Guion de prácticas adicional

Entorno de desarrollo software NetBeans (Parte I)

Febrero de 2020

Metodología de la Programación - ADE

Curso 2019/2020

Índice

1. Introducción

Esta sesión de prácticas está dedicada a aprender a utilizar el entorno de desarrollo de software multi-lenguage `NetBeans` como una alternativa para la gestión de proyectos. NetBeans es un entorno de desarrollo integrado libre y multiplataforma, hecho 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 IDE 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

Para poder ejecutar Netbeans con el plugging de C++ debemos arrancar con la versión Ubuntu 16 - 32 bits. Una vez que hemos arrancado Ubunto, podemos encontrar Netbeans dentro de los menús para el desarrollo de software o desde la línea de comando como

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

2.1. Instalación de Netbeans

Si queremos trabajar con nuestro propio ordenador, es necesario instalar NetBeans con JDK[1] y el plugin para C++[2] instalados en ese orden.

Podéis ver el siguiente tutorial en youtube `https://www.youtube.com/watch?v=z21aUmt1yig` o mejor `https://www.youtube.com/watch?v=ifiAYdgt42g` donde se os muestra la instalación en Windows.

De una forma mas cómoda, podemos descargar una versión que incluya el plugin si utilizamos Netbeans 8.2. En la siguiente dirección podemos encontrar el link para su descarga `https://netbeans.org/downloads/8.2/`

En cualquier caso, se recomienda encarecidamente consultar la guía C/C++ Application Learning Trails en NetBeans `https://netbeans.org/kb/trails/cnd.html`, ya que nos indica detalladamente los pasos que tenemos que dar para conseguir una correcta instalación en distintos sistemas operativos, además de ofrecernos tutoriales interesantes sobre el uso de C/C++ con NetBeans

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

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

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

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

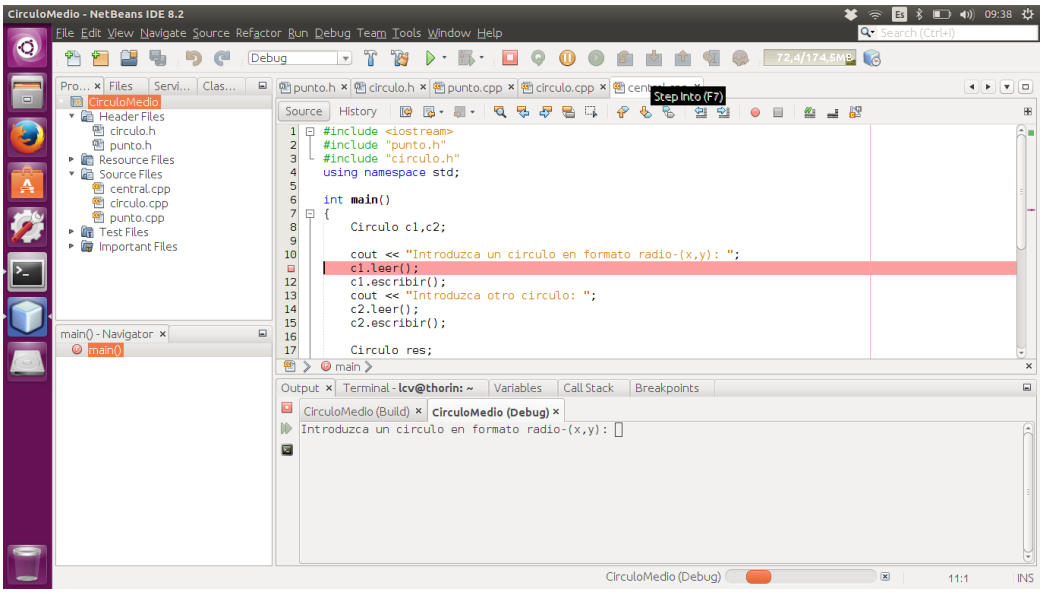


Figura 1: Entorno de trabajo en NetBeans con C++

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 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.
- 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 llamadas (depuración).
  - Breakpoints. Lista de puntos de ruptura activos (depuración).

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

3. Una primera aplicación

Esta sección describe cómo se crearía nuestro primer proyecto con NetBeans, que como no, será el clásico ”Hola Mundo!”. Para mas información, se puede consultar el tutorial `C/C++ Projects Quick Start Tutorial`, en https://netbeans.org/kb/docs/cnd/quickstart.html.

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

Asegurarse de que la opción ”Create Main File” está activada, por lo que nos generará un fichero denominado `main.cpp` en el que se encuentra la función main. También es importante prestar atención a la versión del compilador C++ que vamos a utilizar (en nuestro caso usaremos c++11).

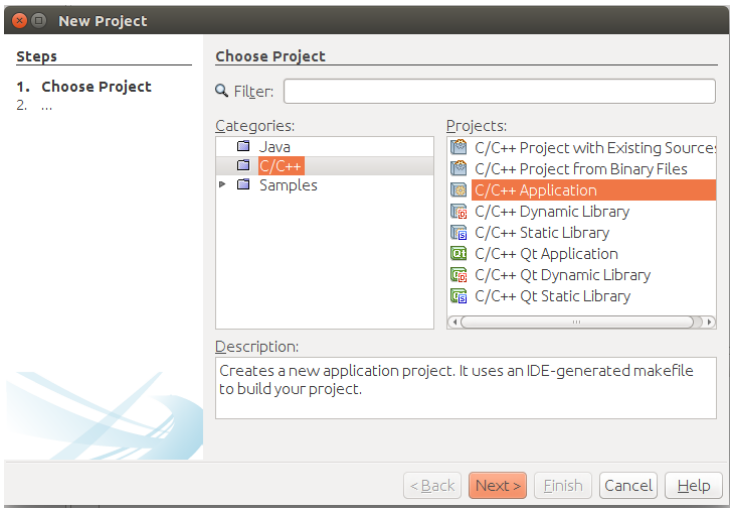
En el navegador de proyectos (cuadrante superior izquierdo) aparecerán los árboles lógicos y físicos del proyecto recién creado (Figura 3). Estas vistas no tienen porqué coincidir, de hecho no suelen hacerlo.

Nos centraremos en el árbol lógico, bajo la pestaña `Projects`. En este caso aparecen 5 carpetas lógicas, la que por ahora nos interesa es la carpeta Source Files que es donde encontraremos nuestro código, esto es el fichero donde encontramos la función main (por defecto Netbeans lo llama `main.cpp`). El resto lo iremos viendo a lo largo del curso.

