



## Guion de prácticas

### *Entorno de desarrollo de software con NetBeans*

Febrero 2019



## Metodología de la Programación

DGIM

Curso 2018/2019



# Índice

<b>1. Introducción</b>	<b>5</b>
<b>2. Empezando</b>	<b>5</b>
<b>3. Un primer proyecto</b>	<b>7</b>
3.1. Creación del proyecto NetBeans . . . . .	7
3.2. Creación de un fichero fuente . . . . .	8
3.3. Generando y ejecutando el binario . . . . .	8
<b>4. Compilación separada</b>	<b>10</b>
4.1. Crear la estructura del proyecto . . . . .	12
4.2. Configurar la vista lógica del proyecto . . . . .	12
4.3. Configurar los parámetros del proyecto . . . . .	13
<b>5. Bibliotecas</b>	<b>13</b>
5.1. Crear una nueva biblioteca a partir de ficheros fuentes ya existentes . . . . .	13
5.2. Usar la nueva biblioteca en el proyecto anterior . . . . .	16
<b>6. Depurador</b>	<b>18</b>
6.1. Conceptos básicos . . . . .	18
6.2. Ejecución de un programa paso a paso . . . . .	18
6.3. Inspección y modificación de datos . . . . .	19
6.4. Inspección de la pila . . . . .	19
6.5. Reparación del código . . . . .	20
<b>7. Otras características de NetBeans</b>	<b>20</b>
<b>8. Personalización de NetBeans</b>	<b>24</b>
8.1. Integración de doxygen . . . . .	24
8.2. Empaquetado de las prácticas para su entrega . . . . .	25
<b>9. Apéndice 1: circulomedio.cpp original</b>	<b>26</b>
<b>10. Apendice 2. Modularización</b>	<b>28</b>
10.1.punto.h . . . . .	28
10.2.circulo.h . . . . .	28
10.3.punto.cpp . . . . .	28
10.4.circulo.cpp . . . . .	29
10.5.central.cpp . . . . .	29



## 1. Introducción

Esta sesión de prácticas está dedicada a aprender a utilizar el entorno de desarrollo de software multi-language **NetBeans** como alternativa a la gestión de proyectos desde la línea de comandos. NetBeans es un entorno de desarrollo integrado libre y multiplataforma, creado principalmente para el lenguaje de programación Java, pero que ofrece soporte para otros muchos lenguajes de programación. Existe además un número importante de módulos para extenderlo. NetBeans es un producto libre y gratuito sin restricciones de uso. En este guión se explicará cómo utilizarlo en un **Linux** que ya tenga instalado **g++** y **make**.

## 2. Empezando

Es necesario instalar NetBeans con JDK <sup>1</sup> y el plugin para **C++** <sup>2</sup> instalados en ese orden. También se puede instalar el plugin de **C++** desde el menú “Tools-Plugins” del menú de NetBeans.

Una vez instalado se ejecuta desde la línea de comandos (para la versión 8.2)

```
/usr/local/netbeans-8.2/bin/netbeans
```

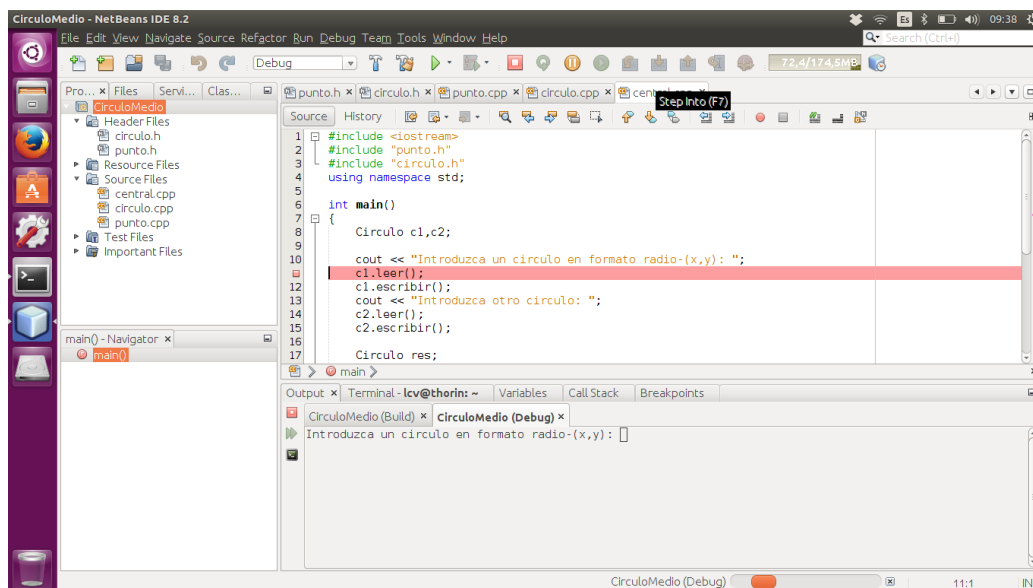


Figura 1: Entorno de trabajo en NetBeans con C++

En la Figura 1 se puede ver la distribución de las áreas de trabajo en NetBeans:

- La parte superior contiene un menú de opciones típico de un entorno de desarrollo de software del que se irá detallando lo más importante

<sup>1</sup><http://www.oracle.com/technetwork/es/java/javase/downloads/jdk-netbeans-jsp-3413139-esa.html>

<sup>2</sup><https://netbeans.org/features/cpp/>

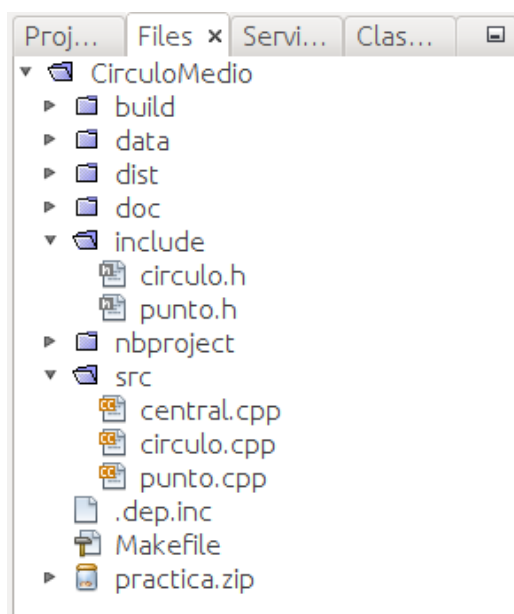


Figura 2: Estructura en disco de un proyecto NetBeans

a lo largo de este guión. En cualquier caso, para más información se puede consultar la guía oficial de NetBeans<sup>3</sup>.

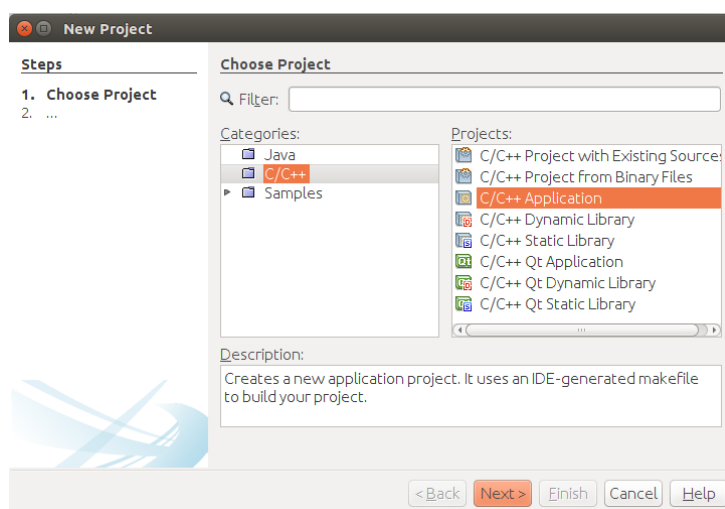
- El cuadrante superior izquierdo muestra el navegador del proyecto, el cual puede estar en la visión lógica del proyecto, tal y como muestra la figura, o la visión física, que muestra la estructura de carpetas y ficheros real en disco (Figura 2).
- El cuadrante superior derecho muestra las pestañas para editar ficheros fuente.
- El cuadrante inferior izquierdo, que muestra el contexto del código (funciones, pila, variables locales) durante las sesiones de depuración.
- El cuadrante inferior derecho que muestra múltiples pestañas asociadas con la ejecución del proyecto:
  - Output. Muestra las salidas estándar y de error y recoge la entrada estándar.
  - Terminal. Línea de comandos en la carpeta principal del proyecto.
  - Variables. Inspección de variables durante la depuración.
  - Call Stack. Estado de la pila de llamadas (depuración).
  - Breakpoints. Lista de puntos de ruptura activos (depuración).

