horas.

Metodología de la Programación

Convocatoria de Septiembre. Curso 2011/2012

Examen práctico

11 de Septiembre de 2012

Una empresa dedicada al transporte de viajeros por carretera dispone de un sistema informático en el que almacena, entre otra información, los datos relativos a los servicios que realiza. Dispone de los siguientes ficheros, que almacenan la información en formato **binario**:

Conductores

- Código conductor (alfanumérico, 5 caracteres)
- Apellidos (alfanumérico, 40 caracteres)
- Nombre (alfanumérico, 20 caracteres)
- Día alta (numérico, int)
- Mes alta (numérico, int)
- Año alta (numérico, int)

Rutas

- Código de ruta (alfanumérico, 10 caracteres)
- Ciudad origen (alfanumérico, 20 caracteres)
- Ciudad destino (alfanumérico, 20 caracteres)
- Kilómetros (numérico, double)

Viajes2012

- Código de conductor (alfanumérico, 5 caracteres)
- Código de ruta (alfanumérico, 10 caracteres)
- Dia viaje (numérico, int)
- Mes viaje (numérico, int)
- Año viaje (numérico, int)

La empresa desea, con urgencia, disponer de una información que debe extraerse de estos ficheros. Para ello realizará unos programas que se ejecutarán desde la línea de órdenes, proporcionando los argumentos necesarios en la llamada a cada programa.

1. Dado el código de un conductor, ¿Cuántos kilómetros recorrió y cuántos viajes hizo?

Kilometros cod-conductor

Indicar, además de la información solicitada, el nombre completo (*nombre* y *apellidos*) del conductor.

2. Mostrar un listado, con todos los conductores, en el que se incluyan los siguientes datos: código conductor, nombre completo de conductor y número de viajes.

3. ¿Qué ruta ha tenido más conductores?

4. Dado un conductor y un mes, ¿Cuántos kilómetros recorrió ese mes?

Kilometros cod-conductor mes

Duración del examen: 1 hora.



Metodología de la Programación

Convocatoria de Junio. Curso 2012/2013

24 de Junio de 2013

Se desea resolver el problema de sumar un número indeterminado de enteros no negativos, con la dificultad añadida de que dichos números pueden llegar a ser muy largos, siendo imposible usar el tipo *int* del lenguaje. Para resolverlo, se propone crear una clase **BigInt** (entero largo) que puede almacenar un entero no negativo de longitud indeterminada.

La clase representará un entero mediante un array -de longitud variable- de objetos de tipo *int*, reservado en memoria dinámica, para poder almacenar todos y cada uno de los dígitos de un entero de longitud indeterminada. Tenga en cuenta que un objeto *int* puede almacenar un rango muy amplio de valores, sin embargo, por simplicidad, será el tipo que usaremos para almacenar cada dígito -del 0 al 9- del *BigInt*.

Además, los dígitos del *BigInt* se almacenarán de forma que el menos significativo -las unidades- se situará en la posición cero del array. Por ejemplo, a continuación se presentan dos ejemplos: los enteros largos 9530273759835 y 0. Observe que para el primer caso se ha reservado y usado un array de objetos *int* de longitud 13 y, para el cero, un array de longitud uno, con el dígito 0.

0	1	2	3	4	5	6	7	8	9	10	11	12
5	3	8	9	5	7	3	7	2	0	3	5	9

0
0

Considerando este problema, **resuelva las siguientes preguntas:**

1. **(0.25 puntos)** Implemente el constructor sin parámetros y el destructor. Dicho constructor crea un entero largo con valor cero.
2. Implemente dos constructores adicionales:
 - a) **(0.75 puntos)** Constructor que crea un *BigInt* a partir de una cadena de caracteres. Esta cadena de caracteres contiene los dígitos del entero largo en formato decimal.
 - b) **(1.5 puntos)** Constructor que crea un *BigInt* a partir de un objeto de tipo *unsigned int*.
3. **(1 punto)** Implemente el constructor de copia y operador de asignación.
4. **(2 puntos)** Sobrecargue el operador de suma para que, a partir de dos objetos *BigInt*, se obtenga un nuevo objeto que corresponde al entero largo resultado de su suma.
5. **(1 punto)** Sobrecargue el operador de salida (operador `<<`) para la clase *BigInt* de forma que nos permite escribir el entero largo en un flujo de salida. Tenga en cuenta que deberá presentar todos los dígitos desde el más significativo al menos significativo (escritura habitual de enteros).
6. **(1.5 puntos)** Sobrecargue el operador de entrada (operador `>>`) para la clase *BigInt* de forma que nos permite leer el entero largo desde un flujo de entrada. Tenga en cuenta que deberá leer todos los dígitos desde el más significativo al menos significativo. El operador de entrada debe comportarse de forma similar al caso del tipo *int*, es decir, asumiendo que cada *BigInt* -secuencia de dígitos consecutivos- se encuentra separado de otros datos por “espacios blancos” (espacios, tabuladores, saltos de línea, etc.).
7. **(1 punto)** Considerando todas las operaciones que se han descrito en los puntos anteriores, escriba el correspondiente archivo de cabecera (“*bigint.h*”) teniendo en cuenta que no se incluye ninguna función en línea (*inline*). Es decir, sólo aparecerán cabeceras de funciones, sin su definición.
8. **(1 punto)** Considere que tiene correctamente implementadas las preguntas anteriores. Supongamos que tenemos un archivo con un número indeterminado de enteros largos en formato texto, separados por “espacios blancos”. Implemente un programa “*suma.cpp*” que use la clase *BigInt* para leer todos los valores que hay en el fichero y escribir, como resultado final, la suma de todos ellos. Por ejemplo, si el archivo “*datos.txt*” contiene los siguientes enteros:

9347829037470000000000000000
9327887198348931

Una posible ejecución del programa podría ser la siguiente:

```
% suma datos.txt  
9347829037479327887198348931
```

Duración del examen: 2 horas y media.

Metodología de la Programación

Examen de teoría. Septiembre 2013.

1. (2.5 puntos) Considere la siguiente clase:

```
class Entero {  
    int *v;  
public:  
    Entero (int i=0) { v= new int; *v=i; }  
    ~Entero () { delete v; }  
    int Set(int i) { *v= i; }  
    int Get() const { return *v; }  
};
```

Realice las modificaciones que crea convenientes sobre la clase *Entero* para que el siguiente código funcione correctamente, describiendo brevemente la razón de dichos cambios.

```
Entero Doble(Entero e)  
{ return e.Get()*2; }  
int main()  
{ Entero x(5), y;  
y= Doble(x);  
cout << "Resultado: " << y << endl;  
}
```

2. Se desea crear una clase *Menu* para simplificar el desarrollo de programas que usan menús en modo texto. Para ello, se propone la siguiente representación:

```
class Menu {  
    char *titulo; // Encabezado que damos al menú (0 si no existe)  
    char **opc; // Cadenas de longitud variable que describen cada una de las opciones  
    int nopc; // Número de opciones actualmente en el menú  
public:  
    ...  
};
```

- a) (1.5 puntos) Escriba la implementación de las funciones miembro correspondientes al constructor por defecto (crea un menú vacío), destructor, constructor de copias y operador de asignación.
- b) (1.5 puntos) Escriba tres funciones:
- *SetTitulo* que asigne un nuevo valor al título.
 - *GetNumeroOpciones* que devuelva el número de opciones que hay actualmente en el menú.
 - *AddOpcion* que reciba una cadena de caracteres y la añada como una nueva opción después de la última.
- c) (1.5 puntos) Sobrecargue los operadores `<<` y `>>` para poder usar el menú para seleccionar una de las opciones. Concretamente:
- Sobrecargue el operador `<<` para que podamos imprimir el menú. Un ejemplo de llamada podría ser `cout << menu`, que imprime en la salida estándar el título del menú seguido por cada una de las opciones (numerándolas, del 1 al número de opciones, en líneas distintas).
 - Sobrecargue el operador `>>` para que podamos escoger una nueva opción. La sintaxis de llamada será `menu >> entero`, y provoca la solicitud de una opción (como un número del 1 al número de opciones desde la entrada estándar) de entre las que incluye el menú. El efecto es que imprime el menú en la salida estándar y pide un valor entero de entre las opciones disponibles. Esta petición se repite hasta que el valor es una opción válida.
- d) (0.5 puntos) Complete el entorno *Class*, que hemos presentado en la pregunta, con la parte pública correspondiente.

3. (2.5 puntos) Escriba un programa *alreves* que recibe en la línea de órdenes el nombre de un fichero, y escribe en la salida estándar el mismo flujo de caracteres, pero en orden inverso. Por ejemplo, si el archivo *abecedario.txt* contiene los caracteres:

```
abcdefghijklmn  
ñopqrstuvwxyz
```

una posible ejecución del programa obtendría lo que sigue:

```
% alreves abecedario.txt  
zyxwvutsrqpoñ  
nmlkjihgfedcba
```

Duración del examen: 2 horas y 30 minutos.

Metodología de la Programación

Convocatoria de Septiembre. Curso 2012/2013

Examen práctico

20 de Septiembre de 2013

1. El problema

El objetivo es implementar completamente una solución al problema de calcular la clasificación final de un circuito de n carreras de fondo.

Cada corredor inscrito participa con el mismo dorsal en todas las pruebas del circuito y puede participar en un número indeterminado de pruebas (podría, incluso, no participar en ninguna).

El sistema de puntuación es el siguiente: en cada prueba se asigna 1 punto al primer clasificado, 2 al segundo, 3 al tercero, y así sucesivamente. El ganador del circuito es el que **menos** puntos totales obtiene, siempre que complete un número mínimo de pruebas, m . En el caso de completar más pruebas que el número mínimo, se contabilizarán para el total las m mejores clasificaciones.

2. Los datos

Los **resultados de cada carrera** se almacenan en un fichero *binario* (un fichero por cada prueba). Cada fichero consiste en una secuencia de datos *int* (codificados en binario) que representan los dorsales, por orden de llegada, de los corredores que han terminado la carrera.

Evidentemente, los ficheros pueden tener diferentes tamaños (el número de corredores que terminan cada prueba es diferente) y los datos estarán en distinto orden (el orden de llegada será diferente).

3. El programa

La aplicación que se va a desarrollar podría aplicarse en diferentes circuitos, por lo que el número de corredores inscritos, el número de pruebas del circuito y el número mínimo de pruebas a completar no se fijan de antemano.

El programa se ejecutará:

```
% clasificacion <m> <fich_1> [<fich_2>...<fich_n>]
```

donde:

- m es el número mínimo de carreras que deben completar los inscritos al circuito para poder optar a la clasificación final.
- $fich_1, fich_2, \dots$ son los nombres de los ficheros de resultados, descritos en el punto 2. Observe que el número de ficheros es n (uno por cada carrera del circuito). Se requiere, al menos, un fichero y no se establece un número máximo de ficheros.

El programa calculará la clasificación del circuito y mostrará en la salida estándar un listado (ordenado por los puntos totales) indicando en cada línea la siguiente información de cada uno de los corredores que han completado el número mínimo de carreras: número de orden, dorsal, número de carreras completadas y puntos totales.

Escribir un fichero *makefile* que se encargue de generar el ejecutable *clasificacion* a partir de todos los ficheros fuente (*.cpp* y *.h*) que se consideren necesarios.

4. Otras consideraciones

Para la resolución de este problema se respetarán las siguientes restricciones:

- a) El mínimo dorsal es el 1, pero el máximo no se conoce cuando se ejecuta el programa.
- b) Se valorará especialmente la eficiencia del programa desarrollado.
- c) Se valorará especialmente la modularidad de la solución.

Duración del examen: 1.5 horas.

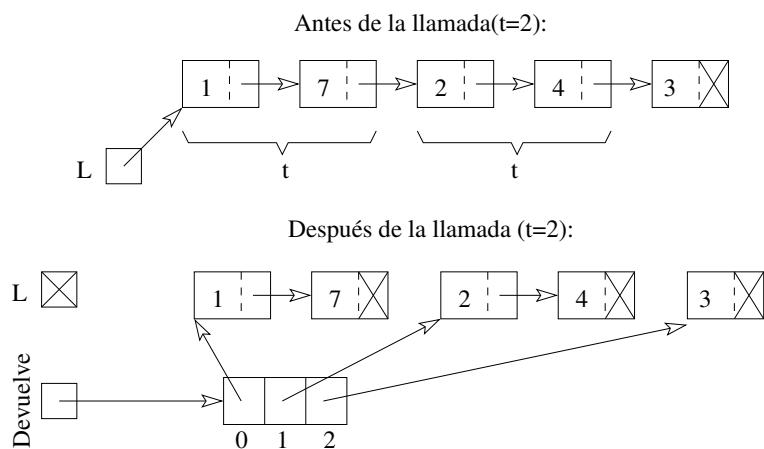


Metodología de la Programación

Convocatoria ordinaria. Curso 2013/2014

7 de Julio de 2014

1. (1.0 puntos) Se desea dividir una lista de celdas enlazadas en secuencias de celdas de tamaño t . Escriba una función que recibe una lista de celdas enlazadas que almacena enteros y devuelva un vector de listas que contiene cada una de estas secuencias. La siguiente figura muestra gráficamente el efecto de la llamada:



Observe que en la parte superior se muestra la lista original y el valor $t=2$. La lista, que contiene 5 elementos tendrá que dividirse en 3 trozos. En la parte inferior se muestra el efecto de la llamada. Tenga en cuenta que:

- La lista original queda vacía.
- No es necesario reservar ni liberar ninguna celda.
- La última lista del vector podría quedar con menos de t casillas. En este ejemplo, se queda sólo con 1.
- Será necesario reservar el vector para almacenar cada una de las sublistas. En este caso es un vector con 3 listas.
- Cada lista es una secuencia de celdas enlazadas terminada en el puntero nulo.
- Puede suponer que la lista no está vacía.

Define una estructura adecuada para almacenar cada una de las celdas enlazadas y escriba la función que implementa la operación descrita.

2. Se desea resolver el problema del juego de los barquitos. Para ello, se propone diseñar una clase *Barquitos* que contiene el tablero de un jugador. La clase debe incluir una matriz de enteros para codificar el estado del tablero. El estado se codifica con una estructura de dos dimensiones de tamaño variable, reservada en memoria dinámica.

En la siguiente figura se presenta una matriz de codificación con enteros, la correspondiente representación gráfica y las indicaciones para entender la representación.

En este ejercicio debe implementar parte de la clase *Barquitos*. Para realizarlo, comience proponiendo la **representación de la parte privada de la clase**. Una vez establecida la representación, resuelva los siguientes apartados:

	1	2	3	4	5	6	7	8	9	10	11
A											
B	.		■				.				
C	■■■■		.		
D			■■			.					
E	■		■■			■■					
F					.		■■■■				
G											
H	.		.			■■■■					
I	.		.								
J		.						■■			
K	■■■■				.	.					
L					■■■■		.				

