Guion de prácticas

Compilación separada (1)

Febrero de 2019

Metodología de la Programación

Curso 2018/2019

Contents

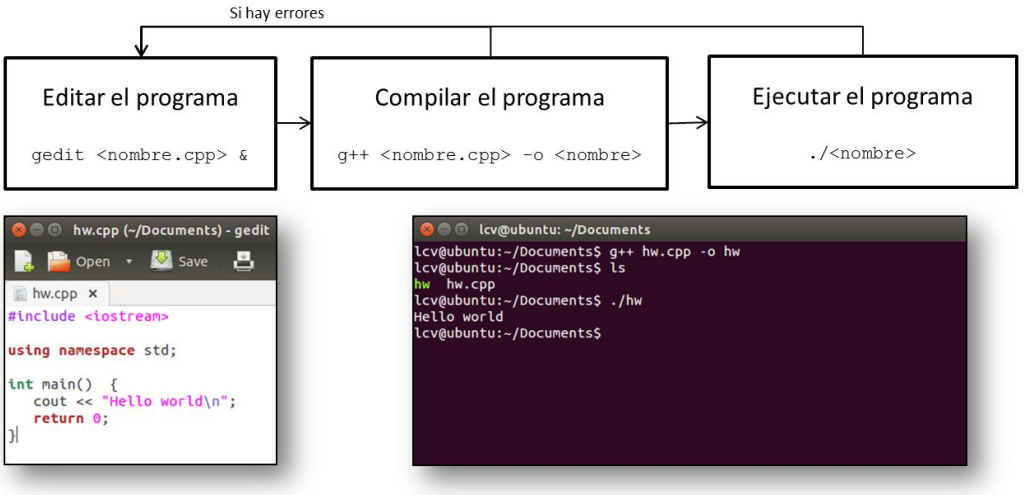

Figure 1: Ciclo de vida de la compilación y ejecución de programas en C++ en Linux

1 Objetivos

Para esta sesión de prácticas, el alumno deberá entender los conceptos relacionados con la compilación separada al tiempo que se revisan los conceptos de clases y vectores de objetos. (ver tema 12 del libro Garrido, A. "**Fundamentos de programación en C++**", Delta Pub., 2005). Se recuerda que el trabajo en estos ejercicios es por parejas, aunque el conocimiento adquirido es individual, esto es, la defensa es individual, además es indispensable la asistencia a clase de prácticas para una defensa. La copia de código no aporta nada al aprendizaje y será considerada como un incumplimiento de las normas de la asignatura con las consecuencias que ello implica, según la normativa de la Universidad de Granada.

2 Un breve apunte sobre la compilación de C++ en Linux

Vamos a seguir el ciclo de edición-compilación-ejecución en una instalación de Linux estándar.

2.1 Edición

Los ficheros con extensión cpp se pueden editar con programas estandar de la consola de Linux como **vi**, **vim** o en modo gráfico (Gnome) con programas como **gedit** (Ver Figura 1).

2.2 Compilación

Una vez guardado el programa en un fichero con extensión **cpp** el siguiente paso es compilarlo con **g++** mediante la orden mostrada en la Figura 1.

Los errores y/o advertencias que pueda generar el compilador son del mismo tipo de las que se obtenían al utilizar tanto **DevC++** como **Code-Blocks**. Dichos entornos utilizan una versión de **g++** para Windows, y por tanto, la gestión de los errores no deberá representar ningún problema[1].

2.3 Ejecución

Si la compilación ha funcionado correctamente, la ejecución del programa se realiza tal y como muestra la Figura 1, observando en la consola el resultado.

3 Un proyecto de ejemplo: circulomedio

El objetivo es la creación del tipo de dato **Punto**, como una estructura con dos campos (las coordenadas de un punto 2D), y la del tipo de dato **Circulo**, como una estructura con dos campos (un punto correspondiente al centro y un valor indicando el radio).

