PROGRAMACIÓN ORIENTADA A OBJETOS

GRADO EN INGENIERÍA INFORMÁTICA
SEGUNDO CURSO

DEPARTAMENTO DE INFORMÁTICA Y ANÁLISIS NUMÉRICO
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD DE CÓRDOBA

CURSO ACADÉMICO: 2012 – 2013

- **Número de práctica: 7**

- **Objetivo**
  - Definición de **plantillas de clases**
    - Plantilla para la clase Punto2D
  - Definición de **plantillas de funciones**
    - calcularDistanciaEuclidea2D
    - calcularDistanciaManhattan2D
    - calculardistanciaAjedrez2D

- **Definición de la plantilla de la clase Punto2D**
  - **Descripción**
    - Se debe definir la clase **Punto2D** mediante el uso de una plantilla que permita parametrizar el tipo **T** de sus coordenadas
    - Posteriormente se podrán declarar objetos del tipo Punto2D<int> o Punto2D<float>, por ejemplo.

  - **Atributos**
    - _x: atributo de tipo T que representa la abscisa del Punto2D
    - _y: atributo de tipo T que representa la ordenada del Punto2D

  - **Funciones o métodos públicos**

    - **Constructor**:
      - Versión 1
        - **Constructor parametrizado** con todos los argumentos con el valor por defecto 0.0
      - Versión 2
        - **Constructor de copia**
          - Permite declarar un Punto2D que toma como valores iniciales las coordenadas de un Punto2D pasado como parámetro mediante referencia constante.

- **Funciones de acceso**
  - **getX**:
    - Función de tipo **T** que devuelve el valor del atributo _x
  - **getY**:
    - Función de tipo **T** que devuelve el valor del atributo _y
  - **Observación**
    - Las funciones de acceso deberán ser calificadas con **const** debido al constructor de copia

- **Funciones de modificación**
  - **setX**:
    - Función de tipo **void** que recibe un parámetro de tipo **T** que es asignado al atributo _x de Punto2D
  - **setY:**
    - Función de tipo **void** que recibe un parámetro de tipo **T** que es asignado al atributo _y de Punto2D

- **Sobrecarga del operador de asignación "="**
  - Asigna al Punto2D actual las coordenadas del Punto2D pasado como parámetro
  - Parámetro: q, punto2D pasado como referencia constante
  - Resultado: referencia al Punto2D actual

- **Funciones de lectura y escritura**
  - **leerPunto2D**
    - Lee desde el teclado las coordenadas y se las asigna al Punto2D.
  - **escribirPunto2D**
    - Escribe por pantalla las coordenadas del Punto2D de la forma "(x, y)"

- **Características de la plantillas de clases**
  - Una plantilla de clase tiene la siguiente sintaxis

```
template <typename Tipo>
class NombreDeClase
{
   ...
}
```

o también

```
template <class Tipo>
class NombreDeClase
{
   ...
}
```

- La palabra reservada **template** indica al compilador que el código que sigue es una plantilla o patrón de clase y que se utilizará **Tipo** como tipo

genérico

- Los parámetros de **Tipo** aparecen entre operadores relacionales de desigualdad “<>”.

- La definición de cada función miembro tiene la sintaxis

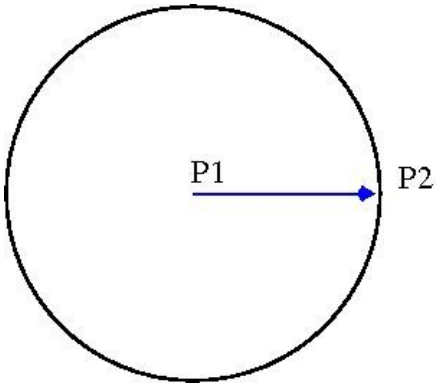
  template *<* ***class*** *Tipo >*
  TipoDeResultado NombreClase< Tipo >::nombreDeFuncion(…)

