



Guion de prácticas

*Entorno de desarrollo de software
con NetBeans*

Marzo 2018



Metodología de la Programación

Curso 2017/2018

Índice

1. Introducción	5
2. Empezando	5
3. Un primer proyecto	6
3.1. Creación del proyecto NetBeans	6
3.2. Estructura de carpetas del proyecto	6
3.3. Personalizando el proyecto	7
3.4. Generando y ejecutando el binario	7
3.5. Personalización del Makefile de NetBeans	8
4. Bibliotecas	8
4.1. Crear una nueva biblioteca a partir de ficheros fuentes ya existentes	9
4.2. Usar la nueva biblioteca en el proyecto anterior	9
5. Depurador	10
5.1. Conceptos básicos	10
5.2. Ejecución de un programa paso a paso	10
5.3. Inspección y modificación de datos	11
5.4. Inspección de la pila	11
5.5. Reparación del código	11
6. Otras características de NetBeans	11
7. Práctica a entregar: envíos de paquetes	12
8. Entrega	14
9. Pantallas de referencia	17

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 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
```

En la Figura 3 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 a lo largo de este guión. En cualquier caso, para más información se puede consultar la guía oficial de NetBeans³.
- 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 4).
- 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.

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

²<https://netbeans.org/features/cpp/>

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

- 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).

3. Un primer proyecto

Esta sección describe cómo se crearía el mismo proyecto de compilación separada del **circulomedio** y se resuelve de forma paralela pero desde NetBeans. Por tanto, se parte desde la Versión 3 en la que el programa principal **circulomedio.cpp** se divide en la siguiente estructura de ficheros

```
./
├── include
│   ├── punto.h
│   └── circulo.h
├── src
│   ├── central.cpp
│   ├── circulo.cpp
│   └── punto.cpp
├── bin
├── obj
├── lib
├── data
└── zip
```

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 5.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 5.b). Asegurarse de que la opción "Create Main File" está desactivada.

En el navegador de proyectos (cuadrante superior izquierdo) aparecerán los árboles lógicos y físicos del proyecto recién creado (Figura 6).

3.2. Estructura de carpetas del proyecto

Sobre el árbol físico del proyecto (pestaña **Files**), crear la típica estructura de carpetas comentada en la Práctica 1 (Figura 7). En este caso hay varias excepciones:

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 (Figura 11).
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 4.

Una vez creada la estructura de carpetas, desde el árbol lógico se crean los ficheros fuente que compondrán el proyecto, se colocan en las carpetas apropiadas (Figura 8) y se copian los contenidos desde la Versión 3 de la Práctica 1.

1. Source Files - New - C++ Source File
2. Header Files - New - C++ Header file

Alternativamente, se pueden arrastrar tanto las carpetas como los ficheros que contiene la Versión 3 de la Práctica 1 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** bien desde la línea de comandos, bien desde NetBeans y el árbol físico del proyecto.

3.3. Personalizando el proyecto

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 9).

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**
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.

3.4. 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

Cuya salida aparece en el cuadrante inferior derecho, pestaña **Output** (Figura 10).

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 recién generado se encuentra en la carpeta **dist** tal y como muestra la Figura 11.

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 12) 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 13 para indicar que el binario se va a ejecutar redireccionando la entrada estándar desde **data/circulomedio.dat**.

3.5. Personalización 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 solamente **make**. Desde la línea de comandos, 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.

4. 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 16). 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.

4.1. 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 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 17 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 9) 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 caso, no se ha generado un binario, sino una biblioteca. En la Figura 18 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 19.
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 20).

4.2. 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 21.
2. En las propiedades del proyecto (Figura 22 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 22 abajo), añadir la biblioteca recién creada en la opción **Libraries**.

4. Crear el nuevo binario (**Clean and Build Project**) y observar (Figura 23) cómo se enlaza con la nueva biblioteca.

5. Depurador

5.1. 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.

5.2. Ejecución de un programa paso a paso

Para comenzar a ejecutar un programa bajo control del depurador es conveniente colocar un punto de ruptura⁴. 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 24). Una vez colocado este punto de ruptura se puede comenzar la ejecución del programa con el menú⁵

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 25). 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.

⁴Un 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

⁵Consultar <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.

5.3. 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 26).

5.4. 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 27).

5.5. 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**).

6. 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 28). 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 29). 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 30). 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 31). 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.

```
unzip -t paquetes_nb.zip
Archive:  paquetes_nb.zip
  testing: Makefile                                OK
  testing: build/                                  OK
  testing: data/                                  OK
  testing: data/paquete.dat                        OK
  testing: dist/                                  OK
  testing: doc/                                    OK
  testing: doc/paquetes.doxy                       OK
  testing: include/                               OK
  testing: include/paquete.h                       OK
  testing: include/secuenciapaquete.h              OK
  testing: nbproject/                             OK
  testing: nbproject/configurations.xml             OK
  testing: nbproject/Makefile-Debug.mk             OK
  testing: nbproject/Makefile-impl.mk              OK
  testing: nbproject/Package-Release.bash          OK
  testing: nbproject/Makefile-Release.mk           OK
  testing: nbproject/private/                      OK
  testing: nbproject/Makefile-variables.mk         OK
  testing: nbproject/project.xml                   OK
  testing: nbproject/Package-Debug.bash            OK
  testing: src/                                    OK
  testing: src/main.cpp                            OK
  testing: src/paquete.cpp                         OK
  testing: src/secuenciapaquete.cpp                OK
  testing: zip/                                    OK
No errors detected in compressed data of paquetes_nb.zip.
```

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

7. Práctica a entregar: envíos de paquetes

Se pide desarrollar un programa para llevar a cabo la facturación de paquetes dentro de la provincia para una central de correos.

Descargue el fichero **paquetes_nb.zip** (Figura 1), descomprímalo en una carpeta independiente y cárguelo en NetBeans con el menu “File-Open Project”. Complete las funciones y/o métodos incompletos. Para realizar esta tarea, tenga en cuenta que el objetivo es que escriba el programa de manera individual, entendiendo perfectamente todo el código que lo compone.

Pero antes, un pequeño apunte:

Peso volumétrico

Un paquete menos denso generalmente ocupa más volumen de espacio, en comparación a su peso real. El peso volumétrico se calcula y se compara con el peso real del envío para establecer cuál es el mayor; el peso más alto se utiliza para calcular el coste del envío. En correos se aplica la siguiente fórmula para el transporte terrestre:

$$\text{PesoVolumetrico} = \frac{\text{largo} * \text{ancho} * \text{alto} \text{ cm}^3}{6000 \text{ cm}^3/\text{kg}}$$

se compara éste con el peso real, el peso Volumétrico es el mayor de los dos pesos.

El programa debe leer una serie de valores asociados a cada paquete: la **identificación** de paquete (un número grande positivo), el **peso** en kg), el **largo**, el **ancho**, y el **alto** cada uno expresado en centímetros. El programa termina con un valor de -1 en la identificación del paquete.

Para calcular el precio del envío de un paquete se utiliza la siguiente tabla en base a su **peso volumétrico**:

P. Vol.	Precio	IVA ⁶	precio+IVA
Hasta 2 kgrs.	12,97	2,72	15,69
Ms de 2 hasta 5 kgrs.	16,83	3,53	20,36
Ms de 5 hasta 15 kgrs.	29,32	6,16	35,48

Se pide implementar la clase **Paquete** con los datos miembro, constructores y algunos métodos necesarios para resolver este problema de la facturación por paquete y las funciones especificadas en el fichero **se-
cuenciapaquete.h** que realizan la facturación por envíos diarios. Para ello se han de usar las declaraciones en los ficheros de cabecera suministrados en el fichero **paquetes.nb.zip** y suponiendo que en un día no se gestionan más de 25 paquetes.

Diseñar un programa que lea desde el teclado la información relativa a cada paquete, y devuelva la siguiente información:

1. Facturación completa del día en la central
2. Precio medio global
3. Número de paquetes que superan 50cm en altura
4. Número de paquetes que superan el precio medio global del día
5. Identificador del paquete de mayor volumen m^3
6. PesoVolumetrico del paquete de mayor volumen
7. Volumen del paquete de mayor volumen

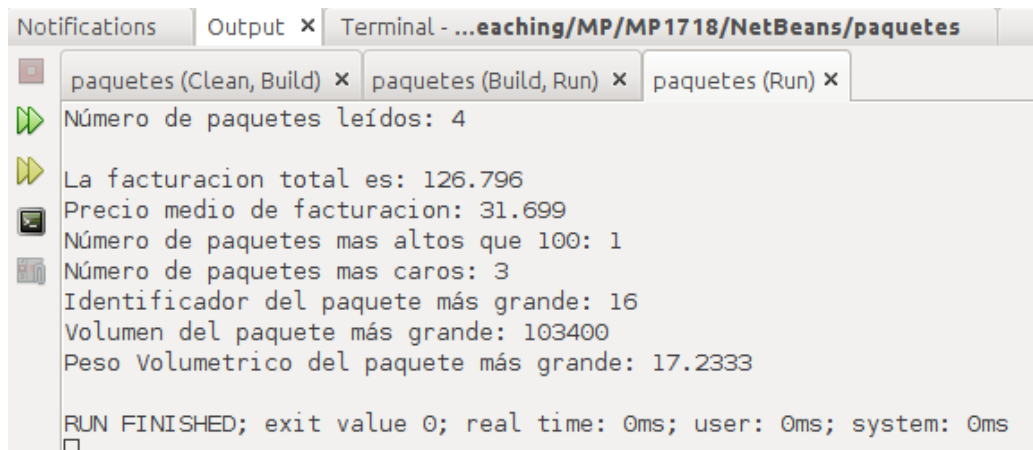
Implementa los métodos y/o funciones externas necesarias para realizar las tareas indicadas.

Se proporciona el fichero **paquetes.dat** un fichero de validación. Así, para el siguiente registro de entradas:

```
1 5.5 30 40 20
3 2.1 15 15 60
5 8.4 50 50 20
16 10.1 55 40 47
-1
```

la salida del programa debería ser la siguiente:

El proyecto debe estar integrado en NetBeans, según lo explicado en este guión.



```

paquetes (Clean, Build) x paquetes (Build, Run) x paquetes (Run) x
Número de paquetes leídos: 4
La facturación total es: 126.796
Precio medio de facturación: 31.699
Número de paquetes mas altos que 100: 1
Número de paquetes mas caros: 3
Identificador del paquete más grande: 16
Volumen del paquete más grande: 103400
Peso Volumetrico del paquete más grande: 17.2333
RUN FINISHED; exit value 0; real time: 0ms; user: 0ms; system: 0ms

```

Figura 2: Salida del programa **paquetes** desde NetBeans

8. Entrega

Se debe de entregar através de **DECSAI** un fichero zip, **practica3.zip** la estructura de directorios ya expuesta: **bin**, **data**, **include**, **lib**, **obj**, **src**, **doc** y **comprimida** con la regla zip que aparece en el makefile de la sección **3.5**.

El fichero **practica3.zip** entregado deberá 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.

```

$$ unzip practica3.zip
Archive:  practica3.zip
  inflating: Makefile
    creating: build/
    creating: data/
  inflating: data/paquete.dat
    creating: dist/
    creating: doc/
  inflating: doc/paquetes.doxy
    creating: include/
  inflating: include/paquete.h
  inflating: include/secuenciapaquete.h
    creating: nbproject/
  inflating: nbproject/configurations.xml
  inflating: nbproject/Makefile-Debug.mk
  inflating: nbproject/Makefile-impl.mk
  inflating: nbproject/Package-Release.bash
  inflating: nbproject/Makefile-Release.mk
    creating: nbproject/private/
  inflating: nbproject/Makefile-variables.mk
  inflating: nbproject/project.xml
  inflating: nbproject/Package-Debug.bash
    creating: src/
  inflating: src/main.cpp
  inflating: src/paquete.cpp
  inflating: src/secuenciapaquete.cpp
    creating: zip/

$$ make
"make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= [...]
make[1]: se entra en el directorio '/home/lcv/Tmp/paquetes'
"make" -f nbproject/Makefile-Debug.mk dist/Debug/GNU-Linux/[...]
make[2]: se entra en el directorio '/home/lcv/Tmp/paquetes'
mkdir -p build/Debug/GNU-Linux/src
rm -f "build/Debug/GNU-Linux/src/main.o.d"
g++ -c -g -Iinclude -MMD -MP -MF "build/Debug/GNU-Linux/[...]"
mkdir -p build/Debug/GNU-Linux/src
rm -f "build/Debug/GNU-Linux/src/paquete.o.d"
g++ -c -g -Iinclude -MMD -MP -MF "build/Debug/GNU-Linux/[...]"
mkdir -p build/Debug/GNU-Linux/src
rm -f "build/Debug/GNU-Linux/src/secuenciapaquete.o.d"
g++ -c -g -Iinclude -MMD -MP -MF "build/Debug/GNU-Linux/[...]"
mkdir -p dist/Debug/GNU-Linux
g++ -o dist/Debug/GNU-Linux/paquetes build/Debug/[...]
make[2]: se sale del directorio '/home/lcv/Tmp/paquetes'
make[1]: se sale del directorio '/home/lcv/Tmp/paquetes'

$$ dist/Debug/GNU-Linux/paquetes < data/paquete.dat
Introduce identificador de paquete:  Introduce peso en [...]

Número de paquetes leídos: 4

La facturacion total es: 126.796
Precio medio de facturacion: 31.699
Número de paquetes mas altos que 100: 1

```