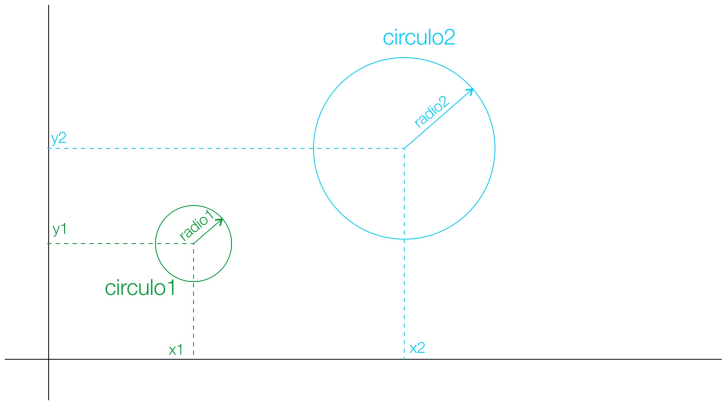

Figure 2: Dos círculos representados en base a los puntos de sus centros y sus radios

Con el fin de centrarnos en la compilación separada, se proporciona en DECSAI la totalidad del código que implementa un programa que lee dos círculos desde la entrada estándar, y escribe en la salida estándar el círculo que tiene, como centro, el punto medio de los dos centros de entrada, y cuyo radio es la mitad de la distancia entre éstos.

Descargue este programa (fichero **circulomedio.zip**) en una carpeta llamada `Version1` y descomprímalo (Figura 4).

A continuación compílelo.

[1] Para más detalles sobre el proceso de compilación y ejecución en Linux, consulte el Guión “Introducción a la Compilación de Programas en C++”

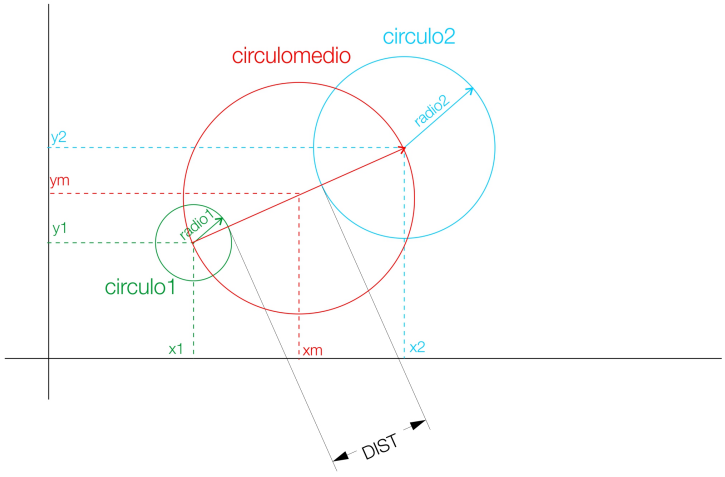

Figure 3: Cálculo del círculo medio y la distancia entre dos círculos

```
./
circulomedio.cpp
circulomedio.dat
circulomedio.doxy
doc
    tex
    html
        index.html
```

Figure 4: Contenido del fichero **circulomedio.zip**

```
g++ circulomedio.cpp -o circulomedio
```

A continuación ejecute el programa, para comprobar su correcto funcionamiento.

```
./circulomedio
```

Se proporciona también un fichero de validación `circulomedio.dat`.

```
./circulomedio < circulomedio.dat
```

El zip descargado también incorpora un fichero de documentación automática **circulomedio.doxy** que se explicará en un guión de prácticas independiente y requiere tener instalado el programa **doxygen**. Para generar y visualizar esta documentación de forma automática se procederá de la siguiente forma

```
doxygen circulomedio.doxy
firefox doc/html/index.html &
```

La Figura 5 muestra parte de esta documentación generada de forma automática.