	1	2	3	4	5	6	7	8	9	10	11
A	9	9	9	9	9	9	9	9	9	9	9
B	9	-9	1	9	9	9	-9	9	9	9	9
C	9	9	-9	-9	-9	-9	-3	-3	-3	-9	9
D	9	9	9	-4	9	9	-9	9	9	9	9
E	9	1	9	4	9	9	-1	9	9	9	9
F	9	9	9	4	9	9	-9	9	9	2	2
G	9	9	9	4	9	9	9	9	9	9	9
H	9	-9	9	9	-9	9	3	9	9	9	9
I	9	-9	9	9	-9	9	3	9	9	9	9
J	9	9	-9	9	9	9	3	9	9	-1	9
K	9	2	2	9	9	9	9	-9	-9	9	9
L	9	9	9	9	9	9	9	-2	-2	-9	9

Codificación:

Valor negativo: se ha disparado en la casilla

Valor 9: agua

Valor entero i: parte de un barco de i casillas

Ejemplos:

B3: valor 1. Barco de 1 casilla

B2: valor -9. Agua donde se ha disparado

D4: valor -4. Barco de 4 alcanzado

C8: valor -3. Barco de 3 alcanzado

a) **(0.75 puntos)** Implemente el constructor de la clase y el destructor. Este constructor recibe las dimensiones del tablero –filas y columnas– y construye un tablero en memoria dinámica con todas las casillas con agua.

b) **(1.25 puntos)** Implemente el constructor de copias y la sobrecarga del operador de asignación.

c) **(0.75 puntos)** Implemente un método que recibe la posición de un barco y devuelve si es posible colocarlo. La posición de un barco viene determinada por:

- Una casilla: un carácter para codificar la fila (desde la 'A') y un entero para la columna (desde el 1). Puede suponer que no habrá más filas que letras consecutivas en la tabla ASCII.
- El tamaño del barco: un número positivo mayor que cero y menor que 9.
- La dirección de dibujo: 'H' o 'V' indicando horizontal-derecha y vertical-abajo, respectivamente.

Por ejemplo, en la figura anterior, el barco de tamaño 3 hundido (con todas las casillas tachadas) está en la posición: casilla C7, de tamaño 3 y dirección H.

Nota: El barco se puede poner si las casillas no están ocupadas y no hay ningún barco con el que "se toque". Por ejemplo, en la figura anterior no se puede poner un barco de tamaño 1 en la posición A2.

d) **(1.0 puntos)** Implemente un método para insertar un barco en el tablero. Este método debe recibir el tamaño del barco e insertarlo en una posición y dirección aleatorias. Supondremos como precondición que seguro que hay algún sitio en el tablero que permite insertarlo.

Para resolverlo suponga que dispone del método del apartado anterior y la siguiente función:

```
int Aleatorio (int min, int max); // un valor aleatorio del intervalo [min,max]
```

Nota: Tenga en cuenta que para generar una dirección no tiene más que generar un valor Aleatorio(1,2) para escoger entre dos posibilidades.

3. En este ejercicio se desea ampliar la clase *Barquitos* con operaciones de E/S. Se desea almacenar el contenido de un tablero en un fichero. El formato de almacenamiento es el siguiente:

- Una cadena “mágica” compuesta por los caracteres “MP-BARQ-V1.0” seguida de un salto de línea.
- Un entero (número de filas), un espacio, un entero (número de columnas) y un salto de línea.
- Tantos valores enteros como casillas tiene el tablero codificados en binario.

a) **(0.75 punto)** Implemente un método *Leer* que recibe el nombre de un fichero y lee el contenido del tablero. Devuelve si ha tenido éxito.

b) **(0.5 puntos)** Implemente un método *Escribir* que recibe el nombre de un fichero y guarda el contenido del tablero. Devuelve si ha tenido éxito.

Duración del examen: 2 horas y media.

Metodología de la Programación

Examen de teoría. Convocatoria extraordinaria. Curso 2013/2014

5 de Septiembre de 2014

Se desea crear un programa para el análisis de rutas terrestres. Una ruta es una secuencia de 0 o más localizaciones sobre la superficie terrestre. Una ruta con cero localizaciones se considera la ruta *NULA*. Una localización está determinada por la latitud y la longitud. Para resolverlo, se proponen dos nuevos tipos:

```
struct Localizacion {
    double longitud, latitud, altura; // Localización geográfica
};

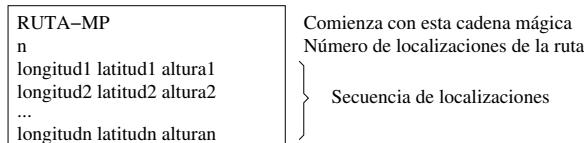
class Ruta {
    Localizacion *locs; // Secuencia de localizaciones: locs[i] != locs[i+1]
    int n;              // Números de localizaciones en locs: n >= 0
    ...
};
```

Observe que hemos indicado las condiciones que debe cumplir la representación del tipo **Ruta**. Por ejemplo, una ruta de 5 localizaciones tendrá 4 tramos: 0-1, 1-2, 2-3 y 3-4.

1. Implemente las siguientes operaciones:

- a) (0.5 puntos) El constructor sin parámetros y el destructor. El constructor inicializa la ruta como *NULA*.
- b) (1.5 puntos) El constructor de copia y el operador de asignación.
- c) (1 punto) Sobrecargar el operador + para poder añadir una localización al final o al principio de la ruta. Deberá implementar dos funciones (observe que, en ambos casos, la ruta inicial **no** se modifica):
 - ruta + loc. Se crea una **nueva** ruta añadiendo una localización al final de la misma.
 - loc + ruta. Se crea una **nueva** ruta añadiendo una localización al inicio de la misma.

2. (1.5 puntos) El almacenamiento de una ruta se realiza en formato de texto con el siguiente formato:



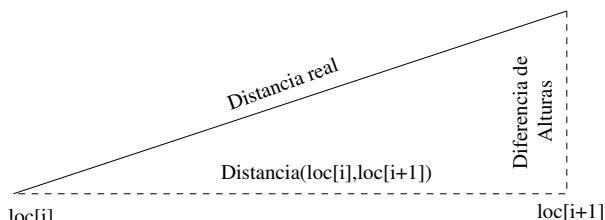
Implemente dos métodos en la clase **Ruta** para la lectura y escritura, respectivamente. En ambos casos, los métodos reciben el nombre de un archivo y devuelven un booleano que indica si ha tenido éxito.

3. Considere un módulo **ruta.h** donde está implementada la clase anterior y la siguiente operación:

```
double Distancia(const Localizacion& l1, const Localizacion& l2);
```

que corresponde a la distancia entre dos localizaciones. Resuelva lo siguiente:

- a) (0.5 puntos) Implemente un método que devuelve la longitud *L* de la ruta. La longitud corresponde a la suma de las distancias de los tramos que componen la ruta. Tenga en cuenta que la distancia entre dos localizaciones consecutivas de la ruta corresponde a la distancia real, es decir, debe tener en cuenta el valor devuelto por la función anterior y la diferencia de alturas:



- b) (1 punto) Escriba un programa *longitud.cpp* que se llama desde la línea de órdenes con el nombre de un archivo y usa todo lo anterior para escribir en la salida estándar la longitud de la ruta almacenada en el archivo. Un ejemplo de ejecución sería el siguiente:

```
% longitud excursion.trk
La longitud de la ruta "excursion.trk" es: 4590.456
```

Duración del examen: 2 horas y media.

Metodología de la Programación

Examen de Prácticas.

Convocatoria extraordinaria de Septiembre. Curso 2013/2014

5 de Septiembre de 2014

El objetivo del proyecto es realizar estadísticas simples sobre el número de apariciones de una serie de palabras en un conjunto de ficheros de texto. Los resultados del análisis se almacenarán en un *fichero resumen*. Este fichero contendrá la frecuencia de aparición de cada palabra en los ficheros de búsquedas.

En este examen el alumno debe realizar un proyecto de programación completo que consiste en la generación de dos ejecutables con la ayuda de un fichero *makefile*. Los programas son:

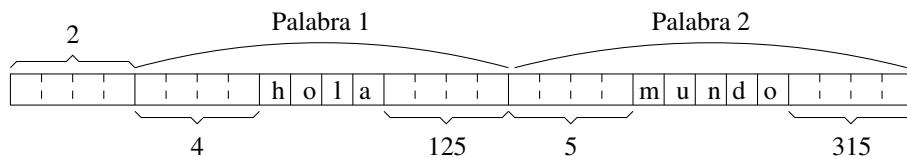
1. *estadistica_palabras*: recibe los nombres de una serie indeterminada de ficheros de texto y genera el fichero resumen resultado. El formato de ejecución será el siguiente:

```
% estadistica_palabras <fich_palabras> <fich_1> [<fich_2>...<fich_n>] <fich_resumen>
```

de manera que:

- <fich_palabras> es un fichero de texto y contiene las palabras sobre las que se va a hacer el estudio. Estas palabras estarán separadas por espacios blancos (espacio, tabulador, salto de línea, etc.).
- <fich_1>, <fich_2>, ..., <fich_n> son los ficheros de texto en los que se van a buscar las palabras contenidas en <fich_palabras>. Al menos hay que proporcionar uno.
- <fich_resumen> es el fichero binario que se creará y que contiene el número de palabras y los datos sobre cada una de ellas (longitud, caracteres que la componen y número total de apariciones en los archivos).

Por ejemplo, en la siguiente figura se presenta un fichero que contiene los resultados sobre dos palabras (el fichero de palabras contiene “hola” y “mundo”) que se han repetido un total de 125 y 315 veces, respectivamente:



Observe que se ha dibujado sobre una secuencia de bytes, es decir, se ha especificado que en este caso cada entero ocupa 4 bytes. Por tanto, será un archivo de 29 bytes.

Durante la ejecución, debe mostrar información en la consola con los resultados parciales: el programa buscará cada una de las palabras del primer fichero en los ficheros de texto proporcionados y mostrará en la consola, para cada palabra, el nombre de cada fichero donde se ha buscado y el número de veces que aparece en ese fichero. Finalmente, mostrará el número total de apariciones de dicha palabra.

2. *mostrar*: recibe únicamente el nombre del fichero de frecuencias y muestra en la salida estándar el contenido. Por tanto, mostrará el número de palabras analizadas y, para cada una, su contenido y el total de apariciones. Por ejemplo, la ejecución sobre un archivo “*resultado.dat*” que corresponde a la figura anterior será:

```
% mostrar_palabras resultado.dat
Palabras totales: 2
125 "hola"
315 "mundo"
```

IMPORTANTE: Restricciones

- No se puede usar el tipo *string* de la STL.
- Los ficheros de entrada, tanto el de palabras como los de texto, sólo se pueden leer una vez.

Debe especificarse claramente el nombre de cada uno de los ficheros que componen el proyecto, así como el directorio en el que se van a alojar. **El alumno escribirá el fichero makefile que permita gestionar adecuadamente el proyecto.**

Duración del examen: 1 hora y media.



Normas para la realización del examen:

Duración: 2.5 horas

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

◀ Ejercicio 1 ▷ Celdas enlazadas

[1 punto]

Considere que una lista de celdas enlazadas se controla con un único puntero que vale cero cuando la lista está vacía y apunta a la primera celda cuando hay al menos un elemento. Defina una estructura de tipo *Celda* que permita tener una lista de objetos de tipo *double*. Escriba una función que recibe una lista e imprime todos sus elementos desde el último al primero.

◀ Ejercicio 2 ▷ Frecuencias de enteros: clase

[2 puntos]

Se desea crear un programa para calcular el número de repeticiones en una secuencia de números enteros. Por ejemplo, en el caso de la secuencia 9 3 9 3 2 4 existen 4 datos distintos, dos de ellos –2 y 4– se repiten una vez y los otros dos se repiten dos veces. Para resolver el problema, se propone crear dos tipos de datos:

```
struct Pareja {  
    int dato;  
    int nveces;  
};  
class Frecuencias {  
    Pareja *parejas; // "npares" datos de tipo "Pareja" ordenados por "dato". 0 si no hay parejas.  
    int npares;  
public:  
    // ... interfaz pública de la clase  
};
```

Defina las siguientes funciones:

1. (0.5 puntos) El constructor por defecto y el destructor.
2. (1.5 puntos) El constructor de copia y operador de asignación.

◀ Ejercicio 3 ▷ Frecuencias de enteros: miembro público

[1 punto]

Considere la clase *Frecuencias* anterior. Se decide añadir una función que facilite contabilizar un número entero. Implemente una función miembro *Add* que recibe un entero y lo añade a la clase. Si el entero ya se había añadido anteriormente, deberá incrementar el contador correspondiente. Si no, deberá añadir una nueva pareja con dicho entero y el contador con valor 1. Tenga en cuenta que las parejas de datos están ordenadas por el valor del campo *dato*.

◀ Ejercicio 4 ▷ Frecuencias de enteros: Sobrecarga de operadores

[1 punto]

Considere la clase *Frecuencias* anterior. Implemente las siguientes funciones:

1. Sobrecarga de *+=* que permite añadir un entero a la secuencia. Puede suponer que está disponible la función *Add*.
2. Sobrecarga de *<<* que imprime las parejas de valores en un flujo de salida, una por línea.

◀ Ejercicio 5 ▷ Frecuencias de enteros: uso de la clase

[1 punto]

Considere la clase *Frecuencias* anterior, para la que se ha creado un archivo de cabecera. Escriba un programa –*contar.cpp*– que lee una secuencia de enteros y escribe en la salida estándar las parejas con las frecuencias. Tenga en cuenta que:

- Si no se da ningún parámetro en la línea de órdenes, el programa lee los datos desde la entrada estándar.
- Si se ejecuta con un parámetro en la línea de órdenes, el programa lee los datos desde este archivo.

Recuerde que el programa deberá mostrar un mensaje de error adecuado si falla la entrada.



Normas para la realización del examen:

Duración: 3.5 horas

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

△ Ejercicio 1 ▷ Celdas enlazadas

[1 punto]

Considere que una lista de celdas enlazadas se controla con un único puntero que vale cero cuando la lista está vacía y apunta a la primera celda cuando hay al menos un elemento. Defina una estructura de tipo *Celda* que permita tener una lista de objetos de tipo *double*. Escriba una función que recibe una lista e imprime todos sus elementos desde el último al primero.

△ Ejercicio 2 ▷ Frecuencias de enteros: clase

[2 puntos]

Se desea crear un programa para calcular el número de repeticiones en una secuencia de números enteros. Por ejemplo, en el caso de la secuencia 9 3 9 3 2 4 existen 4 datos distintos, dos de ellos –2 y 4– se repiten una vez y los otros dos se repiten dos veces. Para resolver el problema, se propone crear dos tipos de datos:

```
struct Pareja {  
    int dato;  
    int nveces;  
};  
class Frecuencias {  
    Pareja *parejas; // "npares" datos de tipo "Pareja" ordenados por "dato". 0 si no hay parejas.  
    int npares;  
public:  
    // ... interfaz pública de la clase  
};
```

Defina las siguientes funciones:

1. (0.5 puntos) El constructor por defecto y el destructor.
2. (1.5 puntos) El constructor de copia y operador de asignación.

△ Ejercicio 3 ▷ Frecuencias de enteros: miembro público

[1 punto]

Considere la clase *Frecuencias* anterior. Se decide añadir una función que facilite contabilizar un número entero. Implemente una función miembro *Add* que recibe un entero y lo añade a la clase. Si el entero ya se había añadido anteriormente, deberá incrementar el contador correspondiente. Si no, deberá añadir una nueva pareja con dicho entero y el contador con valor 1. Tenga en cuenta que las parejas de datos están ordenadas por el valor del campo *dato*.

△ Ejercicio 4 ▷ Frecuencias de enteros: Sobrecarga de operadores

[1 punto]

Considere la clase *Frecuencias* anterior. Implemente las siguientes funciones:

1. Sobrecarga de *+=* que permite añadir un entero a la secuencia. Puede suponer que está disponible la función *Add*.
2. Sobrecarga de *<<* que imprime las parejas de valores en un flujo de salida, una por línea.

△ Ejercicio 5 ▷ Frecuencias de enteros: uso de la clase

[1 punto]

Considere la clase *Frecuencias* anterior, para la que se ha creado un archivo de cabecera. Escriba un programa *-contar.cpp-* que lee una secuencia de enteros y escribe en la salida estándar las parejas con las frecuencias. Tenga en cuenta que:

- Si no se da ningún parámetro en la línea de órdenes, el programa lee los datos desde la entrada estándar.
- Si se ejecuta con un parámetro en la línea de órdenes, el programa lee los datos desde este archivo.