```
Número de paquetes mas caros: 3  
Identificador del paquete más grande: 16  
Volumen del paquete más grande: 103400  
Peso Volumetrico del paquete más grande: 17.2333
```


9. Pantallas de referencia

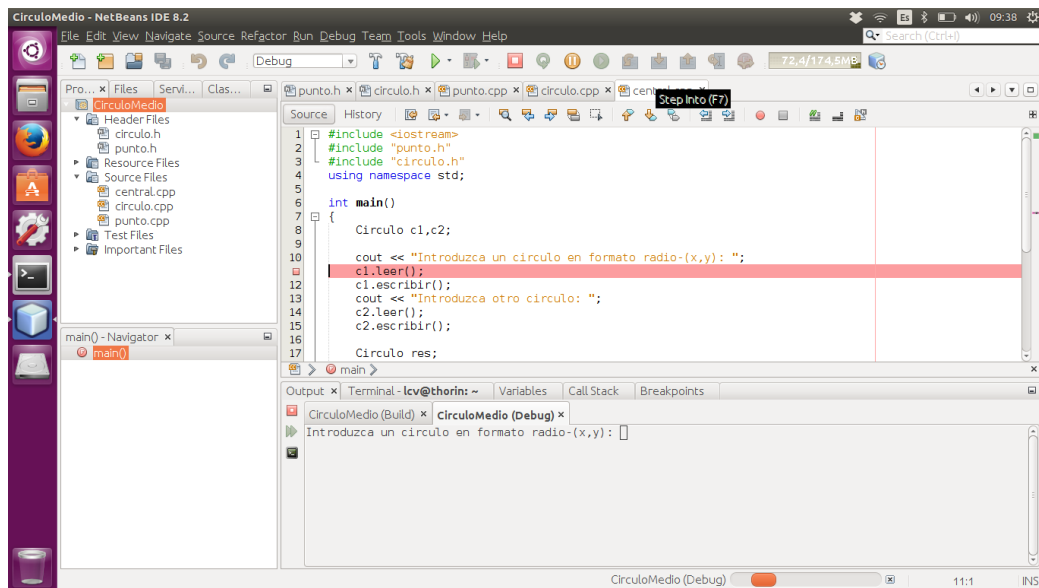


Figura 3: Entorno de trabajo en NetBeans con C++

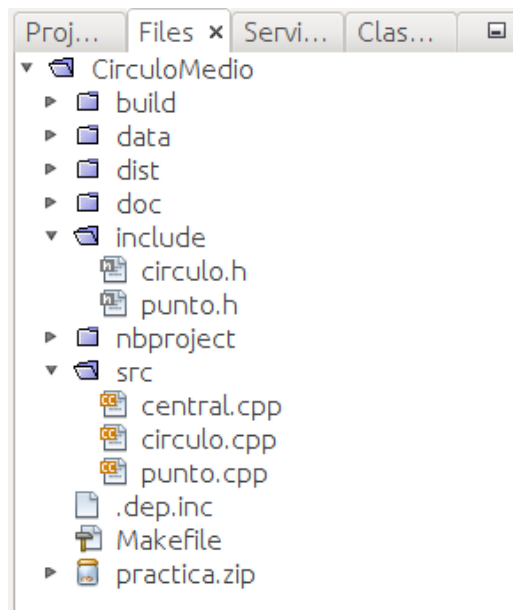
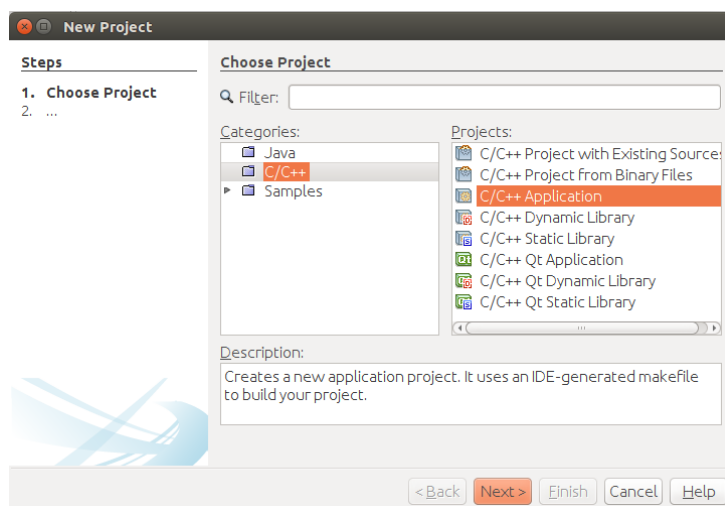
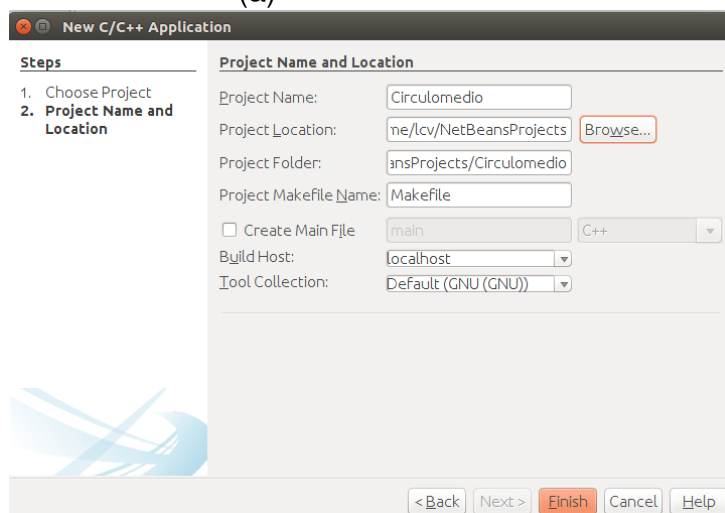


Figura 4: Estructura en disco de un proyecto NetBeans



(a)



(b)

Figura 5: Creando un proyecto nuevo

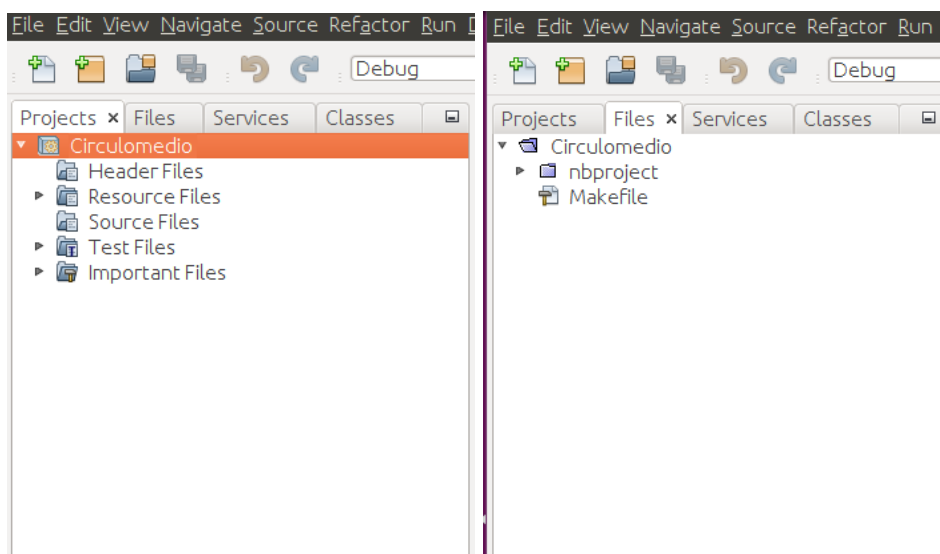


Figura 6: Árbol lógico (izquierda) y físico (derecha) de un proyecto nuevo

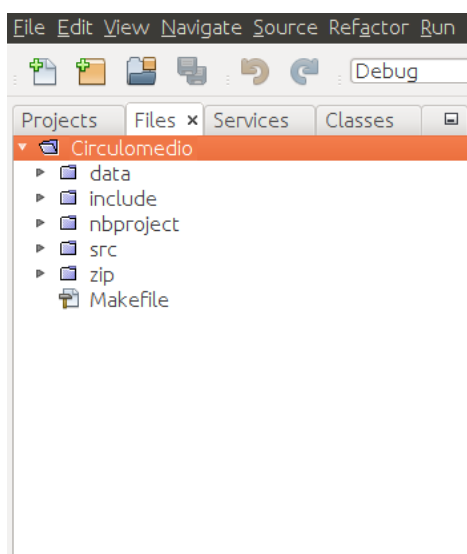


