Guion de prácticas

Entorno de desarrollo de software con NetBeans

Febrero 2020

Metodología de la Programación

GII Grupos B y C
Curso 2019/2020

Contents

1 Introducción

Esta sesión de prácticas está dedicada a aprender a utilizar el entorno de desarrollo de software multi-lenguaje **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 [1] y el plugin para **C++** [2] instalados en ese orden. También se puede instalar el plugin de **C++** desde el menu "Tools-Plugins" del menu de NetBeans.

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

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

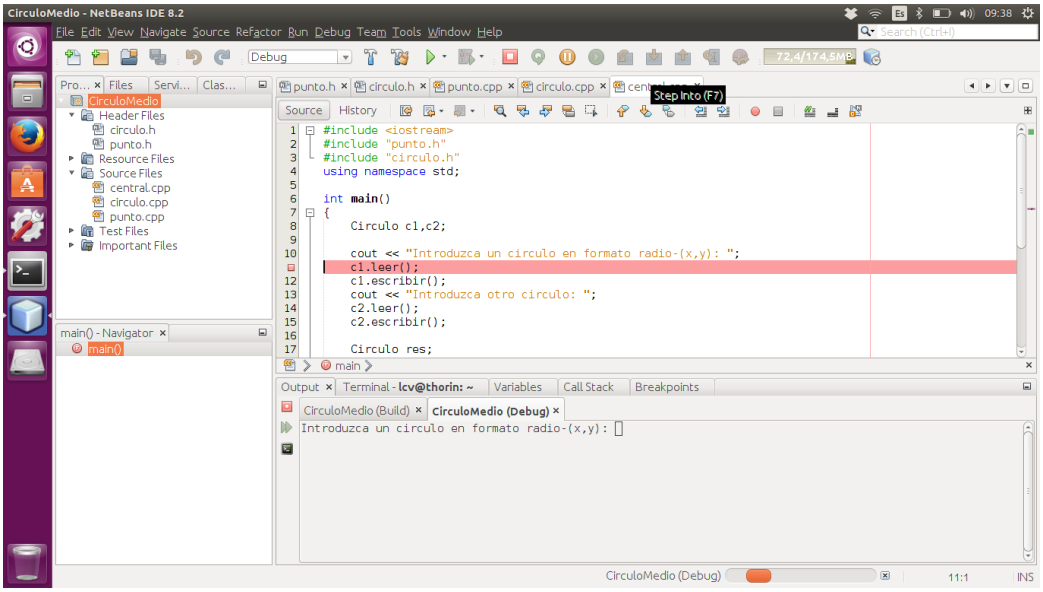


Figure 1: Entorno de trabajo en NetBeans con C++

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

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

[2] https://netbeans.org/features/cpp/

a lo largo de este guión. En cualquier caso, para más información se puede consultar la guía oficial de NetBeans[3].

- 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 1).
- 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. Muesta 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 llamakefilemadas (depuración).
  - Breakpoints. Lista de puntos de ruptura activos (depuración).

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 2), entre los múltiples lenguajes soportados por NetBeans, en este caso ”C/C++ Application”. Seleccionar el nombre del proyecto (”NuevoProyecto”) y la ubicación que se le quiere dar en disco ”Project location / Browse”.

3.2 Creación de un fichero fuente

Vamos a crear el primer fichero fuente del proyecto. Para ello, en el recién creado fichero **main.cpp** se introduce un primer programa en C++, como el conocido “Hola Mundo”[4] (Figura 3).

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

[3] https://netbeans.org/kb/docs/java/quickstart.html

[4] El programa “Hola mundo” en más de 200 lenguajes de programación http://helloworldcollection.de

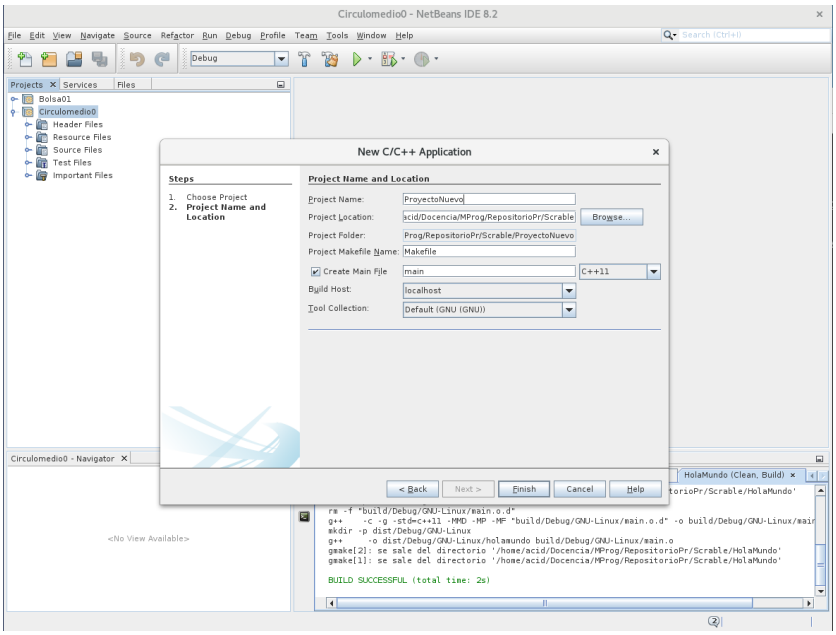


Figure 2: Creando un proyecto nuevo, ProyectoNuevo

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 5).

El binario recien generado se encuentra en la carpeta **dist** tal y como muestra el cuadrante superior izquierdo de la Figura 5.

La ejecución del binario que se acaba de generar se ordena como

Run - Run Project

Y se puede seguir su ejecución, desde la pestaña **Output**, cuadrante inferior, como si fuese una consola de órdenes del sistema operativo.

3.4 El programa en un único fichero. circulomedio.cpp

Recuperamos el fichero **circulomedio.cpp** (contenido en **circulomedio.zip**) para su utilización. Crea un nuevo proyecto llamado **Circulomedio0** y copia este código en el fichero **main.cpp**. Desde la vista de ficheros, crea una carpeta llamada **src** y mueve el fichero dentro de esta carpeta. Copia el fichero **circulomedio.dat** en la carpeta del proyecto, bien mediante interfaz gráfico o por comandos, observa que todos los ficheros se reflejan en el árbol físico. Ejecute el programa desde la consola con redireccionamiento de la entrada con <, pero para ello, hemos de localizar el ejecutable en el directorio **dist...**, ver figura 6.

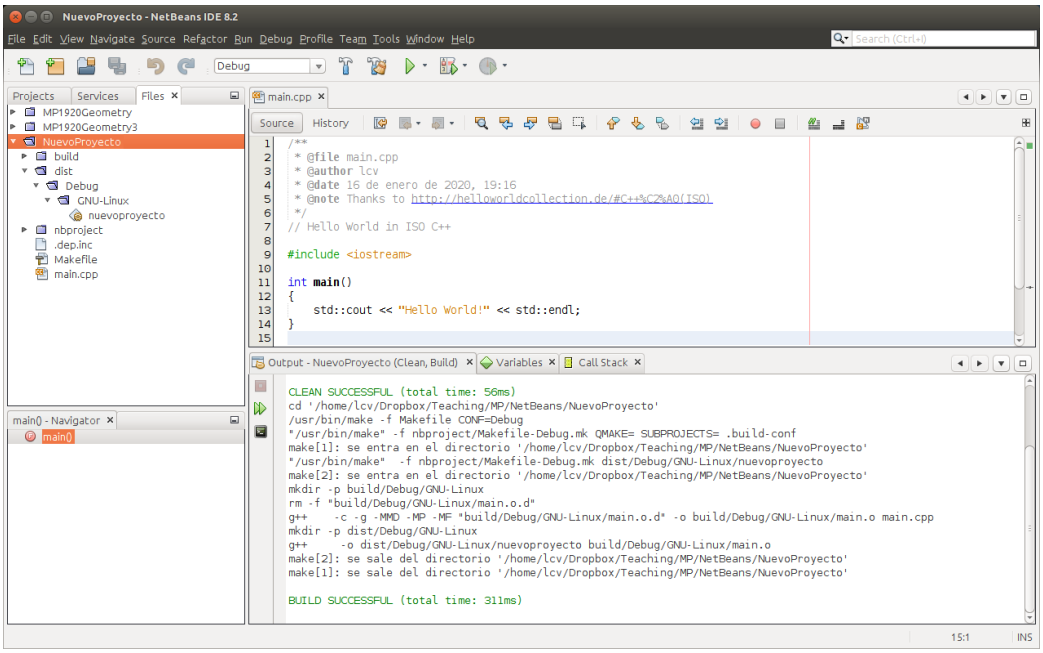


Figure 3: Creación de un nuevo fichero fuente en el proyecto

4 Un proyecto de compilación separada

Esta sección describe cómo crear un proyecto más complejo y dividirlo en múltiples ficheros.

4.1 Creación de un nuevo proyecto. CirculoMedio

Vamos a crear un nuevo proyecto denominado (”CirculoMedio”) en esta ocasión hay que asegurarse de que la opción ”Create Main File” está **desactivada**.