Recuerde que el programa deberá mostrar un mensaje de error adecuado si falla la entrada.

△ Ejercicio 6 ▷ Frecuencias de enteros: ampliando la clase

[1.5 puntos]

Considere la clase *Frecuencias* anterior. Para hacerla más útil, se decide añadir una nueva función miembro *Repeticiones*, que recibe un número entero *d* y devuelve el número de repeticiones de *d*. Lógicamente, será cero si no está en



ninguna pareja. Implemente esta función miembro tal y como aparecería en el archivo *cpp*. Tenga en cuenta que debe hacerla lo más eficiente posible.

▷ **Ejercicio 7 ▷ Frecuencias de enteros: módulos**

[1.25 puntos]

Considere el tipo de dato *Frecuencias* anterior. Escriba el contenido del archivo de cabecera *frecuencias.h*, donde aparece todo lo necesario para usar la clase, sin incluir ninguna definición de función.

▷ **Ejercicio 8 ▷ Frecuencias de enteros: makefile**

[1.25 puntos]

Considere los archivos que implementan el tipo *Frecuencias* así como el programa *contar.cpp* anterior. Escriba el archivo *Makefile* que permite compilar el programa. Incluya las correspondientes dependencias, así como un objetivo *clean* que permite limpiar los archivos intermedios.



Normas para la realización del examen:

Duración: 2.5 horas

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

Los dispositivos GPS registran posiciones periódicamente y las almacenan. La posición de un punto está determinada por tres datos: su coordenada sobre la superficie terrestre (*latitud* y *longitud*) y su *altura*. Una sucesión de puntos registrados consecutivamente forman un recorrido (en inglés, *track*). A partir de un recorrido se pueden calcular distintas medidas que permiten analizarlo, visualizarlo en mapas, utilizarlo como guía para repetir el mismo recorrido, ...

El alumno deberá implementar diferentes métodos para clases diseñadas para la gestión de recorridos.

Se proponen las siguientes clases secundarias:

```
class Fecha {           class Hora {           class Instante {           class Punto {  
    private:             private:             private:  
        int dia;           int hora;           Fecha fecha;  
        int mes;           int minuto;         Hora hora;  
        int anio;           int segundo;       public:  
    public:               .....           .....  
    .....           .....           .....  
};           };           };           };
```

Nota: Suponemos disponibles los métodos públicos *Get...* en las clases enumeradas anteriormente para obtener los valores de los campos privados. Por ejemplo, para la clase Hora disponemos de *GetHora*, *GetMinuto* y *GetSegundo*.

Estos tipos se utilizan para poder crear el tipo de dato *principal* que será el encargado de gestionar recorridos. La representación propuesta es la siguiente:

```
class Recorrido {  
private:  
    PuntoDeRecorrido *puntos; // Puntero a un array de "num_puntos" datos "PuntoDeRecorrido"  
    int num_puntos;           // Número de puntos del recorrido  
    bool activo;              // Indica si el recorrido está activo  
public:  
    .....  
};
```

donde el tipo *PuntoDeRecorrido* se define como sigue:

```
struct PuntoDeRecorrido {  
    Punto     punto;  
    Instante instante;  
};
```

Al iniciar la actividad se crea un objeto de la clase Recorrido. En ese momento, el campo activo se pone a true y periódicamente se van añadiendo puntos -datos de la clase *PuntoDeRecorrido*- hasta que se da por finalizada la actividad. Entonces el campo activo se pone a false y se pueden calcular diferentes parámetros del recorrido.

◀ Ejercicio 1 ▷ Constructores y destructor. Método FinRecorrido

[1.5 puntos]

- Implemente un **constructor** para la clase Recorrido con un parámetro de tipo booleano de manera que:
 - si es false se crea un recorrido *nulo* (con ningún elemento *PuntoDeRecorrido*).
 - si es true indicará que el objeto que se va a crear debe incluir un primer elemento de tipo *PuntoDeRecorrido*, cuyos valores se calculan a partir de las coordenadas del punto en el que se está situado y con la fecha y hora actual. Para poder obtener estos valores suponga que existen las funciones globales:

```
Punto     GetPosicion (void); // (Clase Punto) Devuelve la posición actual  
Instante GetInstante (void); // (Clase Instante) Devuelve la fecha y hora actual
```

Note que en ambos casos el recorrido queda activo.

- Implemente el **destructor** de la clase Recorrido.
- Implemente el **método** FinRecorrido.

Termina un recorrido añadiendo el último dato de tipo *PuntoDeRecorrido* con la posición e instante actual y poniendo activo a false. **Muy importante:** este método no elimina el objeto, simplemente impide la adición de nuevos puntos.



▫ Ejercicio 2 ▷ Constructor de copia y operador de asignación

[1 punto]

Implemente el **constructor de copia** y el **operador de asignación** para la clase Recorrido.

¿Es necesario y/o conveniente implementar estos operadores para las clases secundarias? ¿Por qué?

▫ Ejercicio 3 ▷ Métodos DistanciaRecorrido, TiempoRecorrido y VelocidadMedia

[1 punto]

Escribir tres **métodos** de la clase Recorrido:

- **DistanciaRecorrido**: calcula y devuelve la distancia total recorrida (en metros). La distancia total se calcula como la suma de las distancias entre cada par de puntos consecutivos.
- **TiempoRecorrido**: calcula y devuelve el tiempo empleado en el recorrido (en segundos).
- **VelocidadMedia**: calcula y devuelve la velocidad media de un recorrido expresada en km/hora.

Suponga que existen los métodos de las clases Punto e Instante, respectivamente:

```
double DistanciaAPunto (const Punto & otro) const; // Distancia en metros a otro punto
double IntervaloTiempo (const Instante & otro) const; // Tiempo en segundos a otro instante
```

▫ Ejercicio 4 ▷ Almacenar y recuperar recorridos

[1.5 puntos]

Un recorrido se almacena en un fichero de **texto** con el siguiente formato:

```
RECORRIDO_MP
dia_inicio mes_inicio anio_inicio
hora_inicio minuto_inicio segundo_inicio
dia_fin mes_fin anio_fin
hora_fin minuto_fin segundo_fin
num_puntos
longitud_1 latitud_1 altura_1 dia_1 mes_1 anio_1 hora_1 minuto_1 segundo_1
longitud_2 latitud_2 altura_2 dia_2 mes_2 anio_2 hora_2 minuto_2 segundo_2
...
longitud_n latitud_n altura_n dia_n mes_n anio_n hora_n minuto_n segundo_n
```

Implemente dos **métodos** en la clase Recorrido para la lectura y escritura, respectivamente de recorridos.

En ambos casos, los métodos reciben el nombre del archivo (el argumento formal es una cadena clásica –tipo C–) y devuelven un booleano que indica si ha tenido éxito.

Nota: Suponga que existen los constructores con parámetros para las clases secundarias.

▫ Ejercicio 5 ▷ Uso de la clase

[1 punto]

Escriba un programa (`recorrido_mas_rapido.cpp`) que lea una serie de ficheros de recorridos e indique en cual de ellos se tiene la mayor velocidad media. Mostrará su *nombre* y el *valor* de la mayor velocidad media.

El programa debe recibir en la línea de órdenes nombres de fichero de recorrido (uno al menos).

Si alguno de los ficheros no puede procesarse, se pasará al siguiente (el programa no detiene su ejecución). Mostrar los mensajes de error adecuados a cada problema.



Normas para la realización del examen:

Duración: 1 hora

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

Los dispositivos GPS registran posiciones periódicamente y las almacenan. La posición de un punto está determinada por tres datos: su coordenada sobre la superficie terrestre (*latitud* y *longitud*) y su *altura*. Una sucesión de puntos registrados consecutivamente forman un recorrido (en inglés, *track*). A partir de un recorrido se pueden calcular distintas medidas que permiten analizarlo, visualizarlo en mapas, utilizarlo como guía para repetir el mismo recorrido, ...

Se diseña un sistema para trabajar con recorridos, incluyendo clases para gestionar tiempos y posiciones geográficas:

```
class Fecha {           class Hora {           class Instante {           class Punto {  
    private:             private:             private:  
        int dia;         int hora;         Fecha fecha;  
        int mes;         int minuto;       Hora hora;  
        int anio;        int segundo;      public:  
    public:               public:           .....  
    .....           .....           };           .....  
};           };           };           };  
};           };           };           };
```

Estos tipos se utilizan para poder crear el tipo de dato *principal* que será el encargado de gestionar recorridos. La representación propuesta es la siguiente:

```
class Recorrido {  
private:  
    PuntoDeRecorrido *puntos; // Puntero a un array de "num_puntos" datos "PuntoDeRecorrido"  
    int num_puntos;          // Número de puntos del recorrido  
    bool activo;             // Indica si el recorrido está activo  
public:  
    .....  
};
```

donde el tipo *PuntoDeRecorrido* se define como sigue:

```
struct PuntoDeRecorrido {  
    Punto punto;  
    Instante instante;  
};
```

Notas:

- La dinámica de funcionamiento será que al iniciar la actividad se crea un objeto de la clase *Recorrido*. En ese momento, el campo *activo* se pone a *true* y periódicamente se van añadiendo puntos –datos de la clase *PuntoDeRecorrido*– hasta que se da por finalizada la actividad. Entonces el campo *activo* se pone a *false* y se pueden calcular diferentes parámetros del recorrido.
- El alumno puede asumir que están resueltos e implementados –es decir, puede usar– los siguientes métodos de la clase *Recorrido*:
 - Constructor sin parámetros que crea un recorrido *nulo* (sin puntos).
 - Constructor de copias y operador de asignación.
 - Destructor.
 - *NumeroPuntos*: devuelve el número de puntos que hay almacenados.
 - *GetPunto*: recibe un índice *i* –desde 0 al número de puntos menos 1– para devolver el punto *i*-ésimo del recorrido.
 - *Finalizado*: devuelve si el recorrido se ha dado por finalizado (no está activo).
 - *DistanciaRecorrido*: calcula y devuelve la distancia total recorrida (en metros).
 - *TiempoRecorrido*: calcula y devuelve el tiempo empleado en el recorrido (en segundos).
 - *VelocidadMedia*: calcula y devuelve la velocidad media en un recorrido expresada en km/hora.
 - *Leer*: recibe un nombre de archivo –una Cadena-C– que contiene un recorrido y devuelve un booleano indicando si se ha cargado con éxito.



▷ Ejercicio 1 ▷ Módulo recorrido.h

[1 punto]

Suponemos que se dispone de los módulos que definen los tipos Fecha, Hora, Instante y Punto, para los que se han definido los correspondientes pares de archivos .h y .cpp. Teniendo en cuenta todos los tipos y métodos que se han listado anteriormente, escriba el correspondiente archivo de cabecera recorrido.h. **Notas:** Sólo debe incluir cabeceras, sin ninguna definición.

▷ Ejercicio 2 ▷ Ampliación de la biblioteca

[1 punto]

Se decide crear un nuevo módulo donde añadir nuevas utilidades para procesar recorridos. Para comenzarlo, se decide crear una función global PendienteMedia que recibe como parámetro un recorrido y devuelve la pendiente media del recorrido. La pendiente media es una media ponderada con la siguiente expresión:

$$\frac{1}{L} \sum_{i=1}^n l_i \cdot \text{pend}_i$$

donde L es la longitud total del recorrido, l_i es la longitud de cada segmento –tramo formado por dos puntos consecutivos del recorrido– y pend_i es la pendiente de ese segmento. En el caso de que un recorrido no tenga sentido –menos de dos puntos o esté aún activo– devolverá pendiente cero. Observe que la sumatoria incluye n segmentos, es decir, el recorrido contiene $n + 1$ puntos.

Implemente los dos módulos utils.h y utils.cpp donde se incluye esta función. Tenga en cuenta que es una función global que usa la clase Recorrido y que para implementarla dispone de las siguientes funciones:

```
double Distancia(const Punto& p1, const Punto& p2);  
double Pendiente(const Punto& p1, const Punto& p2);
```

que devuelven, respectivamente, la distancia en metros y la pendiente como porcentaje. Estas funciones ya se encuentran implementadas en el módulo diseñado para el tipo Punto.

▷ Ejercicio 3 ▷ Programa

[1 punto]

Escriba un programa intensidad.cpp que recibe en la línea de órdenes una serie de nombres de archivo que almacenan recorridos, y escribe en la salida estándar cada uno de esos nombres ordenados por su pendiente media. Un ejemplo de su ejecución es:

```
prompt> intensidad hornillo.trk trevenque.trk vega.trk  
Pendiente media 2.05%: vega.trk  
Pendiente media 15.75%: hornillo.trk  
Pentiente media 30.40%: trevenque.trk
```

Recuerde que en caso de error, el programa deberá mostrar los mensajes correspondientes.

▷ Ejercicio 4 ▷ makefile

[1 punto]

Teniendo en cuenta todos los módulos que se han comentado, escriba un archivo Makefile que permita generar el ejecutable del programa propuesto en la pregunta 3 simplemente escribiendo en la línea de órdenes la orden “make”. Incluya las correspondientes dependencias, así como un objetivo clean que permite limpiar los archivos intermedios.



Normas para la realización del examen:

Duración: 2.5 horas

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

△ Ejercicio 1 ▷ Vectores-C

[0.75 puntos]

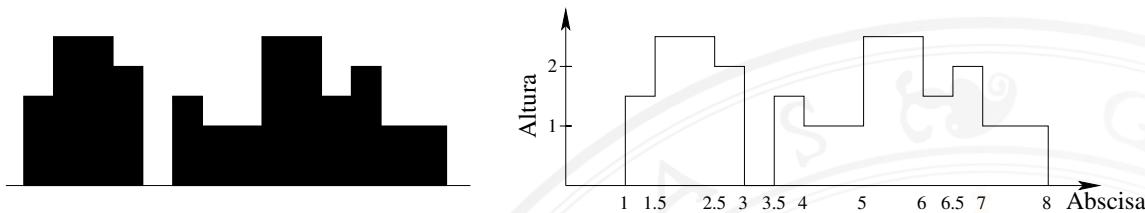
Implemente una función MezclarUnico de tipo int que recibe como entrada dos vectores-C de datos de tipo double y los mezcla en un tercer vector. Tenga en cuenta que:

- Los vectores de entrada están ordenados y sin valores repetidos.
- El vector de salida tendrá los elementos ordenados y sin repetidos.
- Puede asumir que el vector de salida tiene capacidad suficiente para todos los elementos.
- La función devuelve un entero que es el número de elementos que contiene el vector de salida. Nota: Será menor o igual que la suma de los de entrada.

△ Ejercicio 2 ▷ SkyLine: clase

[2.25 puntos]

Se define un SkyLine como la silueta urbana que refleja la vista general de los edificios de una ciudad. Se propone crear un programa para procesar SkyLines simplificados, donde los edificios no son más que un rectángulo que se eleva desde una línea base horizontal. La siguiente figura muestra un ejemplo:



donde podemos ver que se puede codificar guardando los valores de abscisas y alturas de cada cambio de altura.

Se desea crear un programa para procesar SkyLines. Para ello, se propone la siguiente representación:

```
class SkyLine {  
    double *abscisas; // abscisas[i] > abscisas[i-1]  
    double *alturas; // alturas[i]>=0 && alturas[n-1] == 0  
    int n; // tamaño de los vectores anteriores.  
public:  
    // ... interfaz pública de la clase  
};
```

En el ejemplo de la figura anterior, el valor de n sería 11 y los dos vectores de este tamaño contendrían:

- Abscisas: 1, 1.5, 2.5, 3, 3.5, 4, 5, 6, 6.5, 7, 8.
- Alturas: 1.5, 2.5, 2, 0, 1.5, 1, 2.5, 1.5, 2, 1, 0.