Figura 7: Estructura de carpetas del proyecto

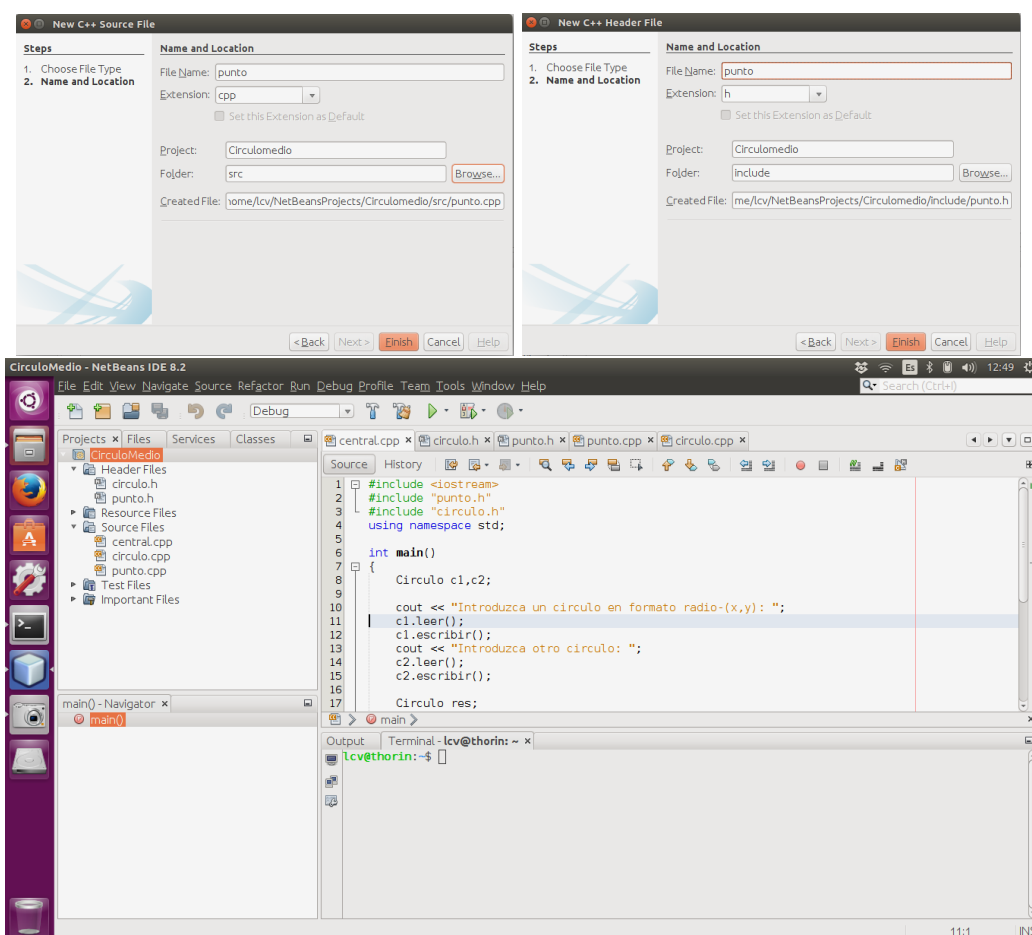


Figura 8: Desde el árbol lógico, crear cada fichero del proyecto (cpp, h) y colocarlo en la carpeta correspondiente (src, include).

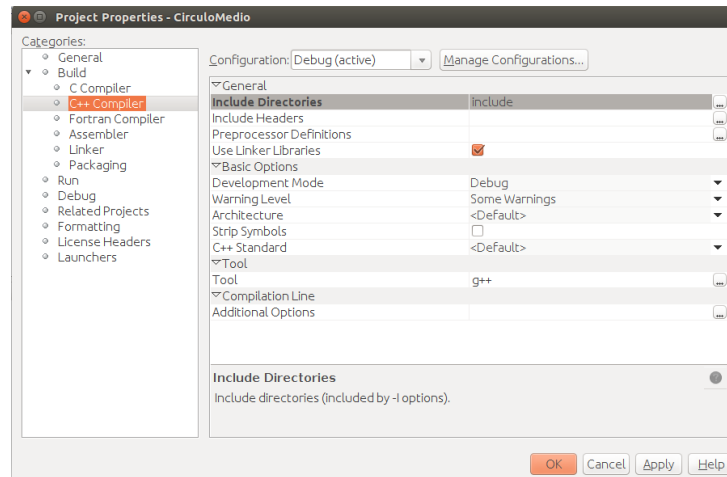


Figura 9: Propiedades del proyecto

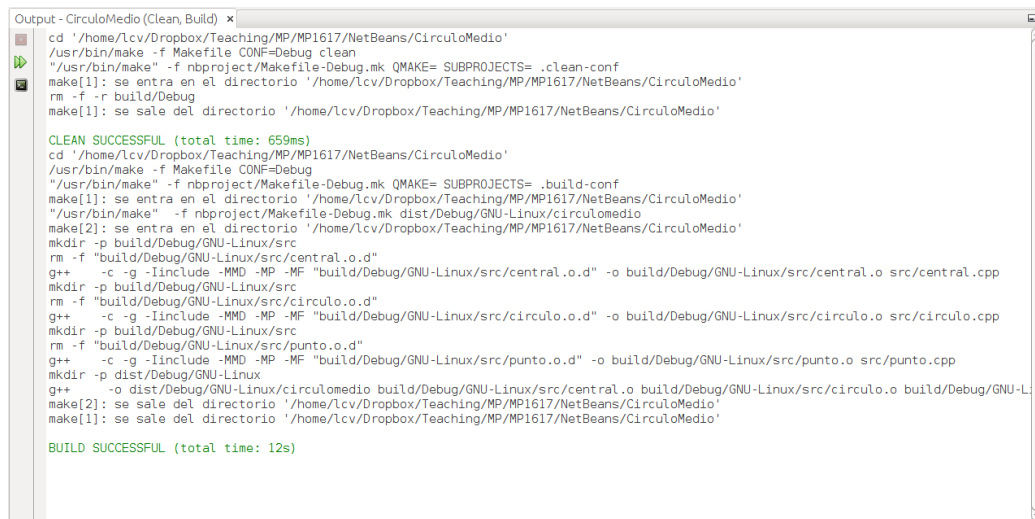


Figura 10: Generando el binario

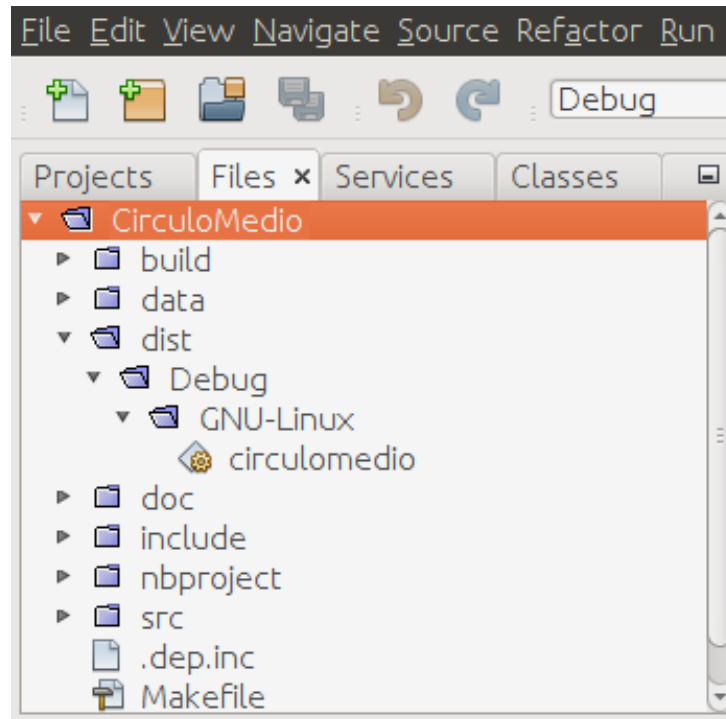


Figura 11: Situación del binario en el árbol físico del proyecto, dentro de la carpeta **dist**

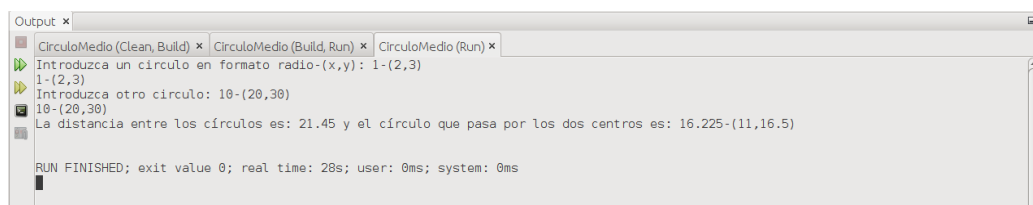


Figura 12: Ejecutando el binario

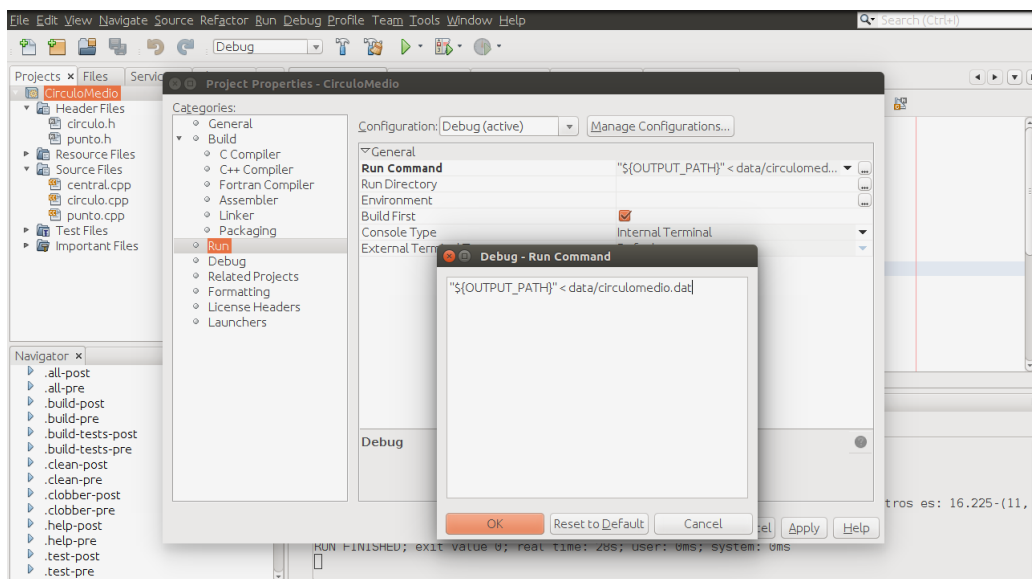


Figura 13: Ejecutando el binario con argumentos o redireccionamientos