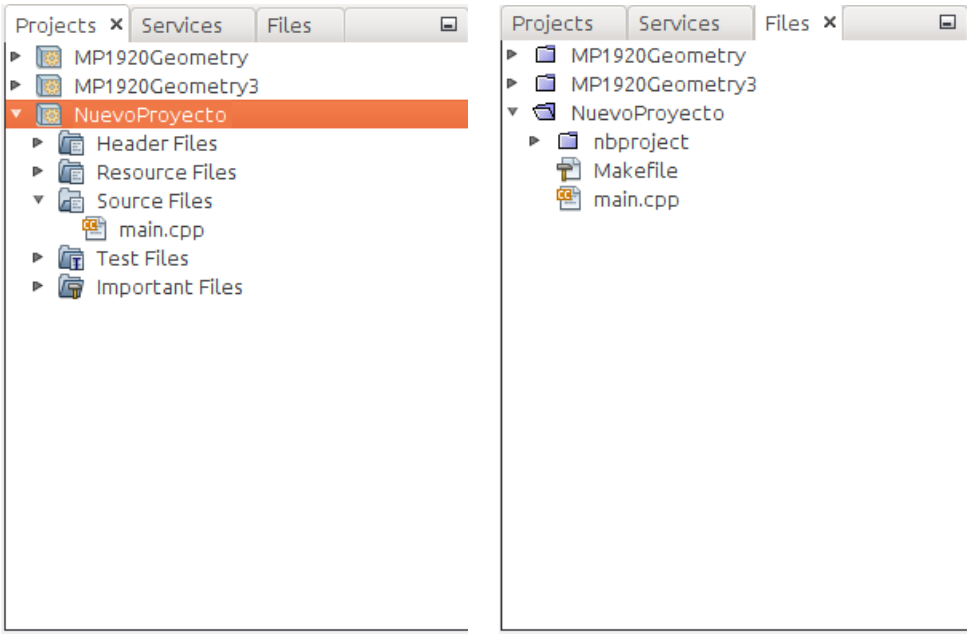
4.2 Configurar la estructura de carpetas del proyecto, la vista física

Sobre el árbol físico del proyecto (pestaña **Files**), crear la típica estructura de carpetas comentada en la Práctica 1, `doc,data,include,zip ...`. Pero en este caso hay varias **diferencias**:

1. La carpeta **./bin** en la que se colocará el binario queda sustituida por la carpeta **dist** que se creará la primera vez que se compile un proyecto

2. La carpeta **./obj** no es necesario indicarla pues es gestionada automáticamente por NetBeans.

3. La carpeta **./lib** y el uso de bibliotecas recibirá una especial atención en la Sección 5.



(a) (b)

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

Luego, vamos a crear los directorios **src, include, data, zip, doc**, y dado que tenemos los ficheros elaborados anteriormente, se pueden copiar los ficheros que contiene el directorio **solucionVersion2** (en circulomedio.zip ya utilizado en la práctica anterior) desde un explorador de archivos hasta la vista física del proyecto. En este caso, y siguiendo los mismos pasos que en la Práctica 1, se incluirá también el fichero de validación de datos en la carpeta **data** y el fichero doxy en la carpeta **doc**.

4.3 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, *para montar el makefile propio*.

4.4 Configurar los parámetros del proyecto

Ya casi está terminado. Las opciones del proyecto se definen desde la pestaña de propiedades del proyecto (Figura 10), la cual aparece con

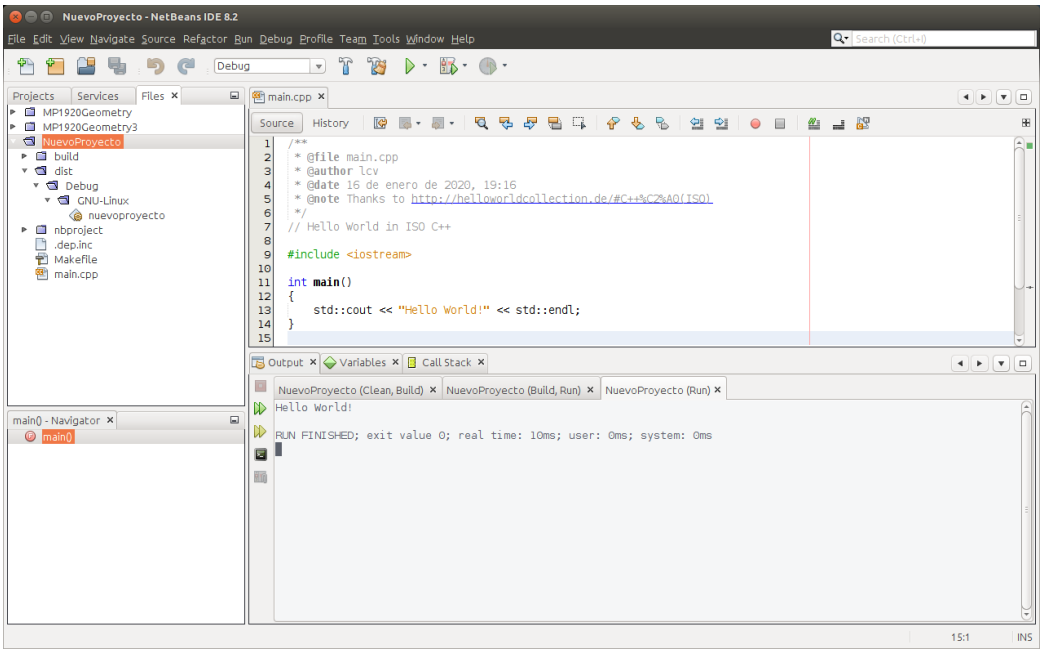


Figure 5: Ejecutando el binario de ProyectoNuevo

botón derecho, “Project properties”, “C++ compiler”, “Include directories” y se selecciona la carpeta “include” perteneciente al 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 estándard de C++ se va a compilar el proyecto. Seleccionaremos siempre “C++11”.

Para introducir parámetros en la llamada al programa (redireccionamientos de la entrada o salida del programa para nuestros ficheros de validación) es necesario modificar las propiedades del proyecto; concretamente opciones de Run hay que editar el valor actual para que quede con los valores de `"${OUTPUT_PATH}" < data/circulomedio.dat`. Cada vez que se ejecuta el programa utilizará como entrada el fichero indicado, ver figura 13.

4.5 Personalizando el proyecto para el makefile de Netbeans

Algunas opciones del proyecto que se hacían directamente en el Makefile en la Práctica 1, se llevan a cabo desde la pestaña de propiedades del proyecto (Figura 10).

Project - Projects Properties

1. Describir la carpeta propia de ficheros de cabecera
   C++ Compiler - Include Directories. Esta carpeta será la que se pase a **g++** con la opción **-I**

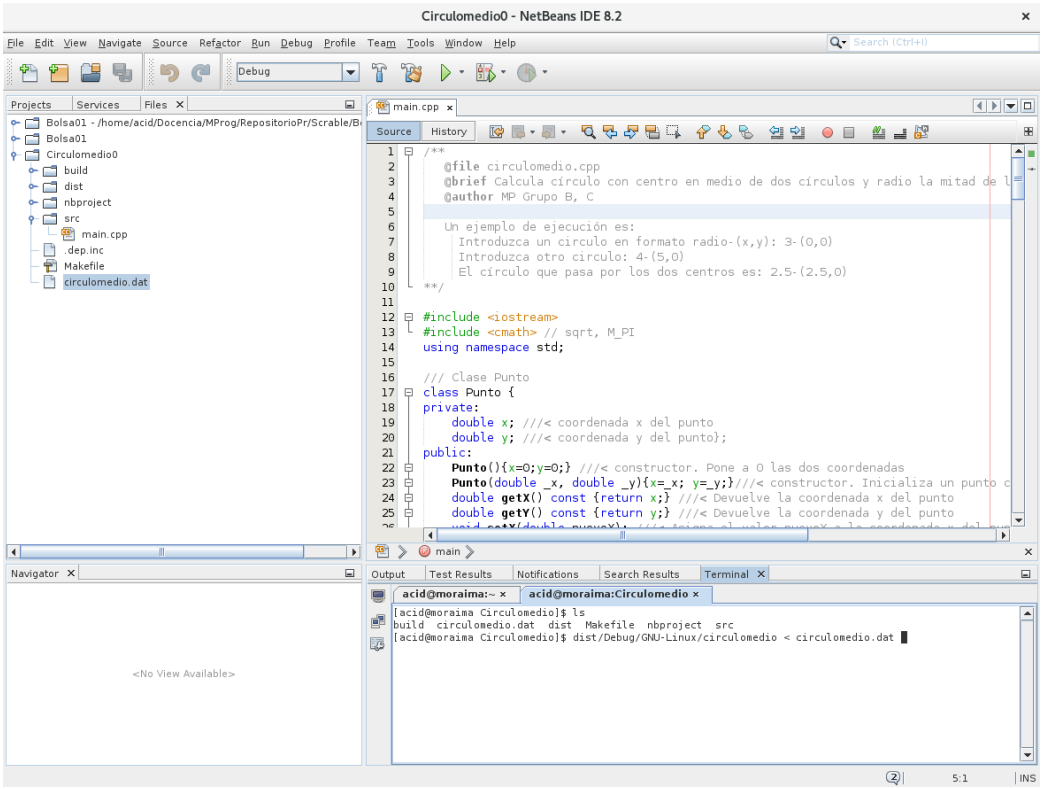


Figure 6: Ejecutando el binario, Circulomedio0

2. Modo de depuración o de producción. Para recompilar todo el proyecto con la opción **-g** o no, esto se elige en el campo **Configuration** y se elige **Debug** o **Release** respectivamente.

3. Opciones del compilador `std=c++11`, para nuestro proyecto mediante **C++ Standard** seleccionar `C++11`. Se pueden fijar niveles superiores de warnings en **warnings level**.

4.6 Generando y ejecutando el binario

A partir de esta información NetBeans permite generar un fichero makefile de forma automática (colocado en la carpeta raíz del proyecto y que también se puede editar y ejecutar desde la línea de comandos tradicional) y generar los binarios con la opción

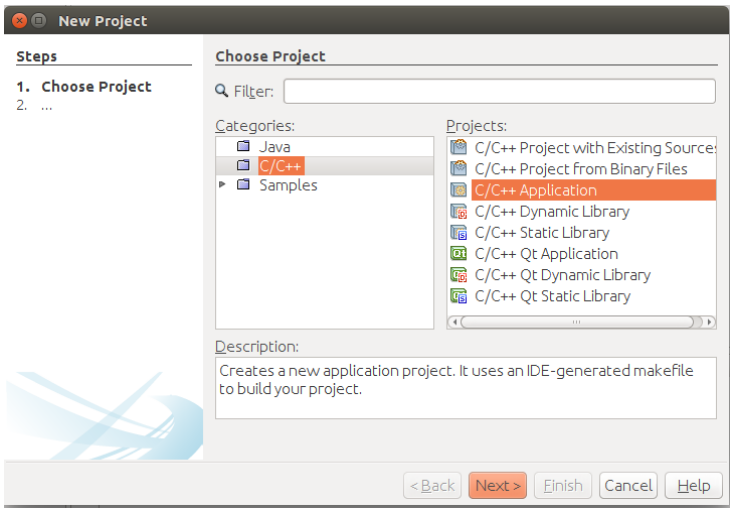
Run - Clean and Build Project

Cuando se selecciona se puede ver que esta llamada ejecuta **make** contra el makefile generado automáticamente por NetBeans y muestra la salida de **g++** habitual pero con algunos parámetros adicionales. El binario recien generado se encuentra en la carpeta **dist** como ya se indicó.