<sup>3</sup><https://netbeans.org/kb/docs/java/quickstart.html>

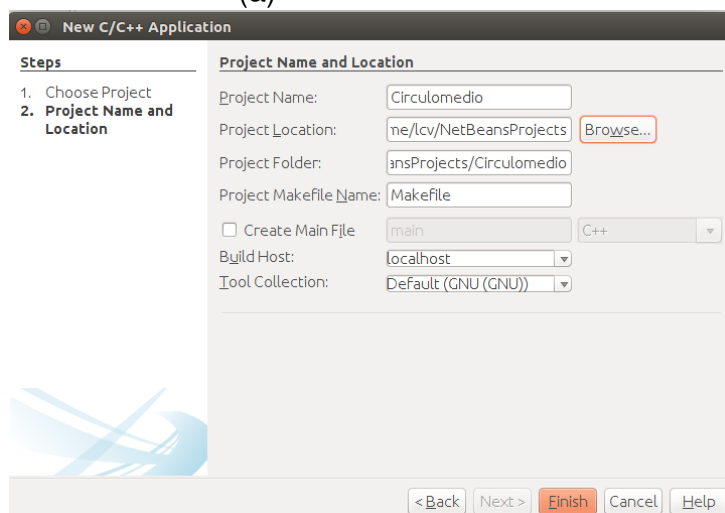
## 3. Un primer proyecto

### 3.1. Creación del proyecto NetBeans

En el navegador de proyecto (cuadrante superior izquierda) con la pestaña "Projects" activa, pulsar con el botón derecho y seleccionar "New Project" (alternativamente "File" - "New Project"). Seleccionar el tipo de aplicación que se quiere construir (Figura 3.a), entre los múltiples lenguajes soportados por NetBeans, en este caso "C/C++ Application". Seleccionar el nombre del proyecto ("CirculoMedio") y la ubicación que se le quiere dar en disco "Project location / Browse" (Figura 3.b). Asegurarse de que la opción "Create Main File" está desactivada.



(a)



(b)

Figura 3: Creando un proyecto nuevo

En el navegador de proyectos (cuadrante superior izquierdo) aparecerán los árboles lógicos y físicos del proyecto recién creado (Figura 4).

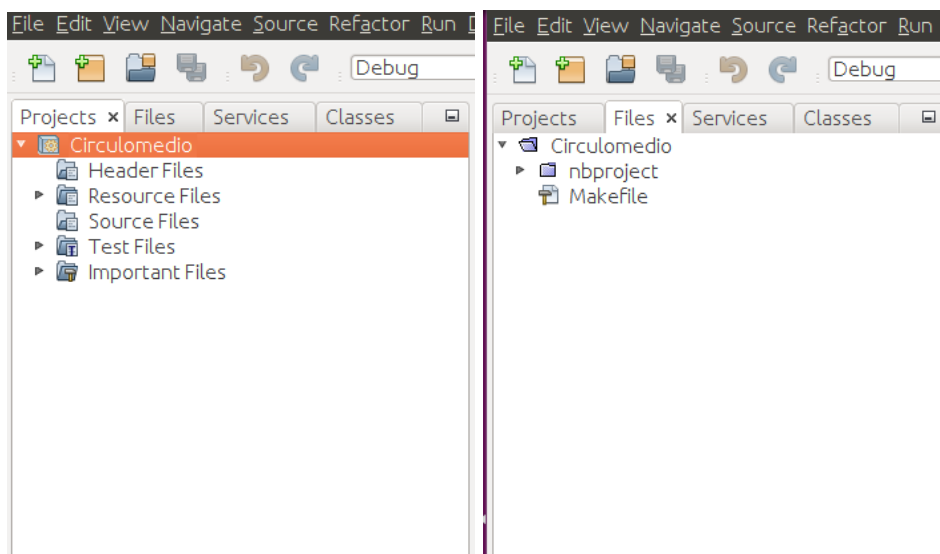


Figura 4: Árbol lógico (izquierda: pestaña “Projects”) y físico (derecha: pestaña “Files”) de un proyecto nuevo

### 3.2. Creación de un fichero fuente

Vamos a crear el primer fichero fuente del proyecto. Para ello se procede con el menú “File”-“New File” y se selecciona “C++ Source File” (Figura 5).

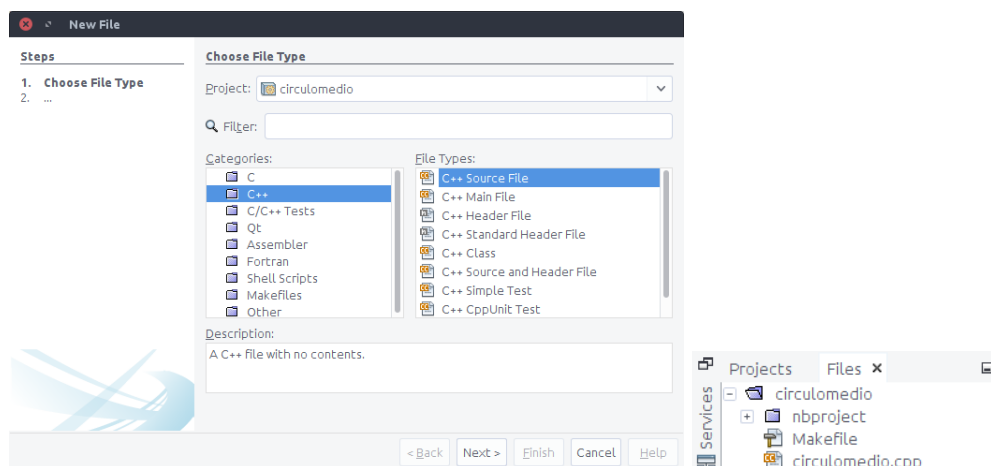


Figura 5: Creación de un nuevo fichero fuente en el proyecto

A continuación se introduce el código de ejemplo “circulomedio.cpp” y se guarda “File-Save”.


### 3.3. Generando y ejecutando el binario

Desde NetBeans se puede compilar el proyecto y ejecutarlo en un mismo paso:

**Run - Clean and Build Project**



Cuya salida aparece en el cuadrante inferior derecho, pestaña **Output** (Figura 6).



```

Output - CirculoMedio (Clean, Build) x
cd '/home/lcv/Dropbox/Teaching/MP/MP1617/NetBeans/CirculoMedio'
/usr/bin/make -f Makefile CONF=Debug clean
"/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .clean-conf
make[1]: se entra en el directorio '/home/lcv/Dropbox/Teaching/MP/MP1617/NetBeans/CirculoMedio'
rm -f -r build/Debug
make[1]: se sale del directorio '/home/lcv/Dropbox/Teaching/MP/MP1617/NetBeans/CirculoMedio'

CLEAN SUCCESSFUL (total time: 659ms)
cd '/home/lcv/Dropbox/Teaching/MP/MP1617/NetBeans/CirculoMedio'
/usr/bin/make -f Makefile CONF=Debug
"/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .build-conf
make[1]: se entra en el directorio '/home/lcv/Dropbox/Teaching/MP/MP1617/NetBeans/CirculoMedio'
"/usr/bin/make" -f nbproject/Makefile-Debug.mk dist/Debug/GNU-Linux/circulomedio
make[2]: se entra en el directorio '/home/lcv/Dropbox/Teaching/MP/MP1617/NetBeans/CirculoMedio'
mkdir -p build/Debug/GNU-Linux/src
rm -f "build/Debug/GNU-Linux/src/central.o.d"
g++ -c -g -Iinclude -MMD -MP -MF "build/Debug/GNU-Linux/src/central.o.d" -o build/Debug/GNU-Linux/src/central.o src/central.cpp
mkdir -p build/Debug/GNU-Linux/src
rm -f "build/Debug/GNU-Linux/src/circulo.o.d"
g++ -c -g -Iinclude -MMD -MP -MF "build/Debug/GNU-Linux/src/circulo.o.d" -o build/Debug/GNU-Linux/src/circulo.o src/circulo.cpp
mkdir -p build/Debug/GNU-Linux/src
rm -f "build/Debug/GNU-Linux/src/punto.o.d"
g++ -c -g -Iinclude -MMD -MP -MF "build/Debug/GNU-Linux/src/punto.o.d" -o build/Debug/GNU-Linux/src/punto.o src/punto.cpp
mkdir -p dist/Debug/GNU-Linux
g++ -o dist/Debug/GNU-Linux/circulomedio build/Debug/GNU-Linux/src/central.o build/Debug/GNU-Linux/src/circulo.o build/Debug/GNU-Linux/src/punto.o
make[2]: se sale del directorio '/home/lcv/Dropbox/Teaching/MP/MP1617/NetBeans/CirculoMedio'
make[1]: se sale del directorio '/home/lcv/Dropbox/Teaching/MP/MP1617/NetBeans/CirculoMedio'

BUILD SUCCESSFUL (total time: 12s)