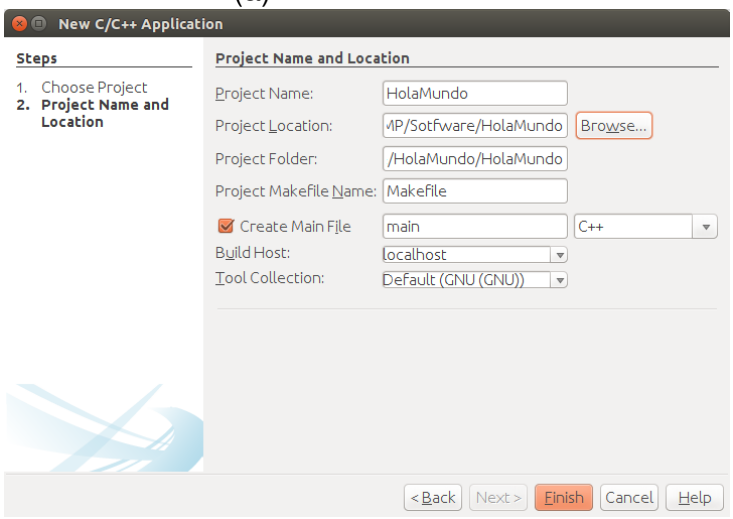
Una vez que hemos escrito el código asociado,

```
#include <cstdlib>
#include <string>
#include <iostream>

using namespace std;

/** funcion ejemplo simple
 *
 */
int main(int argc, char** argv) {
    cout << "Hola Mundo" << endl;
    return 0;
}
```

Podemos compilar y ejecutar el programa. Antes de compilar, debemos decidir que versión queremos obtener, una de prueba/depuración o la definitiva/producción. La primera está asociada a la opción `Debug` y la segunda a la opción `Release`, ver Fig. 4. La primera genera un código objeto más grande, pues almacena elementos que serán utilizados para la



(a)



(b)

Figura 2: Creando un proyecto nuevo

posible depuración del código, la segunda nos generará códigos objetos mas compactos, asociados a la versión final de la aplicación.

En cualquier caso, los ficheros generados por la fase de compilación son almacenados en dos directorios distintos

- build, donde se meten los ficheros intermedios generados durante el proceso de compilación. Lo veremos posteriormente.
- dist, donde se almacenan los ejecutables.

3.2. Compilando y ejecutando

Ahora ya podemos compilar, para ello tenemos dos opciones: nos podemos ir a la ventana Project y sobre el nombre del proyecto pulsar con el botón derecho del ratón la opción `Clean and Build` o, por el contrario irnos al menu de la parte superior y pulsar el icono con el martillo y la

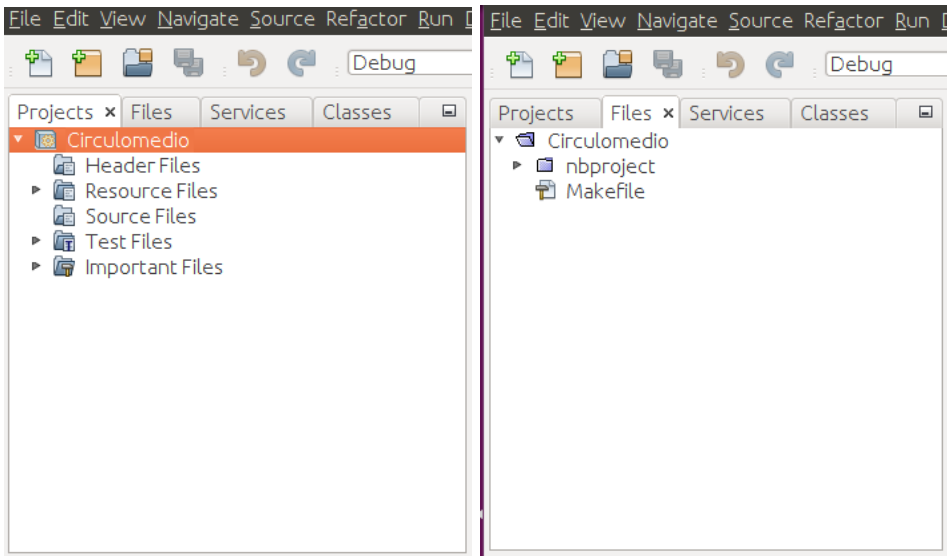


Figura 3: Árbol lógico (izquierda) y físico (derecha) de un proyecto nuevo

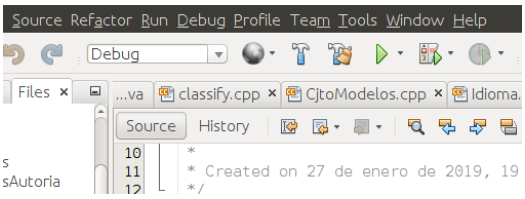


Figura 4: Opción de Compilación Debug/Release

escoba (clean and build), ver Fig 4. Los resultados de la compilación nos lo deja en una nueva ventana `Clean,Build)` de la interfaz.

Una vez compilado, podemos ejecutarlo, al igual que antes tenemos dos opciones, desde la pestaña del proyecto (seleccionar Run) o desde los iconos del menú utilizando el triángulo. Al ejecutarlo, encontraremos una nueva ventana `Run` con la salida de nuestra aplicación, ver Fig 6.

Ejercicio 1 *Crear un proyecto Netbeans que calcule la media de un array de 10 enteros.*

Ejercicio 2 *Crear un proyecto Netbeans que calcule el mayor valor de un array de 10 enteros.*

Ejercicio 3 *Relieve. (El código del mismo lo podemos encontrar en el archivo* `relieve.cpp`*). Debes crear un nuevo proyecto, llamado* `relieve`*, y copiar el código en el fichero main.cpp del proyecto (en la Sección 7 veremos que esto se puede automatizar).*

El código es una solución al siguiente ejercicio del examen ordinario de Fundamentos de Programación:

El relieve de una región geográfica se puede representar mediante una tabla rectangular `alt`*, donde cada elemento* `alt[fil][col]` *representa la altura (sobre el nivel del mar), de la parcela* `(fil,col)` *del terreno (*`fil` *y* `col` *son enteros). Una parcela se define como "pico" si sus 8 parcelas vecinas tienen una altura menor. Por simplicidad, se supone que las*