```
# 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/paquetes.doxy
    firefox doc/html/index.html

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

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

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

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

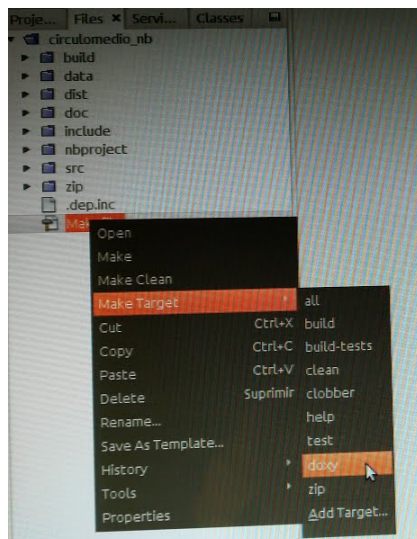


Figura 15: Ejecutando el makefile de NetBeans con objetivos personalizados

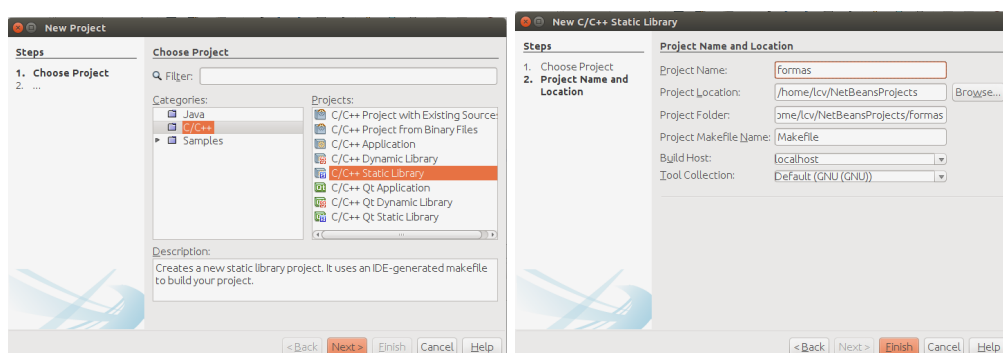


Figura 16: Creando un nuevo proyecto para gestionar una biblioteca

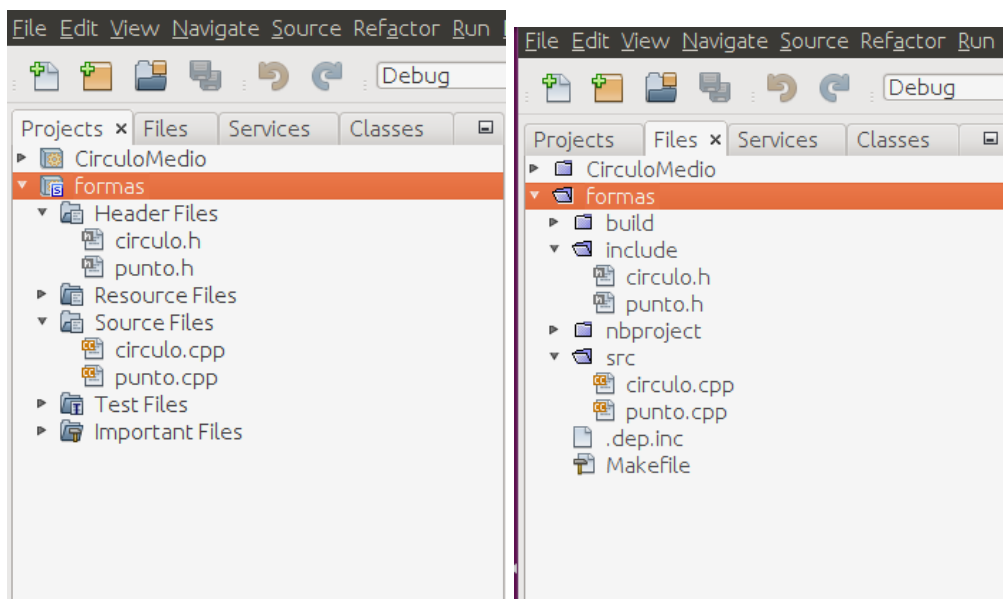


Figura 17: Árboles lógico (izqda) y físico (dcha) del nuevo proyecto de biblioteca



Figura 18: Creación de la biblioteca de forma automática desde NetBeans

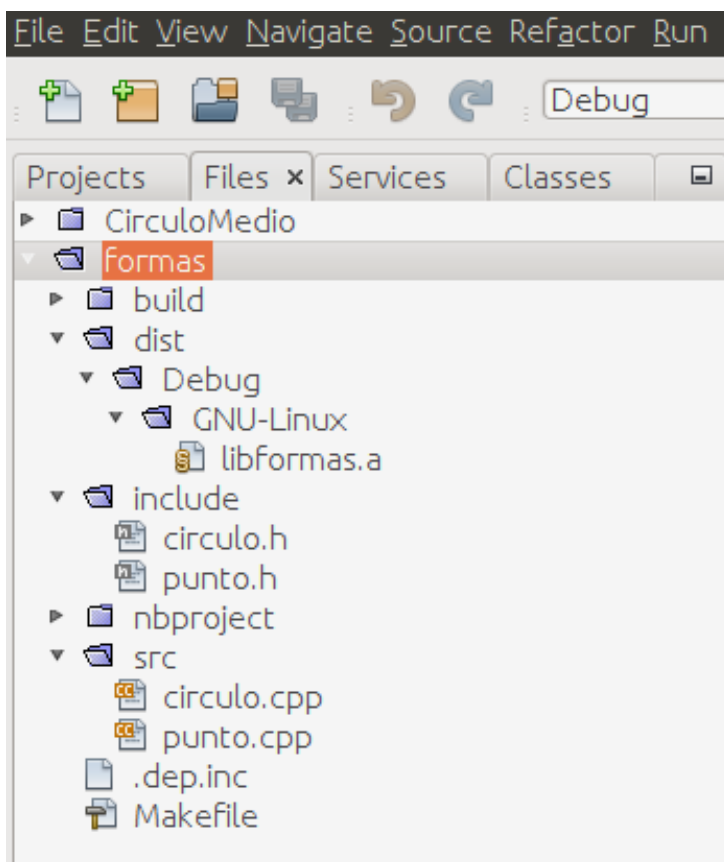


Figura 19: Situación de la biblioteca en el árbol físico del proyecto dentro de la carpeta **dist**

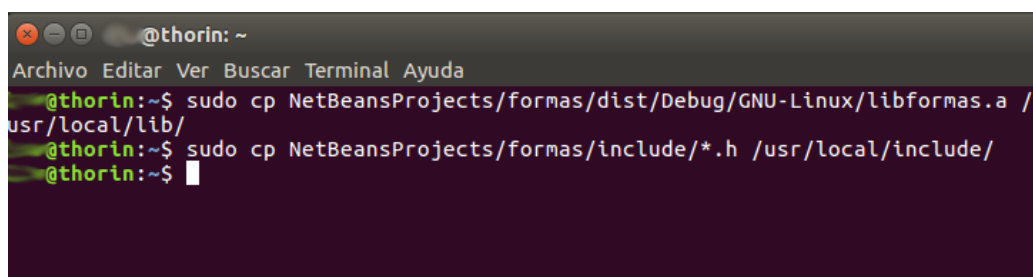


Figura 20: Colocar los .h y los .a en las carpetas del sistema para su uso posterior en otros proyectos