```

Figura 6: Generando el binario

El binario recién generado se encuentra en la carpeta **dist** tal y como muestra la Figura 7.

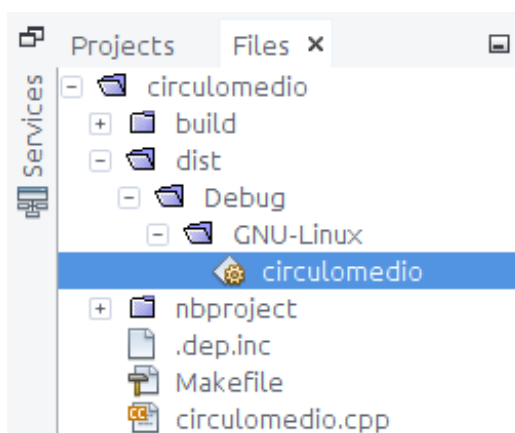


Figura 7: Situación del binario en el árbol físico del proyecto, dentro de la carpeta **dist**

La ejecución del binario que se acaba de generar se ordena como **Run - Run Project**

Y se puede seguir su ejecución, introduciendo los valores correspondientes desde la pestaña **Output** (Figura 8) como si fuese una consola de órdenes del sistema operativo.

Para introducir parámetros en la llamada al programa o redireccionamientos de la entrada o salida del programa (por ejemplo con ficheros de validación de datos) es necesario modificar las propiedades del proyecto.

**Project - Project Properties - Run - Run Command**

e introducir las modificaciones según se muestra en la Figura 9 para indicar que el binario se va a ejecutar redireccionando la entrada estándar

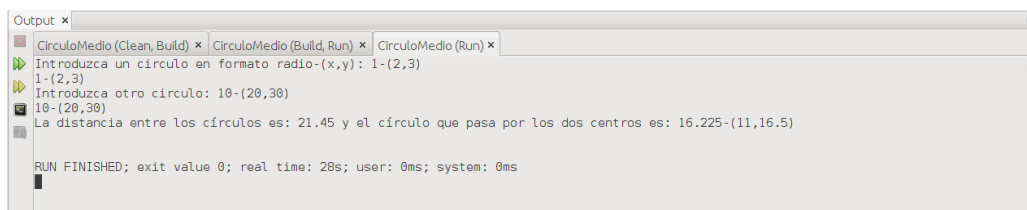


Figura 8: Ejecutando el binario

desde **circulomedio.dat**. Este fichero de validación deberá colocarse en la carpeta raíz del proyecto (bien desde la línea de comandos, bien usando el explorador de archivos o desde dentro de NetBeans).

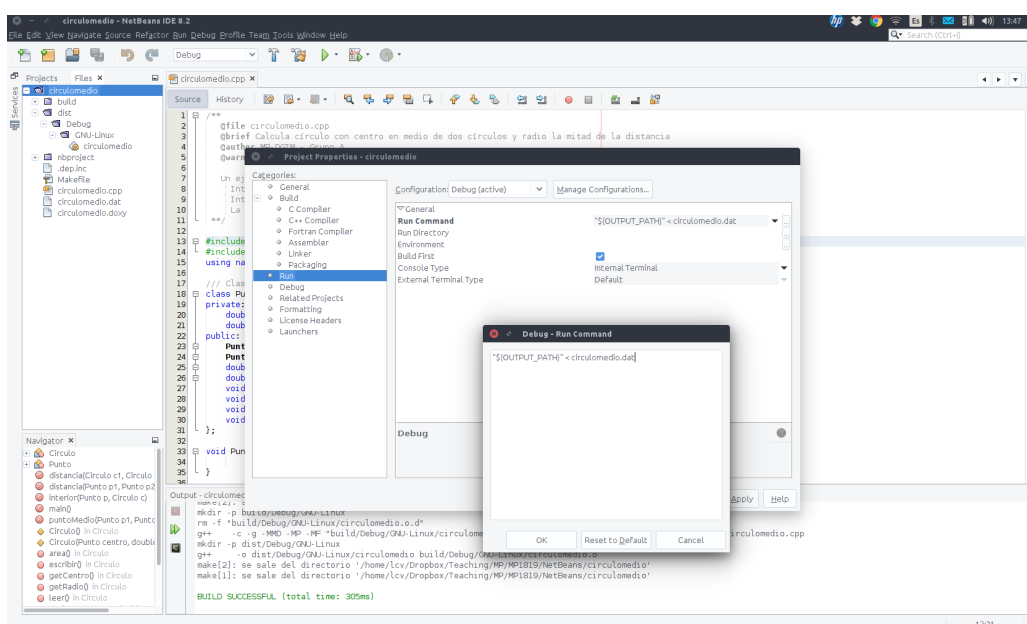


Figura 9: Ejecutando el binario con argumentos o redireccionamientos

## 4. Compilación separada

Esta sección describe cómo crear una nueva versión del programa, para lo cual 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*.

```

./
├── dist
├── build
├── nbproject
├── data
│   └── circulomedio.dat
├── doc
│   └── circulomedio.doxy
├── include
│   ├── circulo.h
│   └── punto.h
├── src
│   ├── central.cpp
│   ├── circulo.cpp
│   └── punto.cpp
└── zip

```

Figura 10: Estructura básica de las carpetas de un proyecto con múltiples módulos. Las tres primeras subcarpetas (dist, build y nbproject) son creadas y mantenidas automáticamente por NetBeans. El resto deben crearse a mano en cada proyecto.