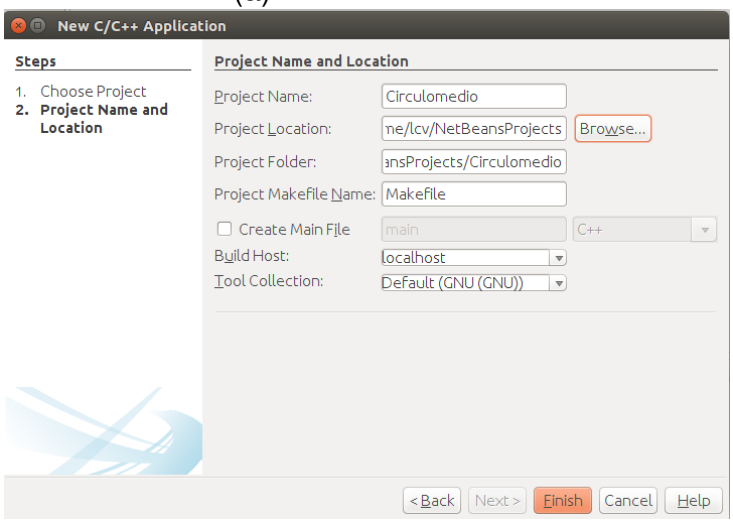
La ejecución del binario que se acaba de generar se ordena como

Run - Run Project

Y se puede seguir su ejecución, introduciendo los valores correspondientes desde la pestaña **Output** (Figura 18) como si fuese una consola de órdenes del sistema operativo.



(a)



(b)

Figure 7: Creando un proyecto nuevo. CirculoMedio

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 13 para indicar que el binario se va a ejecutar redireccionando la entrada estándar desde **data/circulomedio.dat**.

4.7 Nuevos objetivos del Makefile de NetBeans

NetBeans genera automáticamente un fichero makefile, el cual es perfectamente editable desde el entorno y al cual podemos añadirle reglas personalizadas como las reglas **zip** o **doxy** que se muestran en la Figura 14. Este makefile del NetBeans es totalmente estándar y se puede ejecutar desde la línea de comandos, desde la carpeta raíz del proyecto de NetBeans, y ordenando sólamente **make**. Desde la línea de comandos,

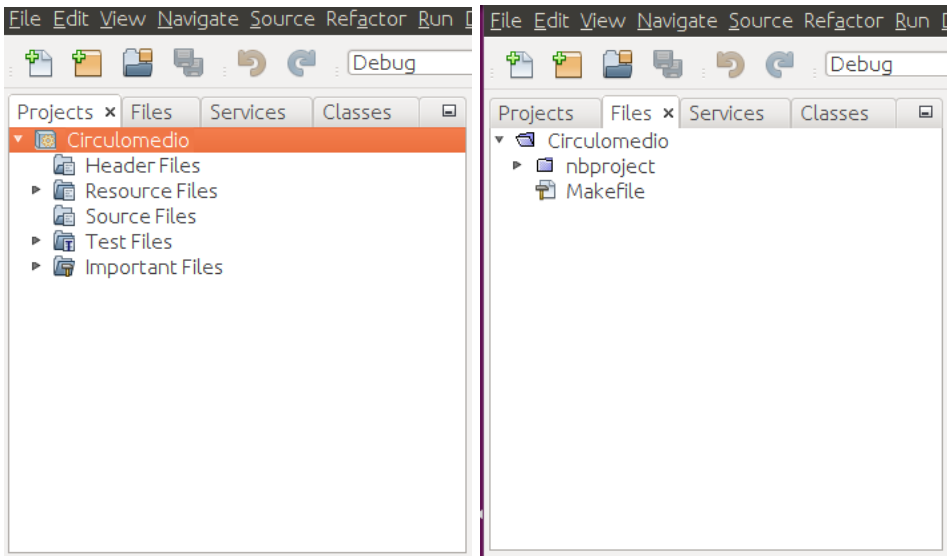


Figure 8: Árbol lógico (izquierda) y físico (derecha) de CirculoMedio

por tanto, también se pueden invocar estos objetivos personalizados del makefile sin más que ejecutar **make zip** o **make doxy**.

Alternativamente se puede invocar cualquier objetivo del makefile desde NetBeans, vista física, pulsar sobre el fichero makefile con el botón derecho y seleccionar el objetivo que se desea invocar (ver Figura 15). NetBeans ejecutará el makefile sobre ese objetivo concreto.

Con el objetivo **zip** además de limpiar varios binarios etc., se obtiene el paquete como se muestra en `practica0.zip` 15. El fichero **practica0.zip** debe ser tal que se pueda descomprimir y compilar tanto desde la línea de comandos como desde dentro de NetBeans, tal y como se ha explicado en este guión, esta es la secuencia de pasos, ver Figura 16.

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 19). 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

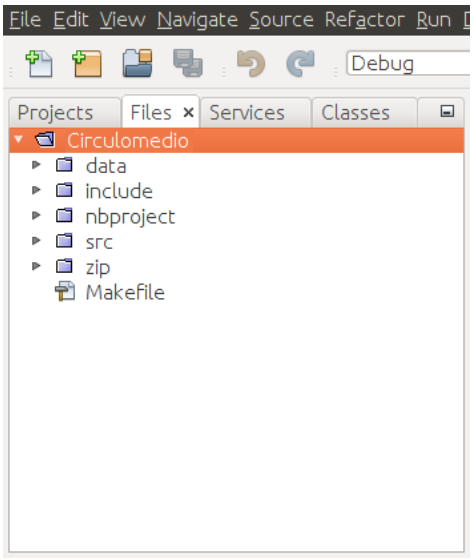


Figure 9: Estructura de carpetas del proyecto, CirculoMedio

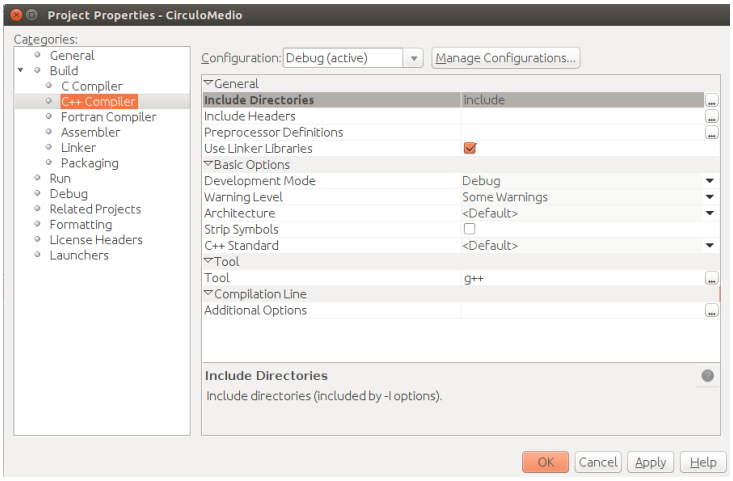


Figure 10: Propiedades del proyecto

desde la carpeta desde donde se encuentran actualmente, bien 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 20 muestra las vistas lógica (izquierda) y física (derecha) 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 10) en el parámetro **Include Directories**.

4. Construir el proyecto de biblioteca (**Clean and Build Project**) igual

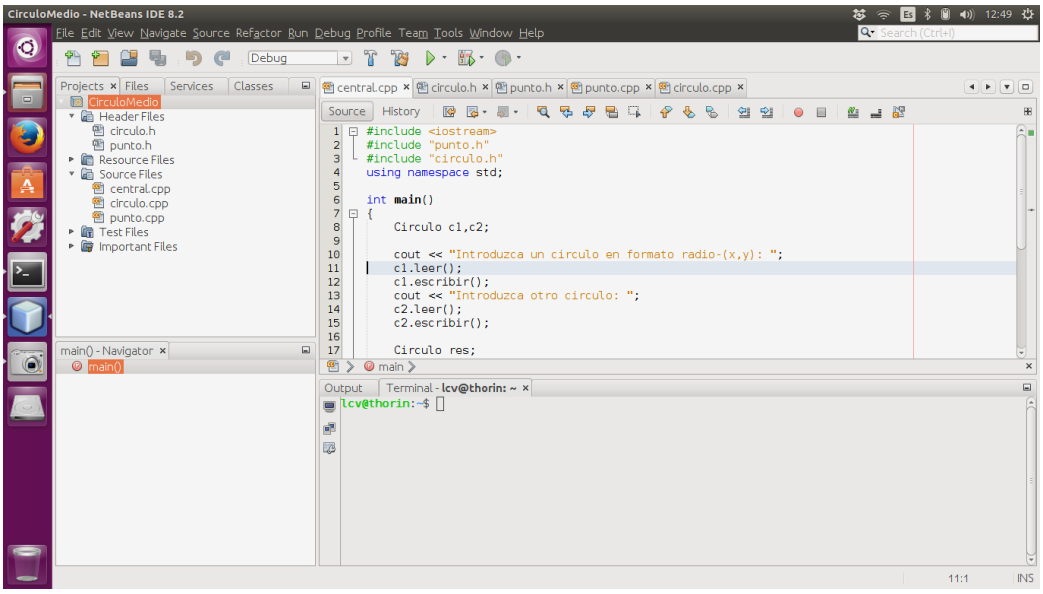


Figure 11: Desde el árbol lógico, crear cada fichero del proyecto (cpp, h) y colocarlo en la carpeta correspondiente (src, include).

que se hizo en el proyecto anteior, con la diferencia de que, en este caso, no se ha generado un binario, sino una biblioteca. En la Figura 21 se puede ver cómo NetBeans llama adecuadamente al compilador para compilar las fuentes y, posteriormente, al programa **ar** para generar la biblioteca. La biblioteca recien generada se encuentra en la carpeta **dist** tal y como muestra la Figura 22.