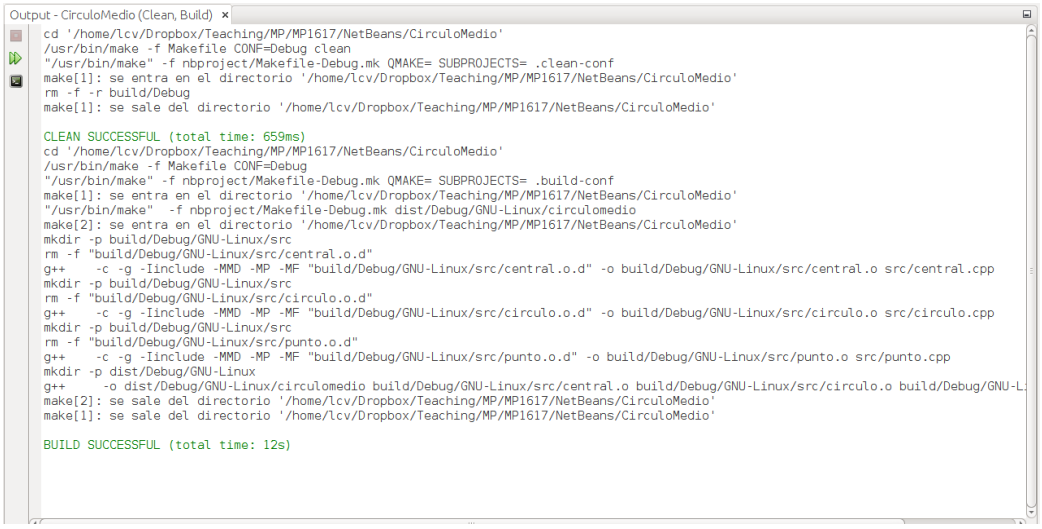


Figura 5: Generando el binario

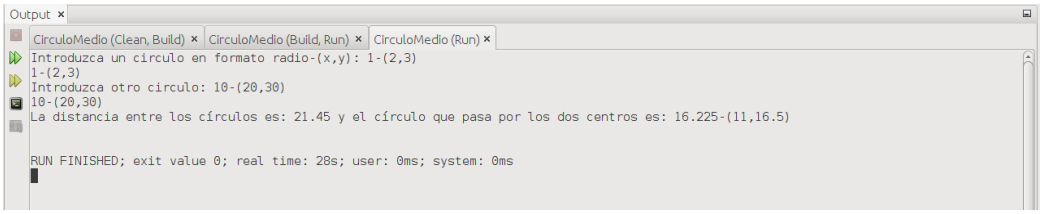


Figura 6: Ejecutando el binario

parcelas de los bordes no pueden ser picos. Se propone la representación para la clase `Relieve` *mostrada en la tabla 1. Remarcar que* **todos los objetos de esta clase estarán registrados respecto a la coordenada** $(0, 0)$.

Relieve
- static const int NUM_FILAS = 100 - static const int NUM_COLS = 100 - int alt[NUM_FILAS][NUM_COLS] - int filas_utilizadas, cols_utilizadas
+ Relieve(int num_filas, int num_cols) + int FilasUtilizadas() + int ColumnasUtilizadas() + int Altura(int fil, int col) + void SetAltura(int fil, int col, int altura) + void Pinta( )

Cuadro 1: Propuesta para la `clase Relieve`

Implemente los siguientes métodos:

- `EsPico`*: Devuelve* `true` *si la parcela situada en las coordenadas* `(fil,col)`*, es un pico.*
- `ObtenerPicos`*: Construye y devuelve un objeto de la* `clase SecuenciaPuntos`

con las coordenadas de las parcelas que son picos.

- `Fusion`*: Involucra dos objetos de la clase* `Relieve`*. Construye y devuelve un nuevo objeto con:*
  - *El número de filas/columnas útiles del nuevo objeto será el mínimo entre el número de filas/columnas útiles de los objetos involucrados.*
  - *Cada parcela* `(fil,col)` *del nuevo objeto tendrá como altura el valor máximo entre las alturas de dicha parcela en los objetos involucrados.*

Para implementar estos métodos, puede incorporar los métodos adicionales que considere oportunos. Si lo hace, justifique si deben ser públicos o privados. Además, considere que dispone de la implementación ya terminada de las clases `Punto2D` *y* `SecuenciaPuntos`*, mostradas en la tabla 2.*

Punto2D
- int x - int y
+ Punto2D() + Punto2D(int abscisa, int ordenada) + int Abscisa() + int Ordenada() + double Distancia(Punto2D otro) + void Pinta( )
SecuenciaPuntos
- static const int TAMANIO = 100 - Punto2D vector_privado[TAMANIO] - int total_utilizados
+ SecuenciaPuntos() + int Capacidad() + int TotalUtilizados() + void Aniade(Punto2D pto) + Punto2D Elemento(int indice) + void Pinta( )

Cuadro 2: Clases disponibles y métodos que **NO hace falta implementar**.

Suponga que en `main` *dispone de objetos* `r1` *y* `r2` *ya creados de la clase* `Relieve`*. Escriba el código necesario para:*

1. *Mostrar las coordenadas y la altura del pico más alto de* `r1`*.*
2. *Calcular la longitud de cable requerida para unir los picos de* `r1` *(orden: el dado por* `ObtenerPicos`*). Debe tener en cuenta la altura de los picos.*
3. *Mostrar las coordenadas de los picos del relieve que se obtendría al fusionar los objetos* `r1` *y* `r2`*.*

4. Documentando un programa

Es de todos bien conocido la necesidad de documentar un código software. Una buena forma de hacerlo es sobre el mismo fichero fuente puesto que, de esta manera, la documentación se tiene siempre a la vista y se puede modificarse fácilmente al tiempo que se modifica el código fuente a lo largo del ciclo de vida del mismo.

Una documentación correcta de un programa es algo más que poner comentarios según el criterio del programador. Todo lo contrario, es necesario seguir ciertas normas o reglas, que nos vendrán impuestas por aplicaciones específicas para la gestión de la documentación de un proyecto. Existen distintas alternativas, pero nosotros nos decantaremos por Doxygen, ya que de forma automática se puede integrar dentro del entorno de desarrollo Netbeans. La documentación de Doxygen se introduce en comentarios estándar de C++ y se identifican como documentación de Doxygen porque comienzan por `/** <los comentarios> */`

Hay múltiples comandos que su pueden utilizar para comentar el código, un listado completo lo podéis encontrar en `http://www.doxygen.nl/manual/commands.html`.

De forma simple, y como norma general debemos incluir

```
/**
Documentación del fichero, usando las órdenes
referentes a fichero

Documentación larga referente al fichero */

#include<los_necesarios>


/**
Documentación de las clases, si existen,
usando las órdenes correspondientes

Documentación larga referente a la clase */
class punto2D{
public:

/**
Documentación de los métodos, usando las órdenes
correspondientes

Documentación larga referente al método */
punto2D(){...};
...
};
```

En el fichero `relieve.txt` podéis encontrar un ejemplo (debeís completarlo) de cómo se puede documentar un fichero c++.

4.1. Documentación general del fichero

La documentación general de un fichero irá al comienzo del mismo, indicando el propósito general del mismo, se debe utilizar el siguiente comando:

`@file [<nombre-fichero>]`

Indica que el bloque comentado contiene documentación de un fichero con el nombre nombre-fichero. Si se omite el nombre de fichero se entiende que la documentación se refiere al fichero actual. IMPORTANTE: Si no se especifica `@file` al comienzo de un fichero, en la salida no se incluirá la documentación de las funciones, estructuras, clases, constantes globales, etc. de ese fichero.

4.2. Documentación general de una clase

Irá justo antes de la declaración de la clase, en este caso el formato es más libre

`@class [<nombre-clase>]`

Indica que el bloque comentado contiene documentación de una clase

Por ejemplo:

```
/** @class Test

descripcion detallada ....
*/
```

4.3. Documentación de un método o función

4.3.1. Comandos básicos

El texto que necesita la mayoría de estas órdenes continúa hasta que se encuentra una línea en blanco o aparece alguna otra orden de Doxygen.

`@brief {descripción breve}`

Comienza un párrafo que sirve como una descripción breve del cometido del elemento documentado. Se puede omitir en un bloque de documentación si la descripción breve es la primera línea del bloque y ocupa sólo una línea.

`@param <nombre-parametro> {descripción del parámetro}`

Comienza la descripción de un parámetro llamado `nombre-parametro` que pertenece a un función. La existencia o no existencia del parámetro no se comprueba.