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) {this->x=x;this->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"
```

Estos módulos deberán colocarse en una típica estructura de proyecto con subcarpetas dedicadas:

Para crear esta estructura en NetBeans hay que seguir los siguientes pasos.

## 4.1. Crear la estructura del proyecto

Desde la pestaña “Files” del navegador de proyectos, crear la estructura que muestra la Figura 11 a la derecha, y organizar cada fichero de los comentados anteriormente.

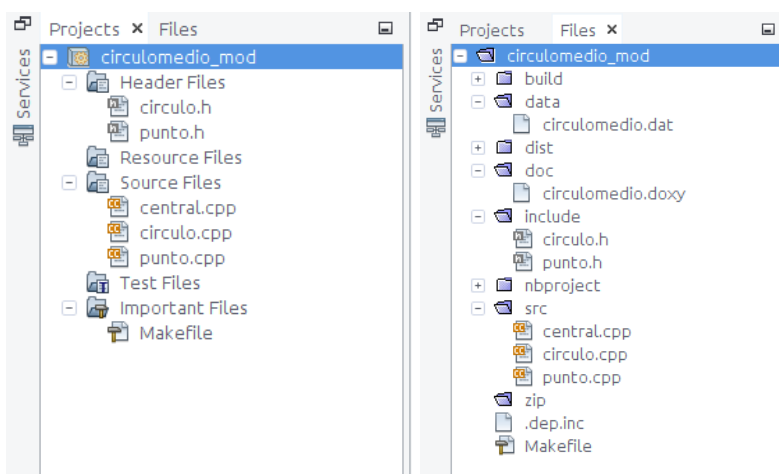


Figura 11: Estructura de carpetas del proyecto, en su visión lógica (izquierda) y física (derecha)

## 4.2. Configurar la vista lógica del proyecto

En la vista lógica del proyecto, pestaña “Projects” es necesario indicar en qué carpeta se encuentran los ficheros .h y .cpp. Para ello se procede como sigue:

1. Header files. Pulsar con el botón derecho, “Add existing item...” y navegar hasta donde están los ficheros .h, seleccionarlos y añadirlos.
2. Source files. Pulsar con el botón derecho, “Add existing item...” y navegar hasta donde están los ficheros .cpp, seleccionarlos y añadirlos.

Esta operación no añade nuevos ficheros al proyecto, tan sólo le indica a NetBeans dónde están estos. Al concluir esta etapa, el proyecto aparece como muestra la Figura 11 a la izquierda.

### 4.3. Configurar los parámetros del proyecto

Ya casi está terminado. Las opciones del proyecto se definen a cabo desde la pestaña de propiedades del proyecto (Figura 12), la cual aparece con botón derecho, “Project properties”, “C++ compiler”, “Include directories” y se selecciona la carpeta “include” perteneciente al proyecto.

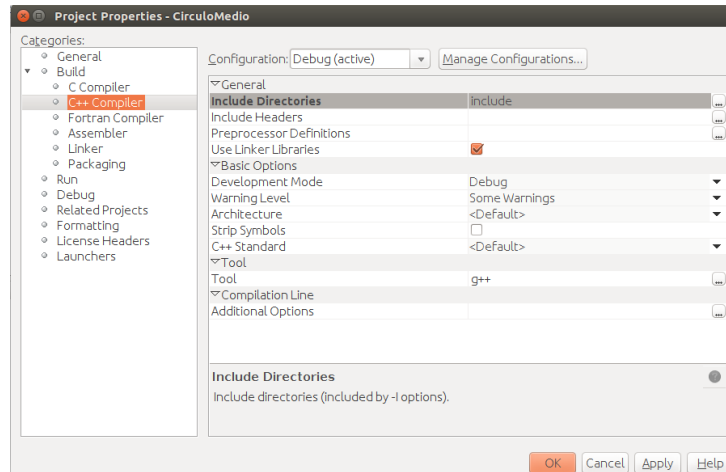


Figura 12: Propiedades del proyecto

1. Modo de depuración o de producción. Para recompilar todo el proyecto se elige en el campo **Configuration** y se elige **Debug** o **Release** respectivamente.
2. “C++ Standard”. Selecciona bajo qué estándar de C++ se va a compilar el proyecto. Seleccionaremos siempre “C++14”.

## 5. Bibliotecas

En NetBeans cada proyecto que se crea sirve para generar un único binario o biblioteca, de forma que para generar más de un binario habrá que crear más de un proyecto. Igualmente pasa con las bibliotecas para las que hace falta crear un proyecto individual (Figura 13). Desde el menú **File** o desde el navegador de proyectos será necesario crear un nuevo proyecto e indicar **C/C++ Static Library**, en vez de **C/C++ Application**. Le daremos un nombre al proyecto, que será el nombre de la biblioteca y se gestionará como un proyecto independiente de NetBeans.

### 5.1. Crear una nueva biblioteca a partir de ficheros fuentes ya existentes

1. En el árbol físico del proyecto de la biblioteca crear la estructura necesaria (carpetas **src** e **include** nada más) y duplicar los archivos desde la carpeta desde donde se encuentran actualmente, bien

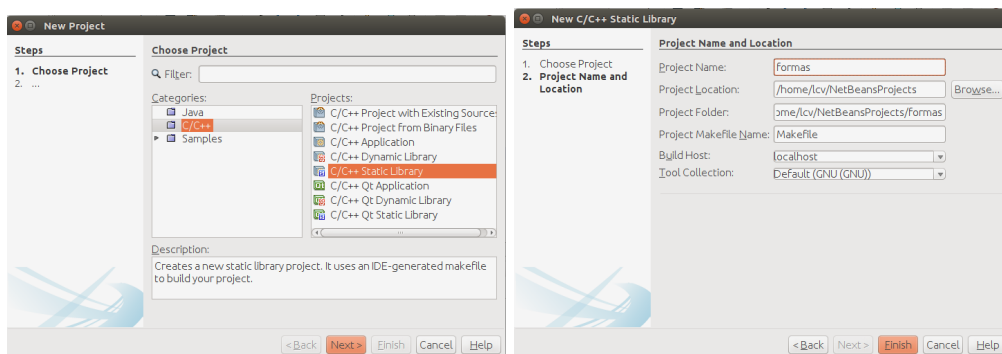


Figura 13: Creando un nuevo proyecto para gestionar una biblioteca

arrastrándolos y duplicándolos desde el navegador de proyectos, bien duplicándolos desde la línea de comandos.

2. En el árbol lógico del proyecto, añadir los **Header Files** y los **Source Files** que acabamos de duplicar usando el menú contextual (botón derecho del ratón) **Add Existing Item** y seleccionando los ficheros que acabamos de duplicar en el proyecto de la biblioteca. La Figura 14 muestra las vistas lógica (izquierda) y física (derecha) del nuevo proyecto de biblioteca.

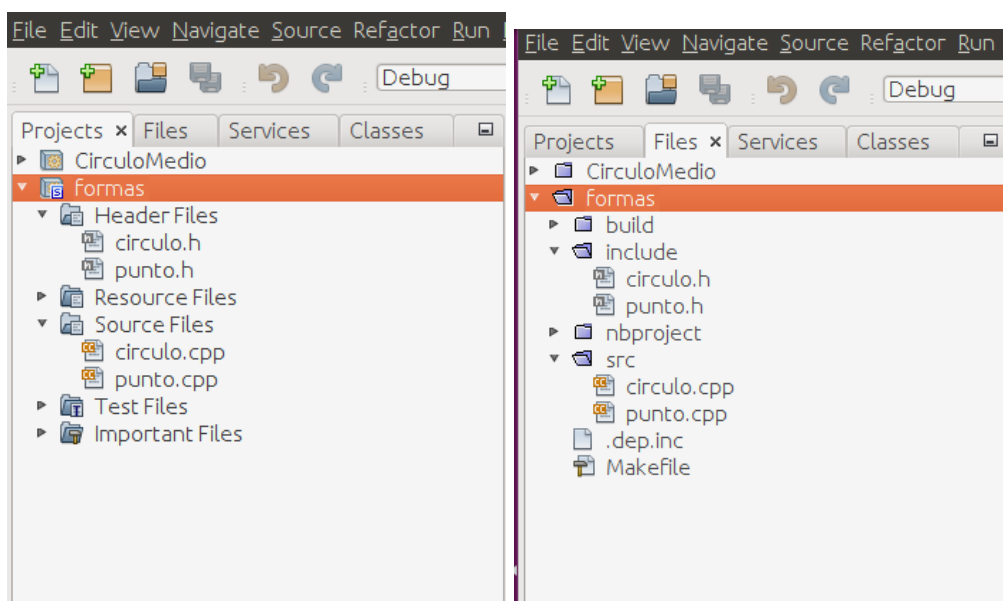


Figura 14: Árboles lógico (izqda) y físico (dcha) del nuevo proyecto de biblioteca

3. Añadir al proyecto de la biblioteca la carpeta de ficheros de cabeceras que se usará en el proyecto, igual que se hizo en el proyecto anterior (ver Figura 12) en el parámetro **Include Directories**.
4. Construir el proyecto de biblioteca (**Clean and Build Project**) igual que se hizo en el proyecto anterior, con la diferencia de que, en este



```

cd '/home/lcv/NetBeansProjects/formas'
/usr/bin/make -f Makefile CONF=Debug clean
"/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .clean-conf
make[1]: se entra en el directorio '/home/lcv/NetBeansProjects/formas'
rm -f -r build/Debug
make[1]: se sale del directorio '/home/lcv/NetBeansProjects/formas'

