

# 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;
};
```

- a) (2 puntos) Escribir la implementación del operador de asignación.  
b) (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  $n*m$  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.  
c) (2 puntos) Para cada elemento  $a_{ij}$  de la matriz, se define su residuo  $r_{ij}$  como:

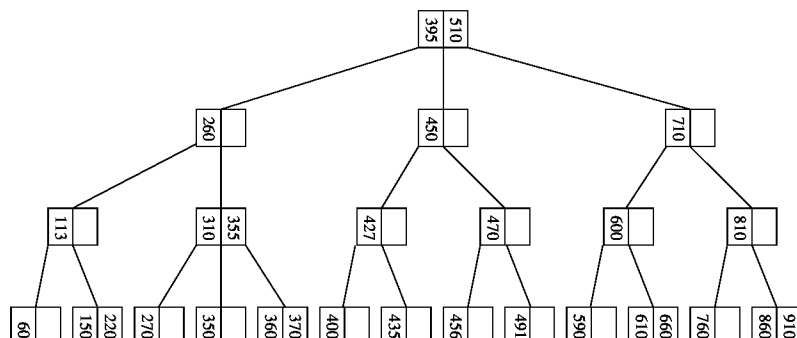
$$r_{ij} = d_{IJ} + a_{ij} - d_{Ij} - d_{iJ} \text{ donde } d_{IJ} = \frac{\sum_{j=1}^m \sum_{i=1}^n a_{ij}}{n*m} \quad d_{Ij} = \frac{\sum_{i=1}^n a_{ij}}{n} \quad d_{iJ} = \frac{\sum_{j=1}^m a_{ij}}{m}$$
$$\text{Se define el residuo de la matriz como } r_{IJ} = \frac{\sum_{j=1}^m \sum_{i=1}^n \|r_{ij}\|}{n*m}.$$

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:

- a) (0.5 puntos) Insertar los valores 365, 275, 364, 272, 352.  
b) (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.

- a) (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).
- b) (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\_descripciones* es el del fichero que proporciona la información para su reconocimiento.

- c) (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.**

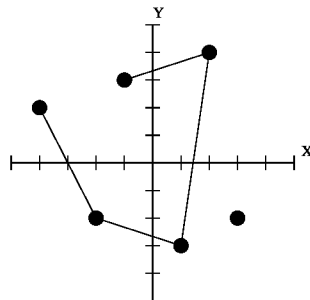
**Duración del examen: 2 horas y media.**

## 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
    ....
};
```



- Implementa el constructor de copias de la clase **PoliLinea**.
  - Implementa la sobrecarga del operador de asignación para la clase **PoliLinea**.
  - ¿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.
  - Sobrecarga el operador de suma para poder añadir nuevos puntos a una polilínea, de manera que podamos ejecutar operaciones de la forma:
    - punto + polilínea. Se crea una nueva polilínea añadiendo un punto al comienzo de la misma.
    - 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

- Implementa un método para cargar una polilínea desde un fichero a un objeto en memoria:  
`void PoliLinea::Leer(const char *nomfich);`
  - 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:

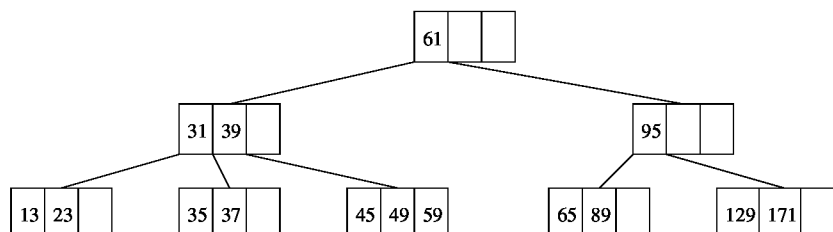
$$x_i = \begin{cases} 1 & \text{si } i = 0 \\ \frac{1}{2} \left( x_{i-1} + \frac{a}{x_{i-1}} \right) & \text{si } i > 0 \end{cases}$$

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
};
```

- Escribe una función que reciba un puntero a la primera celda de la lista y que devuelva una nueva lista invertida.
  - 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.

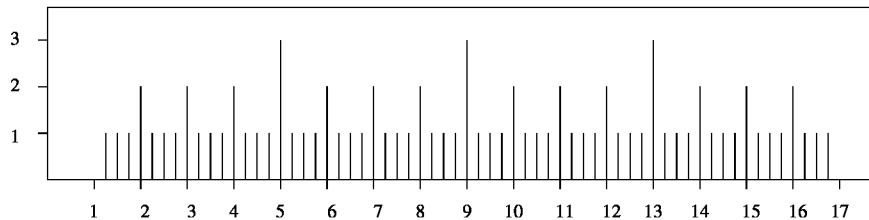
1. (2 puntos) Implemente una función recursiva *Reglar* con la siguiente cabecera:

```
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:  
    ...  
};
```

- (1 punto) Implemente el constructor por defecto y destructor.
  - (1 punto) Implemente el constructor de copias.
  - (1 punto) Sobrecargue el operador de asignación.
  - (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:
- La primera línea contiene la cadena mágica CJTMP2 seguida por un salto de línea.
  - Opcionalmente, puede haber una línea que contiene un carácter ‘#’ seguido por una secuencia de hasta 100 caracteres que finalizan en un salto de línea.
  - Una línea que contiene un número entero ( $N$ ) indicando el número de elementos del conjunto seguido por un salto de línea.
  - Una línea que contiene  $N$  números enteros separados por espacios, y que corresponden a los elementos que contiene el conjunto.

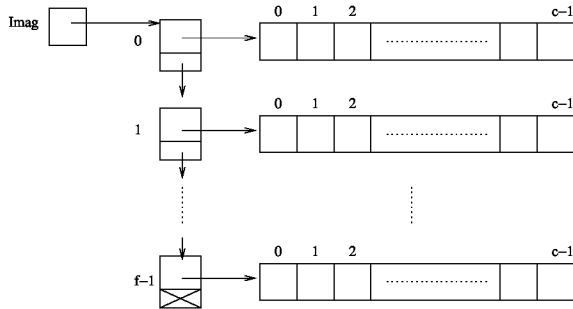
Implemente las funciones miembro “Leer” y “Escribir” para la lectura y escritura de un conjunto en disco. Las funciones reciben como entrada el nombre de un fichero en disco y devuelven un código entero indicando si ha habido errores.

**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 (*Imag*), 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.

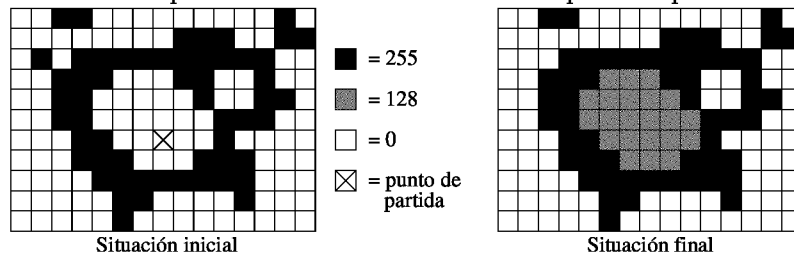
- (0.75 punto) Implemente el constructor por defecto (crea una imagen vacía) y el destructor.
- (0.75 punto) Implemente el constructor de copias.
- (1 punto) Sobrecargue el operador de asignación.
- (1 punto) Implemente dos funciones *Set* y *Get* que permitan modificar y acceder al contenido de una 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;
```

- (1 punto) Sobrecargue 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:

- Lamarlo sin ningún argumento. Los datos con las sumas a realizar se leerán desde la entrada estándar.
  - Lamarlo 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:
- (0.75 punto) Un programa que transforme un fichero de texto a binario.
  - (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
zyxwvutsrqpon
mnlkjihgfedcba
```

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;
        ...
};
#endif
```

### fecha.h

```
#ifndef _FECHA_H_
#define _FECHA_H_

class Fecha {
    ...
};
#endif
```

### profesor.h

```
#ifndef _PROFESOR_H_
#define _PROFESOR_H_

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

### 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;
        ...
};
#endif
```

### 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

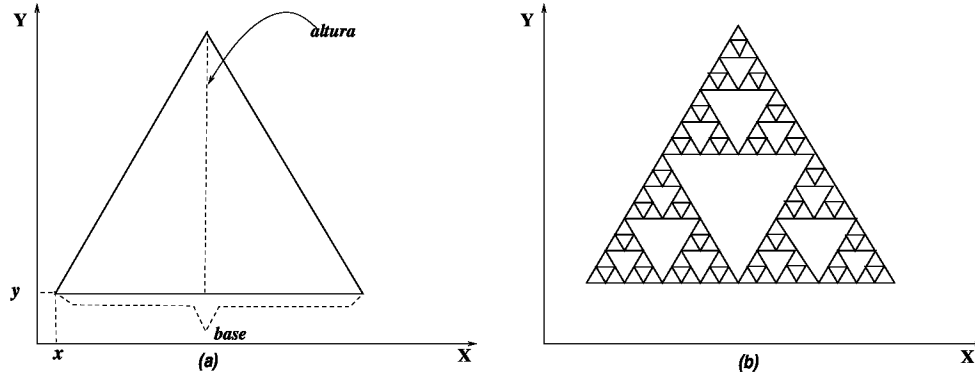
Examen de teoría. 11 de Septiembre de 2007