- **Modelos de compilación de plantillas de clases**
  - Modelo de compilación por **inclusión**
    - **Versión 1**
      - Codificar un único fichero “nombre.hpp” que contenga la declaración de clase, así como la declaración de los prototipos y las definiciones de las funciones miembro
    - **Versión 2**
      - En el fichero “nombre.hpp”, se declara la clase: atributos y prototipos de los métodos o funciones miembro.
      - En el fichero “nombre.cpp” se definen los métodos de la clase que no hayan sido declarados como inline en el fichero “nombre.hpp”
      - Por último, se debe incluir el fichero “nombre.cpp” dentro del fichero “nombre.hpp” mediante el uso de una sentencia include
        #include “nombre.cpp”
    - El **compilador g++** utiliza este método de compilación por **inclusión**
  - Modelo de compilación **separada**
    - Es similar a la declaracion de clases en C++
    - En el fichero “nombre.hpp”, se declara la clase: atributos y prototipos de los métodos o funciones miembro.
    - Se debe utilizar la palabra reservada **export** antes de la palabra **template** para indicar al compilador que puede ser necesaria para la generación de instanciaciones en otros archivos
    - En el fichero “nombre.cpp” se definen los métodos de la clase que no hayan sido declarados como inline en el fichero “nombre.hpp”
    - El compilador **g++ no** permite el uso de **export**.

- **Distancias**
  - Si P1=(x1, x2), y P2(x2, y2) son dos puntos entonces

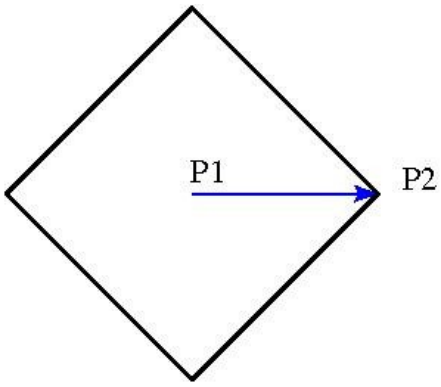
    - **Distancia euclídea 2D**

      d(P1,P2) = $\sqrt{(x2-x1)^2+(y2-y1)^2}$



    - **Distancia de Manhattan 2D**

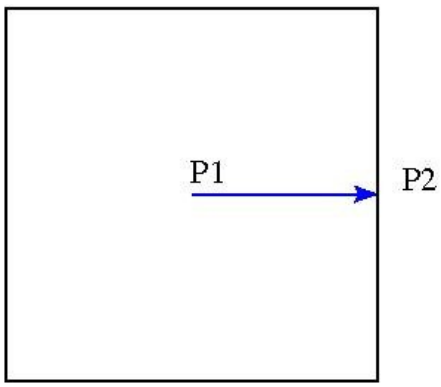
      d(P1,P2) = $|x2-x1|+|y2-y1|$



      - Esta distancia también se denomina "distancia de la ciudad de los bloques" o "*taxicab*"

    - **calcularDistanciaAjedrez2D**

      d(P1,P2) = $max|(x2-x1)|,|(y2-y1)|$



      - Esta distancia también se denomina "distancia de Tchebychev"

- **Definición de plantillas de funciones**
  - Se debe codificar un fichero denominado **distancias2D.hpp** que contenga el prototipo y la definición de las **plantillas** de funciones que calculan **distancias** entre puntos de dos dimensiones
    - calcularDistanciaEuclidea2D
    - calcularDistanciaManhattan2D
    - calcularDistanciaAjedrez2D
  - Observaciones
    - Cada una de las plantillas de funciones recibirá las **coordenadas** (x1, y1) y (x2, y2) de los puntos y devolverán el resultado de calcular la distancia.
    - Para calcular el valor absoluto de un número, se recomienda utilizar la función "fabs" en vez de la función "abs", que no permite argumentos de tipo "int".