5. 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 23).

5.2 Usar la nueva biblioteca en el proyecto anterior

En este caso, se va a mostrar cómo introducir la biblioteca recien 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 24.

2. En las propiedades del proyecto (Figura 25 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 25 abajo), añadir la biblioteca recién creada en la

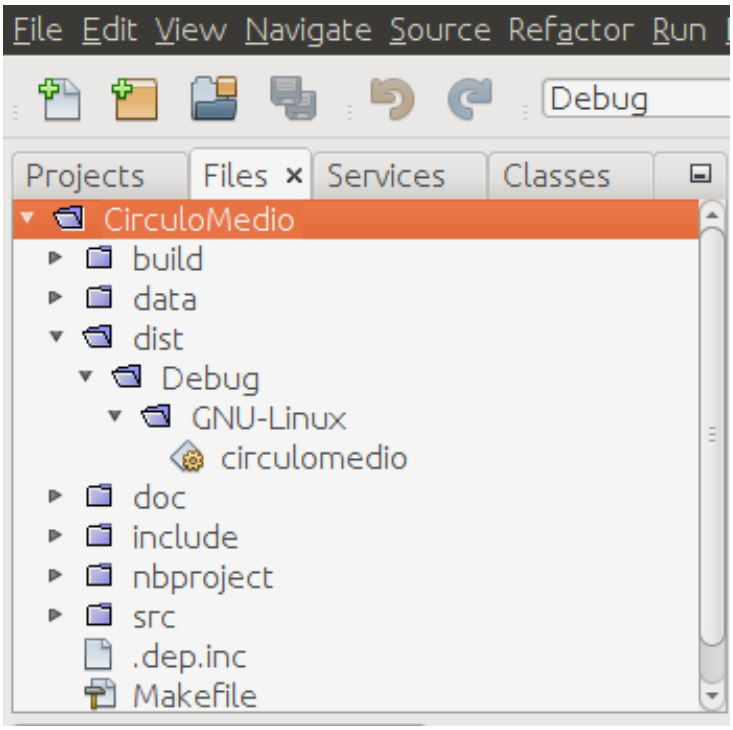


Figure 12: Situación del binario en el árbol físico del proyecto, dentro de la carpeta **dist**

opción **Libraries**.

4. Crear el nuevo binario (**Clean and Build Project**) y observar (Figura 26) cómo se enlaza con la nueva biblioteca.

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[5]. NetBeans permite crear un PR simplemente haciendo click sobre el número de línea correspondiente en

[5]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

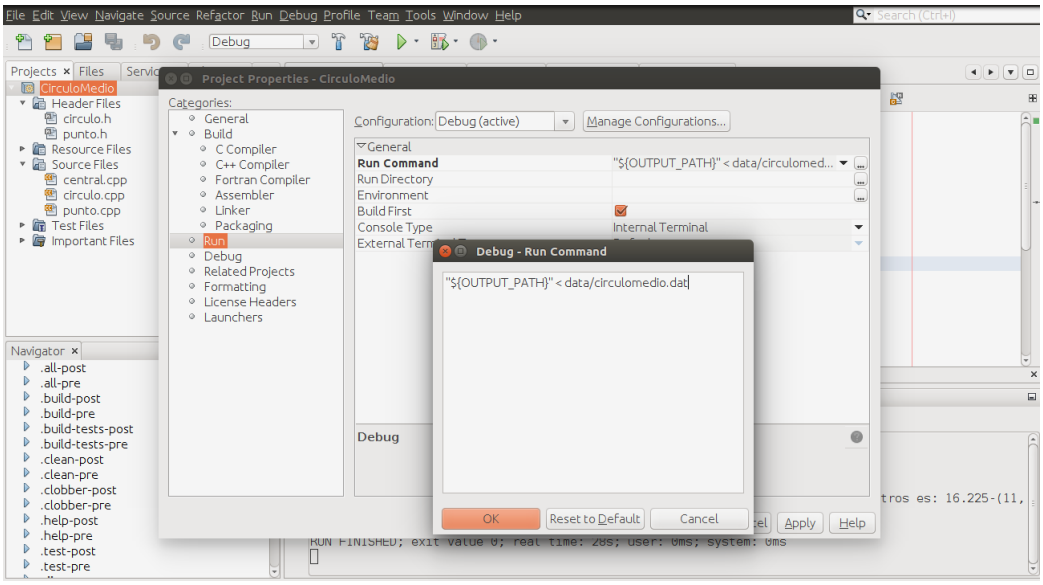


Figure 13: Ejecutando el binario con argumentos o redireccionamientos

la ventana de visualización de código y visualiza esta marca como una línea en rojo (Figura 27). Una vez colocado este punto de ruptura se puede comenzar la ejecución del programa con el menú[6]

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 28). 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.

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 ac-

[6]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.

```
# help
help: .help-post

.help-pre:
# Add your pre 'help' code here...

.help-post: .help-impl
# Add your post 'help' code here...

doxy:
        doxygen doc/circulomedio.doxy
        firefox doc/html/index.html

zip: clobber
        rm -rf zip/*
        rm -rf doc/html doc/latex
        zip -r zip/practica0.zip * -x nbproject/private/*

# include project implementation makefile
include nbproject/Makefile-impl.mk

# include project make variables
include nbproject/Makefile-variables.mk
```

Figure 14: Se le pueden añadir reglas personalizadas al Makefile automático que genera NetBeans, a parte de disponer de otras reglas conocidas como **clean**.

ceder a la pestaña **Variables** y desplegar las variables deseadas y modificarlas, en caso necesario (Figura 29).

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 momendo dado de la ejecución de un programa (Figura 30).

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**).

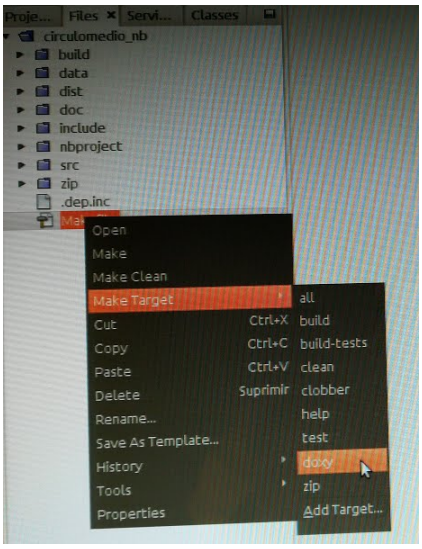


Figure 15: Ejecutando el makefile de NetBeans con objetivos personalizados

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 31). 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 32). 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 33). 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 34). 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.

8 Ampliando el intervalo, en la sesión de prácticas

Se pide continuar la práctica de intervalos en netbeans, crear un proyecto nuevo Intervalo con el código anterior se va a pedir que incluya una nueva función externa llamada `interseccion` que haga lo propio con dos intervalos.

```
$$ unzip practica0.zip
Archive:  practica0.zip
   creating: build/
   creating: data/
  inflating: data/circulomedio.dat
   creating: dist/
   creating: doc/
  inflating: doc/circulomedio.doxy
   creating: include/
  inflating: include/punto.h
  inflating: include/utilidades.h
  inflating: include/circulo.h
  inflating: Makefile
   creating: nbproject/
  inflating: nbproject/project.xml
   creating: nbproject/private/
  inflating: nbproject/Package-Debug.bash
  inflating: nbproject/Makefile-Debug.mk
  inflating: nbproject/Makefile-variables.mk
  inflating: nbproject/Package-Release.bash
  inflating: nbproject/Makefile-Release.mk
  inflating: nbproject/configurations.xml
  inflating: nbproject/Makefile-impl.mk
   creating: src/
  inflating: src/utilidades.cpp
  inflating: src/circulo.cpp
  inflating: src/central.cpp
  inflating: src/punto.cpp
   creating: zip/
$$ make
make[1]: se entra en el directorio '/home/acid/Docencia/MProg/circulomedio/Alumno/
"make"  -f nbproject/Makefile-Release.mk dist/Release/GNU-Linux/p3
make[2]: se entra en el directorio '/home/acid/Docencia/MProg/circulomedio/Alumno/
mkdir -p build/Release/GNU-Linux/src
rm -f "build/Release/GNU-Linux/src/central.o.d"
g++    -c -O2 -Wall -Iinclude -std=c++11 -MMD -MP -MF "build/Release/GNU-Linux/src
mkdir -p build/Release/GNU-Linux/src
rm -f "build/Release/GNU-Linux/src/circulo.o.d"
g++    -c -O2 -Wall -Iinclude -std=c++11 -MMD -MP -MF "build/Release/GNU-Linux/src
mkdir -p build/Release/GNU-Linux/src
rm -f "build/Release/GNU-Linux/src/punto.o.d"
g++    -c -O2 -Wall -Iinclude -std=c++11 -MMD -MP -MF "build/Release/GNU-Linux/src
mkdir -p build/Release/GNU-Linux/src
rm -f "build/Release/GNU-Linux/src/utilidades.o.d"
g++    -c -O2 -Wall -Iinclude -std=c++11 -MMD -MP -MF "build/Release/GNU-Linux/src
mkdir -p dist/Release/GNU-Linux
g++     -o dist/Release/GNU-Linux/p3 build/Release/GNU-Linux/src/central.o build/R
make[2]: se sale del directorio '/home/acid/Docencia/MProg/circulomedio/Alumno/Ent
make[1]: se sale del directorio '/home/acid/Docencia/MProg/circulomedio/Alumno/Ent

$$ ./dist/Release/GNU-Linux/p3 < data/circulomedio.dat
Introduzca un circulo en formato radio-(x,y): 1-(0,0)
Introduzca otro circulo: 2-(3,4)
La distancia entre los círculos es: 2 y el círculo que pasa por los dos centros es
```

Figure 16: Una vez terminado, comprobación de su funcionamiento. Secuencia completa.