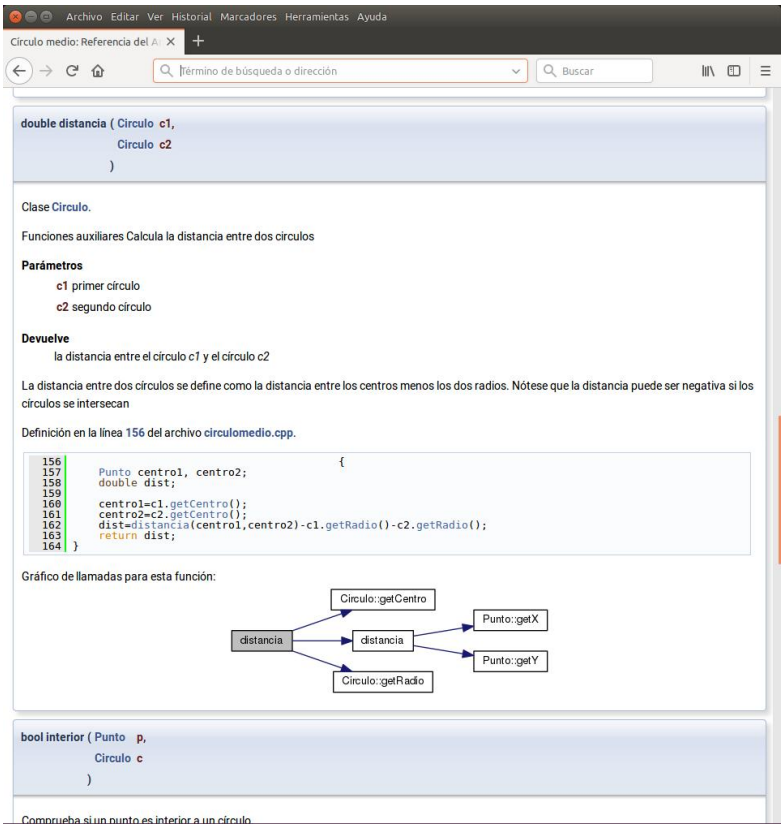

Figure 5: Parte de la documentación automática generada por doxygen

4 Versión 2. Implementación en módulos separados

Cree una nueva carpeta llamada **Version2** donde se hará una nueva versión del programa. A continuación, deberá dividir el programa que ya ha implementado y ejecutado en la sección anterior en los siguientes módulos distintos:

1. Módulo *Punto*: implementado en *punto.h* y *punto.cpp*. Contiene el código para manejar el tipo de dato *Punto*.

2. Módulo *Circulo*: implementado en *circulo.h* y *circulo.cpp*. Contiene el código para manejar el tipo de datos *Circulo*. Hace uso del módulo *Punto*.

3. Módulo *Central*: implementado en *central.cpp*. Contiene el código que implementa el programa de cálculo del círculo central y su área. Hace uso de los módulos *Punto* y *Circulo*.

Tenga en cuenta que para evitar dobles inclusiones, los ficheros `.h` deben contener directivas del preprocesador de la forma siguiente:

```
#ifndef _FICHERO_H_
#define _FICHERO_H_
...
#endif
```

En los ficheros `.h` se incluirán la definición de las estructuras y los prototipos de las funciones (su cabecera más punto y coma). Por ejemplo, el contenido de `punto.h` sería algo así:

```
#ifndef _PUNTO_H
#define _PUNTO_H

class Punto {
private:
    double x;
    double y;
public:
    Punto(){x=0;y=0;}
    Punto(double _x, double _y){x=_x;y=_y;}
...
};

double distancia(Punto p1, Punto p2);
Punto puntoMedio(Punto p1, Punto p2);
```

Para poder compilar bien los ficheros `cpp` deberán incluirse donde sea necesario, los ficheros `.h` con la directiva:

```
#include "punto.h"
```

Cuando tenga los tres módulos, se deberán compilar para obtener los archivos **punto.o**, **circulo.o** y **central.o**.

```
g++ -c central.cpp -o central.o
g++ -c punto.cpp -o punto.o
g++ -c circulo.cpp -o circulo.o
```

Una vez que dispone de los dos archivos objeto, deberá enlazarlos para obtener el ejecutable **circulomedio** de la aplicación propuesta.

```
g++ central.o punto.o circulo.o -o circulomedio
```

Ejecute el nuevo programa para comprobar el funcionamiento.