`@return {descripción del valor de retorno}`

Se usa para describir el valor de retorno de una función de forma genérica. Por ejemplo:

```
@return la posición del máximo en el vector
```

`@retval <valor de retorno> {descripción}`

Se describe un valor de retorno concreto de una función. Por ejemplo:

```
@retval true si el número es primo
@retval false si el número no es primo
```

`@pre {descripción}`

Se describe las precondiciones que se deben cumplir para llamar al método. Por ejemplo:

```
@pre el parámetro n debe ser positivo
```

`@post {descripción}`

Se describe las postcondiciones que se cumplirán tras llamar al método. Por ejemplo:

```
@post se aumenta en 1 la longitud
```

Una vez que tenemos comentados los métodos de una clase, la propia herramienta Netbeans, nos proporcionará la ayuda que hemos descrito cuando tratamos de llamar al método. Esto nos permitirá de forma rápida poder conocer cual va a ser el comportamiento del método, aunque lo hubiésemos implementado hace tiempo. Para ello, basta con escribir sobre el editor el nombre del objeto y el punto y nos muestra documentadas las posibles opciones.

Ejercicio 4 *Documenta brevemente las clases SecuenciaPuntos y Relieve, junto a sus métodos.*

5. Otras características de NetBeans

Además de estas caraceterísticas, 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 15). 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 8). 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 9). 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 10). 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.

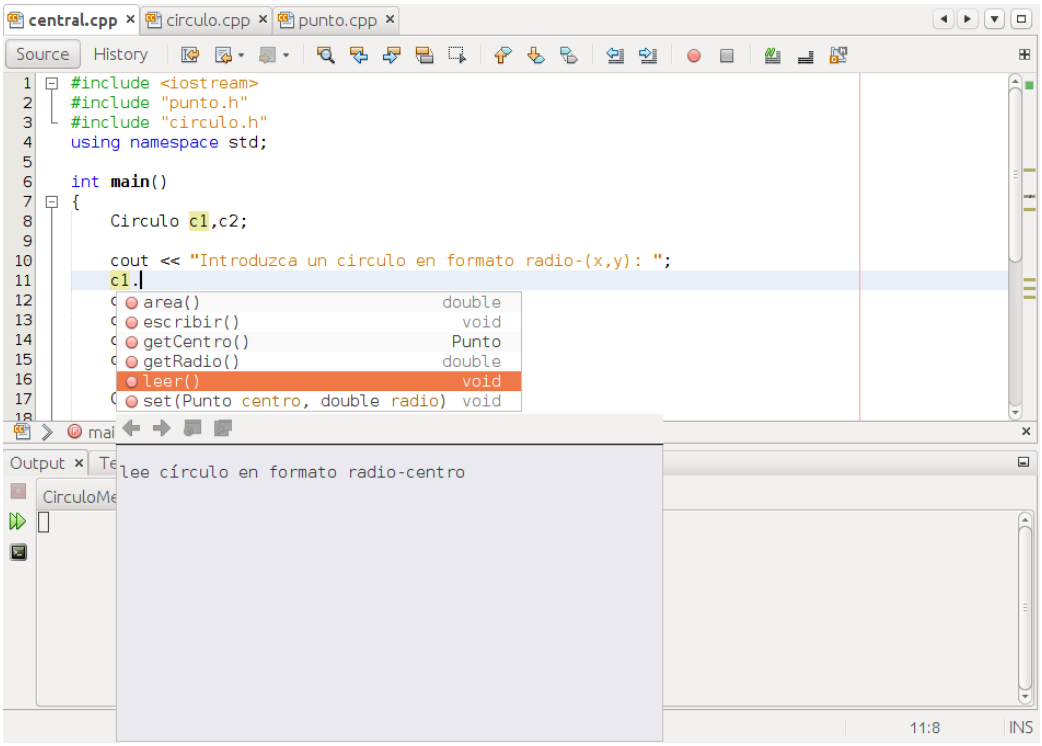


Figura 7: Auto-rellenado de código

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

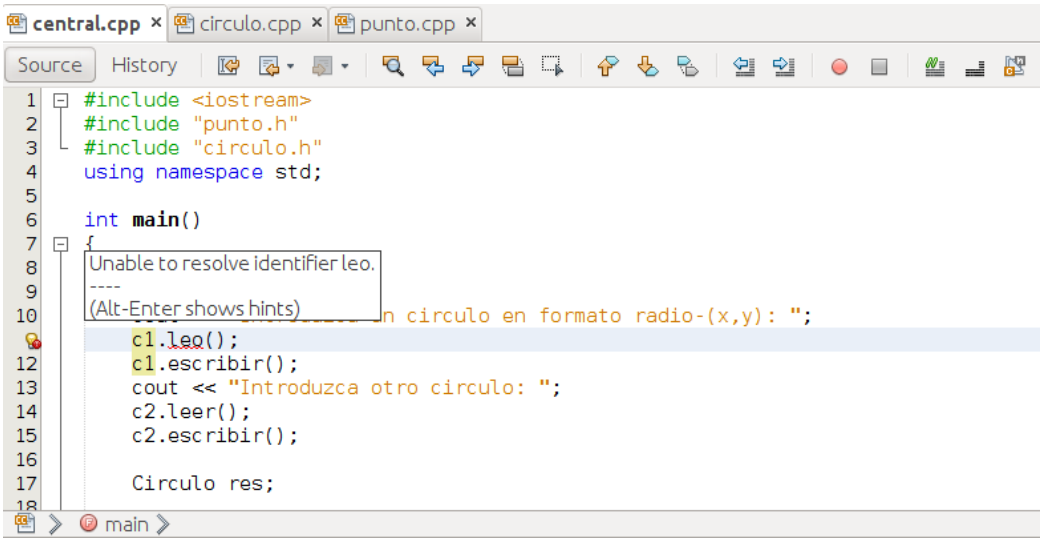


Figura 8: Detección inmediata de llamadas erróneas

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.

El entorno de depuración incorporado nos permite recorrer nuestro código paso a paso y chequear aspectos de la aplicación que se está ejecutando, tales como valores de variables, el estado de la pila, la creación de objetos, etc. Utilizando el depurador nos ahorramos ensuciar nuestro código (la salida del mismo) con incómodos `cout` para visualizar el estado de nuestras variables. Es más cómodo, podemos indicar en que línea de nuestro código queremos que se detenga la ejecución, puntos de ruptura PR[4] (breakpoint) y evaluar el estado de todas las variables que nos puedan interesar en ese momento justo. Estos PR forman parte del entorno Netbeans y no de nuestro código.

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. 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 11). Una vez colocado este punto de ruptura se puede comenzar la ejecución del programa con el menú[5]

`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

[4]Un punto de ruptura 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

[5]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.