La intersección $\cap$ de dos intervalos resulta un intervalo. $i1 \cap i2 = ir$, dónde ir es un intervalo simple de los definidos en la clase intervalo, incluido el vacío.
Para definir el intervalo resultado se comparan las cotas inferiores de $i1$ e $i2$, y se obtiene como cota inferior la mayor de las dos. La cota superior del resultado será la menor de las cotas superiores de $i1$ e $i2$. Por último, en resultado se establece la propiedad de pertenencia de cada una de las cotas asignadas anteriormente en el resultado.
Finalmente, el resultado puede ser un intervalo imposible (las cotas se cruzan, ...), en ese caso se devuelve un intervalo vacío. [7]
Nota: es necesario **revisar** la declaración y **definir** la función **interseccion** además, de definir el método **set** y un **constructor** con parámetros de la clase intervalo.

Se proporciona un main que declara un vector de intervalos $interv$. El programa lee una serie de intervalos y muestra el resultado de realizar la intersección de éstos y muestra el intervalo resultante.

8.1 Casos de prueba

Algunos de los casos de prueba a considerar y sus respectivas salidas son:

Caso de prueba 1

```
2
[0,10 ]
(0,10 )
```

./data/datos1.dat

```
(0,10)
```

./data/salida1.txt

Caso de prueba 2

```
2
[0,10 ]
[1,1 ]
```

./data/datos2.dat

```
[1,1]
```

./data/salida2.txt

Caso de prueba 3

```
2 [9,13 ] (0,10)
```

./data/datos3.dat

```
[9,10)
```

./data/salida3.txt

[7] Caso de que no tengan ningun punto común, el intervalo es el vacío..

Caso de prueba 4

```
2  [0,5] [6,10]
```

./data/datos4.dat

```
(0)
```

./data/salida4.txt

Para facilitar la tarea se le facilita parte del código que debe de incluir dentro de la función interseccion

```
/**
 * @brief realiza la interseccion de dos intervalos, puede resultar un intervalo vacío en caso de que
 * no tengan puntos comunes, en caso contrario se revisan las cotas.
 * @param i1 primer intervalo de entrada
 * @param i2 segundo intervalo de entrada
 * @return devuelve el intervalo resultante de realizar la interseccion entre los dos intervalos de entrada
 */

interseccion( ){
   int cotaInf, cotaSup;        // variables locales
   bool cerradoInf, cerradoSup;// variables locales

   if (i1.getCotaInf() < i2.getCotaInf()){  // para la cota inferior
       cotaInf = i2.getCotaInf();
       cerradoInf = i2.dentroCotaInf();
   }
   else if (i1.getCotaInf() > i2.getCotaInf()){
       cotaInf = i1.getCotaInf();
       cerradoInf = i1.dentroCotaInf();
   }

   ...
```

9 Pantallas de referencia

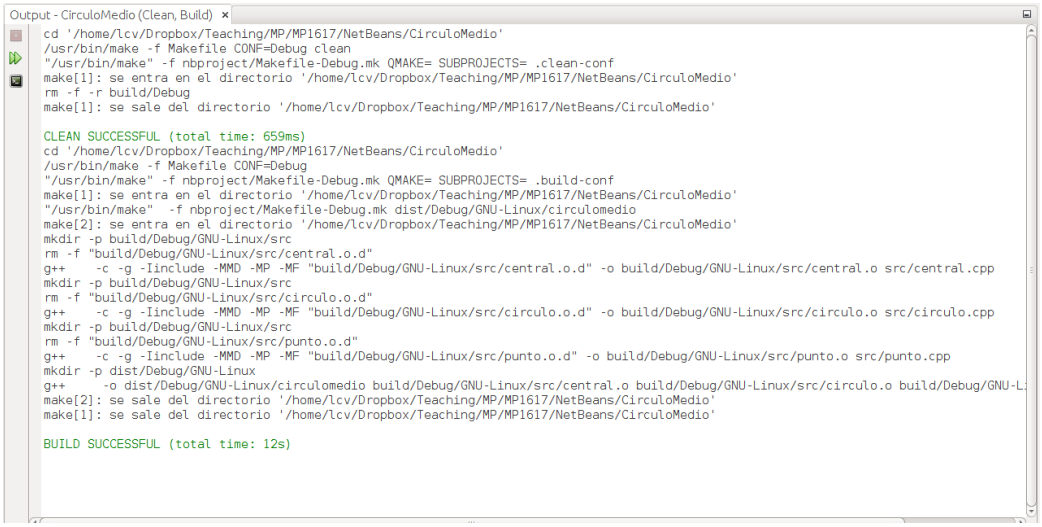


Figure 17: Generando el binario

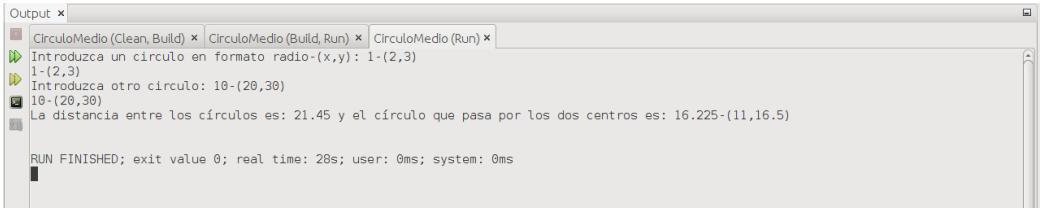


Figure 18: Ejecutando el binario

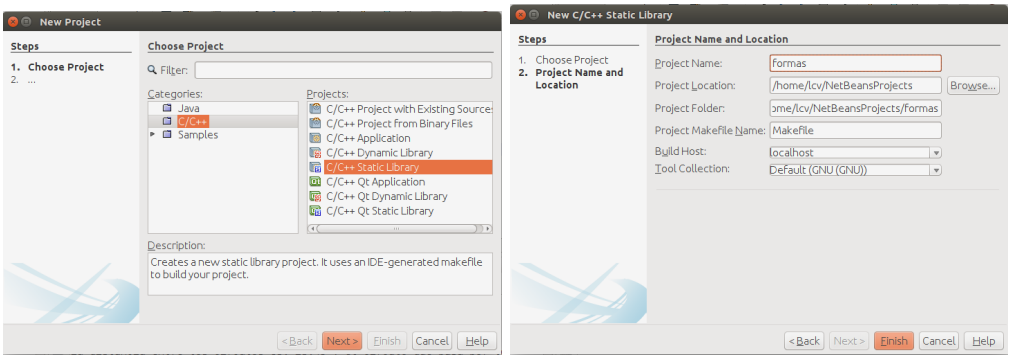


Figure 19: Creando un nuevo proyecto para gestionar una biblioteca

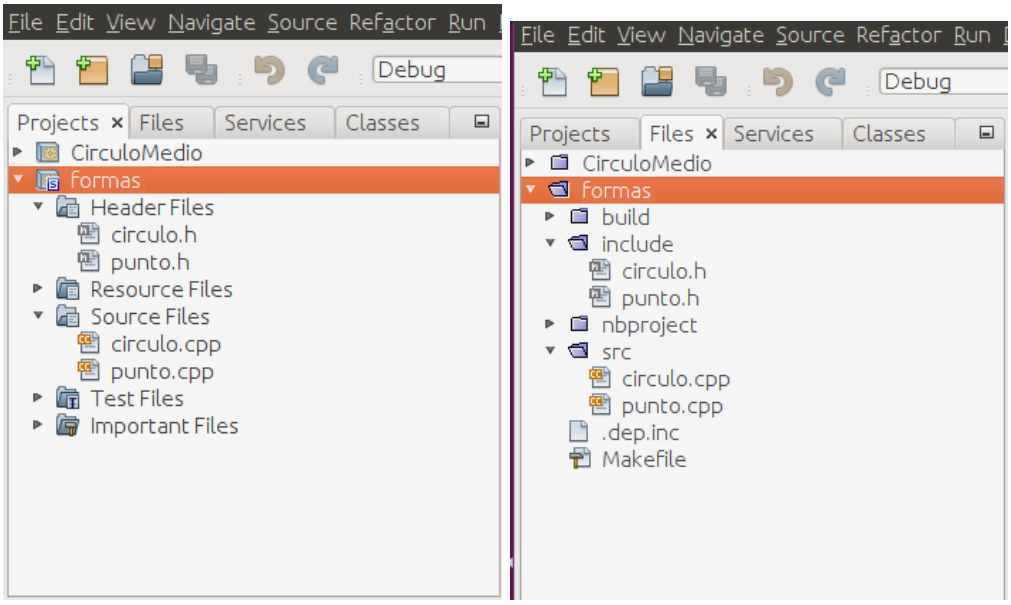


Figure 20: Árboles lógico (izqda) y físico (dcha) del nuevo proyecto de biblioteca