Defina las siguientes funciones:

1. (0.5 puntos) El constructor por defecto y el destructor. El constructor por defecto crea el SkyLine nulo, representado con todos sus campos a cero.
2. (0.75 puntos) Un constructor para un edificio. Recibe la descripción de un edificio compuesta por tres valores: dos abscisas y una altura; construye el skyline correspondiente.
3. (1 punto) El constructor de copia y operador de asignación.

△ Ejercicio 3 ▷ Skyline: miembro público

[0.5 puntos]

Considere la clase Skyline anterior. Se decide añadir una función para calcular la altura del skyline en cualquier punto. Implemente una función miembro Altura que recibe una abscisa y devuelve la altura en ese punto. Tenga en cuenta que la altura fuera del skyline es cero.



ugr

▷ Ejercicio 4 ▷ Skyline: Sobrecarga de operadores

[1.5 puntos]

Considere la clase Skyline anterior. Implemente las siguientes funciones miembro:

1. (0.75 puntos) Sobrecarga del operador de suma para poder obtener el skyline que corresponde a superponer dos skylines. El algoritmo consiste en crear las abscisas como la mezcla de los dos vectores abscisas de entrada y situar las alturas como el máximo de entre los dos skylines de entrada. Puede considerar resueltas y usar las funciones de los ejercicios 1 y 3.
2. (0.25 puntos) Sobrecarga del operador de salida << para poder obtener en un flujo de salida el skyline en formato texto. En concreto, deberá aparecer el número de abscisas y, en distintas líneas, cada una de las parejas que componen los valores de abscisa y altura.
3. (0.5 puntos) Sobrecarga del operador de entrada >> para poder obtener desde un flujo de entrada el skyline en formato texto. Suponga que el formato está compuesto por el número de abscisas y cada una de las parejas que componen los valores de abscisa y altura; todos los valores separados por "espacios blancos".

▷ Ejercicio 5 ▷ Skyline: Funciones externas y ficheros

[1 punto]

Se desea añadir la posibilidad de cargar/salvar el contenido de un skyline desde/a un fichero. Para ello, se establece un formato de texto que consiste en:

- Una cadena mágica "MP-SKYLINE-1.0" y un salto de línea para poder distinguir que es un fichero tipo skyline.
- El skyline en formato de texto tal como se indica en el ejercicio 4.

Considere que tiene disponibles las funciones del ejercicio 4. Implemente las siguientes funciones externas:

1. Una función Cargar que recibe un nombre de archivo y un objeto de tipo SkyLine y carga en éste el contenido del archivo. Devuelve si ha tenido éxito.
2. Una función Salvar que recibe un nombre de archivo y un objeto de tipo Skyline y lo salva en el archivo. Devuelve si ha tenido éxito.



Normas para la realización del examen:

Duración: 3.5 horas

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

△ Ejercicio 1 ▷ Vectores-C

[0.75 puntos]

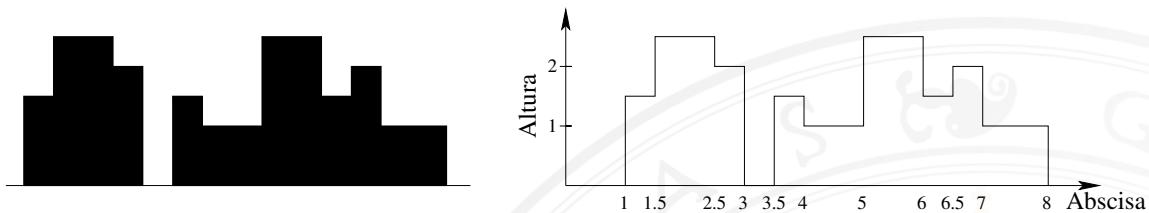
Implemente una función MezclarUnico de tipo int que recibe como entrada dos vectores-C de datos de tipo double y los mezcla en un tercer vector. Tenga en cuenta que:

- Los vectores de entrada están ordenados y sin valores repetidos.
- El vector de salida tendrá los elementos ordenados y sin repetidos.
- Puede asumir que el vector de salida tiene capacidad suficiente para todos los elementos.
- La función devuelve un entero que es el número de elementos que contiene el vector de salida. Nota: Será menor o igual que la suma de los de entrada.

△ Ejercicio 2 ▷ SkyLine: clase

[2.25 puntos]

Se define un SkyLine como la silueta urbana que refleja la vista general de los edificios de una ciudad. Se propone crear un programa para procesar SkyLines simplificados, donde los edificios no son más que un rectángulo que se eleva desde una línea base horizontal. La siguiente figura muestra un ejemplo:



donde podemos ver que se puede codificar guardando los valores de abscisas y alturas de cada cambio de altura.

Se desea crear un programa para procesar SkyLines. Para ello, se propone la siguiente representación:

```
class SkyLine {  
    double *abscisas; // abscisas[i] > abscisas[i-1]  
    double *alturas; // alturas[i]>=0 && alturas[n-1] == 0  
    int n; // tamaño de los vectores anteriores.  
public:  
    // ... interfaz pública de la clase  
};
```

En el ejemplo de la figura anterior, el valor de n sería 11 y los dos vectores de este tamaño contendrían:

- Abscisas: 1, 1.5, 2.5, 3, 3.5, 4, 5, 6, 6.5, 7, 8.
- Alturas: 1.5, 2.5, 2, 0, 1.5, 1, 2.5, 1.5, 2, 1, 0.

Defina las siguientes funciones:

1. (0.5 puntos) El constructor por defecto y el destructor. El constructor por defecto crea el SkyLine nulo, representado con todos sus campos a cero.
2. (0.75 puntos) Un constructor para un edificio. Recibe la descripción de un edificio compuesta por tres valores: dos abscisas y una altura; construye el skyline correspondiente.
3. (1 punto) El constructor de copia y operador de asignación.

△ Ejercicio 3 ▷ Skyline: miembro público

[0.5 puntos]

Considere la clase Skyline anterior. Se decide añadir una función para calcular la altura del skyline en cualquier punto. Implemente una función miembro Altura que recibe una abscisa y devuelve la altura en ese punto. Tenga en cuenta que la altura fuera del skyline es cero.



◀ **Ejercicio 4 ▷ Skyline: Sobrecarga de operadores**

[1.5 puntos]

Considere la clase `Skyline` anterior. Implemente las siguientes funciones miembro:

1. (0.75 puntos) Sobrecarga del operador de suma para poder obtener el skyline que corresponde a superponer dos skylines. El algoritmo consiste en crear las abscisas como la mezcla de los dos vectores abscisas de entrada y situar las alturas como el máximo de entre los dos skylines de entrada. Puede considerar resueltas y usar las funciones de los ejercicios 1 y 3.
2. (0.25 puntos) Sobrecarga del operador de salida << para poder obtener en un flujo de salida el skyline en formato texto. En concreto, deberá aparecer el número de abscisas y, en distintas líneas, cada una de las parejas que componen los valores de abscisa y altura.
3. (0.5 puntos) Sobrecarga del operador de entrada >> para poder obtener desde un flujo de entrada el skyline en formato texto. Suponga que el formato está compuesto por el número de abscisas y cada una de las parejas que componen los valores de abscisa y altura; todos los valores separados por “espacios blancos”.

◀ **Ejercicio 5 ▷ Skyline: Funciones externas y ficheros**

[1 punto]

Se desea añadir la posibilidad de cargar/salvar el contenido de un skyline desde/a un fichero. Para ello, se establece un formato de texto que consiste en:

- Una cadena mágica "MP-SKYLINE-1.0" y un salto de línea para poder distinguir que es un fichero tipo skyline.
- El skyline en formato de texto tal como se indica en el ejercicio 4.

Considere que tiene disponibles las funciones del ejercicio 4. Implemente las siguientes funciones externas:

1. Una función `Cargar` que recibe un nombre de archivo y un objeto de tipo `SkyLine` y carga en éste el contenido del archivo. Devuelve si ha tenido éxito.
2. Una función `Salvar` que recibe un nombre de archivo y un objeto de tipo `Skyline` y lo salva en el archivo. Devuelve si ha tenido éxito.

◀ **Ejercicio 6 ▷ Skyline: uso de la clase**

[1 punto]

Considere la clase `Skyline` anterior. Escriba un programa —`montar.cpp`— que lee desde la entrada estándar una secuencia de edificios y escribe en un fichero el skyline resultado. Tenga en cuenta que:

- En la entrada, cada edificio se especifica como dos valores que indican las abscisas de la base más un tercero para indicar la altura. La entrada termina cuando se introduce una base con dos valores idénticos.
- El nombre del fichero resultado se especifica en la línea de órdenes. Puede suponer resuelta y usar la función de salvado del ejercicio 5.

◀ **Ejercicio 7 ▷ Skyline: Modificando la parte interna**

[1 punto]

El algoritmo de suma de dos skylines propuesto da lugar a un resultado que puede contener varios valores de altura consecutivos idénticos. En este caso, se puede simplificar manteniendo únicamente el primero de ellos. Escriba una función miembro `Simplificar` privada para minimizar el número de puntos almacenados en el skyline y así el espacio que ocupa. Implemente esta función miembro tal y como aparecería en el archivo `cpp`.

◀ **Ejercicio 8 ▷ Skyline: módulos**

[1 punto]

Considere la clase `Skyline` anterior. Escriba el contenido del archivo de cabecera `skyline.h`, donde aparece todo lo necesario para usar la clase, sin incluir ninguna definición de función (suponga que estarán en `skyline.cpp`).

◀ **Ejercicio 9 ▷ Skyline: makefile**

[1 punto]

Considere los archivos que implementan el tipo `skyline` así como el programa `montar.cpp` anterior. Escriba el archivo `Makefile` que permite compilar el programa. Incluya las correspondientes dependencias, así como un objetivo `clean` que permite limpiar los archivos intermedios.



Normas para la realización del examen:

Duración: 2.5 horas

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

Se pretende implementar una clase que permita representar de forma eficiente matrices dispersas, donde sólo un número relativamente bajo de valores son distintos de cero. En este caso sólo es necesario almacenar información de aquellos valores significativos, no cero. Para cada valor significativo hay que almacenar: **fila**, **columna** (de tipo *int*) y **valor** (de tipo *double*). Esta información puede gestionarse de la forma que consideréis oportuna (mediante estructura o clase; asumamos que su nombre es **Valor** y están disponibles ya todos los métodos de asignación y acceso a sus datos). De esta forma se propone la siguiente representación para la clase:

```
class MatrizDispersa {  
    private:  
        int nfilas;           // Número de filas  
        int ncolumnas;        // Número de columnas  
        Valor *valores;       // Vector-C no ordenado, solo valores significativos  
        int numeroValores;   // Número de valores significativos almacenados  
    public:  
        // ... interfaz pública de la clase  
};
```

NOTA: los objetos de tipo **Valor** no tienen por qué estar almacenados por orden de filas y columnas en el vector-C **valores**. A modo de ejemplo consideremos la matriz dispersa a la derecha. Su tratamiento requiere de un vector-C con 4 datos de tipo **Valor**, con la información:

fila = 1, columna = 1, valor = 1.0	1.0	0.0	0.0	0.0
fila = 2, columna = 2, valor = 2.0	0.0	2.0	0.0	0.0
fila = 3, columna = 3, valor = 3.0	0.0	0.0	3.0	0.0
fila = 4, columna = 4, valor = 4.0	0.0	0.0	0.0	4.0

▷ Ejercicio 1 ▷ Vectores-C y función externa a la clase

[0.75 puntos]

Implemente la función externa **combinarSuma** que reciba como entrada dos vectores-C de datos de tipo **Valor** y produce como resultado un tercer vector del mismo tipo (puede asumir que el vector de salida tiene espacio suficiente para almacenar todos los valores que se produzcan). Tenga en cuenta que:

- el valor de una posición se determina de la siguiente forma: si un valor aparece sólo en un vector de entrada, entonces mantiene su valor; si aparece en ambos vectores, entonces el valor almacenado será la suma de ambos (incluso si la suma es 0).
- la función puede contener los argumentos que considere necesarios para su funcionamiento.
- la función devolverá el número de datos de tipo **Valor** almacenados en el vector-C de resultado.
- puede implementar funciones auxiliares si se estima oportuno.

▷ Ejercicio 2 ▷ Métodos básicos de la clase

[2 puntos]

Defina los siguientes métodos para la clase **MatrizDispersa**:

1. (0.5 puntos) **constructor por defecto** y **destructor**. El constructor por defecto crea una matriz sin elementos (0 filas, 0 columnas, 0 valores).
2. (0.5 puntos) **constructor con parámetros**. Recibe dos enteros con las dimensiones de la matriz, junto con tres vectores-C y un entero que indica sus tamaños (total: 6 parámetros). Cada posición de los vectores corresponde a una tripleta fila/columna/valor de la matriz. Se asume que no habrá dos tripletas que correspondan al mismo par fila/columna, que las parejas fila/columna están dentro de las dimensiones indicadas y que ningún valor es igual a cero.
3. (1 punto) **constructor de copia** y **operador de asignación**.



UGR

△ Ejercicio 3 ▷ Obtención y asignación de valor

[1 punto]

Se decide añadir a la clase **MatrizDispersa** los métodos:

- (0.5 puntos) **obtenerValor**: permite recuperar el valor asociado a una determinada posición. Recibe la fila y columna de interés; devuelve 0 en caso de no haber valor significativo en esa posición ((1, 2) en el ejemplo anterior) o el valor almacenado (4.0 para posición (4, 4)). Se supone que los valores de fila y columna siempre serán válidos.
- (0.5 puntos) **asignarValor**: permite modificar el contenido de la matriz. Recibe la fila y columna de interés y el valor de dicha posición. Si existe un valor para dicha posición, lo sustituye (a menos que el nuevo valor a asignar sea 0, en cuyo caso se debe reducir el número de valores significativos de la matriz). Si se trata de una posición no asignada previamente, se agrega el nuevo valor (siempre que sea distinto de 0).

△ Ejercicio 4 ▷ Sobrecarga de operadores

[1.25 puntos]

Implemente la siguiente funcionalidad para la clase **MatrizDispersa**:

1. (0.5 puntos) **operador de suma** para poder obtener la suma de dos matrices dispersas. Puede usar las funciones pedidas en los ejercicios previos (**combinarSuma** y **asignarValor**). Se asume que las matrices sumadas tienen el mismo número de filas y columnas.
2. (0.25 puntos) **operador de salida <<** para poder obtener en un flujo de salida el contenido de la matriz en formato texto. En concreto, mostrará: número de filas, número de columnas, número de valores significativos y, en distintas líneas, el contenido de cada objeto de tipo **Valor**: fila, columna y valor.
3. (0.5 puntos) **operador de entrada >>** para poder obtener desde un flujo de entrada el contenido de un objeto. Se asume el mismo formato indicado anteriormente.

△ Ejercicio 5 ▷ Funciones externas y ficheros

[1 punto]

Se desea añadir la posibilidad de cargar y salvar el contenido de una matriz dispersa desde fichero. Para ello, se establece un formato de texto que consiste en:

- una cadena mágica "MP-MATRIZDISPERSA-1.0" y un salto de línea (para poder distinguir que es un fichero usado para almacenar información de objetos de la clase de interés).
- el contenido de la matriz en formato de texto, tal como se indica en el ejercicio 4.

Considerando que tiene disponibles los métodos de los ejercicios previos implemente las siguientes funciones externas:

1. (0.5 puntos) función **cargar**, que recibe un nombre de archivo y un objeto de tipo **MatrizDispersa** y carga en éste el contenido del archivo. Devuelve si la operación ha tenido éxito.
2. (0.5 puntos) función **salvar**, que recibe un nombre de archivo y un objeto de tipo **MatrizDispersa** y lo salva en el archivo. Devuelve si la operación ha tenido éxito.