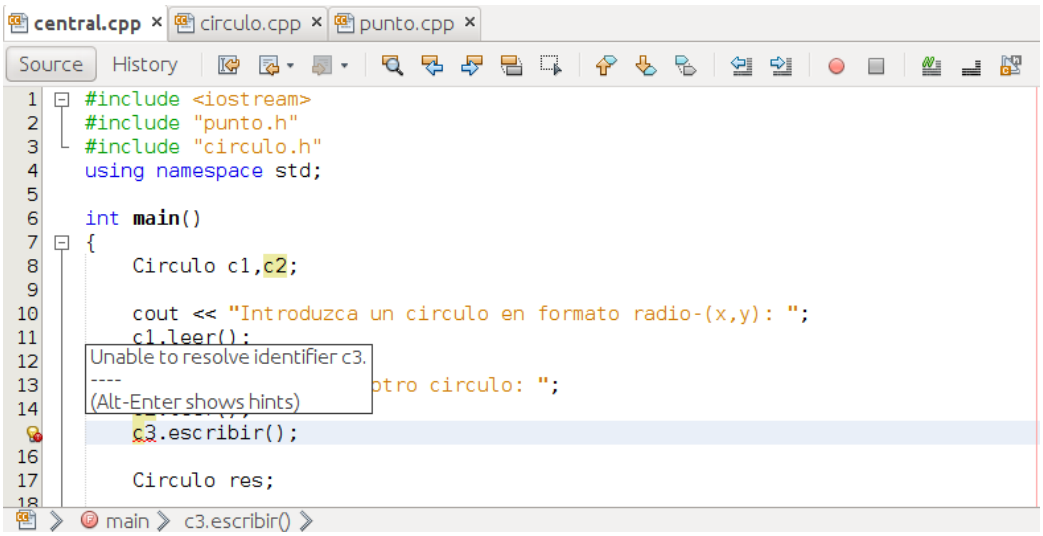


Figura 9: Detección inmediata del uso de identificadores erróneos

(Figura 12). 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.
- `Debug - Run to cursor`
  Ejecuta el programa hasta alcanzar la línea de código en la que se encuentra el cursor.

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 13).

Ejercicio 5 *Copiar este código dentro del un nuevo proyecto y utilizar el depurador para tratar de ver donde está el error. El objetivo del programa es contar en número de veces que aparece un determinado valor leido desde teclado en un array de 10 enteros;*

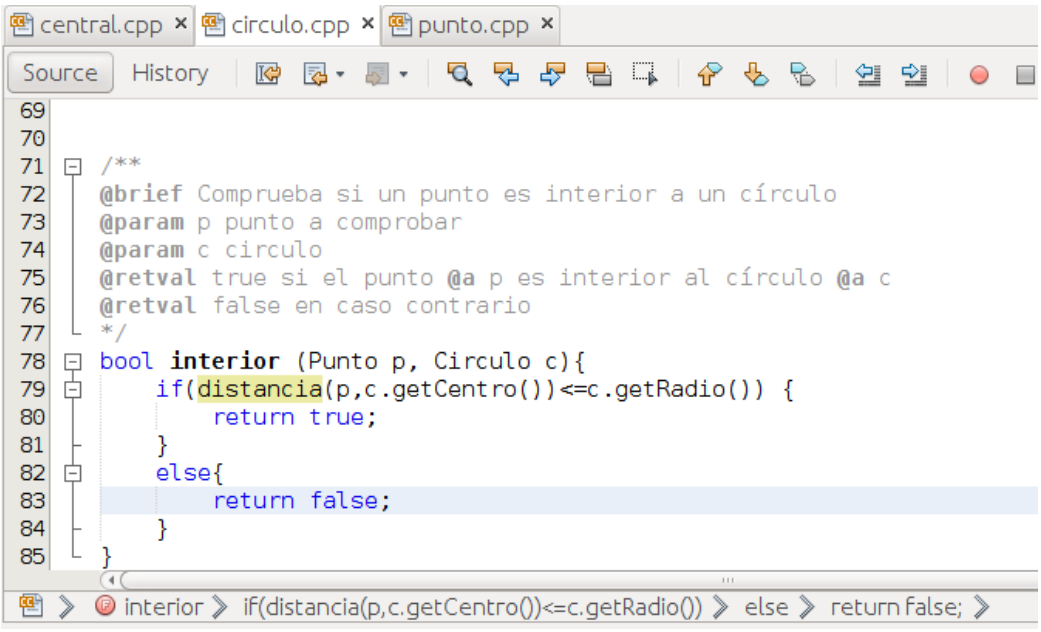


Figura 10: Detección inmediata del nivel de anidación del código. En la parte inferior del editor se puede observar la localización precisa de la línea que se está editando (línea número 83): Función `interior`, dentro de un `if` y en la parte `else` del mismo

```
int entero[10]={1,2,3,4,5,0,0,8,9,0};
int contador,valor;

    cout << "Introduce valor: ";
    cin >> valor;
    for (int i = 0;  i<20; i++){
        if (entero[i]=valor)
            contador++;
    }
    cout << contador << endl;
    if (contador==0){
        cout << "No hay ningun " << valor << endl;
    } else cout<< "Hay "<< contador << " veces " << valor<<
  endl;
```

Ejercicio 6 *Copiar este código dentro del un nuevo proyecto y utilizar el depurador para tratar de ver donde está el error.*

El objetivo del programa es intentar adivinar un número que piense la máquina. Nos dice cuantos intentos hemos realizado. Una posible salida del programa, suponiendo que el número a adivinar es 96, sería la siguiente:

```
introduce numero entre 1 y 100 :12
es mayor
introduce numero entre 13 y 100 :50
es mayor
introduce numero entre 51 y 100 :75
```

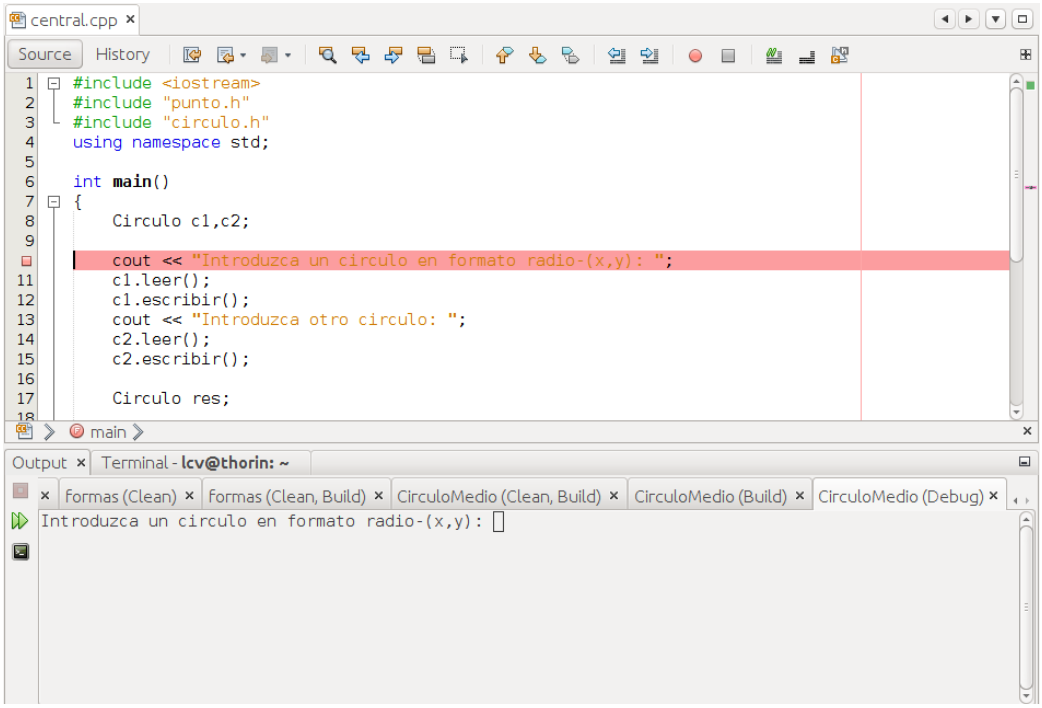


Figura 11: Creando un punto de ruptura para depurar un programa. El PR aparece como una línea de color rojo

```
es mayor
introduce numero entre 76 y 100 :80
es mayor
introduce numero entre 81 y 100 :90
es mayor
introduce numero entre 91 y 100 :95
es mayor
introduce numero entre 96 y 100 :98
es menor
introduce numero entre 96 y 97 :97
es menor
introduce numero entre 96 y 96 :96
lo adivinaste en 9 intentos
```