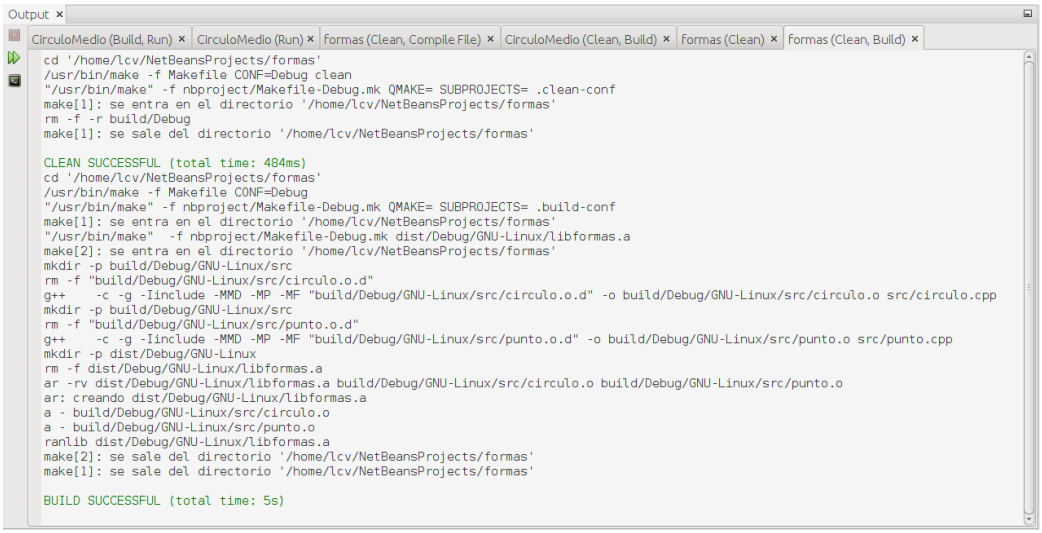


Figure 21: Creación de la biblioteca de forma automática desde NetBeans

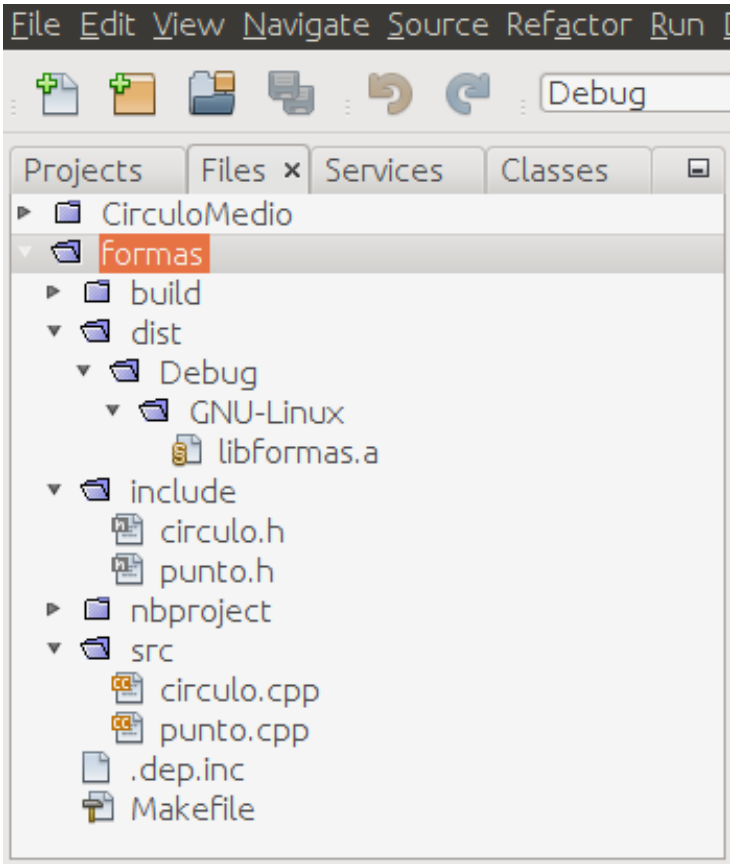


Figure 22: Situación de la biblioteca en el árbol físico del proyecto dentro de la carpeta **dist**

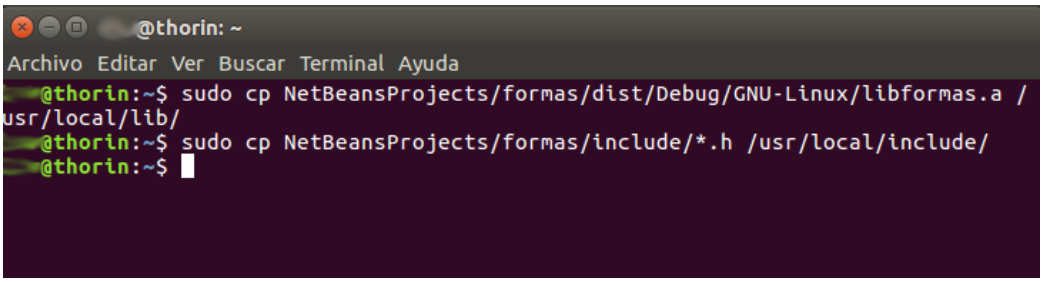


Figure 23: Colocar los .h y los .a en las carpetas del sistema para su uso posterior en otros proyectos

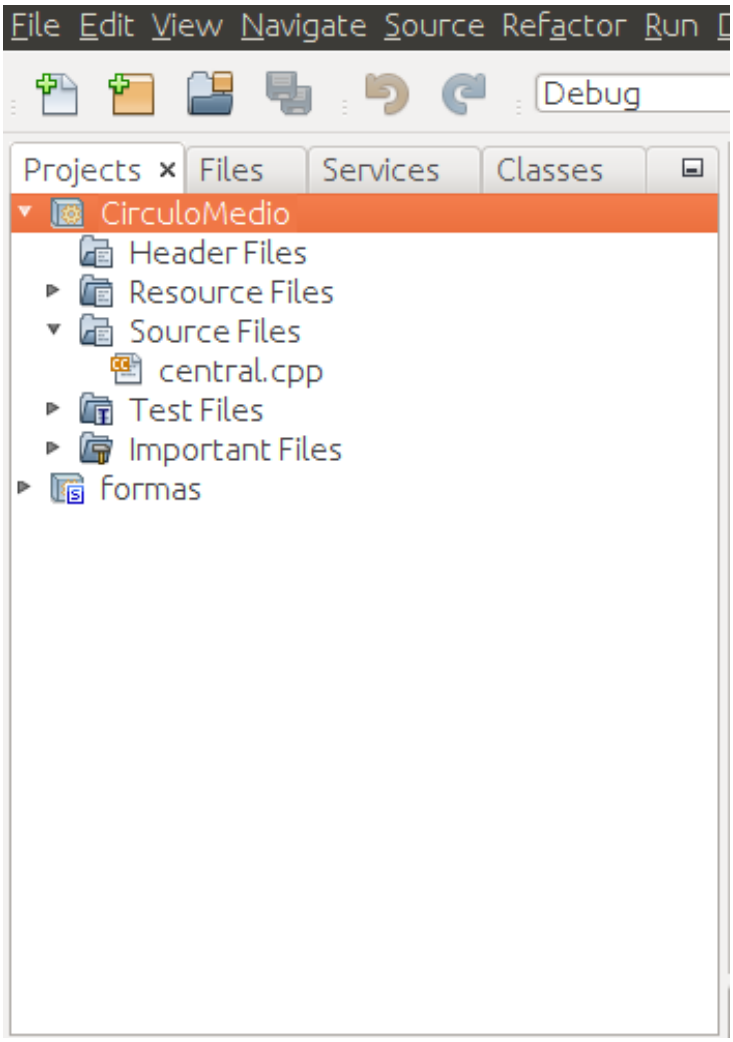


Figure 24: Nuevo árbol lógico del proyecto, que excluye a todos los .h y .cpp que se han movido a la biblioteca recien creada

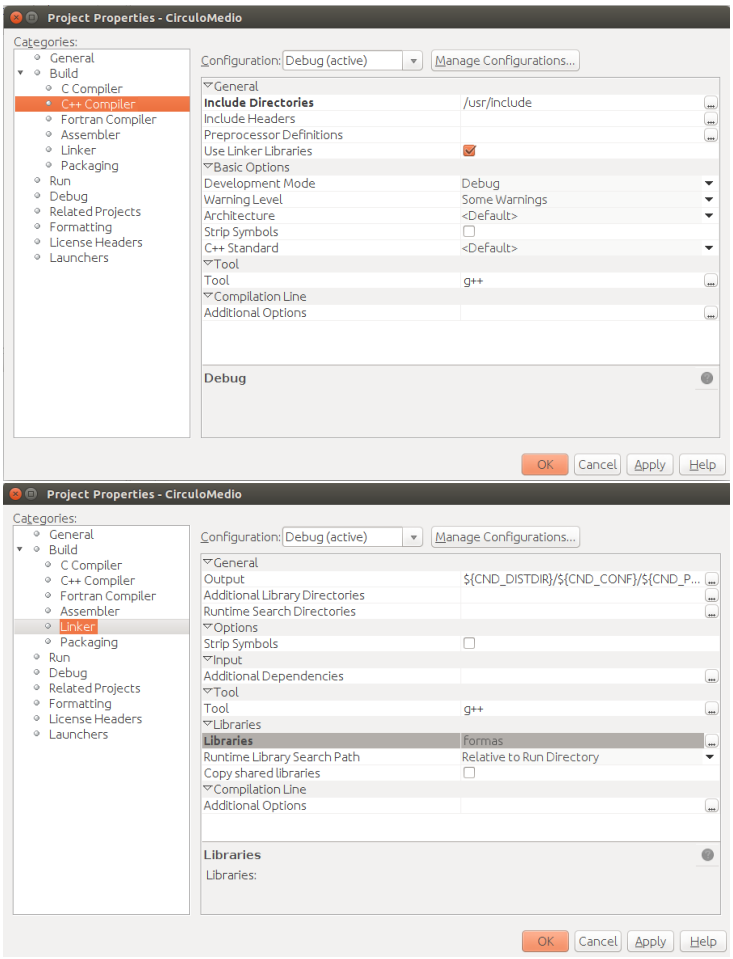


Figure 25: Configuración del proyecto para incluir la carpeta de cabeceras del sistema (arriba) y la librería recien creada (abajo)