CLEAN SUCCESSFUL (total time: 484ms)
cd '/home/lcv/NetBeansProjects/formas'
/usr/bin/make -f Makefile CONF=Debug
"/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .build-conf
make[1]: se entra en el directorio '/home/lcv/NetBeansProjects/formas'
"/usr/bin/make" -f nbproject/Makefile-Debug.mk dist/Debug/GNU-Linux/libformas.a
make[2]: se entra en el directorio '/home/lcv/NetBeansProjects/formas'
mkdir -p build/Debug/GNU-Linux/src
rm -f "build/Debug/GNU-Linux/src/circulo.o.d"
g++ -c -g -Iinclude -MMD -MP -MF "build/Debug/GNU-Linux/src/circulo.o.d" -o build/Debug/GNU-Linux/src/circulo.o src/circulo.cpp
mkdir -p build/Debug/GNU-Linux/src
rm -f "build/Debug/GNU-Linux/src/punto.o.d"
g++ -c -g -Iinclude -MMD -MP -MF "build/Debug/GNU-Linux/src/punto.o.d" -o build/Debug/GNU-Linux/src/punto.o src/punto.cpp
mkdir -p dist/Debug/GNU-Linux
rm -f dist/Debug/GNU-Linux/libformas.a
ar -rv dist/Debug/GNU-Linux/libformas.a build/Debug/GNU-Linux/src/circulo.o build/Debug/GNU-Linux/src/punto.o
ar: creando dist/Debug/GNU-Linux/libformas.a
a - build/Debug/GNU-Linux/src/circulo.o
a - build/Debug/GNU-Linux/src/punto.o
ranlib dist/Debug/GNU-Linux/libformas.a
make[2]: se sale del directorio '/home/lcv/NetBeansProjects/formas'
make[1]: se sale del directorio '/home/lcv/NetBeansProjects/formas'

BUILD SUCCESSFUL (total time: 5s)

```

Figura 15: Creación de la biblioteca de forma automática desde NetBeans

caso, no se ha generado un binario, sino una biblioteca. En la Figura 15 se puede ver cómo NetBeans llama adecuadamente al compilador para compilar las fuentes y, posteriormente, al programa **ar** para generar la biblioteca. La biblioteca recién generada se encuentra en la carpeta **dist** tal y como muestra la Figura 16.

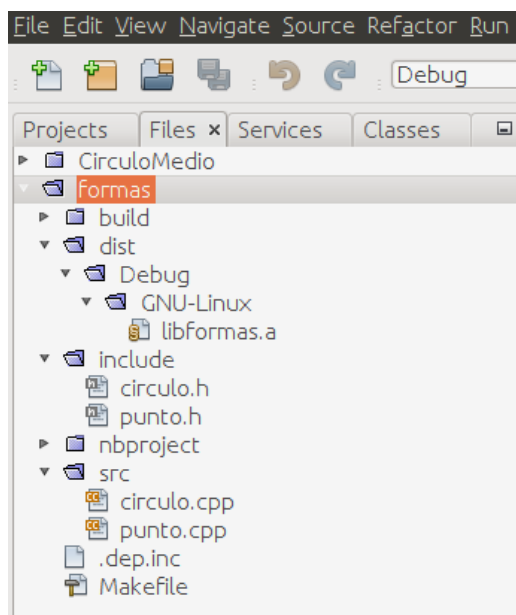


Figura 16: Situación de la biblioteca en el árbol físico del proyecto dentro de la carpeta **dist**

- Colocar los ficheros **.h** y **.a** de la nueva biblioteca en las carpetas que Linux tiene preparadas para esto y que son **/usr/local/include** y **/usr/local/lib** para que se puedan reutilizar de ahora en adelante por todos los proyectos futuros (Figura 17).

