LENGUAJES DE PROGRAMACIÓN

INSTRUCCIONES
Por favor, entregue esta primera hoja de enunciado junto con el examen.
Dispone de 2 horas para realizar el examen.
MATERIAL PERMITIDO: Ninguno.

Pregunta 1 (3 puntos)

Indique la veracidad o falsedad de cada una de las afirmaciones siguientes, explicando detalladamente en cada caso el motivo de su respuesta.

A. (0.5 puntos) En el lenguaje ALGOL las variables vectoriales pueden tener cualquier número de dimensiones.

B. (0.5 puntos) La sentencia `v = 17 % 4` asigna a la variable `v` el valor 4.

C. (0.5 puntos) Dada la variable `v`, declarada

```
std::vector<double> v(4,2)
```

la sentencia `v.clear()` es equivalente a la sentencia `v.assign(4,0)`.

D. (0.5 puntos) En el lenguaje Ada no se produce el problema del *else ambiguo*.

E. (0.5 puntos) La función sobre el flujo de entrada `std::cin.get()` devuelve el primer caracter del flujo de entrada, pero sin extraerlo del flujo.

F. (0.5 puntos) Cualquier nodo de un árbol posee un único nodo padre.

Pregunta 2 (1 punto)

Escriba la salida por consola producida al ejecutar el programa en C++ escrito a continuación.

```
#include <iostream>

int main () {
   double* p;
   double a[] = {10, 20, 30};
   double b[] = {15, 25, 35};
   double x= 1.0,  y= 2.0,  z = 3.0;
   int i = 1;
   p = &a[0];
   for (int i = 0; i<2; i++) {
      double x = 40;
      x += i*z;
      y += (x+z);
      a[i] = x;
      std::cout << a[i]   << std::endl;
      std::cout << *(p+i) << std::endl;
   }
   std::cout << x    << std::endl;
   *(p+1) = b[i];
   std::cout << p[0] << std::endl;
   std::cout << p[1] << std::endl;
   std::cout << p[2] << std::endl;
   std::cout << a[0] << std::endl;
   std::cout << a[1] << std::endl;
   std::cout << a[2] << std::endl;
   return 0;
}
```

Pregunta 3 (1 punto)

Escriba la salida por consola producida al ejecutar el programa en C++ escrito a continuación.

```
#include <iostream>
#include <vector>

int main() {
   std::vector<double> v(4,5);
   std::vector<double> x(4,3);
   v.assign(x.begin()+1, x.begin()+3);
   v.push_back(10);
   for (unsigned int i = 0; i<v.size();i++)
      std::cout << v[i] << " ";
   std::cout << std::endl;
   v.at(2) = 50;
   v.insert(v.end(), x.begin(), x.begin()+2);
   v.pop_back();
   for (unsigned i = 0; i<v.size(); i++)
      std::cout << v[i] << " ";
   std::cout << std::endl;
   for (unsigned int i = 0; i<v.size()-1; i++)
      std::cout << x[i] << " ";
   return 0;
}
```

Pregunta 4 (2.5 puntos)

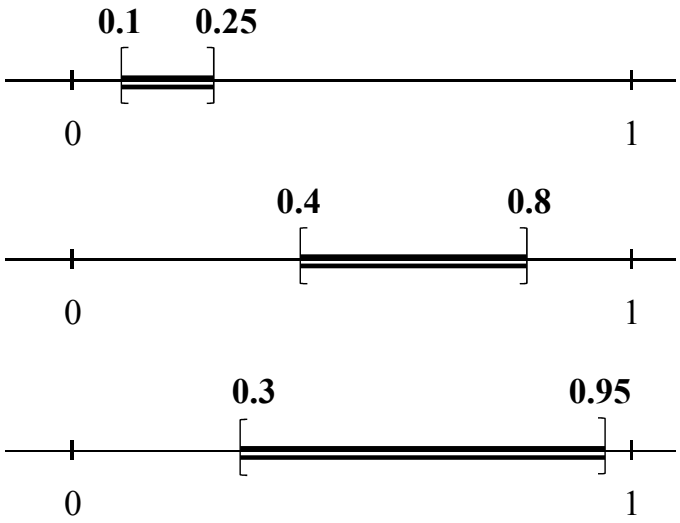
En un fichero de texto llamado *intervalos.txt* se encuentran almacenadas dos columnas de números reales que describen intervalos cerrados de la recta real. Los dos números de cada fila describen un intervalo: el número de la primera columna es el extremo inferior del intervalo y el número de la segunda columna el extremo superior.