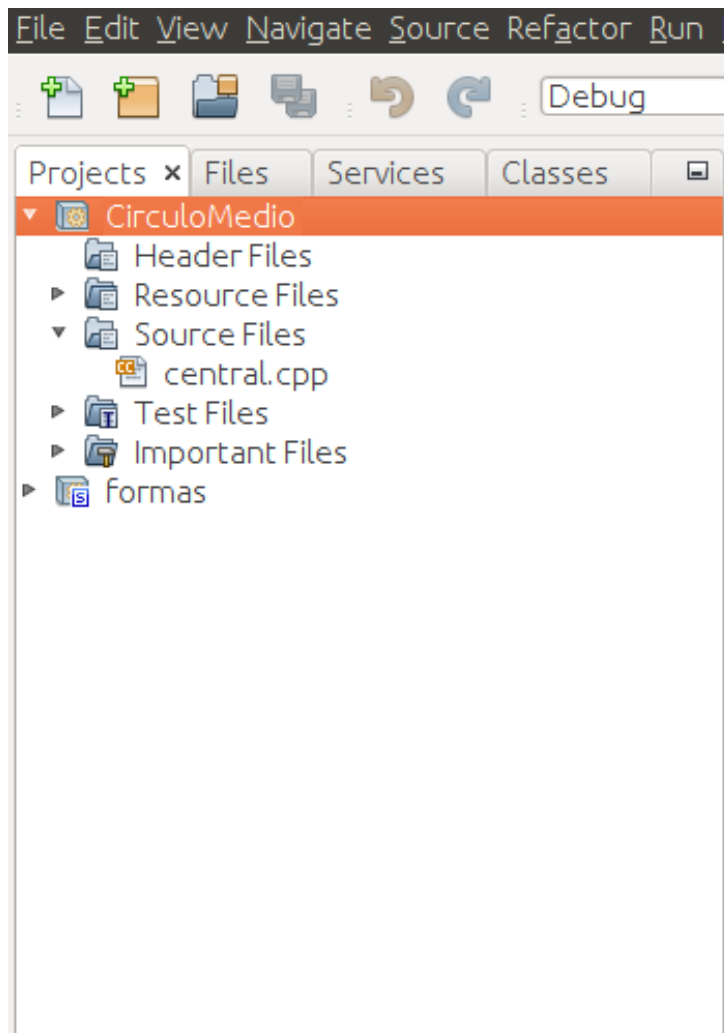


Figura 21: Nuevo árbol lógico del proyecto, que excluye a todos los .h y .cpp que se han movido a la biblioteca recién creada

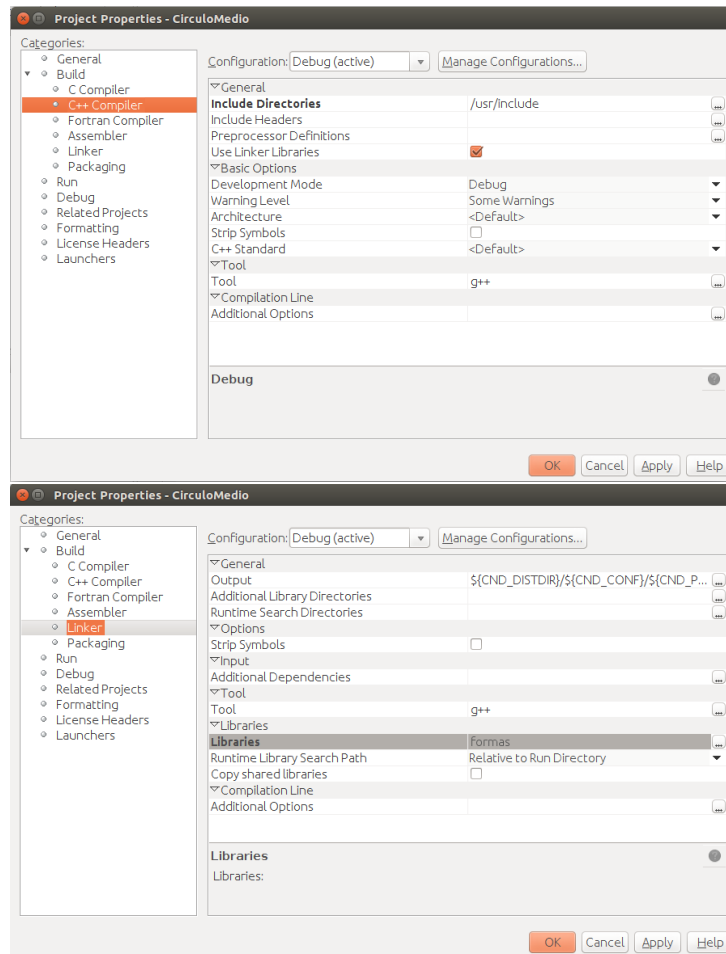


Figura 22: Configuración del proyecto para incluir la carpeta de cabeceras del sistema (arriba) y la librería recién creada (abajo)



Figura 23: Salida de la creación del nuevo proyecto (**Clean and Build Project**) en la que se puede apreciar el enlazado con la biblioteca recién creada

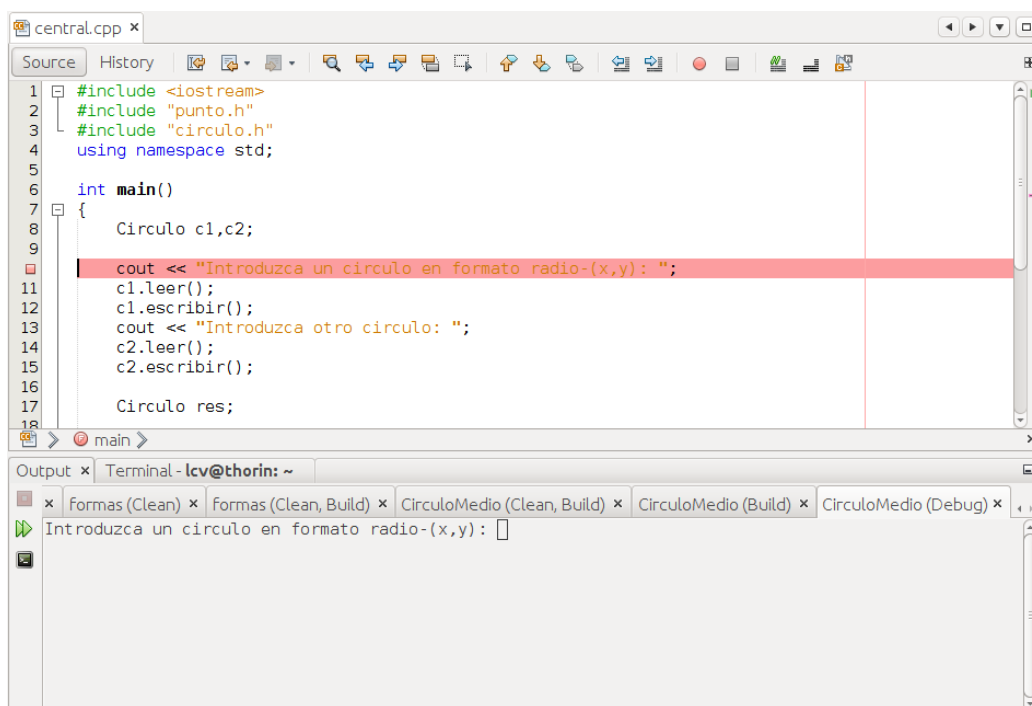


Figura 24: Creando un punto de ruptura para depurar un programa. El PR aparece como una línea de color rojo