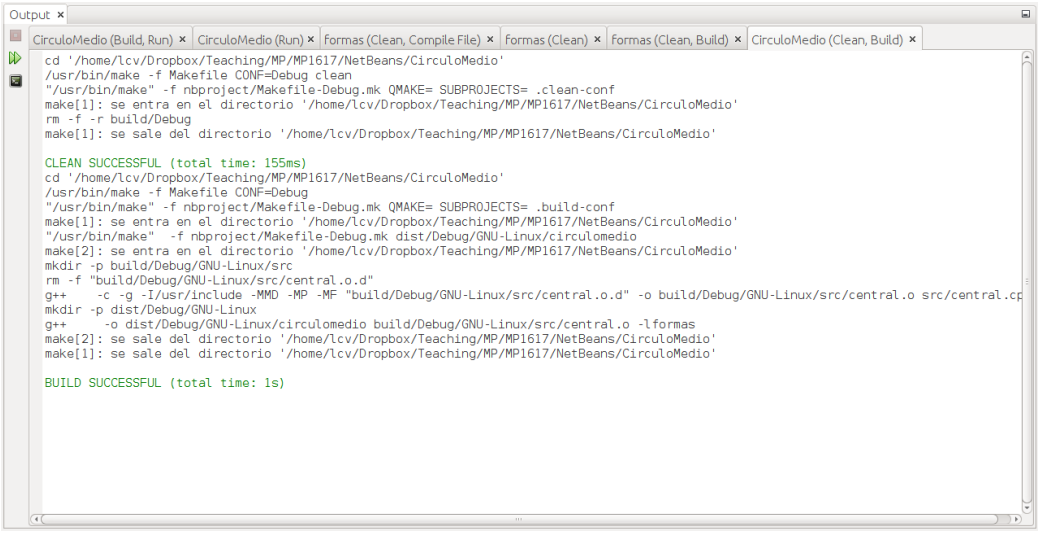


Figure 26: Salida de la creación del nuevo proyecto (**Clean and Build Project**) en la que se puede apreciar el enlazado con la biblioteca recien creada

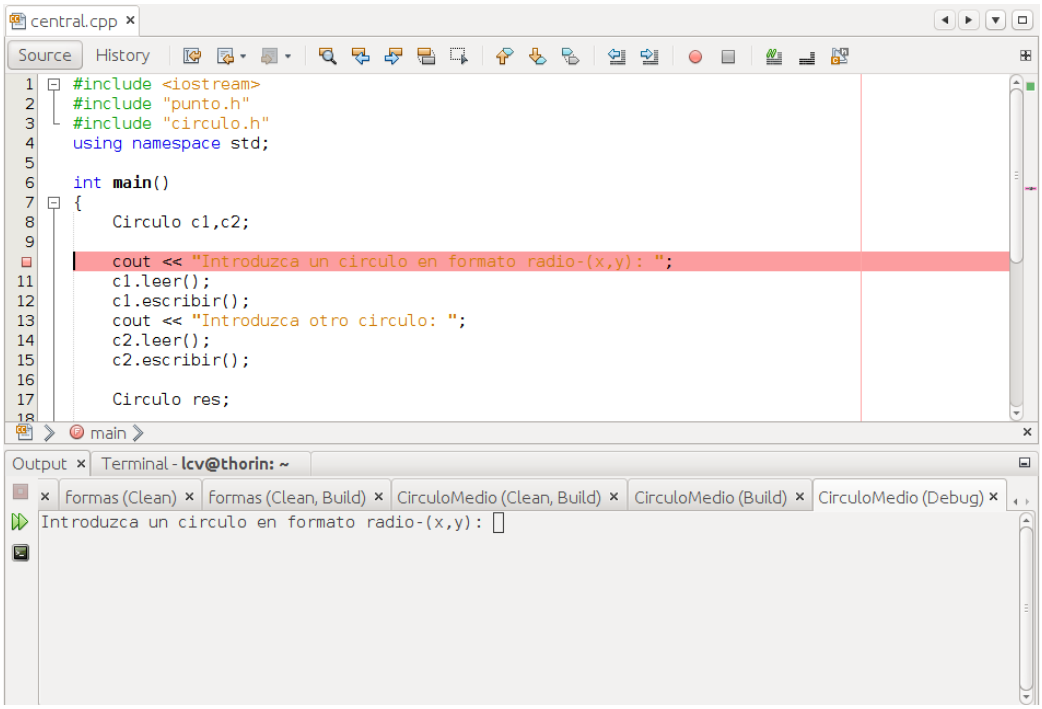


Figure 27: Creando un punto de ruptura para depurar un programa. El PR aparece como una línea de color rojo

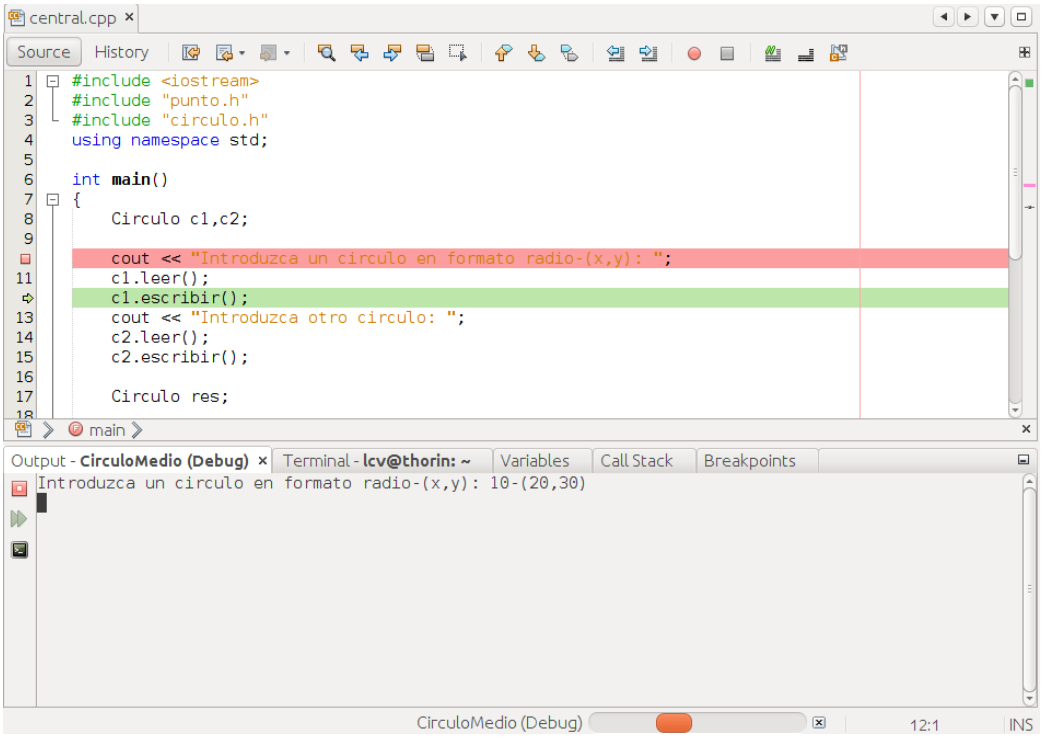


Figure 28: Ejecutando un programa paso a paso. La línea que se va a ejecutar a continuación aparece marcada en color verde

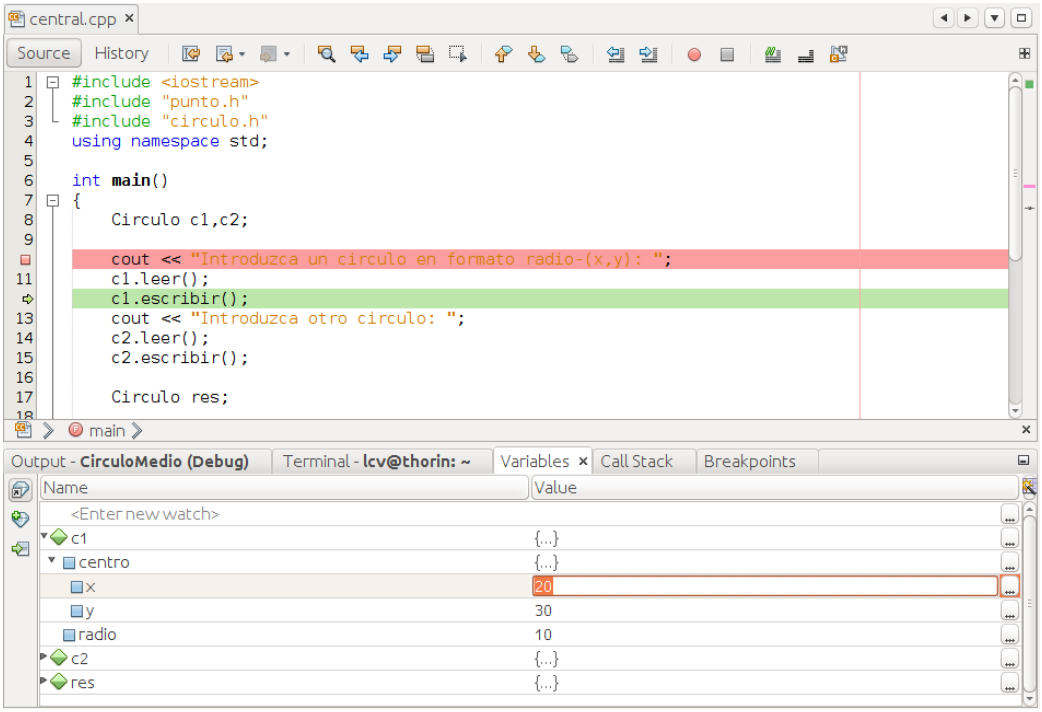


Figure 29: Inspeccionado y modificando, si fuese necesario, las variables del programa

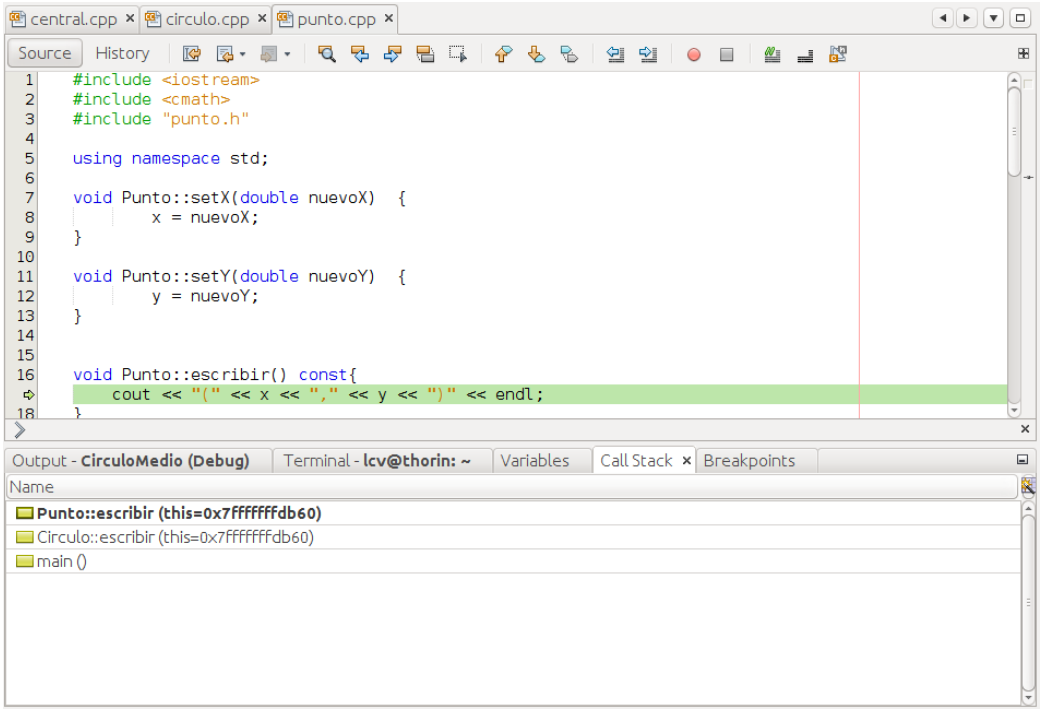


Figure 30: Inspeccionado y modificando, si fuese necesario, las pila de llamadas del programa