Normas para la realización del examen:

Duración: 1 hora

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

Se pretende implementar una clase que permita representar de forma eficiente matrices dispersas, donde sólo un número relativamente bajo de valores son distintos de cero. En este caso sólo es necesario almacenar información de aquellos valores significativos, no cero. Para cada valor significativo hay que almacenar: **fila**, **columna** (de tipo *int*) y **valor** (de tipo *double*). Esta información puede gestionarse de la forma que consideréis oportuna (mediante estructura o clase; asumamos que su nombre es **Valor** y están disponibles ya todos los métodos de asignación y acceso a sus datos). De esta forma se propone la siguiente representación para la clase:

```
class MatrizDispersa {  
    private:  
        int nfilas;           // Número de filas  
        int ncolumnas;        // Número de columnas  
        Valor *valores;      // Vector-C no ordenado, solo valores significativos  
        int numeroValores;   // Número de valores significativos almacenados  
    public:  
        // ... interfaz pública de la clase  
};
```

NOTA: los objetos de tipo **Valor** no tienen por qué estar almacenados por orden de filas y columnas en el vector-C **valores**. A modo de ejemplo consideremos la matriz dispersa a la derecha. Su tratamiento requiere de un vector-C con 4 datos de tipo **Valor**, con la información:

fila = 1, columna = 1, valor = 1.0	1.0	0.0	0.0	0.0
fila = 2, columna = 2, valor = 2.0	0.0	2.0	0.0	0.0
fila = 3, columna = 3, valor = 3.0	0.0	0.0	3.0	0.0
fila = 4, columna = 4, valor = 4.0	0.0	0.0	0.0	4.0

△ Ejercicio 1 ▷ Vectores-C y función externa a la clase

Implemente la función externa **combinarSuma** que reciba como entrada dos vectores-C de datos de tipo **Valor** y produce como resultado un tercer vector del mismo tipo (puede asumir que el vector de salida tiene espacio suficiente para almacenar todos los valores que se produzcan). Tenga en cuenta que:

- el valor de una posición se determina de la siguiente forma: si un valor aparece sólo en un vector de entrada, entonces mantiene su valor; si aparece en ambos vectores, entonces el valor almacenado será la suma de ambos (incluso si la suma es 0).
- la función puede contener los argumentos que considere necesarios para su funcionamiento.
- la función devolverá el número de datos de tipo **Valor** almacenados en el vector-C de resultado.
- puede implementar funciones auxiliares si se estima oportuno.

△ Ejercicio 2 ▷ Métodos básicos de la clase

Defina los siguientes métodos para la clase **MatrizDispersa**:

1. **constructor por defecto** y **destructor**. El constructor por defecto crea una matriz sin elementos (0 filas, 0 columnas, 0 valores).
2. **constructor con parámetros**. Recibe dos enteros con las dimensiones de la matriz, junto con tres vectores-C y un entero que indica sus tamaños (total: 6 parámetros). Cada posición de los vectores corresponde a una tripleta fila/columna/valor de la matriz. Se asume que no habrá dos tripletas que correspondan al mismo par fila/columna, que las parejas fila/columna están dentro de las dimensiones indicadas y que ningún valor es igual a cero.
3. (1 punto) **constructor de copia** y **operador de asignación**.

△ Ejercicio 3 ▷ Obtención y asignación de valor

Se decide añadir a la clase **MatrizDispersa** los métodos:

- **obtenerValor**: permite recuperar el valor asociado a una determinada posición. Recibe la fila y columna de interés; devuelve 0 en caso de no haber valor significativo en esa posición ((1, 2) en el ejemplo anterior) o el valor almacenado (4.0 para posición (4, 4)). Se supone que los valores de fila y columna siempre serán válidos.



UGR

- **asignarValor**: permite modificar el contenido de la matriz. Recibe la fila y columna de interés y el valor de dicha posición. Si existe un valor para dicha posición, lo sustituye (a menos que el nuevo valor a asignar sea 0, en cuyo caso se debe reducir el número de valores significativos de la matriz). Si se trata de una posición no asignada previamente, se agrega el nuevo valor (siempre que sea distinto de 0).

◀ Ejercicio 4 ▷ Sobrecarga de operadores

Implemente la siguiente funcionalidad para la clase **MatrizDispersa**:

1. **operador de suma** para poder obtener la suma de dos matrices dispersas. Puede usar las funciones pedidas en los ejercicios previos (**combinarSuma** y **asignarValor**). Se asume que las matrices sumadas tienen el mismo número de filas y columnas.
2. **operador de salida <<** para poder obtener en un flujo de salida el contenido de la matriz en formato texto. En concreto, mostrará: número de filas, número de columnas, número de valores significativos y, en distintas líneas, el contenido de cada objeto de tipo **Valor**: fila, columna y valor.
3. **operador de entrada >>** para poder obtener desde un flujo de entrada el contenido de un objeto. Se asume el mismo formato indicado anteriormente.

◀ Ejercicio 5 ▷ Funciones externas y ficheros

Se desea añadir la posibilidad de cargar y salvar el contenido de una matriz dispersa desde fichero. Para ello, se establece un formato de texto que consiste en:

- una cadena mágica "MP-MATRIZDISPERSA-1.0" y un salto de línea (para poder distinguir que es un fichero usado para almacenar información de objetos de la clase de interés).
- el contenido de la matriz en formato de texto, tal como se indica en el ejercicio 4.

Considerando que tiene disponibles los métodos de los ejercicios previos implemente las siguientes funciones externas:

1. función **cargar**, que recibe un nombre de archivo y un objeto de tipo **MatrizDispersa** y carga en éste el contenido del archivo. Devuelve si la operación ha tenido éxito.
2. función **salvar**, que recibe un nombre de archivo y un objeto de tipo **MatrizDispersa** y lo salva en el archivo. Devuelve si la operación ha tenido éxito.

◀ Ejercicio 6 ▷ Ampliación de la clase

[1 punto]

Implemente un nuevo constructor de la clase **MatrizDispersa** que reciba como parámetros las filas y columnas y un tercer parámetro opcional, que es un puntero a los datos. Este puntero tiene 0 como valor por defecto, en cuyo caso representa una matriz con todos los valores a 0.

◀ Ejercicio 7 ▷ Uso de la clase

[1 punto]

Escriba un programa (**principal.cpp**) que lea desde la entrada estándar una secuencia con el contenido de la matriz y genere el archivo correspondiente. El nombre del archivo se especifica en la línea de órdenes. Recuerde que puede asumir resueltas y disponibles las funciones y métodos mencionados en los ejercicios previos.

◀ Ejercicio 8 ▷ Método de poda

[1 punto]

Implemente un método llamado **podar**, que produce un nuevo objeto de la clase, mediante la siguiente simplificación: todos aquellos valores que estén por debajo de un cierto valor umbral se eliminan (reduciéndose así el número de valores almacenados en la matriz). Se valorará la eficiencia del método.

◀ Ejercicio 9 ▷ Organización en módulos y Makefile

[1 punto]

1. (0.5 puntos) Escriba el contenido del archivo de cabecera **matrizDispersa.h**, donde debe aparecer todo lo necesario para usar la clase, sin incluir ninguna definición de función (suponga que estarán en **matrizDispersa.cpp**).
2. (0.5 puntos) Considere los archivos que implementan el tipo **MatrizDispersa**, así como el programa **principal.cpp** indicado con anterioridad. Escriba el archivo **Makefile** que permita la compilación del programa. Incluya las dependencias necesarias y un objetivo **clean** que permita eliminar los archivos intermedios generados en el proceso de compilación.



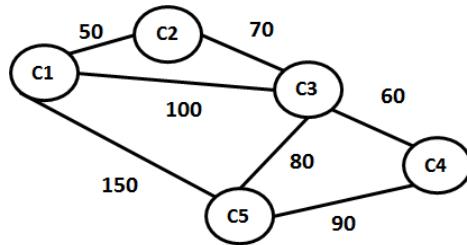
UGR

Normas para la realización del examen:

Duración: 3 horas

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

Se desea construir la clase RedCiudades para almacenar datos de un conjunto de ciudades, y las distancias de los caminos directos que las conectan. Si entre dos ciudades no existe un camino directo, se almacenará un cero. Se supone que la distancia de una ciudad consigo misma será cero y que las distancias son simétricas. Un ejemplo con 5 ciudades sería:



	C1	C2	C3	C4	C5
C1	0	50	100	0	150
C2	50	0	70	0	0
C3	100	70	0	60	80
C4	0	0	60	0	90
C5	150	0	80	90	0

Se propone la siguiente representación para la clase:

```
class RedCiudades {  
private:  
    int num_ciudades; // Número de ciudades  
    InfoCiudad * info; // info[i]: info de la ciudad i  
    double ** distancia; // distancia[i][j]: distancia  
                          // entre las ciudades i y j  
public:  
    // ... interfaz pública de la clase
```

```
        struct InfoCiudad {  
            char * nombre; // Nombre  
            int poblacion; // Num. habs.  
        };
```

Suponga que dispone de los métodos públicos de consulta:

- NumCiudades: devuelve el número de ciudades.
- Distancia: devuelve la distancia entre dos ciudades.
- NombreCiudad: devuelve el nombre de una ciudad (en realidad, la dirección de inicio).
- PoblacionCiudad: devuelve número de habitantes de una ciudad.

Para la realización de los ejercicios propuestos deberá escribir en un folio aparte el fichero RedCiudades.h.

Entenderemos que los métodos pedidos en los ejercicios 1, 2, 3 y 4 serán parte de RedCiudades.cpp y como tales deberá escribirlos.

▷ Ejercicio 1 ▷ Métodos básicos de la clase

[1.25 puntos]

Defina los siguientes métodos para la clase RedCiudades:

1. (0.5 puntos) Constructor por defecto y destructor. El constructor por defecto crea una red vacía (escriba también el método EstaVacia que indique si una red está vacía).
2. (0.75 puntos) Constructor de copia y operador de asignación.



Ugr

△ Ejercicio 2 ▷ Sobre carga de operadores

[1.0 puntos]

Implemente la siguiente funcionalidad para la clase sobrecargando los operadores << y >>:

1. (0.25 puntos) operador de salida << para poder insertar en un flujo de salida el contenido de una red en formato texto. En concreto, deberá aparecer:
 - un número entero que indica el número de ciudades y un salto de línea,
 - una serie de líneas (tantas como ciudades) que contienen el índice de la ciudad (un número entero), su nombre (una serie de caracteres sin espacios intermedios) y su población (un número entero),
 - una serie de líneas (tantas como conexiones entre ciudades) que contienen dos números enteros (números de las ciudades conectadas) y un número real (distancia entre ellas). **Nota:** Cada conexión entre ciudades se escribe una sola vez (no importa cuál es la ciudad de origen o destino).
2. (0.75 puntos) operador de entrada >> para poder obtener desde un flujo de entrada el contenido de un objeto. Se asume el mismo formato indicado en el ejercicio anterior.

△ Ejercicio 3 ▷ Métodos y funciones para E/S

[1.25 puntos]

1. (0.5 puntos) Constructor con argumento. Recibe el nombre de un fichero de **texto** con la información sobre una red. El fichero debe contener en la primera línea la *cadena mágica* "RED" y un salto de línea. A continuación está el contenido de la red en formato de texto, tal como se indica en el ejercicio 2.
2. (0.5 puntos) Método LeerRedCiudades que permite actualizar el contenido de una red con los datos de un fichero de texto con el formato conocido. El contenido de la red será sustituido por los datos leídos.
3. (0.25 puntos) Método EscribirRedCiudades que permite guardar el contenido de una red en un fichero de texto, que tendrá el formato descrito en el apartado anterior.

△ Ejercicio 4 ▷ Métodos de cálculo

[1.25 puntos]

1. (0.5 puntos) Implemente el método CiudadMejorConectada que permita obtener la ciudad (su índice) con mayor número de conexiones directas.
En el ejemplo la ciudad mejor conectada sería la ciudad C3 con 4 conexiones.
2. (0.75 puntos) Implemente el método MejorEscalaEntre para que, dadas dos ciudades i y j no conectadas directamente, devuelva aquella ciudad intermedia z que permita hacer el trayecto entre i y j de la forma más económica posible. Es decir, se trata de encontrar una ciudad z tal que $d(i, z) + d(z, j)$ sea mínima ($d(a, b)$ es la distancia entre las ciudades a y b). El método devuelve -1 si no existe dicha ciudad intermedia.
Por ejemplo, si se desea viajar desde la ciudad C2 a la C5, hacerlo a través de la ciudad C1 tiene un costo de $50 + 150 = 200$ mientras que si se hace a través de la ciudad C3, el costo sería $70 + 80 = 150$.

△ Ejercicio 5 ▷ Aplicación

[1.25 puntos]

Escriba un programa *completo* que reciba desde la línea de órdenes el nombre de un fichero de texto que contiene la descripción de una red (como se indica en el ejercicio 3) y calcule, para todas las parejas de ciudades que *no* estén directamente conectadas, cuál es la mejor escala con una sola ciudad intermedia (su índice). Si no la tuviera deberá indicarlo.



Normas para la realización del examen:

Duración: 2.5 horas

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

Se desea crear un programa para contabilizar el número de repeticiones de cada palabra en un determinado texto.

Para resolver el problema, se propone usar dos tipos de datos:

```
class Entrada {  
    char* palabra; // Palabra (cadena-C reservada dinámicamente)  
    int frecuencia; // Número de repeticiones de la palabra  
  
public:  
    // ... interfaz pública de la clase  
}  
  
class Frecuencias {  
    Entrada* entradas; // array dinámico de Entradas  
    int nEntradas;  
  
public:  
    // ... interfaz pública de la clase  
}
```

Un objeto Entrada representa una palabra (como cadena-C) y el número de veces (frecuencia) que aparece en un determinado texto. Un objeto Frecuencias representa una colección de entradas, en la que cada una representa una palabra y su frecuencia en un determinado texto. El dato miembro entradas de la clase Frecuencias es un array que contiene datos de tipo Entrada **ordenados** por el campo palabra de las entradas (alfabéticamente), o bien 0 si no hay ninguna entrada en el objeto Frecuencias.

El array dinámico de un objeto Frecuencias debe tener siempre el **tamaño justo** para contener el número de entradas añadidas al array hasta el momento.

◀ **Ejercicio 1 ▷ Vectores-C**

[1 punto]

Implemente una función externa mezclarEntradas de tipo int que reciba como parámetro dos vectores-C ordenados de datos de tipo Entrada y los mezcle en un tercer vector. Cuando una determinada cadena aparece en los dos vectores de entradas, en la salida solo aparecerá una vez, con un valor frecuencia calculado como la suma de las dos frecuencias de entrada. Tenga en cuenta que:

- Puede suponer implementadas en la clase Entrada los métodos getFrecuencia y setFrecuencia.
- Los vectores de entrada están ordenados por la cadena de Entrada y sin valores repetidos.
- El vector de salida debe tener los elementos ordenados por la cadena de Entrada y sin repetidos.
- Puede asumir que el vector de salida tiene capacidad suficiente para todos los elementos.
- La función devuelve un entero que es el número de elementos que contiene el vector de salida. Nota: será menor o igual que la suma de los de entrada.

◀ **Ejercicio 2 ▷ Métodos básicos de la clase**

[1.5 puntos]

Defina los siguientes métodos para la clase Frecuencias. Defina además los métodos necesarios en la clase Entrada para su correcto funcionamiento.

1. (0.5 puntos) Constructor por defecto y destructor. El constructor por defecto en la clase Entrada crea un objeto cuya palabra representa la cadena-C vacía, y tiene una frecuencia igual a 0.
2. (1 punto) Constructor de copia y operador de asignación.