Escriba una función en C++ que lea el contenido del fichero *intervalos.txt* y devuelva un valor Booleano que indique si la unión de todos los intervalos leídos del fichero cubre totalmente a $[0,1]$. A continuación se muestra un ejemplo.

Supongamos que el contenido del fichero *intervalos.txt* fuera:

```
0.1   0.25
0.4   0.8
0.3   0.95
```

Dicho contenido indica tres intervalos: $[0.1,0.25]$, $[0.4,0.8]$ y $[0.3,0.95]$. Como puede observarse en la figura, la unión de los tres intervalos no recubre totalmente $[0,1]$. Queda sin recubrir $[0,0.1)\cup(0.25,0.3)\cup(0.95,1]$. La función debería devolver *false*.



Veamos otro ejemplo. Supongamos ahora que el contenido del fichero *intervalos.txt* fuera:

```
-0.1   0.5
 0.3   1.9
```

La unión de los intervalos $[-0.1,0.5]$ y $[0.3,1.9]$ recubre totalmente a $[0,1]$, con lo cual la función debería devolver *true*.

Pregunta 5 (2.5 puntos)

La *Ley de Zipf*, formulada en la década de 1940 por el lingüista de la Universidad de Harvard George Kingsley Zipf, es una ley empírica que describe la frecuencia con la cual aparecen las palabras en un texto. Según esta ley, la frecuencia con la que aparece la palabra más frecuente es el doble de la frecuencia con la que aparece la segunda palabra más frecuente, el triple de la frecuencia con la que aparece la tercera palabra más frecuente y así sucesivamente.

Expresado formalmente, la frecuencia (número de veces que la palabra aparece en el texto dividido por el número total de palabras del texto) es:

$$f(k,N) = \frac{1/k}{\sum_{n=1}^{N} (1/n)} \tag{1}$$

donde k es el rango de la palabra y N es el número de palabras distintas que hay en el texto. El rango de la palabra más frecuente es $k = 1$, el de la segunda palabra más frecuente $k = 2$ y así sucesivamente.

Escriba el código C++ indicado a continuación.

5.a (0.5 puntos) Escriba en C++ una función llamada `leyZipf` que implemente la función $f(k,N)$ descrita mediante la Ec. (1). El prototipo de la función es:

```
double leyZipf(int k, int N)
               throw (std::invalid_argument);
```

La función debe lanzar la excepción cuando cualquiera de sus argumentos sea menor que uno, o cuando k sea mayor que N.

5.b (1 punto) Escriba una función en C++ cuyo prototipo sea:

```
std::map<std::string,int> leeFich(std::string nombreFich)
                     throw (std::invalid_argument);
```

y que realice las acciones siguientes:

1. Abrir para lectura el fichero de texto cuyo nombre viene dado por el parámetro `nombreFich`. Si no es posible abrir el fichero, la función debe lanzar una excepción.

2. El fichero contiene una serie de palabras escritas en minúscula. No contiene signos de puntuación. Leer las palabras del fichero e ir almacenándolas en un mapa. La clave del mapa es la palabra y el valor el número de veces que la correspondiente palabra aparece en el fichero.
3. Devolver el mapa.

5.c (1 punto) Escriba el programa principal, que debe realizar las acciones siguientes:

1. Invocar la función definida en el Apartado 5.b, pasándole como parámetro el siguiente nombre de un fichero de texto: *libro.txt*. Almacenar en una variable llamada `mPalabras` el mapa devuelto por la función. Si la función lanza una excepción, mostrar en la consola un mensaje indicándolo y terminar.
2. Calcular la frecuencia de las 10 palabras más frecuentes según la Ley de Zipf, invocando para ello la función definida en el Apartado 5.a, siendo N el número de palabras distintas almacenadas en el mapa `mPalabras`. Almacenar los 10 valores en un array de **double** llamado `freqZipf`. Si la función lanza una excepción, mostrar en la consola un mensaje indicándolo y terminar.
3. Buscar las 10 palabras con mayor frecuencia del mapa `mPalabras` y mostrar en la consola, para cada una de estas palabras:
   - la palabra,
   - su frecuencia real (número de veces que la palabra aparece en el fichero dividido por el número total de palabras que hay en el fichero),
   - su correspondiente frecuencia teórica calculada según la Ley de Zipf (dicha frecuencia se encuentra almacenada en el array `freqZipf`), y
   - la diferencia entre ambas frecuencias.
4. Terminar.