```
@thorin: ~
Archivo Editar Ver Buscar Terminal Ayuda
@thorin:~$ sudo cp NetBeansProjects/formas/dist/Debug/GNU-Linux/libformas.a /usr/local/lib/
@thorin:~$ sudo cp NetBeansProjects/formas/include/*.h /usr/local/include/
@thorin:~$
```

Figura 17: Colocar los .h y los .a en las carpetas del sistema para su uso posterior en otros proyectos

## 5.2. Usar la nueva biblioteca en el proyecto anterior

En este caso, se va a mostrar cómo introducir la biblioteca recién creada y colocada dentro del proyecto **CirculoMedio** que sirve como ejemplo hasta ahora.

1. Eliminar del árbol lógico del proyecto los ficheros .h y .cpp que han pasado a formar parte de la biblioteca porque ya no se van a usar más desde aquí, sino desde el proyecto de la biblioteca. El nuevo árbol lógico queda como muestra la Figura 18.

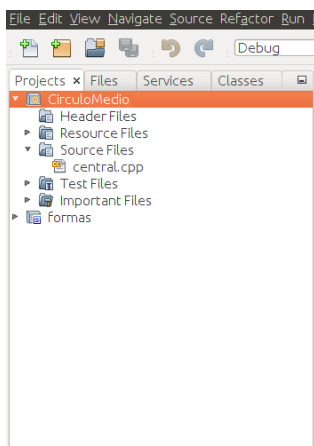


Figura 18: Nuevo árbol lógico del proyecto, que excluye a todos los .h y .cpp que se han movido a la biblioteca recién creada

2. En las propiedades del proyecto (Figura 19 arriba), eliminar la carpeta local y añadir la carpeta de includes del sistema en la opción **Include Directories**. Los includes del código se pueden dejar con comillas dobles o como ángulos indiferentemente a partir de ahora.
3. En las propiedades del proyecto pero en las opciones del enlazador (**Linker** Figura 19 abajo), añadir la biblioteca recién creada en la opción **Libraries**.
4. Crear el nuevo binario (**Clean and Build Project**) y observar (Figura 20) cómo se enlaza con la nueva biblioteca.



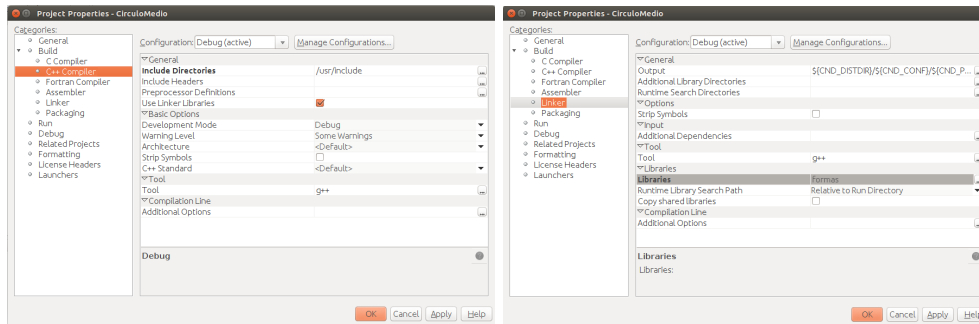


Figura 19: Configuración del proyecto para incluir la carpeta de cabeceras del sistema (arriba) y la librería recién creada (abajo)

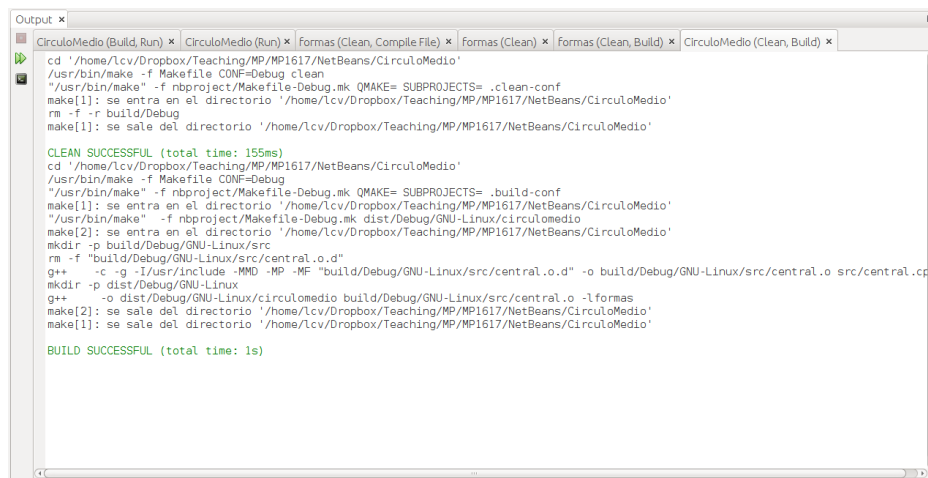


Figura 20: Salida de la creación del nuevo proyecto (**Clean and Build Project**) en la que se puede apreciar el enlazado con la biblioteca recién creada

## 6. Depurador

### 6.1. Conceptos básicos

NetBeans actúa, además, como una interfaz separada que se puede utilizar con un depurador en línea de órdenes. En este caso será la interfaz de alto nivel del depurador **gdb**. Para poder utilizar el depurador es necesario compilar los ficheros fuente con la opción **-g** o, lo que es lo mismo, con la configuración **Debug** en la ventana de propiedades del proyecto NetBeans.

### 6.2. Ejecución de un programa paso a paso

Para comenzar a ejecutar un programa bajo control del depurador es conveniente colocar un punto de ruptura<sup>4</sup>. NetBeans permite crear un PR simplemente haciendo click sobre el número de línea correspondiente en la ventana de visualización de código y visualiza esta marca como una línea en rojo (Figura 21).

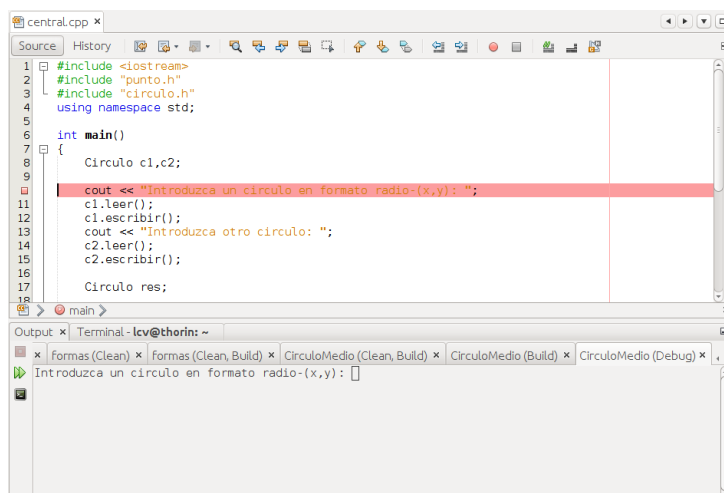


Figura 21: Creando un punto de ruptura para depurar un programa. El PR aparece como una línea de color rojo

Una vez colocado este punto de ruptura se puede comenzar la ejecución del programa con el menú<sup>5</sup>

#### Debug - Debug Project

NetBeans señala la línea de código activa con una pequeña flecha verde a la izquierda de la línea y remarca toda la línea en color verde (Figura 22). A la misma vez, se abren una serie de pestañas adicionales en el cuadrante inferior derecho, tal y como se comentó en la Sección 2.

<sup>4</sup>Un punto de ruptura (abreviadamente PR) es una marca en una línea de código ejecutable de forma que su ejecución siempre se interrumpe antes de ejecutar esta línea, pasando el control al depurador

<sup>5</sup>Consultar <https://netbeans.org/kb/docs/cnd/debugging.html> para una descripción más detallada de las funciones de depuración de NetBeans, tanto para C++ como para otros lenguajes.

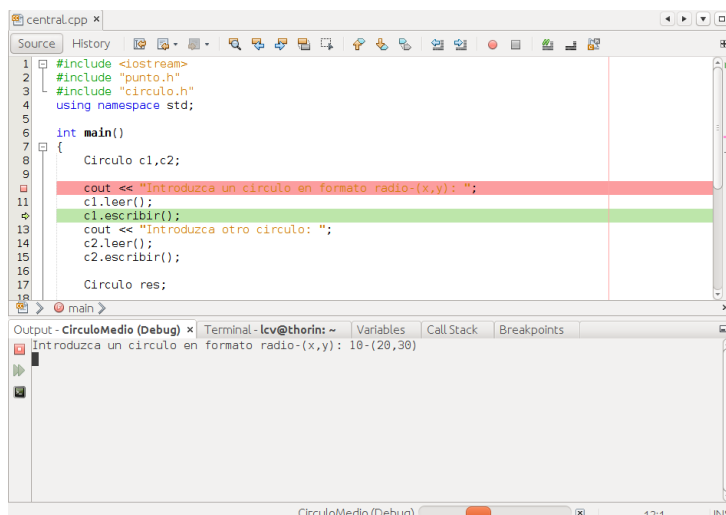


Figura 22: Ejecutando un programa paso a paso. La línea que se va a ejecutar a continuación aparece marcada en color verde

Las siguientes son algunas de las funciones más habituales de ejecución paso a paso:

- **Debug - Step Into**  
Ejecuta el programa paso a paso y entra dentro de las llamadas a funciones o métodos.
- **Debug - Step Over**  
Ejecuta el programa paso a paso sin entrar dentro de las llamadas a funciones o métodos, las cuales las resuelve en un único paso.
- **Debug - Continue**  
Ejecuta el programa hasta el siguiente punto de ruptura o el final del programa.

### 6.3. Inspección y modificación de datos

NetBeans, como cualquier depurador, permite inspeccionar los valores asociados a cualquier variable y modificar sus valores sin más que acceder a la pestaña **Variables** y desplegar las variables deseadas y modificarlas, en caso necesario (Figura 23).

### 6.4. Inspección de la pila

Durante el proceso de ejecución de un programa se suceden llamadas a módulos que se van almacenando en la pila. NetBeans ofrece la posibilidad de inspeccionar el estado de esta pila y analizar qué llamadas se están resolviendo en un momento dado de la ejecución de un programa (Figura 24).

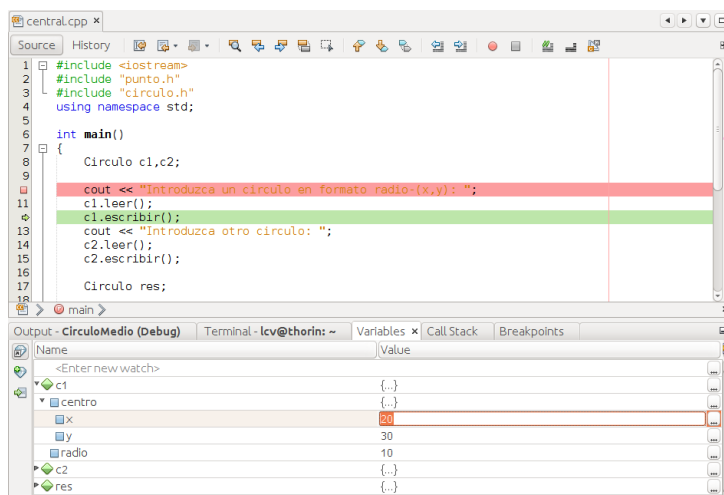


Figura 23: Inspeccionado y modificando, si fuese necesario, las variables del programa

## 6.5. Reparación del código

Durante una sesión de depuración es normal que sea necesario modificar el código para reparar algún error detectado. En este caso es necesario mantener bien actualizada la versión del programa que se encuentra cargada. Para ello lo mejor es interrumpir la ejecución del programa

### Debug - Finish Debugger Session

y re-escribir los cambios y recompilarlos (**Clean and Rebuild Project**).

## 7. Otras características de NetBeans

Además de estas características, el editor de código de NetBeans dispone de algunas ayudas a la escritura de código que incrementan enormemente la eficiencia y la detección temprana de errores de programación. Estas son algunas de estas características.

1. Ayuda a la escritura de llamadas a métodos y funciones (Figura 25). Mientras se escribe la llamada a un método se muestran las distintas posibilidades, sus parámetros y la información de ayuda.
2. Ayuda a la escritura de llamadas a métodos y funciones (Figura 26). Si se escribe una llamada a un método inexistente o con una llamada incorrecta, NetBeans lo señala inmediatamente con un icono rojo en la línea de la llamada.
3. Ayuda a la escritura de variables (Figura 27). Si se escribe una variable no declarada aún, NetBeans lo señala inmediatamente con un icono rojo en la línea de la llamada.
4. Ayuda a la indentación de código (Figura 28). Cada vez que se escribe código en una ventana, al pie de la misma aparece una indicación del nivel de anidamiento de código en el que se encuentra.

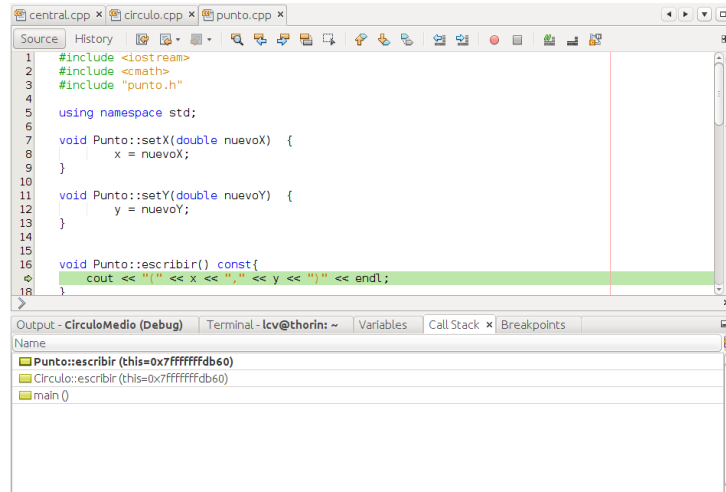


Figura 24: Inspeccionado y modificando, si fuese necesario, las pila de llamadas del programa

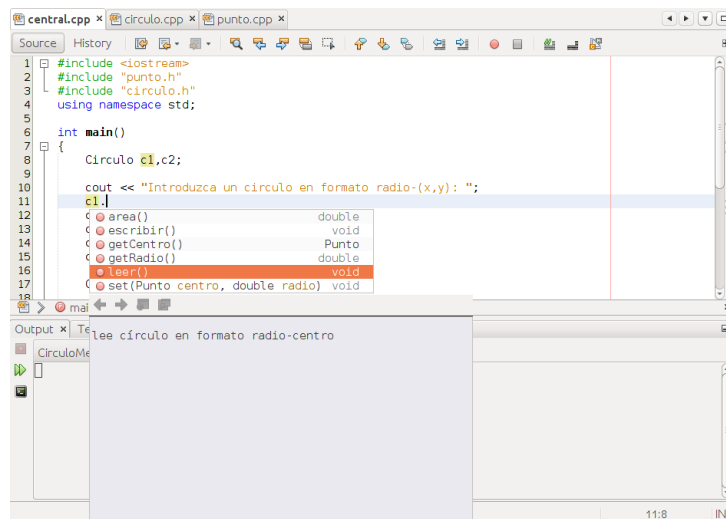
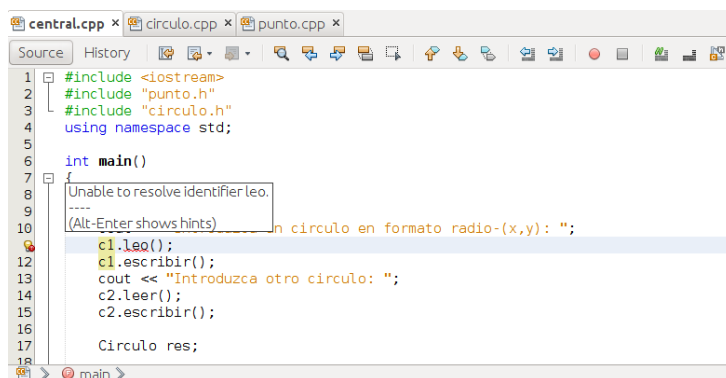


Figura 25: Auto-rellenado de código

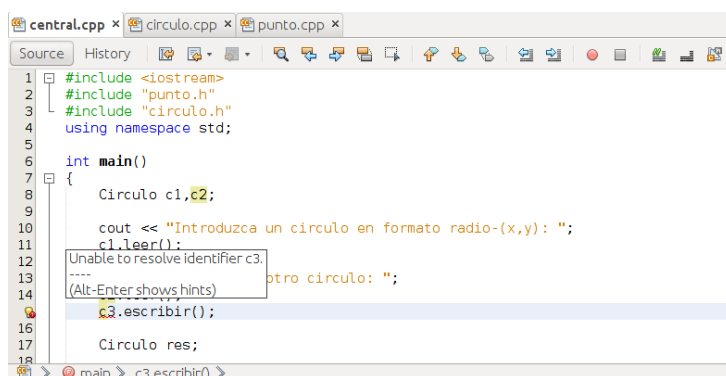