Si modificamos el fichero `punto.cpp`, ¿qué órdenes sería necesario volver a ejecutar para obtener de nuevo el ejecutable? ¿Y si modificamos `punto.h`?

5 Versión 3. Implementación en carpetas separadas

Cree una nueva carpeta llamada `Version3` donde se hará una nueva versión del programa. Dentro de ella, deberá crear los directorios **include** para los ficheros `.h`, **src** para los ficheros `.cpp`, **obj** para los ficheros `.o`, **lib** para alojar las bibliotecas y **bin** para los ejecutables. Se crearán

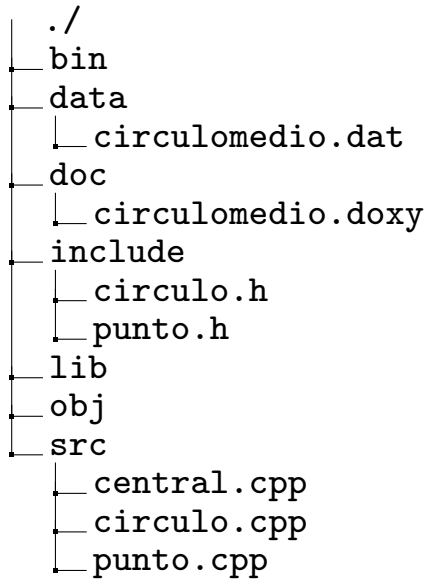

Figure 6: Estructura básica de las carpetas de un proyecto con múltiples módulos antes de compilar y enlazar

también las carpetas **doc** para documentación y **data** para ficheros auxiliares y se colocará cada fichero en su sitio (Figura 6).

Una vez ordenados los ficheros por carpetas, se procede a compilar, cogiendo cada fichero desde su carpeta y colocando cada fichero de salida en su carpeta correspondiente (Figura 7).

```
g++ -c src/central.cpp -o obj/central.o -Iinclude
g++ -c src/punto.cpp -o obj/punto.o -Iinclude
g++ -c src/circulo.cpp -o obj/circulo.o -Iinclude
g++ obj/central.o obj/punto.o obj/circulo.o -o bin/circulomedio
```

6 Versión 4. Creación de una biblioteca

Cree una nueva carpeta llamada `Version4` donde se hará una nueva versión del programa. En esta sesión deberá crear una biblioteca sobre la estructura creada en la Sección 5. Concretamente, debe realizar las siguientes tareas:

1. Compile los ficheros sin enlazar el binario aún

   ```
   g++ -c src/central.cpp -o obj/central.o -Iinclude
   g++ -c src/punto.cpp -o obj/punto.o -Iinclude
   g++ -c src/circulo.cpp -o obj/circulo.o -Iinclude
   ```

2. Cree una biblioteca con el archivos **punto.o** y **circulo.o** con nombre **libformas.a**.

   ```
   ar rvs lib/libformas.a obj/punto.o obj/circulo.o
   ```

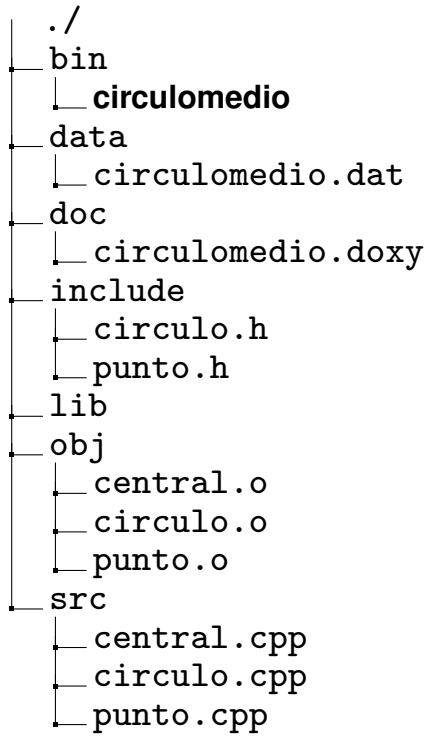

Figure 7: Estructura básica de las carpetas de un proyecto con múltiples módulos después de compilar y enlazar

