Guion de prácticas

Entrega de prácticas:
Clase Intervalo con NetBeans

Febrero 2021

Metodología de la Programación

DGIM-GII-GADE

Curso 2020/2021

Índice

Contents

1 Práctica a entregar: Intervalo

Un intervalo es un espacio métrico comprendido entre dos valores o cotas, a y b, siendo a la cota inferior y b la cota superior. Cada extremo de un intervalo puede ser abierto o cerrado. Para la cota inferior solo se usa (o [y para la cota superior) o].
Ejemplos de intervalos: $(a, b] = \{x \in \mathcal{R}/a < x \leq b\}$,
o $[a, b] = \{x \in \mathcal{R}/a \leq x \leq b\}$. Esto es, se deben distinguir los entre los siguientes 4 intervalos: $[0..10.0], (0., 10.0], [0., 10.0), (0., 10.0)$.

Se quiere implementar la clase Intervalo. Para ello, descargue el fichero **Intervalo.zip** (Figura 1) y descomprímalo en una carpeta independiente. Complete las funciones y/o métodos incompletos. Para realizar esta tarea, tenga en cuenta que el objetivo es escribir el programa mediante la separación de ficheros de la forma vista en el guión de prácticas sobre Netbeans (Sección 4 - Un proyecto de compilación separada).

```
./
├── data
│   └── intervalo*.dat
├── doc
│   └── intervalo.doxy
├── scripts
│   ├── runDocumentation.sh
│   └── doZipProject.sh
└── src
    └── intervalo.cpp
```

Figura 1: Contenido del fichero **Intervalo.zip**

- Defina los datos miembro de la clase y los constructores que se han declarado en el fichero **intervalo.cpp**.
  Debe considerar el intervalo vacío como un intervalo válido y este debe estar asociado al constructor sin parámetros.
  En este problema, no se consideran intervalos con extremos infinitos como por ejemplo $(-\infty, \infty), (-\infty, a]$ o (a, ∞).
- Defina los métodos incompletos para dotar la clase de operatividad.
- Implemente un método para comprobar si un intervalo es vacío.
- Implemente un método **bool estaDentro** que comprueba si un valor numérico está dentro de un determinado intervalo.

Se proporciona un main que realiza la lectura de varios intervalos (en `interv`), la lectura de varios valores (en `v`) y muestra en la salida, por cada uno de los intervalos, los puntos de `v` que caen dentro de `interv`.

Un ejemplo de entrada pueden encontrarlo en **intervalo1.dat**, donde se lee 1 intervalo y 6 puntos. La salida es el intervalo leído y los puntos leídos que están dentro de este.

```
---Caso 1:----------------------------------------
1 [0,10]
6 -1 -0.001 0 5.7 9.6 10
- - - - - - - - - - - - - - - - - - - - - - - - -
[0,10] : 0 5.7 9.6 10
---Caso 2:----------------------------------------
1 (0,10]
6 -1 -0.001 0 5.7 9.6 10
- - - - - - - - - - - - - - - - - - - - - - - - -
(0,10] : 5.7 9.6 10
---Caso 3:----------------------------------------
1 [0,10)
6 -1 -0.001 0 5.7 9.6 10
- - - - - - - - - - - - - - - - - - - - - - - - -
[0,10) : 0 5.7 9.6
---Caso 4:----------------------------------------
1 (0,10)
6 -1 -0.001 0 5.7 9.6 10
- - - - - - - - - - - - - - - - - - - - - - - - -
(0,10) : 5.7 9.6
---Caso 5:----------------------------------------
1 (10,10)
6 -1 -0.001 0 5.7 9.6 10
- - - - - - - - - - - - - - - - - - - - - - - - -
(0) :
---Caso 6:----------------------------------------
5 [0,10] (0,10] [0,10) (0,10) (10,10)
6 -1 -0.001 0 5.7 9.6 10
- - - - - - - - - - - - - - - - - - - - - - - - -
[0,10] : 0 5.7 9.6 10
(0,10] : 5.7 9.6 10
[0,10) : 0 5.7 9.6
(0,10) : 5.7 9.6
 (0) :
```

Para facilitar la lectura y escritura de los datos se le proponen dos funciones y un main que las utiliza en el fichero **intervalo.cpp**. Así mismo, se le dan una serie de prototipos para los métodos de la clase que tendrá que completar.

Una vez completado el programa, descompóngalo en múltiples ficheros y carpetas, configure el proyecto y obtenga el ejecutable binario. Una vez obtenido el ejecutable, compruebe el funcionamiento de los métodos con los datos proporcionados en **data**; estos son algunos conjuntos de prueba, no obstante ¡¡pruebe también con otros datos!! Por útimo, ejecute el script **doZipProject.sh** para la obtención del paquete a entregar en PRADO.

2 Entrega

Se debe de entregar a través de **PRADO** un fichero zip, **entrega2.zip** la estructura de directorios ya expuesta: **data, doc, include, scripts, src, zip**.

El fichero **entrega2.zip** debe contener la estructura de la Figura 2.

```
./
├── data
│   └── intervalo*.dat
├── doc
│   └── intervalo.doxy
├── include
│   └── intervalo.h
├── scripts
│   └── doZipProject.sh
├── src
│   ├── intervalo.cpp
│   └── main.cpp
├── zip
└── Makefile
```

Figura 2: Estructura de la entrega entrega2.zip

Deben mantenerse estos nombres de carpetas para que el código pueda corregirse y compilarse de forma automática tras la entrega. También deben ajustarse los nombres de los archivos, funciones, clases,...., a las indicaciones dadas en los ejercicios. **NOTA: no se considerarán ejercicios que no cumplan estas normas. Tampoco aquellos en que se usen espacios en blanco en los nombres de los archivos o carpetas o caracteres especiales (como acentos)**.

En general, se proporciona el código básico para probar la funcionalidad pedida, con los casos de prueba indicados. Debe observarse la forma de uso de las funciones y/o métodos en el programa principal para especificar de forma correcta los argumentos de las funciones. **NOTA: puede modificarse el contenido del programa principal, pero compruebe que funciona con el main original**. Deben implementarse todas las funciones necesarias para que este se ejecute en la forma indicada.