```

1 #include <iostream>
2 #include "punto.h"
3 #include "círculo.h"
4 using namespace std;
5
6 int main()
7 {
8     // ...
9     // ...
10    cout << "Introduzca un círculo en formato radio-(x,y): ";
11    // ...
12    leo();
13    c1.escribir();
14    cout << "Introduzca otro círculo: ";
15    c2.leer();
16    c2.escribir();
17    Círculo res;
18 }

```

Figura 26: Detección inmediata de llamadas erróneas



```

1 #include <iostream>
2 #include "punto.h"
3 #include "círculo.h"
4 using namespace std;
5
6 int main()
7 {
8     Círculo c1,c2;
9
10    cout << "Introduzca un círculo en formato radio-(x,y): ";
11    c1.leer();
12    // ...
13    // ...
14    c3.escribir();
15    // ...
16    Círculo res;
17 }

```

Figura 27: Detección inmediata del uso de identificadores erróneos

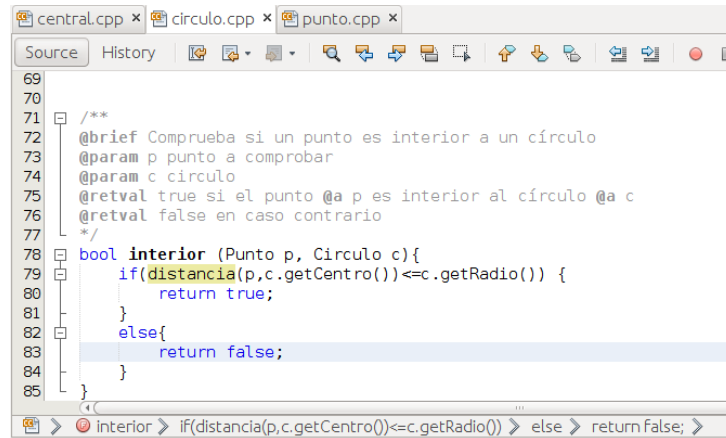


Figura 28: Detección inmediata del nivel de anidación del código. En la parte inferior del editor se puede observar la localización precisa de la línea que se está editando (línea número 83): Función **interior**, dentro de un **if** y en la parte **else** del mismo

## 8. Personalización de NetBeans

NetBeans puede ejecutar scripts del sistema operativo (Linux) para realizar tareas externas al propio entorno. Todas ellas se encuentran en la sección “Tools”-“Options”-“Miscellaneous”, como muestra la Figura 29. Estas scripts se pueden ejecutar sobre cualquier fichero del proyecto y, en particular, en esta sección se propone ejecutarla sobre el fichero **Makefile**, situado en la carpeta raíz del proyecto, con la opción (botón derecho) “Tools”-“Send to...” y el nombre de la script que se quiere ejecutar.

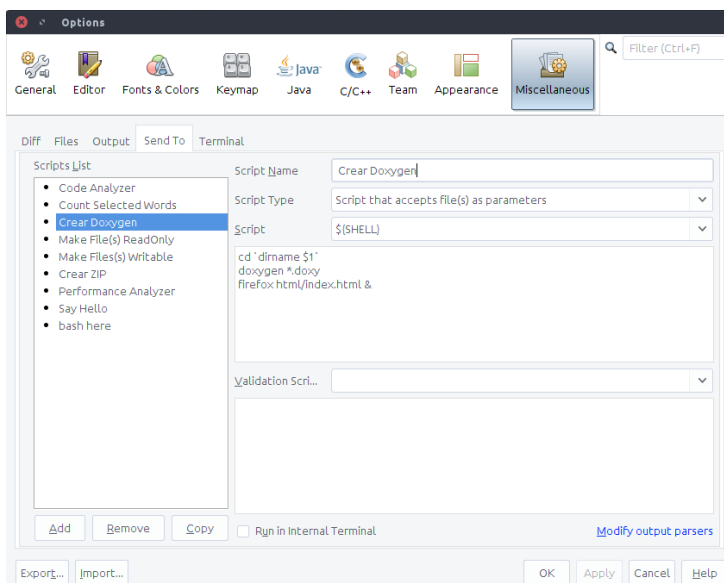


Figura 29: Personalización de NetBeans con scripts externas y la herramienta “Send To...”

### 8.1. Integración de doxygen

Crear la siguiente script para ejecutar doxygen en un proyecto sin estructura de directorios. En el caso de proyectos con una determinada estructura, modificar la script para tenerlo en cuenta de dónde obtener el fichero doxygen y donde dejar la salida.

Script name:

Crear Doxygen

Script type:

Script that accepts file(s) as parameters

Script

\$(SHELL)