El objetivo del ejercicio es utilizar el depurador y ver cómo se modifica el estado de las variables.

```
#include <iostream>
#include <cstdlib>

using namespace std;

int main(){
    int x = rand()%100+1;  // Genera número entre 1 y 100
    int inf, sup,n,pasos;
    bool fin = false;
    inf =1; sup = 100;
    do {
        cout << "introduce numero entre "<< inf << " y "<< sup<<
    " :";
```

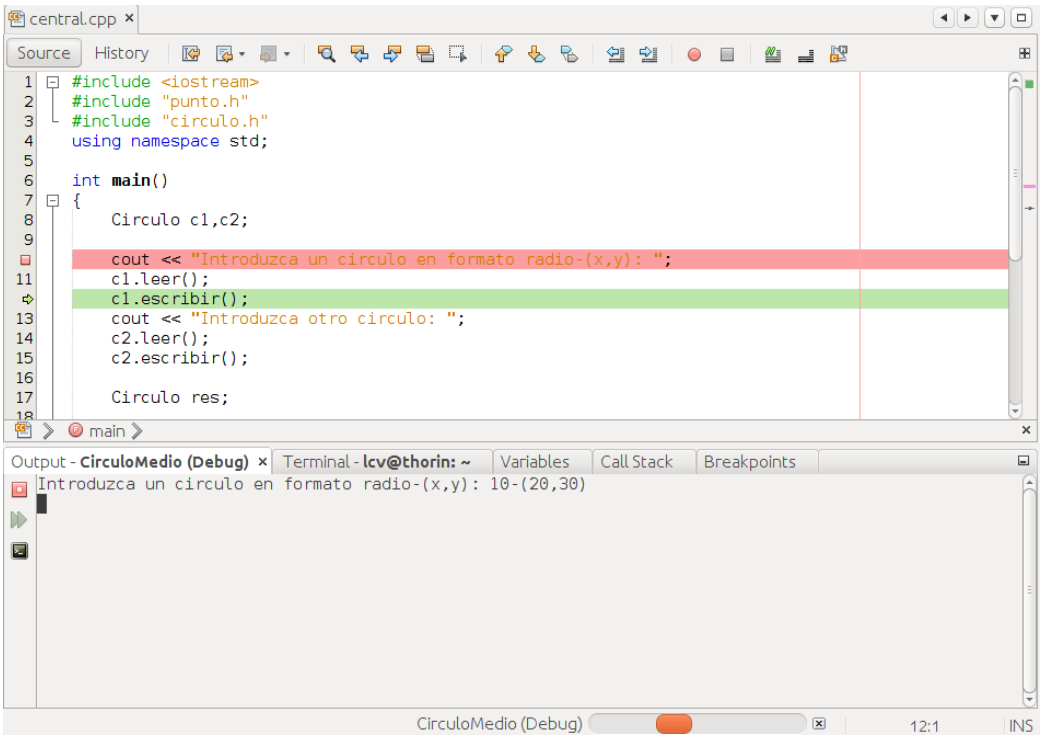


Figura 12: Ejecutando un programa paso a paso. La línea que se va a ejecutar a continuación aparece marcada en color verde

```
        cin >> n;
        cout << endl;
        if (n==x) {
            cout << "lo adivinaste en "<< pasos<< " intentos";
            fin = true;
        }
        else if (n<x) {
            cout << "es mayor";
            inf = n+1;
        } else {
            cout << "es menor";
        }
    }while (fin==false);

}
```

6.4. Punto de ruptura condicional

Una utilidad muy interesante en Netbeans, y en cualquier depurador, es utilizar lo que se de denominan puntos de ruptura condicionales que permite detener la ejecución del programa si se cumple una condición, como por ejemplo que se hayan dado 5000 iteraciones de un bucle o que una determinada variable tome un valor concreto, en el que sabemos que el programa no funciona correctamente.

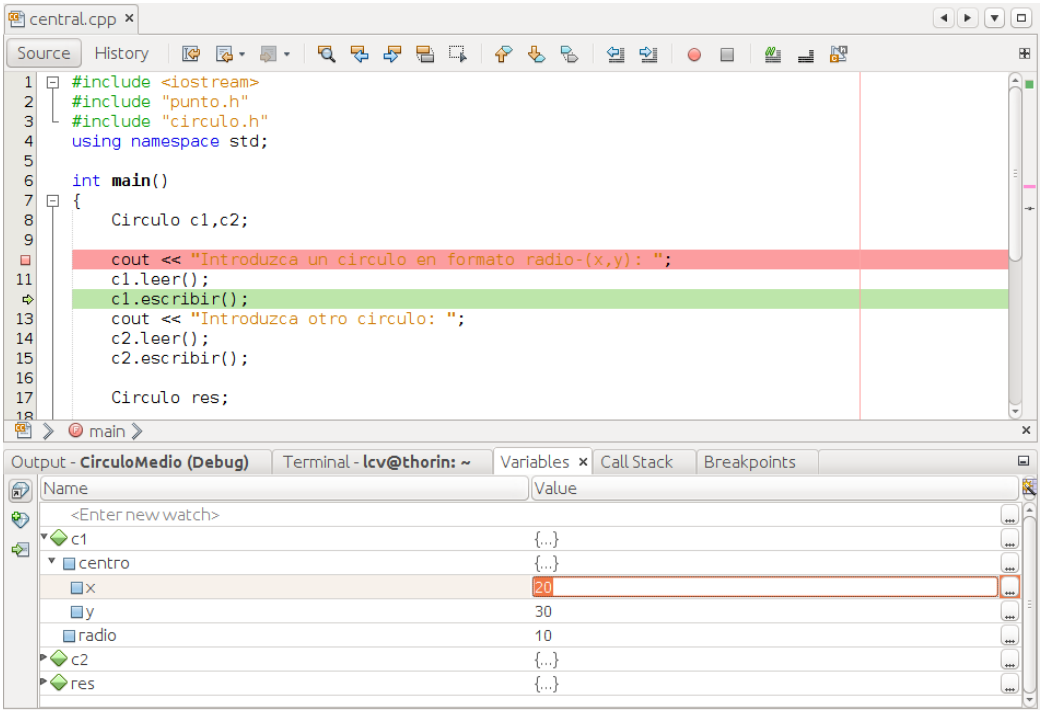


Figura 13: Inspeccionado y modificando, si fuese necesario, las variables del programa

En esta situación el programa se ejecutará normalmente hasta que dicha condición se satisfaga y se detiene en el lugar indicado. Para fijar el PR condicional, primero fijamos un PR normal, nos situamos sobre el y seleccionamos la opción `Properties`, en el menú que nos sale le indicamos la condición de parada, así como la acción Count Limit. Podemos ver que al punto de ruptura le ha salido una pequeña muesca que muestra que la depuración se detiene cuando se satisfaga alguna condición.