- **Características de la plantillas de funciones**
  - Una plantilla de función tiene la siguiente sintaxis

    template <***typename*** *Tipo*>
    Tipo nombreDeFuncion(Tipo parametro1, …)

  - Si Tipo es una clase entonces es preferible cambiar **typename** por **class**
  - La palabra reservada **template** indica al compilador que el código que sigue es una plantilla o patrón de funciones, pero no es la cabecera o la definición real de una función
  - Los parámetros de tipo aparecen entre operadores relacionales de desigualdad "<>".
  - Las plantillas de funciones, al contrario que las funciones normales, **no se pueden separar** en un archivo de cabecera ".hpp" que contenga el prototipo y en un archivo ".cpp" independiente que contenga la definición de la plantilla de la función.
  - Por tanto, se debe usar el **modelo de compilación por inclusión**
    - **Versión 1**
      a) Codificar un fichero "hpp" que contenga el prototipo y la definición de la plantilla de la función.
    - **Versión 2**
      b) Codificar un fichero "nombre.hpp" que contenga el prototipo de la plantilla de la función.
      c) Codificar un fichero "nombre.cpp" que contenga la definición de la plantilla de la función,
      d) Incluir el fichero "nombre.cpp" dentro del fichero "nombre.hpp" mediante el uso de una sentencia include

         #include "nombre.cpp"
- **Ficheros que se han de codificar**
  - Ficheros de la plantilla de clase Punto2D
    - **punto2D.hpp**

- Contendrá la definición de la plantilla de la clase
  - Atributos
  - Prototipo y definición de las funciones miembro declaradas como **inline**
    - constructores, funciones de acceso y funciones de modificación
  - Prototipo de las funciones miembro **no** declaradas inline
    - Funciones de lectura y escritura
- **Observación:**
  - Al trabajar con "plantillas", se va a utilizar el método de compilación por inclusión
  - Por tanto, el fichero punto2D.hpp debe incluir al fichero punto2D.cpp

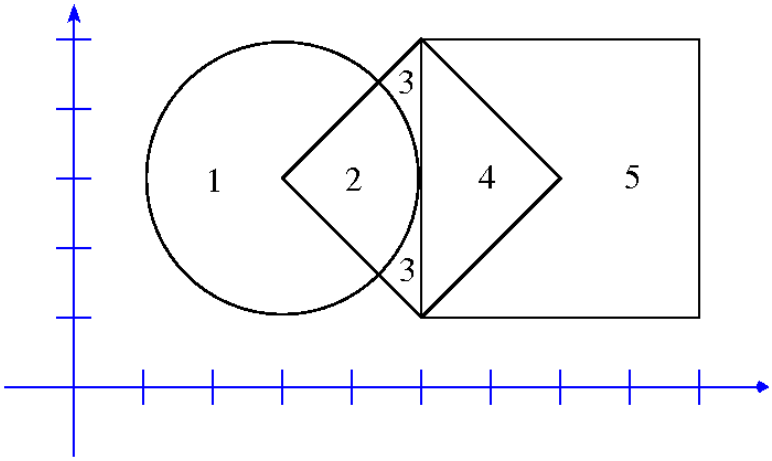
    #include "punto2D.cpp"

- **punto2D.cpp**
  - Definición de las funciones miembro **no** declaradas inline
    - Funciones de lectura y escritura

- Fichero para la declaración de las plantillas de funciones de distancias
  - Se codificará un fichero denominado **distancias2D.hpp** que contenga el prototipo y la definición de las funciones que calculan disntacias
    - calcularDistanciaEuclidea2D
    - calcularDistanciaManhattan2D
    - calcularDistanciaAjedrez2D

- La plantilla de la clase Punto2D y de las funciones de distancias deberán permitir la ejecución del programa de ejemplo contenido en el fichero **practica_7.cpp**
- Se deberá crear un fichero **makefile** de compilación

- **Programa de prueba de distancias**
  - Se debe codificar un programa denominado **figura.cpp** que pida por pantalla las coordenadas de un punto (entero o float) y que calcule el valor que le corresponde según su posición dentro de la figura que se muestra más abajo, donde:
    - El círculo está centrado en el punto (3,3) y su radio es 2
    - El rombo está centrado en en el punto (5,3) y la distancia del centro a los vértices es 2
    - El cuadrado está centrado en en el punto (7,3) y la distancia del centro a los lados es 2
    - Si punto pertenece a la circunferencia o a algún lado del rombo o el cuadrado entonces el valor que le corresponde es cero.



- Se deberá crear un fichero **makefile** de compilación