```
cd `dirname $1`
doxygen *.doxy
firefox html/index.html &
```



## 8.2. Empaquetado de las prácticas para su entrega

Crear la siguiente script para comprimir y exportar un proyecto, listo para ser entregado en la asignatura.

Script name:

Crear ZIP

Script type:

Script that accepts file(s) as parameters

Script

\$(SHELL)

```
cd `dirname $1`
rm -rf zip/* dist/* build/* html latex
zip -r proyecto.zip * -x nbproject/private/*
```

En caso de que el proyecto esté estructurado en carpetas, los comandos a introducir en la script serían:

\$(SHELL)

```
cd `dirname $1`
rm -rf zip/* dist/* build/* doc/html doc/latex
zip -r zip/proyecto.zip * -x nbproject/private/*
```

## 9. Apéndice 1: circulomedio.cpp original

```
#include <iostream>
#include <cmath>
using namespace std;

class Punto {
private:
    double x;
    double y;
public:
    Punto() {x=0;y=0;}
    Punto(double x, double y){this->x=x;this->y=y;}
    double getX() const {return x;}
    double getY() const {return y;}
    void setX(double nuevoX);
    void setY(double nuevoY);
    void escribir() const;
    void leer();
};

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

void Punto::setY(double nuevoY) {
    y = nuevoY;
}

void Punto::escribir() const{
    cout << "(" << x << "," << y << ")" << endl;
}

void Punto::leer(){
    // Formato de lectura del punto: (x,y)
    char car;
    cin >> car >> x >> car >> y >> car;
}

double distancia(Punto p1, Punto p2){
    return sqrt((p1.getX()-p2.getX())*(p1.getX()-p2.getX()) +
        (p1.getY()-p2.getY())*(p1.getY()-p2.getY()));
}

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;
}

class Circulo {
private:
    Punto centro;
    double radio;
public:
    Circulo();
    Circulo(Punto centro, double radio);
    void set(Punto centro, double radio);
    Punto getCentro() const;
    double getRadio() const;
    void escribir() const;
    void leer();
    double area() const;
};

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

Circulo::Circulo(Punto centro, double radio) {
    this->centro = centro;
    this->radio = radio;
    // set(centro,radio);
}

void Circulo::set(Punto centro, double radio) {
    this->centro = centro;
    this->radio = radio;
}

Punto Circulo::getCentro() const {
    return centro;
}

double Circulo::getRadio() const {
    return radio;
}

void Circulo::escribir() const{
    cout << radio << "-";
    centro.escribir();
}
```

```

}

void Circulo::leer(){
    // Formato de lectura del c\{'i}rculo: radio-(x,y)
    char car;

    cin >> radio;
    cin >> car; // Leer -
    centro.leer();
}

double Circulo::area() const{
    return M.PI*radio*radio;
}

double distancia (Circulo c1, Circulo c2){
    Punto centro1, centro2;
    double dist;

    centro1=c1.getCentro();
    centro2=c2.getCentro();
    dist=distancia(centro1,centro2)-c1.getRadio()-c2.getRadio();
    return dist;
}

bool interior (Punto p, Circulo c){
    if(distancia(p,c.getCentro())<=c.getRadio()) {
        return true;
    }
    else{
        return false;
    }
}

int main()
{
    Circulo c1,c2;

    cout << "Introduzca un círculo en formato radio-(x,y): ";
    c1.leer();
    c1.escribir();
    cout << "Introduzca otro círculo: ";
    c2.leer();
    c2.escribir();

    Circulo res;

    res.set(puntoMedio(c1.getCentro(),c2.getCentro()),
            distancia(c1.getCentro(),c2.getCentro())/2);
    cout << "La distancia entre los c\{'i}rculos es: " << distancia(c1,c2) << " y el c\{'i}rculo que pasa_
por los dos centros es: ";
    res.escribir();
    cout << endl;
}

```

## 10. Apendice 2. Modularización

### 10.1. punto.h

```
#ifndef _PUNTO.H
#define _PUNTO.H

class Punto {
private:
    double x;
    double y;
public:
    Punto() {x=0;y=0;}
    Punto(double x, double y){this->x=x;this->y=y;}
    double getX() const {return x;}
    double getY() const {return y;}
    void setX(double nuevoX);
    void setY(double nuevoY);
    void escribir() const;
    void leer();
};

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

### 10.2. circulo.h

```
#ifndef _CIRCULO.H
#define _CIRCULO.H

#include "punto.h"

class Circulo {
private:
    Punto centro;
    double radio;
public:
    Circulo();
    Circulo(Punto centro, double radio);
    void set(Punto centro, double radio);
    Punto getCentro() const;
    double getRadio() const;
    void escribir() const;
    void leer();
    double area() const;
};

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

### 10.3. punto.cpp

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

using namespace std;

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

void Punto::setY(double nuevoY) {
    y = nuevoY;
}

void Punto::escribir() const{
    cout << "(" << x << "," << y << ")" << endl;
}

void Punto::leer(){
    // Formato de lectura del punto: (x,y)
    char car;
    cin >> car >> x >> car >> y >> car;
}

double distancia(Punto p1, Punto p2){
    return sqrt((p1.getX()-p2.getX())*(p1.getX()-p2.getX()) +
                (p1.getY()-p2.getY())*(p1.getY()-p2.getY()));
}

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;
}
```

## 10.4. 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) {
    this->centro = centro;
    this->radio = radio;
    // set(centro, radio);
}

void Circulo::set(Punto centro, double radio) {
    this->centro = centro;
    this->radio = radio;
}

Punto Circulo::getCentro() const {
    return centro;
}

double Circulo::getRadio() const {
    return radio;
}

void Circulo::escribir() const {
    cout << radio << "-";
    centro.escribir();
}

void Circulo::leer() {
    // Formato de lectura del c\{'\i}rculo: radio-(x,y)
    char car;

    cin >> radio;
    cin >> car; // Leer -
    centro.leer();
}

double Circulo::area() const {
    return M_PI*radio*radio;
}

double distancia (Circulo c1, Circulo c2){
    Punto centro1, centro2;
    double dist;

    centro1=c1.getCentro();
    centro2=c2.getCentro();
    dist=distancia(centro1,centro2)-c1.getRadio()-c2.getRadio();
    return dist;
}

bool interior (Punto p, Circulo c){
    if(distancia(p,c.getCentro())<=c.getRadio()) {
        return true;
    }
    else{
        return false;
    }
}
}
```

## 10.5. central.cpp

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

int main()
{
    Circulo c1,c2;

    cout << "Introduzca un circulo en formato radio-(x,y): ";
```

```

c1.leer();
c1.escribir();
cout << "Introduzca otro círculo: ";
c2.leer();
c2.escribir();

Circulo res;

res.set(puntoMedio(c1.getCentro(), c2.getCentro()),
        distancia(c1.getCentro(), c2.getCentro())/2);
cout << "La distancia entre los círculos es: " << distancia(c1, c2) << " y el círculo que pasa
por los dos centros es: ";
res.escribir();
cout << endl;
return 0;
}

```