Ugr

△ Ejercicio 3 ▷ Otros métodos de la clase

[1.25 puntos]

Añade a la clase Frecuencias los siguientes métodos públicos:

1. (0.5 puntos) Un método `repeticiones` de tipo `int` que reciba como parámetro una cadena-C (palabra) y devuelva la frecuencia asociada a esa cadena en el objeto Frecuencias o bien 0 si la cadena no es encontrada. Puede asumir que tiene disponible el método `int buscar(const char* palabra)` en la clase Frecuencias. Este método devuelve la posición en el vector de entradas, donde está esa palabra, o en caso de que la palabra no esté en el vector, la posición donde debería insertarse para que el vector siguiese estando ordenado.
2. (0.75 puntos) Un método `insertar` que reciba como parámetro una cadena-C (palabra). Si la palabra ya estaba en el vector de entradas, sumará 1 a su frecuencia. Si la palabra no estaba en el vector de entradas, insertará en el lugar adecuado, una nueva entrada cuya palabra será una copia en memoria dinámica de la palabra recibida y una frecuencia igual a 1.

△ Ejercicio 4 ▷ Sobrecarga de operadores

[1.5 puntos]

Implemente la siguiente funcionalidad para la clase Frecuencias sobrecargando los operadores `<<` y `>>` y de suma. En caso necesario, implemente también los métodos necesarios de la clase Entrada.

1. (0.25 puntos) Operador de salida `<<` para poder insertar en un flujo de salida el contenido de un objeto Frecuencias. En concreto, deberá aparecer:
 - un número entero que indica el número de entradas y un salto de línea,
 - una serie de líneas (tantas como entradas) que contienen la palabra, un espacio en blanco y la frecuencia (un número entero).
2. (0.5 puntos) Operador de entrada `>>` para poder obtener desde un flujo de entrada el contenido de un objeto. Se asume el mismo formato indicado en el apartado anterior.
3. (0.75 puntos) Operador de suma para obtener el objeto Frecuencias que se obtiene a partir de la la suma de dos objetos Frecuencias. La frecuencia de una palabra en el objeto resultado se obtiene como la suma de las frecuencias de los dos objetos de entrada. Puede considerar resueltas y usar las funciones de los ejercicios anteriores.

△ Ejercicio 5 ▷ Funciones externas para E/S

[0.75 puntos]

Se desea añadir la posibilidad de cargar/salvar el contenido de un objeto Frecuencias desde/a un fichero. Para ello, se establece un formato de texto que consiste en:

- Una cadena mágica "MP-FRECUENCIAS-1.0" y un salto de línea para poder distinguir que es un fichero tipo Frecuencias.
- El objeto Frecuencias en formato de texto tal como se indica en el ejercicio 4.

Considere que tiene disponibles las funciones del ejercicio 4. Implemente las siguientes funciones externas:

1. Función `cargar` que reciba un nombre de archivo y un objeto de tipo Frecuencias y cargue en éste el contenido del archivo. Debe devolver si ha tenido éxito o no.
2. Función `salvar` que reciba un nombre de archivo y un objeto de tipo Frecuencias y lo salve en el archivo. Debe devolver si ha tenido éxito o no.



Normas para la realización del examen:

Duración: 1 hora

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

Se desea crear un programa para contabilizar el número de repeticiones de cada palabra en un determinado texto.

Para resolver el problema, se propone usar dos tipos de datos:

```
class Entrada {  
    char* palabra; // Palabra (cadena-C reservada dinámicamente)  
    int frecuencia; // Número de repeticiones de la palabra  
  
public:  
    // ... interfaz pública de la clase  
}  
  
class Frecuencias {  
    Entrada* entradas; // array dinámico de Entradas  
    int nEntradas;  
  
public:  
    // ... interfaz pública de la clase  
}
```

Un objeto Entrada representa una palabra (como cadena-C) y el número de veces (frecuencia) que aparece en un determinado texto. Un objeto Frecuencias representa una colección de entradas, en la que cada una representa una palabra y su frecuencia en un determinado texto. El dato miembro entradas de la clase Frecuencias es un array que contiene datos de tipo Entrada **ordenados** por el campo palabra de las entradas (alfabéticamente), o bien 0 si no hay ninguna entrada en el objeto Frecuencias.

El array dinámico de un objeto Frecuencias debe tener siempre el **tamaño justo** para contener el número de entradas añadidas al array hasta el momento.

▷ Ejercicio 1 ▷ Vectores-C

Implemente una función externa mezclarEntradas de tipo int que reciba como parámetro dos vectores-C ordenados de datos de tipo Entrada y los mezcle en un tercer vector. Cuando una determinada cadena aparece en los dos vectores de entradas, en la salida solo aparecerá una vez, con un valor frecuencia calculado como la suma de las dos frecuencias de entrada. Tenga en cuenta que:

- Puede suponer implementadas en la clase Entrada los métodos getFrecuencia y setFrecuencia.
- Los vectores de entrada están ordenados por la cadena de Entrada y sin valores repetidos.
- El vector de salida debe tener los elementos ordenados por la cadena de Entrada y sin repetidos.
- Puede asumir que el vector de salida tiene capacidad suficiente para todos los elementos.
- La función devuelve un entero que es el número de elementos que contiene el vector de salida. Nota: será menor o igual que la suma de los de entrada.

▷ Ejercicio 2 ▷ Métodos básicos de la clase

Defina los siguientes métodos para la clase Frecuencias. Defina además los métodos necesarios en la clase Entrada para su correcto funcionamiento.

1. Constructor por defecto y destructor. El constructor por defecto en la clase Entrada crea un objeto cuya palabra representa la cadena-C vacía, y tiene una frecuencia igual a 0.
2. Constructor de copia y operador de asignación.



ugr

△ Ejercicio 3 ▷ Otros métodos de la clase

Añade a la clase Frecuencias los siguientes métodos públicos:

1. Un método *repeticiones* de tipo *int* que reciba como parámetro una cadena-C (palabra) y devuelva la frecuencia asociada a esa cadena en el objeto Frecuencias o bien 0 si la cadena no es encontrada.
Puede asumir que tiene disponible el método *int buscar(const char* palabra)* en la clase Frecuencias.
Este método devuelve la posición en el vector de entradas, donde está esa palabra, o en caso de que la palabra no esté en el vector, la posición donde debería insertarse para que el vector siguiese estando ordenado.
2. Un método *insertar* que reciba como parámetro una cadena-C (palabra). Si la palabra ya estaba en el vector de entradas, sumará 1 a su frecuencia. Si la palabra no estaba en el vector de entradas, insertará en el lugar adecuado, una nueva entrada cuya palabra será una copia en memoria dinámica de la palabra recibida y una frecuencia igual a 1.

△ Ejercicio 4 ▷ Sobrecarga de operadores

Implemente la siguiente funcionalidad para la clase Frecuencias sobrecargando los operadores << y >> y de suma. En caso necesario, implemente también los métodos necesarios de la clase Entrada.

1. Operador de salida << para poder insertar en un flujo de salida el contenido de un objeto Frecuencias. En concreto, deberá aparecer:
 - un número entero que indica el número de entradas y un salto de línea,
 - una serie de líneas (tantas como entradas) que contienen la palabra, un espacio en blanco y la frecuencia (un número entero).
2. Operador de entrada >> para poder obtener desde un flujo de entrada el contenido de un objeto. Se asume el mismo formato indicado en el apartado anterior.
3. Operador de suma para obtener el objeto Frecuencias que se obtiene a partir de la la suma de dos objetos Frecuencias. La frecuencia de una palabra en el objeto resultado se obtiene como la suma de las frecuencias de los dos objetos de entrada. Puede considerar resueltas y usar las funciones de los ejercicios anteriores.

△ Ejercicio 5 ▷ Funciones externas para E/S

Se desea añadir la posibilidad de cargar/salvar el contenido de un objeto Frecuencias desde/a un fichero. Para ello, se establece un formato de texto que consiste en:

- Una cadena mágica "MP-FRECUENCIAS-1.0" y un salto de línea para poder distinguir que es un fichero tipo Frecuencias.
- El objeto Frecuencias en formato de texto tal como se indica en el ejercicio 4.

Considere que tiene disponibles las funciones del ejercicio 4. Implemente las siguientes funciones externas:

1. Función *cargar* que reciba un nombre de archivo y un objeto de tipo Frecuencias y cargue en éste el contenido del archivo. Debe devolver si ha tenido éxito o no.
2. Función *salvar* que reciba un nombre de archivo y un objeto de tipo Frecuencias y lo salve en el archivo. Debe devolver si ha tenido éxito o no.



◀ **Ejercicio 6** ▷ **Aplicación: Frecuencias de palabras**

[1 punto]

Considere la clase `Frecuencias` anterior, para la que se ha creado un archivo de cabecera. Escriba un programa `-contar.cpp` que lea una secuencia de palabras y escriba en la salida estándar las parejas con la palabra y su frecuencia. Tenga en cuenta que:

- Si no se da ningún parámetro en la línea de órdenes, el programa lee las palabras desde la entrada estándar.
- Si se ejecuta con un parámetro en la línea de órdenes, el programa lee las palabras desde este archivo de texto.
- Si se ejecuta con más de un parámetro, leerá las palabras de todos los archivos de texto proporcionados.

Recuerde que el programa deberá mostrar un mensaje de error adecuado si falla la entrada.

◀ **Ejercicio 7** ▷ **Frecuencias de palabras: Método de una clase**

[1 punto]

Considere la clase `Frecuencias` anterior. Teniendo en cuenta que las entradas de un objeto `Frecuencias` están ordenadas según las palabras, implemente el método `int buscar(const char* palabra)` en la clase `Frecuencias`. Este método devuelve la posición en el vector de entradas, donde está esa palabra, o en caso de que la palabra no esté en el vector, la posición donde debería insertarse para que el vector siguiese estando ordenado.

◀ **Ejercicio 8** ▷ **Frecuencias de palabras: módulos**

[1 punto]

Considere el tipo de dato `Frecuencias` anterior. Escriba el contenido del archivo de cabecera `frecuencias.h`, donde aparece todo lo necesario para usar la clase, sin incluir ninguna definición de función.

◀ **Ejercicio 9** ▷ **Frecuencias de palabras: makefile**

[1 punto]

Considere los archivos que implementan el tipo `Frecuencias` así como el programa `contar.cpp` anterior. Escriba el archivo `Makefile` que permite compilar el programa. Incluya las correspondientes dependencias, así como un objetivo `clean` que permita limpiar los archivos intermedios.



Normas para la realización del examen:

Duración: 2.5 horas

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

Se pretende informatizar la gestión de reservas para las entradas de grupos de turistas a la Alhambra, que se producen a lo largo del día. Para ello, se construirá la clase **ReservasGrupos**. Para cada grupo, esta clase almacenará los nombres y apellidos de sus componentes (se usará un cstring para representarlos) y la hora de reserva. Los grupos pueden tener un número diferente de turistas y tendrán distintas horas de reserva asignadas.

Así, se propone la siguiente representación para esta clase:

```
struct Hora{  
    int hh;  
    int mm;  
};  
  
class Turista{  
private:  
    char* nombre;      // Cadena-C reservada dinámicamente para nombre y apellido  
    char *id;          // Cadena-C reservada dinámicamente para DNI o pasaporte  
    int nacionalidad; // Código de nacionalidad (por ej. código telefónico)  
public:  
    // ... interfaz pública de la clase  
};  
  
class GrupoTuristas{  
private:  
    Turista *turistas; // Vector dinámico con los turistas del grupo  
    int nturistas;     // Número de turistas en el grupo  
    Hora hora;         // Hora de reserva del grupo  
public:  
    // ... interfaz pública de la clase  
};  
  
class ReservasGrupos {  
private:  
    GrupoTuristas *grupos; // Vector dinámico con los grupos de turistas  
    int ngrupos;           // Número de grupos con reserva  
public:  
    // ... interfaz pública de la clase  
};
```

NOTAS:

1. Los métodos que modifiquen el estado de los objetos de esta clase deben asegurarse que utilizan la memoria dinámica **exacta** necesaria para almacenar cada elemento de información.
2. Para la realización de los ejercicios del examen hay que implementar, de forma explícita, determinados métodos y, también, se pueden usar otros que se asume que ya están implementados (se especifica con claridad cada caso). En cada pregunta, se pueden emplear directamente los métodos considerados en preguntas anteriores, tanto de un tipo como de otro.
3. Por último, el alumno puede implementar los métodos privados auxiliares y métodos adicionales que vea convenientes.

▷ Ejercicio 1 ▷ Métodos básicos de la clase

[1,25 puntos]

Defina los siguientes métodos para la clase **ReservasGrupos**:

1. (0,5 puntos) **Constructor por defecto y destructor.**
2. (0,75 puntos) **Constructor de copia y operador de asignación.**

Para la realización de estos ejercicios, se puede asumir la existencia (e implementación) de determinados métodos para las clases **Turista** y **GrupoTuristas**. Debe especificar sus cabeceras y justificar brevemente su necesidad.



UGR

△ Ejercicio 2 ▷ Sobre carga de operadores

[1,75 puntos]

Implemente la siguiente funcionalidad para la clase **ReservasGrupos** sobrecargando los operadores +, << y >>.

1. (1 punto) **Operador de suma** para poder obtener la combinación de dos objetos de la clase **ReservasGrupos**. El conjunto de reservas resultado de este operador se obtendrá uniendo las reservas representadas en los dos argumentos. Cuando existan dos grupos pertenecientes a cada argumento con la misma hora de reserva, se combinarán en un único grupo. Este operador será adecuado cuando se realicen reservas desde distintos puntos de venta y, al final, se debe confeccionar un conjunto de reservas global.
2. (0,25 puntos) **Operador de salida** << para poder insertar en un flujo de salida el contenido de las reservas de grupos en formato texto. En concreto, deberá aparecer:
 - Un entero que indica el número de grupos de turistas que tienen reserva.
 - Para cada grupo de turistas, se insertará la siguiente información:
 - Un entero que indica el número de turistas en el grupo.
 - Para cada turista del grupo, se insertará, en líneas separadas, la siguiente información: su identificador, su nombre y apellidos y su código de nacionalidad.
 - Un entero que indica la hora de reserva.
 - Un entero que indica los minutos de la reserva.
3. (0,5 puntos) **Operador de entrada** >> para poder obtener desde un flujo de entrada el contenido de un objeto. Se asume el mismo formato indicado anteriormente.

Para la realización de estos ejercicios, se puede asumir la existencia (e implementación) de determinados métodos para las clases **Turista** y **GrupoTuristas**. Debe especificar sus cabeceras y justificar brevemente su necesidad.

△ Ejercicio 3 ▷ Entradas y Salidas

[1 punto]

Se desea añadir la posibilidad de cargar y salvar las reservas de grupos para un día desde fichero. Para ello, se establece un formato de texto que consiste en:

- Una cadena mágica "RESERVASGRUPOS-ALHAMBRA" y un salto de línea (para poder distinguir que es un fichero usado para almacenar información de objetos de la clase de interés).
- Las reservas en formato de texto, tal como se indica en el ejercicio 2.

Considerando que tiene disponibles los métodos de los ejercicios previos implemente las siguientes funciones externas:

1. (0,5 puntos) Función **Cargar**, que recibe un nombre de archivo y un objeto de tipo **ReservasGrupos** y carga en éste el contenido del archivo. Devuelve si la operación ha tenido éxito.
2. (0,5 puntos) Función **Salvar**, que recibe un nombre de archivo y un objeto de tipo **ReservasGrupos** y lo salva en el archivo. Devuelve si la operación ha tenido éxito.