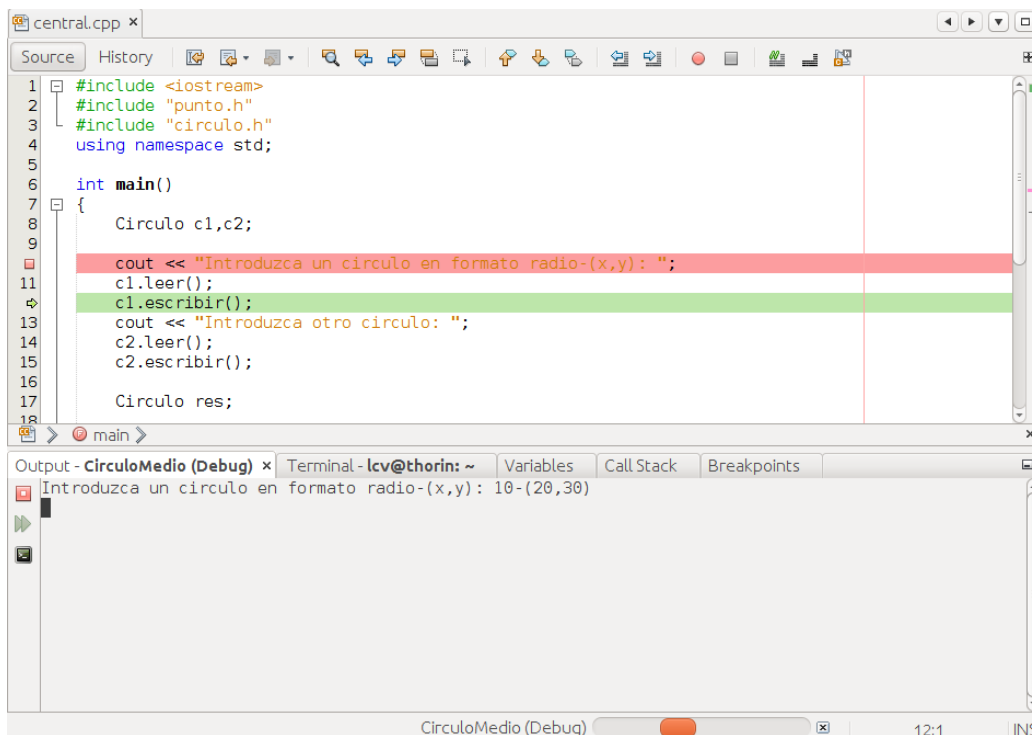


Figura 25: Ejecutando un programa paso a paso. La línea que se va a ejecutar a continuación aparece marcada en color verde

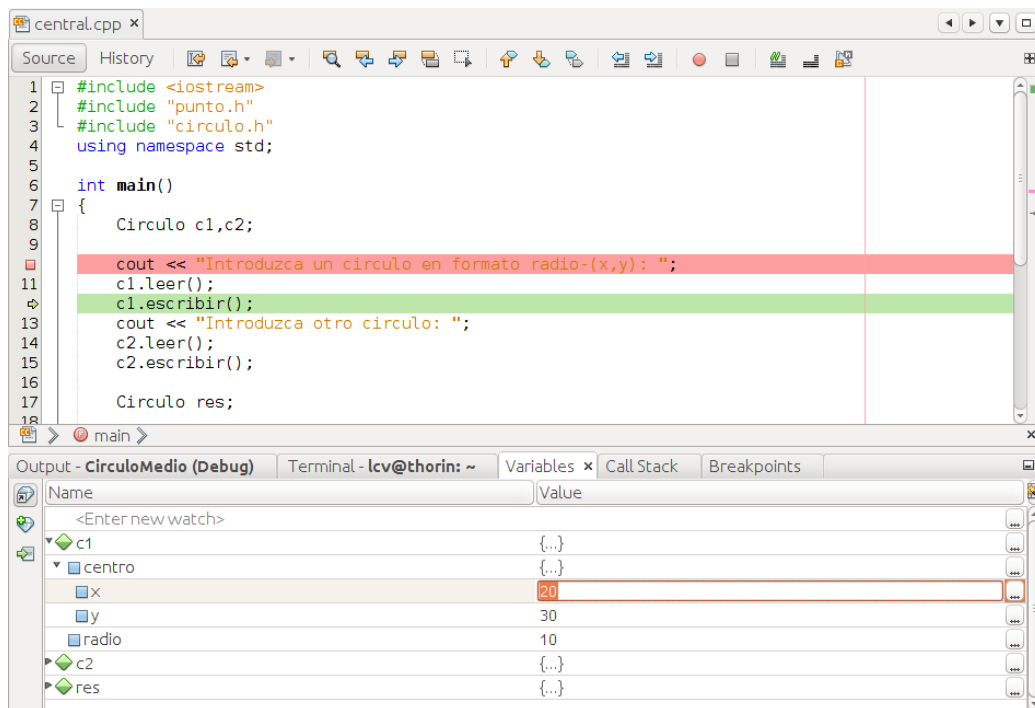


Figura 26: Inspeccionado y modificando, si fuese necesario, las variables del programa

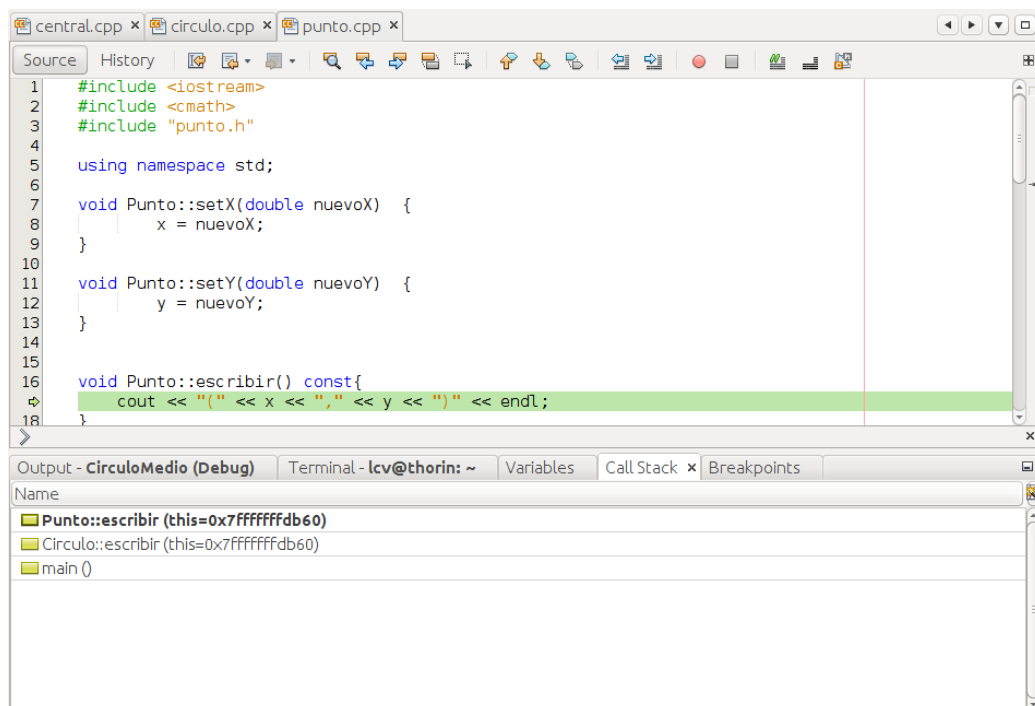


Figura 27: Inspeccionado y modificando, si fuese necesario, las pila de llamadas del programa