Los PR condicionales nos serán muy útiles por lo que os recomiendo hacer distintas pruebas hasta que os sintáis seguros de su funcionamiento.

Ejercicio 7 *Sobre el ejercicio anterior, hacer un punto de ruptura condicional de forma que el programa se detenga cuando el valor de la variable leída desde teclado, n, sea mayor que 50.*

Ejecutar el depurador y asegurarse de que se detiene.

Ejercicio 8 *Sobre el siguiente código*

```
int x[1000], suma=0;
    for (int i=0;i<1000;i++){
        x[i]=rand() %100+1;
        suma+=x[i];
    }
```

Poner un PR condicional sobre la línea `suma+=x[i];` *de forma que el depurador se detenga*

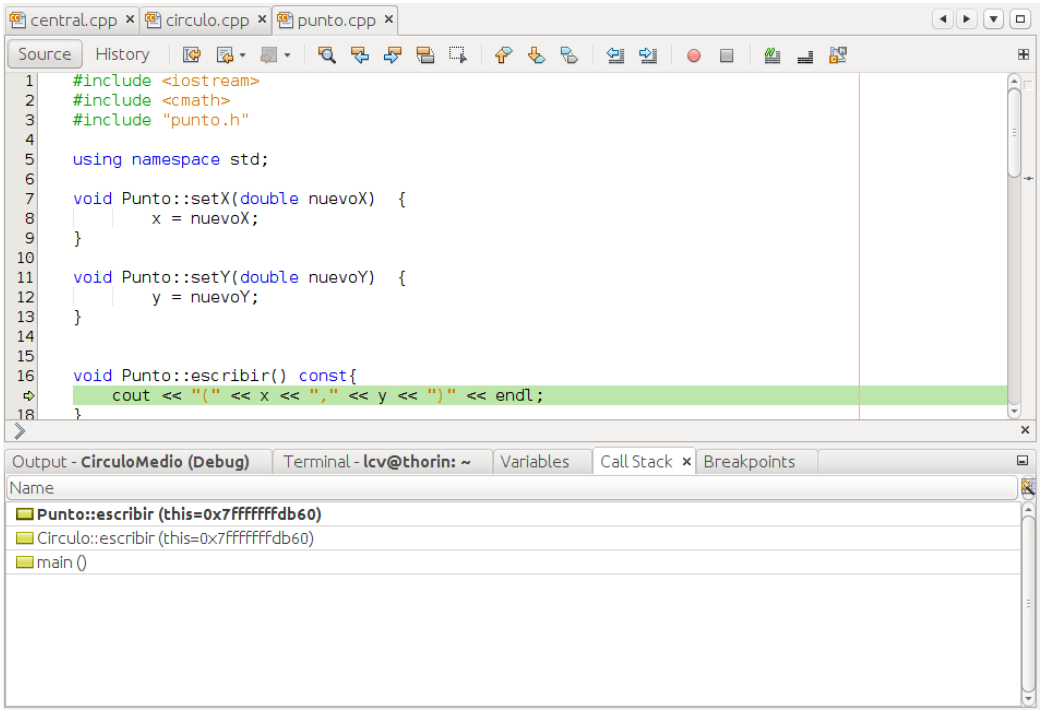


Figura 14: Inspeccionado y modificando, si fuese necesario, las pila de llamadas del programa

- *Después de haber generado 500 datos*
- *El valor de suma sea* > 3000
- *Se cumpla la primera de las dos condiciones anteriores.*

Ejercicio 9 *En el programa relieve hay un problema ya que no es capaz de encontrar bien los picos del relieve. Utilizar el depurador para detectar y corregir el problema.*

6.5. 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 14).

6.6. 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`).

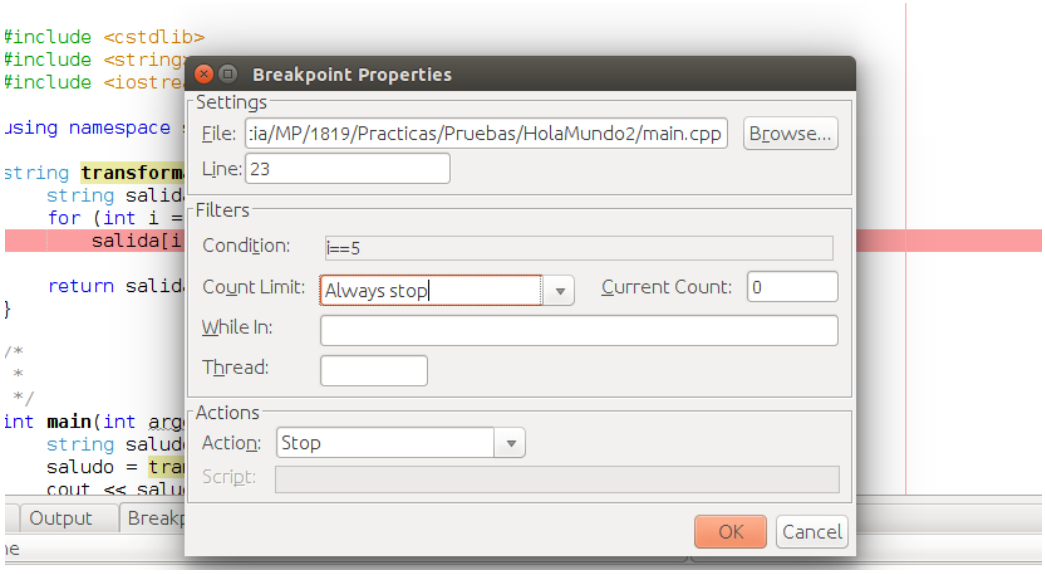


Figura 15: Punto de Ruptura condicional

7. Creación de un proyecto en Netbeans a partir de códigos existentes

7.1. Caso de un único fichero fuente

Hemos visto como podemos crear proyectos de forma simple con Netbeans siguiendo de forma intuitiva los pasos que nos indica el IDE, esta suele ser la forma usual de trabajar. Sin embargo, en muchas circunstancias queremos crear un proyecto a partir de códigos generados por otros usuarios (por ejemplo, los códigos que se os pueden entregar). Si el proyecto original ha sido generado por Netbeans, el proceso es simple y sencillo, pero lo dejaremos para otra sesión de prácticas donde el alumno tenga un mayor dominio sobre el proyectos c++ en general (compilación separada).