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 siguiente 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.**

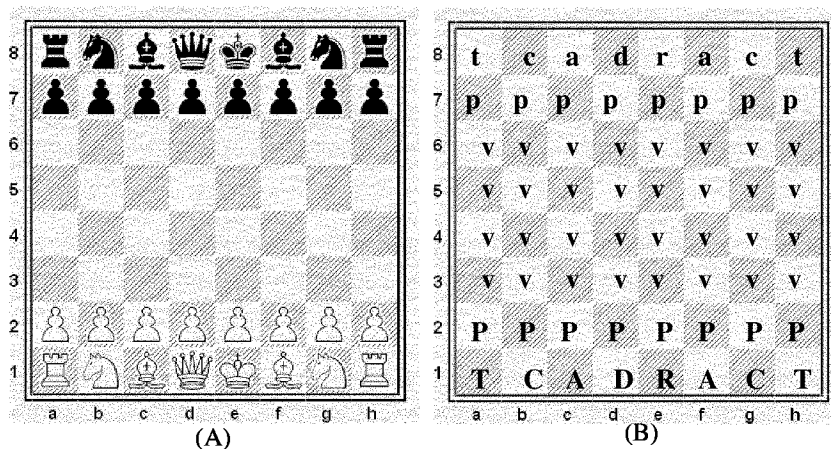
# Metodología de la Programación II

Examen de teoría. 16 de Junio de 2008

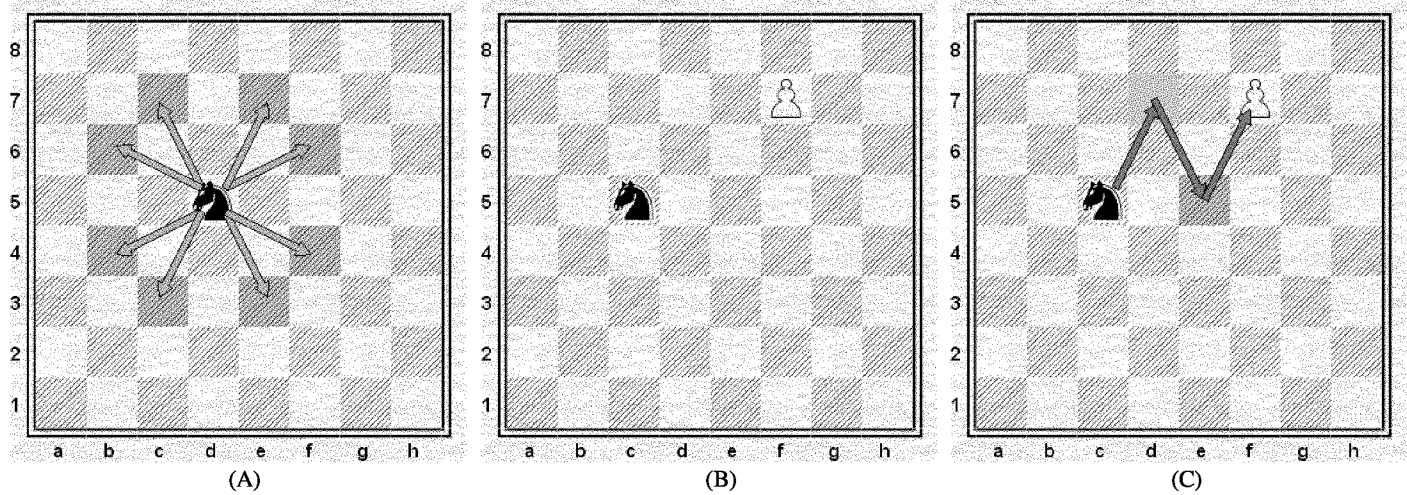
1. Suponga la clase tablero de ajedrez, con la siguiente representación:

```
struct Celda{
    char pieza;
    Celda *sig;
};
class Tablero{
    char **tablero;//situación del tablero
    Celda *perdidas;//colección de piezas perdidas, tanto blancas como negras
    bool turnoblancas;//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)** Sobrecargue 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 | El fichero siempre comienza por esta cadena      |
| tcarvact   |                                                  |
| pppvppp    | El estado del tablero en 8 líneas independientes |
| vvvvvvvv   |                                                  |
| vvvvPvvv   |                                                  |
| vvvvppvv   |                                                  |
| vvvvvvvv   |                                                  |
| PPPvvPPP   |                                                  |
| TCAvRACT   |                                                  |
| DdPp       | Lista de piezas perdidas                         |
| 1          | 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 (basta 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 el 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 NumPages;
    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  $\varepsilon$  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++.