△ Ejercicio 4 ▷ Método que modifica objetos

[1 punto]

Implemente el método **EliminarReserva** que permita eliminar la reserva realizada a un grupo en una hora determinada. Así, su argumento será un objeto de tipo **Hora**.

△ Ejercicio 5 ▷ Aplicación

[1 punto]

Escriba un programa completo que reciba desde la línea de órdenes el nombre de dos ficheros de texto que contienen reservas para grupos en un día particular (como se indica en el ejercicio 3). Deberá combinar tales reservas, eliminar las comprendidas entre las 12 horas y las 13.30 horas, ordenarlas en función de la hora de reserva y, finalmente, salvarlas en un fichero de salida cuyo nombre también se indica en la línea de órdenes.

Para la realización de este ejercicio, puede suponer que ya está implementado el método **OrdenarReservas()** para la clase **ReservasGrupos**.



UGR

Normas para la realización del examen:

Duración: 1 hora

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

Se pretende informatizar la gestión de reservas para las entradas de grupos de turistas a la Alhambra, que se producen a lo largo del día. Para ello, se construirá la clase **ReservasGrupos**. Para cada grupo, esta clase almacenará los nombres y apellidos de sus componentes (se usará un cstring para representarlos) y la hora de reserva. Los grupos pueden tener un número diferente de turistas y tendrán distintas horas de reserva asignadas.

Así, se propone la siguiente representación para esta clase:

```
struct Hora{  
    int hh;  
    int mm;  
};  
  
class Turista{  
private:  
    char* nombre;      // Cadena-C reservada dinámicamente para nombre y apellido  
    char *id;          // Cadena-C reservada dinámicamente para DNI o pasaporte  
    int nacionalidad; // Código de nacionalidad (por ej. código telefónico)  
public:  
    // ... interfaz pública de la clase  
};  
  
class GrupoTuristas{  
private:  
    Turista *turistas; // Vector dinámico con los turistas del grupo  
    int nturistas;     // Número de turistas en el grupo  
    Hora hora;         // Hora de reserva del grupo  
public:  
    // ... interfaz pública de la clase  
};  
  
class ReservasGrupos {  
private:  
    GrupoTuristas *grupos; // Vector dinámico con los grupos de turistas  
    int ngrupos;           // Número de grupos con reserva  
public:  
    // ... interfaz pública de la clase  
};
```

NOTAS:

1. Los métodos que modifiquen el estado de los objetos de esta clase deben asegurarse que utilizan la memoria dinámica **exacta** necesaria para almacenar cada elemento de información.
2. Para la realización de los ejercicios del examen hay que implementar, de forma explícita, determinados métodos y, también, se pueden usar otros que se asume que ya están implementados (se especifica con claridad cada caso). En cada pregunta, se pueden emplear directamente los métodos considerados en preguntas anteriores, tanto de un tipo como de otro.
3. Por último, el alumno puede implementar los métodos privados auxiliares y métodos adicionales que vea convenientes.

▷ Ejercicio 1 ▷ Métodos básicos de la clase

[puntos]

Defina los siguientes métodos para la clase **ReservasGrupos**:

1. **Constructor por defecto y destructor.**
2. **Constructor de copia y operador de asignación.**

Para la realización de estos ejercicios, se puede asumir la existencia (e implementación) de determinados métodos para las clases **Turista** y **GrupoTuristas**. Debe especificar sus cabeceras y justificar brevemente su necesidad.



UGR

▷ Ejercicio 2 ▷ Sobre carga de operadores

[puntos]

Implemente la siguiente funcionalidad para la clase **ReservasGrupos** sobrecargando los operadores +, << y >>.

1. **Operador de suma** para poder obtener la combinación de dos objetos de la clase **ReservasGrupos**. El conjunto de reservas resultado de este operador se obtendrá uniendo las reservas representadas en los dos argumentos. Cuando existan dos grupos pertenecientes a cada argumento con la misma hora de reserva, se combinarán en un único grupo. Este operador será adecuado cuando se realicen reservas desde distintos puntos de venta y, al final, se debe confeccionar un conjunto de reservas global.
2. **Operador de salida <<** para poder insertar en un flujo de salida el contenido de las reservas de grupos en formato texto. En concreto, deberá aparecer:
 - Un entero que indica el número de grupos de turistas que tienen reserva.
 - Para cada grupo de turistas, se insertará la siguiente información:
 - Un entero que indica el número de turistas en el grupo.
 - Para cada turista del grupo, se insertará, en líneas separadas, la siguiente información: su identificador, su nombre y apellidos y su código de nacionalidad.
 - Un entero que indica la hora de reserva.
 - Un entero que indica los minutos de la reserva.
3. **Operador de entrada >>** para poder obtener desde un flujo de entrada el contenido de un objeto. Se asume el mismo formato indicado anteriormente.

Para la realización de estos ejercicios, se puede asumir la existencia (e implementación) de determinados métodos para las clases **Turista** y **GrupoTuristas**. Debe especificar sus cabeceras y justificar brevemente su necesidad.

▷ Ejercicio 3 ▷ Entradas y Salidas

[puntos]

Se desea añadir la posibilidad de cargar y salvar las reservas de grupos para un día desde fichero. Para ello, se establece un formato de texto que consiste en:

- Una cadena mágica "RESERVASGRUPOS-ALHAMBRA" y un salto de línea (para poder distinguir que es un fichero usado para almacenar información de objetos de la clase de interés).
- Las reservas en formato de texto, tal como se indica en el ejercicio 2.

Considerando que tiene disponibles los métodos de los ejercicios previos implemente las siguientes funciones externas:

1. Función **Cargar**, que recibe un nombre de archivo y un objeto de tipo **ReservasGrupos** y carga en éste el contenido del archivo. Devuelve si la operación ha tenido éxito.
2. Función **Salvar**, que recibe un nombre de archivo y un objeto de tipo **ReservasGrupos** y lo salva en el archivo. Devuelve si la operación ha tenido éxito.

▷ Ejercicio 4 ▷ Método que modifica objetos

[puntos]

Implemente el método **EliminarReserva** que permita eliminar la reserva realizada a un grupo en una hora determinada. Así, su argumento será un objeto de tipo **Hora**.

▷ Ejercicio 5 ▷ Aplicación

[puntos]

Escriba un programa completo que reciba desde la línea de órdenes el nombre de dos ficheros de texto que contienen reservas para grupos en un día particular (como se indica en el ejercicio 3). Deberá combinar tales reservas, eliminar las comprendidas entre las 12 horas y las 13.30 horas, ordenarlas en función de la hora de reserva y, finalmente, salvarlas en un fichero de salida cuyo nombre también se indica en la línea de órdenes.

Para la realización de este ejercicio, puede suponer que ya está implementado el método **OrdenarReservas()** para la clase **ReservasGrupos**.



△ Ejercicio 6 ▷ Método redistribuir

[1,5 puntos]

Implemente el método **RedistribuirGrupo** que permita eliminar la reserva realizada a un grupo en una hora determinada, pero, además, debe redistribuir los turistas de ese grupo entre los restantes, aplicando la siguiente estrategia: *iterativamente, cada turista se asignará al grupo actual con menos turistas (en caso de empate, se asigna a cualquiera de ellos).*

△ Ejercicio 7 ▷ Método ordenar grupos

[1,5 puntos]

Implemente el método **OrdenarGrupos** que permita ordenar todos los grupos de turistas almacenados en un objeto de la clase **ReservasGrupos** alfabéticamente por apellidos y nombre (Recuerde que los objetos de esta clase almacenan nombre y apellidos).

Para la realización de este ejercicio, se puede asumir la existencia (e implementación) de determinados métodos para las clases **Turista** y **GrupoTuristas**. Debe especificar sus cabeceras y justificar brevemente su necesidad.

△ Ejercicio 8 ▷ Organización en módulos y Makefile

[1 punto]

1. (0,5 puntos) Escriba el contenido de los archivos de cabecera turistas.h, grupoTuristas.h y reservasGrupos.h, donde debe aparecer todo lo necesario para usar las clases **Turista**, **GrupoTuristas** y **ReservasGrupos**, sin incluir ninguna definición de función (suponga que estarán en turistas.cpp, grupoturistas.cpp y reservasGrupos.cpp).
2. (0,5 puntos) Considere los archivos que implementan las tres clases así como el programa principal.cpp indicado en el ejercicio 5. Escriba el archivo Makefile que permita la compilación del programa. Incluya las dependencias necesarias y un objetivo clean que permita eliminar los archivos intermedios generados en el proceso de compilación.

Normas para la realización del examen:

Duración: 3 horas

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

Se desea construir la clase RedMetro para poder realizar simulaciones sobre una red de transporte. La **red** está compuesta por **líneas**, y cada línea es una sucesión de **paradas**. Ni el número de líneas ni el número de paradas está predeterminado. Algunas paradas pueden ser comunes a distintas líneas. Una parada se identifica por un número (**índice**, toma valores desde 1 y no hay "huecos" en la numeración de las paradas) y puede estar activa -admite el tráfico- para una(s) determinada(s) línea(s) e inactiva (no admite el tráfico) para la(s) otra(s).

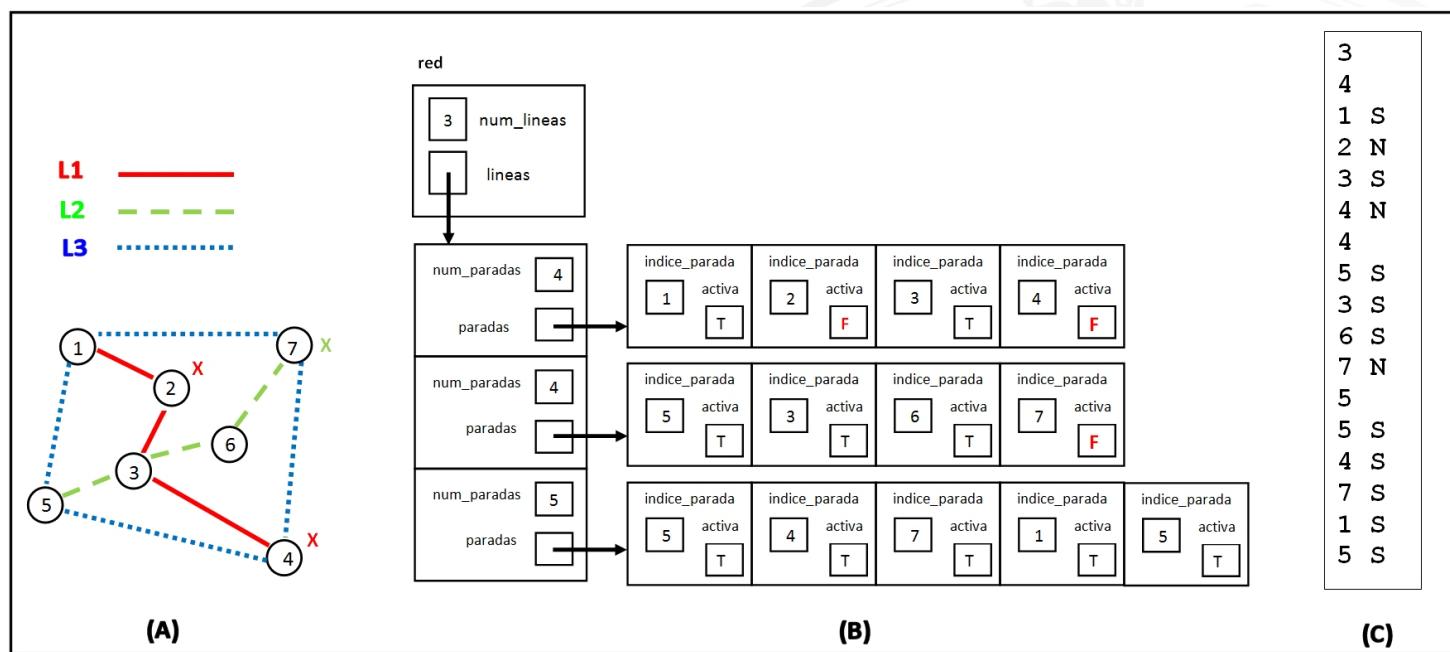
Se propone la siguiente representación para las clases:

```
class RedMetro {
    private:
        int num_lineas;
        Linea * lineas;
    public:
        // ... interfaz
};

class Linea {
    private:
        int num_paradas;
        InfoParada * paradas;
    public:
        // ... interfaz
};

class InfoParada {
    private:
        bool activa;
        int indice_parada;
    public:
        // ... interfaz
};
```

En la figura A mostramos una red formada por tres líneas y siete paradas. Esta red se muestra en la figura B representada en el objeto **red** (clase: RedMetro). La numeración de las líneas empieza en 1 y no hay "huecos". La línea 1 es la primera del array referenciado por **lineas**, la línea 2 es la segunda, y así sucesivamente. Las líneas 1 y 2 tienen 4 paradas mientras que la línea 3 tiene 5 (una línea circular *duplica* la parada de cabecera). Las líneas 1 y 2, p.e. comparten la parada 3. En la línea 1 la parada 4 está inactiva, mientras que en la línea 3 esa misma parada está activa.



Para la realización de los ejercicios propuestos deberá escribir RedMetro.h, Linea.h, e InfoParada.h.

Suponga que dispone de los métodos públicos de **consulta** sobre las clases:

- **InfoParada**: a) **GetIndice**, que devuelve el índice de la parada y b) **EstaActiva**, que indica si la parada está activa o no para la línea en la que se encuentra.
- **Linea**: a) **GetNumParadas**, que devuelve el número de paradas y b) operador **[]**, que devuelve una parada⁽¹⁾.
- **RedMetro**: a) **GetNumLineas**, que devuelve el número de líneas, b) **GetNumeroTotalParadas**, que devuelve el número **total** de paradas (en el ejemplo, 7) y c) operador **[]**, que devuelve una línea⁽¹⁾.

(1) El índice de acceso para los operadores **[]** toma valores 1,2,...

Deberá implementar todo lo que pudiera necesitar y que no se indique explícitamente su disposición.



△ Ejercicio 1 ▷ Métodos básicos de la clase

[1 punto]

Defina los siguientes métodos para la clase RedMetro:

1. (0.5 puntos) Constructor sin argumentos (crea una red vacía) y destructor. Escriba también el método EstaVacia que indique si una red está vacía.
2. (0.5 puntos) Constructor de copia y operador de asignación.

Nota 1: Los constructores y el destructor de las otras clases están implementados, así como el operador de asignación.

△ Ejercicio 2 ▷ Sobrecarga de operadores

[1.5 puntos]

1. (0.5 puntos) Sobrecargue el operador += para la clase Linea. El objetivo es incorporar una nueva parada (InfoParada) a la línea (Linea). El nuevo objeto InfoParada se añade al final del objeto Linea.

Nota 2: Para soluciones futuras (si fuera preciso) suponga que está implementado el operador += para la clase RedMetro (para incorporar una nueva linea (Linea) a la red (RedMetro)).

2. Implemente la siguiente funcionalidad para la clase RedMetro sobrecargando los operadores << y >>:

- (a) (0.25 puntos) Operador de salida << para poder insertar en un flujo de salida el contenido de una red en formato texto. Deberá aparecer (ver figura C):
 - Un número entero que indica el número de líneas de la red y un salto de línea,
 - Para cada línea de la red:
 - Un número entero que indica el número de paradas de la línea y un salto de línea,
 - Tantas líneas de texto como paradas haya en la línea. En cada una aparecerá: un número entero (el índice de la parada), un espacio o tabulador (al menos), un carácter (S si está operativa ó N si no lo está) y un salto de línea.

Nota 3: Suponga que está implementado el operador << para las otras clases.

- (b) (0.75 puntos) operador de entrada >> para poder obtener desde un flujo de entrada el contenido de un objeto. Se asume el formato indicado en el ejercicio anterior.