Aquí supondremos que tenemos en un directorio un fichero que contiene todo nuestro código. En esta práctica consideraremos la estructura de directorios que se obtiene al descomprimir el fichero `relieve.zip`. En este caso, nuestro fichero fuente se encuentra bajo el subdirectorio `src`, con el nombre `relieve.cpp`.

```
RLV
|-- src
|     |-- relieve.cpp
|
|-- data
      |-- relieve.dat
```

Una vez que tenemos estos ficheros en un directorio, pasamos a generar el proyecto Netbeans siguiendo los siguientes pasos

1. Creamos un nuevo proyecto Netbeans: `File->New Proyect` y seleccionamos la opción New C/C++ Application.

   Damos un nombre al proyecto, en nuestro caso utilizaremos `ProyRelieve`, pero podría ser cualquier otro. Además, seleccionamos la carpeta en la que se ubicará nuestro proyecto, indicándole al Netbeans que queremos utilizar la carpeta `RLV`.

   Como ya tenemos un fichero que contiene la función `main` NO seleccionaremos la opción *create Main File*.

   Como resultado tendremos la siguiente estructura de directorios

```
RLV
|__ ProyRelieve
|      |__ Makefile
|      |__ nbproject
|__ src
|     |__ relieve.cpp
|
|__ data
      |__ relieve.dat
```

2. Es buena idea tener todos los ficheros relacionados dentro del proyecto por lo que vamos a moverlos a dicho directorio

```
mv src ProyRelieve
mv data ProyRelieve
```

   Deberíamos tener esta estructura

```
RLV
|__ ProyRelieve
       |__ Makefile
       |__ nbproject
       |__ src
       |     |__ relieve.cpp
       |
       |__ data
             |__ relieve.dat
```

3. Vamos a indicarle a Netbeans como incorporar los ficheros en nuestro proyecto, y lo mas importante, Netbeans se encargará de importarlos de forma correcta. Para ello, desde la pestaña del proyecto de Netbeans nos ubicamos sobre la carpeta lógica *Sources Files*, pulsamos el botón derecho del ratón y seleccionamos

   `Add existing items from Folders`.

   En la ventana que se nos abre pulsamos el botón *Add Folder* y nos movemos hasta seleccionar nuestro directorio `src`.

   Podemos hacer lo mismo con el directorio data, pero en este caso lo incluiremos en la carpeta lógica *Resource Folders*.

4. Modificamos las propiedades del proyecto. Para ello, de nuevo nos situamos sobre la pestaña del proyecto `ProyRelieve` y con el ratón derecho desplegamos el menú. Abajo del todo aparecerá el enlace a las propiedades del proyecto. Aquí se deben introducir los parámetros que pueden afectar a la compilación/ejecución/depuración.

   - En la sección `Build` seleccionamos la opción *C++ Compiler*. Aquí podremos indicarle donde buscar los includes (por ahora no será necesario), y también modificarle la versión del compilador que queremos utilizar **(seleccionaremos C++11 del desplegable)**.
   - En la sección `Run` podremos modificar los parámetros que se pueden utilizar en tiempo de ejecución. Por ahora, no consideramos ninguno.

Ya tenemos nuestro proyecto configurado y ahora ya podemos compilarlo (Clean and Build), ejecutarlo (Run) o depurarlo (o Debug) de la pestaña del proyecto) o mediante los botones que tenemos en el IDE (estos están asociados al proyecto por defecto, si no fuese el nuestro lo tendríamos que asignar como proyecto por defecto, para ello nos situamos sobre el proyecto y pulsamos sobre el ratón derecho para seleccionar "Set as main project").

Cuando compilamos, `Clean and Build` nuestro proyecto, obtenemos como salida la información asociada a la compilación, si todo es correcto tendremos

```
BUILD SUCCESSFUL (total time: xxxms)
```

Si lo ejecutamos, `Run` la salida será

```
Distancia entre a y b es 6.32456
84,87,78,16,94,36,87,93,50,22,
63,28,91,60,64,27,41,27,73,37,
12,69,68,30,83,31,63,24,68,36,
30,3,23,59,70,68,94,57,12,43,
30,74,22,20,85,38,99,25,16,71,
14,27,92,81,57,74,63,71,97,82,
6,26,85,28,37,6,47,30,14,58,
25,96,83,46,15,68,35,65,44,51,
88,9,77,79,89,85,4,52,55,100,
33,61,77,69,40,13,27,87,95,40,
Picos:
(1,2)(2,4)(4,4)(4,6)(5,2)(5,8)(7,1)(7,7)(8,4)
Pico mas alto (4,6) altura=99
longitud cable 32.8016
84,87,78,16,94,36,87,93,50,22,
63,28,91,60,64,27,41,27,73,37,
24,69,68,30,83,31,63,31,68,36,
36,37,38,59,70,68,94,57,44,45,
48,74,50,51,85,53,99,55,56,71,
60,61,92,81,64,74,66,71,97,82,
72,73,85,75,76,77,78,79,80,81,
```

```
84,96,86,87,88,89,90,91,92,93,
96,97,98,99,100,101,102,103,104,105,
108,109,110,111,112,113,114,115,116,117,
Picos en Fusion:
(1,2)(2,4)(4,4)(4,6)(5,2)(5,8)
RUN FINISHED; exit value 0; real time: 0ms; user: 0ms; system:
   0ms
```

8. A entregar

En esta práctica utilizamos el fichero `relieve.zip` que se encuentra en la plataforma de la asignatura y que hemos utilizado para generar nuestro primer proyecto Netbeans.

El objetivo ha sido lograr tener un proyecto que se pueda ejecutar desde Netbeans. Una vez que lo tengáis, debemos hacerle las siguientes modificaciones:

1. Modificar el main de forma que permita conocer la longitud del cable de los relieves `r1, r2` y `rfusion`. Imprimiremos las tres longitudes y nos quedaremos con la menor.

Una vez modificado el proyecto, debemos subirlo a la plataforma para su evaluación. Subiremos el **proyecto NetBeans** completo, no el fichero fuente para ello consultar las siguientes secciones.

8.1. Crear ZIP a partir de un proyecto Netbeans

Es bastante usual que necesitemos crear una copia del proyecto Netbeans para distribuirla, enviarla de un equipo a otro o en vuestro caso, para subirla a la plataforma. Una alternativa bastante cómoda y que permite posteriormente abrir directamente el proyecto Netbeans es generar un fichero .zip donde se encuentre toda la información del proyecto. Así, desde el directorio raíz del proyecto podemos ejecutar el siguiente comando

```
zip -r  zip/proyecto.zip * -x nbproject/private/*
```

este fichero proyecto.zip es el que debéis subir a la plataforma. Alguno se preguntará ¿lo he hecho bien?. Para encontrar respuesta a esa pregunta lo mas simple es abrir el proyecto desde Netbeans, como se indica en la siguiente sección.

8.2. Abrir un proyecto Netbeans existente

Partimos de un fichero `proyecto.zip` creado en la sección anterior, lo descomprimimos en un nuevo directorio, que llamaremos `NuevoProyecto`

1. Una vez que lo tenemos solo tenemos que abrirlo, esto es, `File->Open Project` e indicarle la localización del mismo, esto es, nuestro directorio `NuevoProyecto`. Ya podemos ejecutarlo, editarlo, etc.