3. Ejecute la orden para crear el ejecutable teniendo en cuenta esta biblioteca, es decir, sin enlazar directamente con los archivos objeto.

```
g++ obj/central.o -lformas -o bin/circulomedio -Llib
```

   Indique de nuevo qué debemos hacer si modificamos el fichero `punto.h`, ¿qué órdenes sería necesario volver a ejecutar para obtener de nuevo el ejecutable? ¿Y si modificamos `punto.cpp`?

7 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 pueden ser abierto o cerrado, y se nota respectivamente por $(,)$ y $[,]$. 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 debe poder distinguir entre los siguientes 4 intervalos: $[0..10], (0..10], [0..10), (0..10)$.
Se quiere implementar la clase Intervalo.

Descargue el fichero **intervalo.zip** (Figura 9) y descomprímalo en una carpeta independiente. Complete las funciones y/o métodos incompletos. Para realizar esta tarea, tenga en cuenta que el objetivo es que escriba el programa de forma conjunta, entendiendo perfectamente **todo** el código que lo compone.

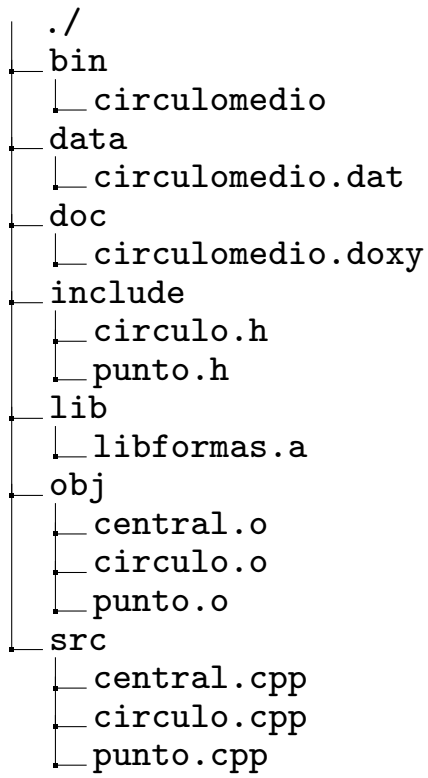

Figure 8: Estructura básica de las carpetas de un proyecto con múltiples módulos después de crear y usar una biblioteca

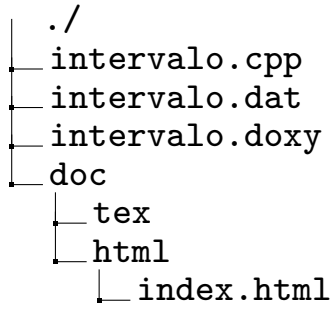

Figure 9: Contenido del fichero **intervalo.zip**

- 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 éste debe estar asociado al constructor sin parámetros.
  En este problema, no se consideran intervalos con extremos infinitos como por ejemplo $(-\infty, \infty)$.
- 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 compruebe si un valor númerico 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. Ejemplo de entrada pueden encontrarlo en **intervalo1.dat**, donde se lee 1 intervalo y 6 puntos.

```
---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 en el fichero **intervalo.cpp** que tendrá que completar.

Una vez completado el programa, descompóngalo en múltiples ficheros y carpetas y compile el proyecto completo tal y como se ha visto en las secciones anteriores.

8 Entrega

Se debe de entregar através de **DECSAI** un fichero zip, **practica1.zip** la estructura de directorios ya expuesta: **bin, data, include, lib, obj, src, doc**.

Donde **src** debe contener los ficheros **intervalo.cpp, main.cpp**, **data** contiene el fichero **intervalo.dat**. **include** el fichero (*h) correspondiente la versión 2 de la práctica. Borre previamente el ejecutable y los objetos, esto es, **bin** y **obj** deben estar vacíos. Los comandos para llevar a cabo el empaquetamiento es el siguiente:

```
rm -f obj/*.o lib/*.a bin/*
zip -r practica1.zip *
```

El fichero **practica1.zip** debe contener la siguiente estructura:

```
./
├── include
│   └── intervalo.h
├── src
│   ├── intervalo.cpp
│   └── main.cpp
├── bin
├── obj
├── lib
├── data
│   └── intervalo.dat
└── doc
    └── intervalo.doxy
```

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 se 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: no se modifica el contenido del programa principal**. Deben implementarse todas las funciones necesarias para que éste se ejecute de la forma indicada.

9 Apendice 1. Modularización

9.1 punto.h

```
#ifndef _PUNTO_H
#define _PUNTO_H
class Punto {
private:
    double x; ///< coordenada x del punto
    double y; ///< coordenada y del punto};
public:
    Punto(){x=0;y=0;} ///< constructor. Pone a 0 las dos coordenadas
    Punto(double _x, double _y){x=_x; y=_y;}///< constructor. Inicializa un punto con dos valores x y
    double getX() const {return x;} ///< Devuelve la coordenada x del punto
    double getY() const {return y;} ///< Devuelve la coordenada y del punto
    void setX(double nuevoX); ///< Asigna el valor nuevoX a la coordenada x del punto
    void setY(double nuevoY); ///< Asigna el valor nuevoY a la coordenada y del punto
    void escribir() const; ///<  Escribe un punto en formato (x,y)
    void leer(); ///< // Lee un punto en el formato (x,y)
};


double distancia(Punto p1, Punto p2);
Punto puntoMedio(Punto p1, Punto p2);
#endif
```

9.2 circulo.h

```
#ifndef _CIRCULO_H
#define _CIRCULO_H
#include "punto.h"

/// Clase CÃrculo
class Circulo {
private:
    Punto centro; ///< Centro del cÃrculo
    double radio; ///< radio del cÃrculo
public:
    Circulo(); ///< Constructor: Pone a 0 el punto y el radio
    Circulo(Punto centro, double radio); ///< Constructor: Inicializa el cÃrculo con un centro y un radio
    void set(Punto centro, double radio); ///< Asigna el centro y el radio a un circulo
    Punto getCentro() const; ///< Devuelve el centro de un circulo
    double getRadio() const; ///< Devuelve el radio de un circulo
    void escribir() const; ///< Escribe cÃrculo en formato radio-centro
    void leer(); ///< lee cÃrculo en formato radio-centro
    double area() const; ///< Devuelve el Ã¡rea de un cÃrculo
};

double distancia (Circulo c1, Circulo c2);
bool interior (Punto p, Circulo c);
#endif
```

9.3 punto.cpp

```
#include <iostream>
#include <cmath>
#include "punto.h"

using namespace std;
```

```
void Punto::setX(double nuevoX)  {
  x = nuevoX;
}
...

Punto puntoMedio(Punto p1, Punto p2){
    Punto pMedio;
    pMedio.setX((p1.getX()+p2.getX())/2.0);
    pMedio.setY((p1.getY()+p2.getY())/2.0);
    return pMedio;
}
```

9.4 circulo.cpp

```
#include <iostream>
#include <cmath>
#include "circulo.h"

using namespace std;

Circulo::Circulo()  {
  centro.setX(0);
  centro.setY(0);
  radio = 0;
}

Circulo::Circulo(Punto centro, double radio)  {

}
...
```

9.5 central.cpp

```
#include <iostream>
#include "punto.h"
#include "circulo.h"
using namespace std;

int main() {
    Circulo c1,c2;
     ...
    return 0;
}
```