△ Ejercicio 3 ▷ Métodos y funciones para E/S

[0.75 puntos]

Implemente un constructor de la clase RedMetro con argumento. Recibe el nombre de un fichero de texto con la información sobre una red. El fichero debe contener en la primera línea la cadena mágica "METRO" y un salto de línea. A continuación está el contenido de la red en formato de texto, tal como se indica en el ejercicio 2.

△ Ejercicio 4 ▷ Sobrecarga de operadores relacionales

[0.75 puntos]

Sobrecargue los operadores relacionales ==, != y > para la clase RedMetro en base a un valor de calidad. Este valor se calculará a partir de: 1) el número de líneas (30%) y 2) el número de paradas activas (70%).

Nota 4: Una parada común a dos líneas contará dos veces, si es común para tres líneas contará tres veces, ... Si una parada no está operativa para una línea, no contará. Lo hará para cualquier otra línea para la que estuviera operativa.

△ Ejercicio 5 ▷ Métodos de cálculo

[1 punto]

1. (0.75 puntos) Implemente el método MejorConectada que permita obtener la parada (su índice) en la que confluyen el mayor número de líneas, independientemente de si las paradas están activas o no.
2. (0.25 puntos) Implemente el método EstaTotalmenteOperativa que indique si una línea está totalmente operativa, o sea, si todas sus paradas están activas.

△ Ejercicio 6 ▷ Aplicación

[1 punto]

Escriba un programa completo que reciba desde la línea de órdenes el nombre de dos ficheros de texto que contienen la descripción de dos redes de metro (como se indica en el ejercicio 3) y muestre:

1. Cuál es la mejor red.
2. Para la mejor red informará sobre la operatividad de cada una de sus líneas.
3. Para la mejor red indicará qué parada es la mejor conectada.



Normas para la realización del examen:

Duración: 2.5 horas

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

En este examen se desea construir varias clases para trabajar con un algoritmo conocido como algoritmo de las K-medias. La entrada a este algoritmo es un conjunto de puntos y un número entero K . El algoritmo tiene como objetivo agrupar los puntos de entrada en K grupos (clústeres) disjuntos. Cada uno de los grupos estará formado por un subconjunto de los puntos originales y un punto que los represente (*centroide*), que es un punto que se calcula como el *vector de medias* de los puntos del grupo. Por ejemplo, en el gráfico de la derecha se muestra el resultado de aplicar el algoritmo al conjunto de puntos del gráfico de la izquierda, tomando $K = 2$. Se marcan con círculos rellenos, los puntos de uno de los grupos, y con cuadrados no rellenos los puntos del otro. También se marcan con una cruz, los centroides de cada uno de los dos grupos.



Para resolver el problema, se usarán los siguientes tipos de datos:

```
class Punto {  
    private:  
        double x;  
        double y;  
    public:  
        // Interfaz de la clase  
};  
  
class Cluster {  
    private:  
        Punto centroide; // centroide del grupo  
        VectorPuntos puntos; // puntos del grupo (clúster)  
    public:  
        // Interfaz de la clase  
};  
  
class VectorPuntos {  
    private:  
        Punto *puntos; // array dinámico de puntos  
        int nPuntos; // nº de puntos actualmente en el array dinámico  
        int reservados; // capacidad del array dinámico  
    public:  
        // Interfaz de la clase  
};  
  
class KMedias {  
    private:  
        int K; // nº de grupos  
        Cluster *clusters; // array dinámico de grupos  
    public:  
        // Interfaz de la clase  
};
```

Suponga que tiene ya implementados los siguientes métodos o funciones:

- Clase Punto
 - Constructor por defecto en la clase Punto que inicializa x e y al valor 0.0.
 - Métodos públicos de consulta: getX(), getY() y getDistancia(const Punto& punto).
 - Métodos públicos de modificación: setXY(double newX, double newY).
 - Operadores >> y << (leen o escriben las coordenadas x e y separadas por un blanco).
- Clase VectorPuntos:
 - Métodos públicos de consulta: size() (devuelve el número de puntos).
 - Operadores >> y << (leen o escriben el número de puntos y el conjunto de puntos).
- Clase Cluster:
 - Métodos públicos de consulta: getCentroide() (devuelve el centroide), getPunto(int n) (devuelve el punto n del grupo) y size() (devuelve el número de puntos en el grupo).
 - Métodos públicos de modificación: setCentroide(const Punto& punto) y add(const Punto& punto).
- Clase KMedias: destructor, constructor de copia y operador de asignación.
- Función int uniforme(int minimo, int maximo) que devuelve un número aleatorio entre minimo y maximo.

NOTA IMPORTANTE: Para la realización de los ejercicios del examen, puede usar los métodos y funciones especificados anteriormente, así como los que usted vaya implementando. Si necesita usar otros métodos adicionales, deberá implementarlos en el ejercicio correspondiente.



△ Ejercicio 1 ▷ Métodos básicos de la clase

[1.5 puntos]

1. (0.5 punto) Constructor por defecto de las clases VectorPuntos y KMedias. El constructor por defecto de VectorPuntos crea un vector de puntos vacío, pero con capacidad para guardar 10 puntos. El constructor por defecto de KMedias crea un array para 2 grupos.
2. (0.75 puntos) Constructor de copia, operador de asignación y destructor de la clase VectorPuntos.
3. (0.25 puntos) Razone si es necesario implementar el constructor por defecto, constructor de copia, operador de asignación y destructor en la clase Cluster, o si no es necesario implementar alguno (o ninguno) de estos métodos.

△ Ejercicio 2 ▷ Métodos de la clase

[1 punto]

1. (0.75 puntos) Método add(punto) en la clase VectorPuntos para añadir un punto. Este método debe redimensionar automáticamente el array dinámico al doble de la capacidad que tenga actualmente cuando la capacidad del VectorPuntos esté agotada.
2. (0.25 puntos) Método clearAndSetK(entero) en la clase KMedias que debe reconfigurar el objeto para tener el número de grupos deseado, destruyendo la información que contenga actualmente el objeto KMedias y reservándole la memoria dinámica necesaria.

△ Ejercicio 3 ▷ Sobrecarga de operadores

[1.75 puntos]

Implemente los siguientes operadores en la clase indicada:

1. (0.5 puntos) Operador [] de la clase VectorPuntos para poder usarlo tanto en el lado derecho como en el lado izquierdo de una asignación.
2. (0.5 puntos) Operador de salida << para poder insertar en un flujo de salida el contenido de un objeto KMedias en formato texto. Implemente también el operador << para la clase Cluster (recuerde que dispone de este operador para las clases VectorPuntos y Punto). En concreto, para un objeto KMedias deberá aparecer:
 - El número de grupos K seguido del carácter fin de línea.
 - La lista de grupos. Para cada uno se mostrará:
 - Las coordenadas x e y del centroide (separadas por un blanco) seguido del carácter fin de línea.
 - El número de puntos del grupo, seguido del carácter fin de línea.
 - El conjunto de puntos del grupo, uno en cada línea.
3. (0.75 puntos) Operador de entrada >> para poder obtener desde un flujo de entrada el contenido de un objeto KMedias. Implemente también el operador >> para la clase Cluster (recuerde que dispone de este operador para las clases VectorPuntos y Punto). Se asume el mismo formato indicado en el apartado anterior.

△ Ejercicio 4 ▷ Métodos de cálculo

[1.75 puntos]

1. (0.5 puntos) Implemente el método calcularCentroide() en la clase Cluster que recalcule las coordenadas x e y del centroide de un grupo. La coordenada x (de forma similar con la y) se calcula como la media de las coordenadas x (de forma similar con la y) de los puntos del grupo.
2. (0.75 puntos) Implemente el método clusterMasCercano(Punto) en la clase KMedias que devuelva el índice (un entero) del grupo más cercano a un punto dado. Para ello se comparará la distancia del punto al centroide de cada grupo.
3. (0.5 puntos) Implemente el método cohesion() en la clase KMedias que devuelva la distancia media entre los centroides de los grupos. Para ello debe sumar la distancia entre todas las combinaciones de centroides tomados dos a dos y dividir por el número de combinaciones posibles. Por ejemplo, si tenemos un conjunto de cuatro grupos $\{A, B, C, D\}$, la cohesión se calcularía como:

$$[d(A, B) + d(A, C) + d(A, D) + d(B, C) + d(B, D) + d(C, D)]/6$$



Normas para la realización del examen:

Duración: 1 hora

- El único material permitido durante la realización del examen es un bolígrafo azul o negro.
- Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.
- No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.

En este examen se van a emplear varias clases para trabajar con un algoritmo conocido como algoritmo de las K-medias. La entrada a este algoritmo es un conjunto de puntos y un número entero K . El algoritmo tiene como objetivo agrupar los puntos de entrada en K grupos (*clústeres*) disjuntos. Cada uno de los grupos estará formado por un subconjunto de los puntos originales y un punto que los represente (*centroide*), que es un punto que se calcula como el *vector de medias* de los puntos del grupo. Por ejemplo, en el gráfico de la derecha se muestra el resultado de aplicar el algoritmo al conjunto de puntos del gráfico de la izquierda, tomando $K = 2$. Se marcan con círculos rellenos, los puntos de uno de los grupos, y con cuadrados no rellenos los puntos del otro. También se marcan con una cruz, los centroides de cada uno de los dos grupos.



Para resolver el problema, se usan los siguientes tipos de datos:

```
class Punto {  
    private:  
        double x;  
        double y;  
    public:  
        // Interfaz de la clase  
};  
  
class Cluster {  
    private:  
        Punto centroide; // centroide del grupo  
        VectorPuntos puntos; // puntos del grupo (clúster)  
    public:  
        // Interfaz de la clase  
};  
  
class VectorPuntos {  
    private:  
        Punto *puntos; // array dinámico de puntos  
        int nPuntos; // nº de puntos actualmente en el array dinámico  
        int reservados; // capacidad del array dinámico  
    public:  
        // Interfaz de la clase  
};  
  
class KMedias {  
    private:  
        int K; // nº de grupos  
        Cluster *clusters; // array dinámico de grupos  
    public:  
        // Interfaz de la clase  
};
```

Dispone de los siguientes métodos o funciones:

- Clase Punto (declaraciones en Punto.h)

```
Punto(); // Constructor. Establece las coordenadas (0.0, 0.0)  
double getX() const; // Devuelve la coordenada x  
double getY() const; // Devuelve la coordenada y  
void setXY(double c_x, double c_y); // Cambia las coordenadas a (c_x, c_y)  
double getDistancia(const Punto & otro) const; // Devuelve la distancia al punto dado
```

- Clase VectorPuntos (declaraciones en VectorPuntos.h):

```
VectorPuntos(const int capacidad_inicial=CAPACIDAD_DEF); // Constructor. Establece la  
// capacidad del array  
VectorPuntos(const VectorPuntos & otro); // Constructor de copia  
~VectorPuntos(); // Destructor  
int size() const; // Devuelve el núm. de puntos guardados  
int capacity() const; // Devuelve el núm. máx. de puntos que pueden guardarse  
void add(const Punto & nuevo_punto); // Añade un punto
```



- Clase Cluster (declaraciones en Cluster.h):

```
Punto getCentroide() const; // Obtener centroide
void setCentroide(const Punto & el_nuevo); // Establecer centroide
void calcularCentroide(); // Calcular centroide
void add(const Punto & nuevo_punto); // Añade un punto
int size() const; // Num. de puntos guardados
Punto getPunto(int num_punto) const; // Devuelve un punto del grupo
```

- Clase KMedias (declaraciones en KMedias.h):

```
KMedias(const int K_inicial=K_DEF); // Constructor. Establece el número de grupos
KMedias(const KMedias & otro); // Constructor de copia
~KMedias(); // Destructor
int getK() const; // Devuelve el núm. de grupos
void clearAndSetK(const int nuevo_K); // Modifica el núm. de grupos.
int clusterMasCercano(const Punto & punto) const; // Devuelve el grupo (su índice) más
// cercano al punto dado
```

- Todas las clases disponen de los operadores << y >>
- Las clases VectorPuntos y KMedias disponen de los operadores [] y =
- Función int uniforme(int minimo, int maximo) que devuelve un número aleatorio entre minimo y maximo.

NOTA IMPORTANTE: Para la realización de los ejercicios puede usar los métodos y funciones especificados anteriormente, así como los que usted vaya implementando. Si necesitara usar otros métodos adicionales, deberá implementarlos en el ejercicio correspondiente.



△ Ejercicio 1 ▷ KMedias: implementación del algoritmo

[2 puntos]

Implemente el método `run(conjunto de puntos)` en la clase `KMedias` que aplica el algoritmo de las KMedias al conjunto de puntos que se le pase (de la clase `VectorPuntos`). El resultado del algoritmo quedará guardado en el dato miembro `clusters`. Los pasos que realiza este algoritmo son los siguientes:

1. Inserta aleatoriamente cada punto en uno de los grupos (**asignación inicial**) del array `clusters`.
2. Repite mientras que cambie de grupo algún punto de algún grupo
 - (a) Para cada uno de los K grupos, calcula el centroide del grupo.
 - (b) Crea un nuevo array con K grupos (`clustersNuevos`).
 - (c) Recorre cada uno de los puntos de `clusters`, e insértalo en el grupo de `clustersNuevos` que nos dé una menor distancia al centroide del correspondiente grupo en `clusters`.
 - (d) Haz que `clustersNuevos` pase a ser el array actual `clusters`.

△ Ejercicio 2 ▷ Aplicación: algoritmo de las K-medias

[1.5 puntos]

Escriba un programa `-kmedias.cpp` que lea un conjunto de puntos y le aplique el algoritmo de las K-medias. El programa podrá recibir tres parejas de **parámetros opcionales** desde la línea de órdenes, que **podrán aparecer en cualquier orden**:

> `kmedias -k K -i ficheroEntradaVectorPuntos -o ficheroSalidaKMedias`

donde:

- `-k K`: permite indicar el número K de grupos a usar. Si no se da, se usarán **dos** grupos.
- `-i ficheroEntradaVectorPuntos`: permite indicar el nombre del fichero de puntos de entrada. Si no se da, los puntos se leen de la entrada estándar.
- `-o ficheroSalidaKMedias`: permite indicar el nombre del fichero de salida, donde se guardará la información sobre los grupos encontrados. Si no se da, la información se muestra en la salida estándar.

Tenga en cuenta que:

- Los ficheros de puntos son ficheros de texto que contienen:
 - El *número de puntos* y un separador (normalmente salto de línea).
 - Los puntos del grupo, uno en cada línea. Para cada punto se indica su coordenada x y su coordenada y , con un separador al menos entre ambas.
- Los ficheros en formato *KMedias*, son también ficheros de texto que deben contener en la primera línea la *cadena mágica KMEDIAS* y un salto de línea. A continuación está el contenido del objeto *KMedias* en formato de texto. Concretamente:
 - El número de grupos K seguido del carácter fin de línea.
 - Los grupos. Para cada uno se mostrará:
 - * Las coordenadas x e y del centroide (con un separador al menos entre ambas) seguido del carácter fin de línea.
 - * El *número de puntos* del grupo, seguido del carácter fin de línea.
 - * Los puntos del grupo, uno en cada línea. Para cada punto se indica su coordenada x y su coordenada y , con un separador al menos entre ambas.

△ Ejercicio 3 ▷ KMedias: Makefile

[0.5 puntos]

Considere los archivos que implementan los tipos `KMedias`, `Cluster`, `VectorPuntos` y `Punto`, así como el programa `kmedias.cpp` anterior. Escriba el archivo `Makefile` que permite compilar el programa. Incluya las correspondientes dependencias, así como un objetivo `clean` que permita limpiar los archivos intermedios.