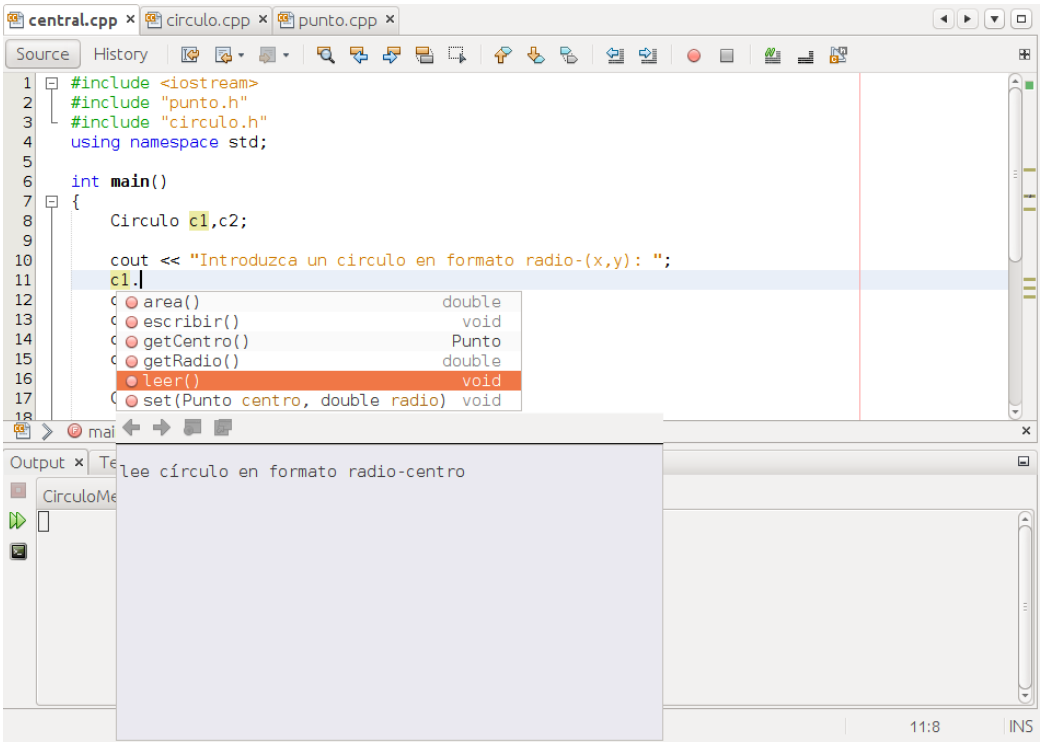


Figure 31: Auto-rellenado de código

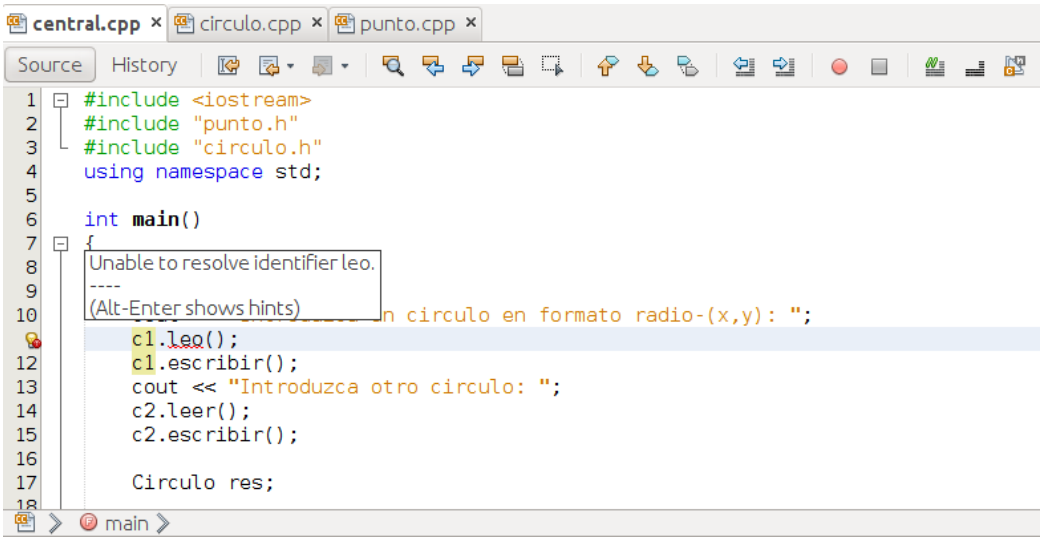


Figure 32: Detección inmediata de llamadas erróneas

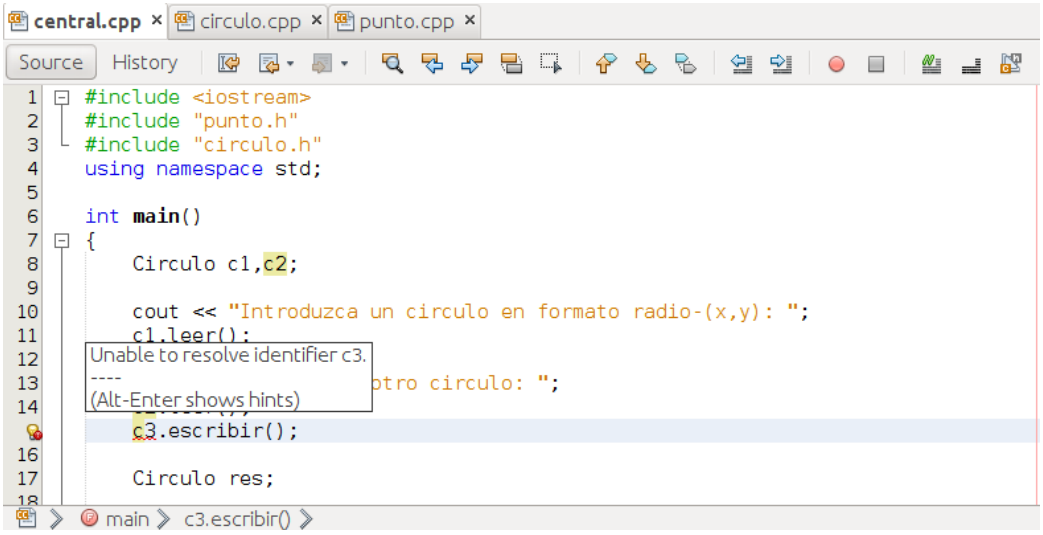


Figure 33: Detección inmediata del uso de identificadores erróneos

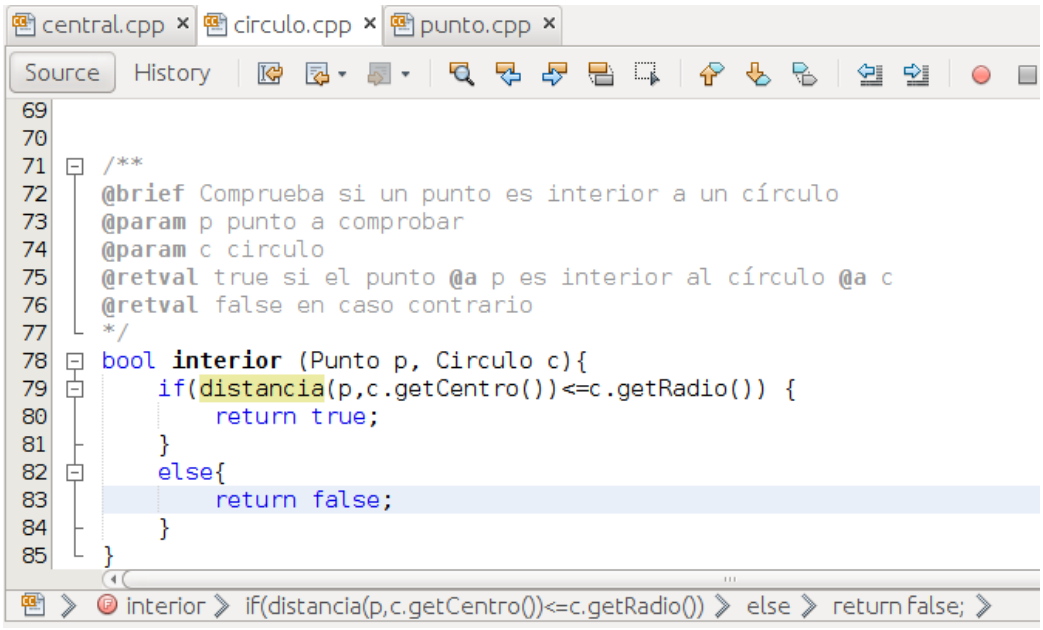


Figure 34: Detección inmediata del nivel de anidación del código. En la parte inferior del editor se puede observar la localización precisa de la línea que se está editando (línea número 83): Función **interior**, dentro de un **if** y en la parte **else** del mismo