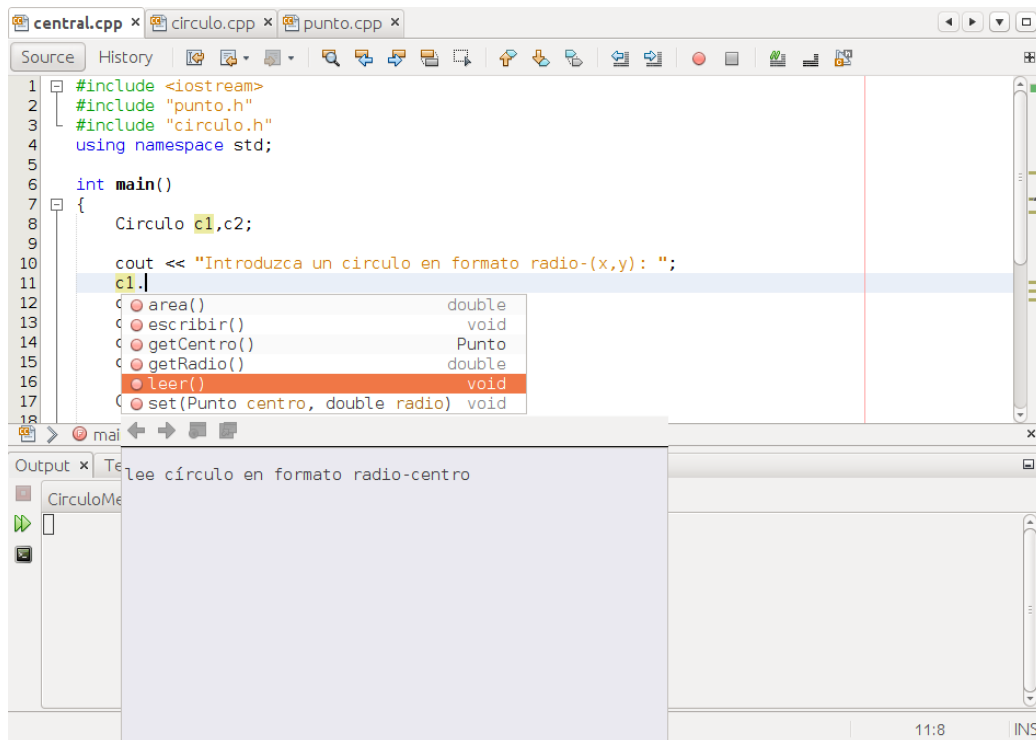


Figura 28: Auto-relleno de código

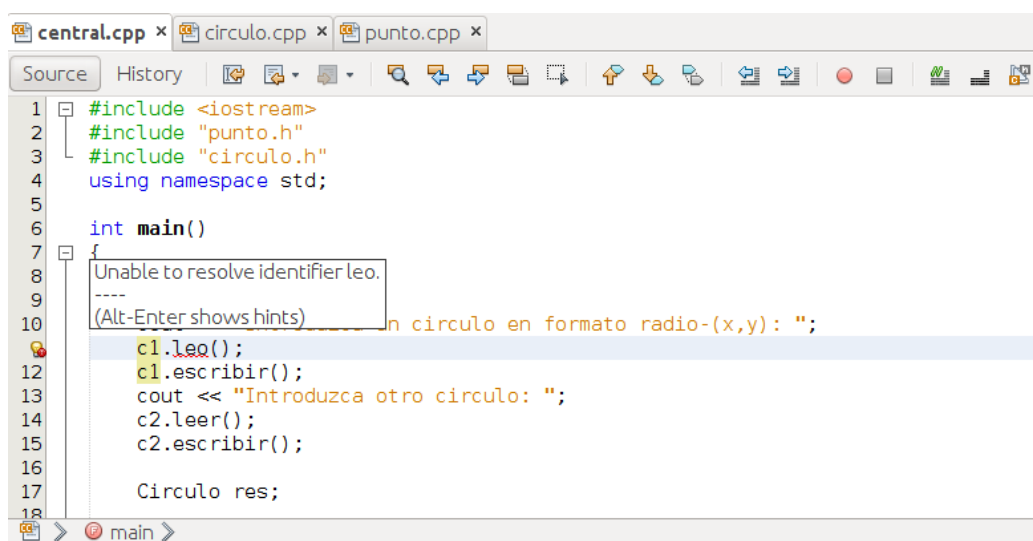
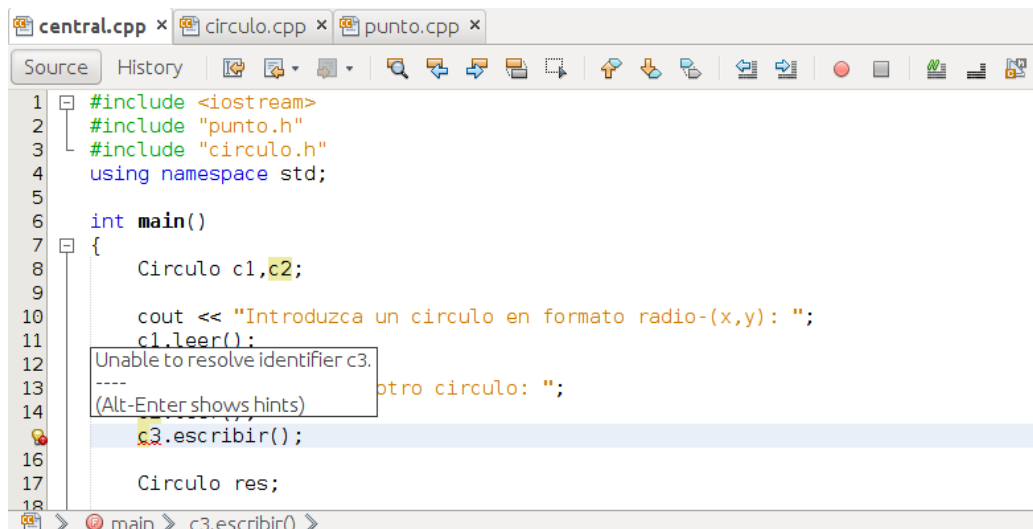


Figura 29: Detección inmediata de llamadas erróneas



```

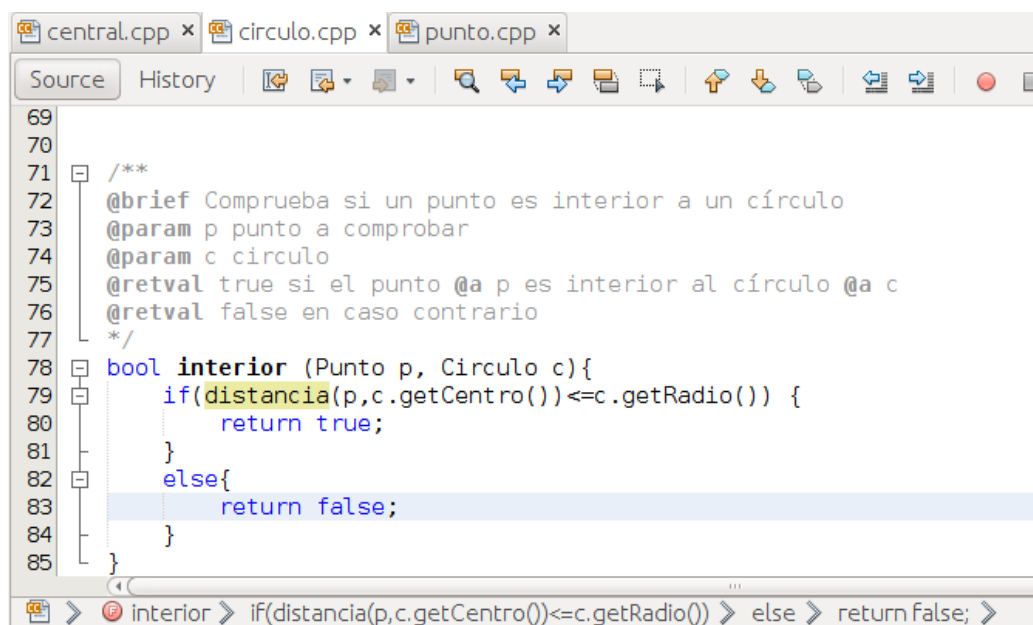
1 #include <iostream>
2 #include "punto.h"
3 #include "circulo.h"
4 using namespace std;
5
6 int main()
7 {
8     Circulo c1, c2;
9
10    cout << "Introduzca un circulo en formato radio-(x,y): ";
11    c1.leer();
12    c3.escribir();
13    cout << "otro circulo: ";
14
15    Circulo res;
16
17 }

```

Unable to resolve identifier c3.

(Alt-Enter shows hints)

Figura 30: Detección inmediata del uso de identificadores erróneos



```

69
70
71 /**
72  * @brief Comprueba si un punto es interior a un círculo
73  * @param p punto a comprobar
74  * @param c círculo
75  * @retval true si el punto @a p es interior al círculo @a c
76  * @retval false en caso contrario
77  */
78 bool interior (Punto p, Circulo c){
79     if(distancia(p,c.getCentro())<=c.getRadio()) {
80         return true;
81     }
82     else{
83         return false;
84     }
85 }

```

interior > if(distancia(p,c.getCentro())<=c.getRadio()) > else > return false; >

Figura 31: 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