Programación 2
Curso 2013–2014

Guía de desarrollo C/C++ con Eclipse

Introducción

Eclipse[1] es un entorno de programación que permite el desarrollo de aplicaciones en diferentes lenguajes. Consta de un entorno de desarrollo (*IDE*) y de un conjunto de complementos (*plugin*) que permiten extender su funcionalidad. Eclipse es un proyecto de código abierto (*open source*) y, como tal, cualquiera que lo desee puede desarrollar e integrar en él sus propios complementos.

El entorno clásico de Eclipse[2] está preparado para el desarrollo de código Java. Sin embargo, existen numerosos complementos que permiten adaptar el entorno a muchos otros lenguajes de programación, como Ada, C, C++, COBOL, Perl, PHP, Python o Ruby[3]. Para poder utilizar Eclipse como entorno de programación en C/C++ existen dos posibilidades:

- Instalar la plataforma Eclipse clásica para Java y añadir a ésta el complemento CDT[4].
- Instalar el paquete de Eclipse para desarrolladores de C/C++ que ya incluye el citado complemento[5]. Esta es la opción recomendada.

Eclipse nos va a permitir crear, editar, compilar y depurar nuestros programas en C/C++. La instalación de Eclipse no incluye las herramientas para compilar y depurar código, sino que proporciona el entorno de trabajo para integrar dichas herramientas de manera consistente. Por tanto, para poder desarrollar proyectos en C/C++ debemos asegurarnos que tenemos instalado en nuestro sistema un compilador y una herramienta de depuración.

En este curso vamos a utilizar el compilador `g++` y la herramienta de depuración `gdb`, ambas pertenecientes al proyecto GNU. Emplearemos también la herramienta `make` para asistir en el proceso de compilación. Estas tres herramientas vienen habitualmente instaladas en cualquier sistema Linux.

La ventana de trabajo

Cuando se ejecuta Eclipse, lo primero que aparece es una ventana de diálogo (Figura 1) que permite indicar cuál va a ser nuestro espacio de trabajo (*workspace*). El espacio de trabajo es la carpeta donde se almacenará el trabajo que vayamos realizando (proyectos creados, código fuente, ejecutables, etc.).

[1] `http://www.eclipse.org/`
[2] `http://www.eclipse.org/downloads/packages/eclipse-standard-431/keplersr1`
[3] `http://marketplace.eclipse.org/`.
[4] `http://www.eclipse.org/cdt/downloads.php`.
[5] `http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/keplersr1`

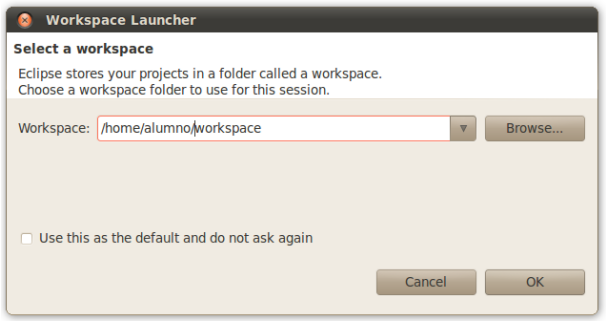

Figura 1: Ventana de diálogo para indicar el espacio de trabajo.

Una vez definido el espacio de trabajo aparece la ventana de trabajo (*workbench*), donde se realiza todo el proceso de desarrollo del programa. La ventana de trabajo de Eclipse contiene una o más perspectivas (*perspectives*). Las perspectivas contienen vistas (*views*) y editores (*editors*), y determinan qué opciones se muestran en ciertos menús y barras de herramientas, adaptando el entorno de Eclipse a la tarea que vayamos a desarrollar. Por ejemplo, existe una perspectiva para editar código en C/C++, otra para depurar programas y otra para trabajar con CVS. Si tenemos instalados complementos en Eclipse para desarrollar programas con distintos lenguajes, tendremos perspectivas diferentes para trabajar con cada uno de ellos. La Figura 2 muestra la perspectiva por defecto para el desarrollo de código en C/C++. Los principales elementos que componen esta perspectiva son los siguientes:

(A) *Barra de herramientas*: contiene accesos directos a las opciones más habituales.

(B) *Cambio de perspectiva*: contiene los botones para cambiar de perspectiva. También se puede hacer a través del menú `Window > Open perspective`.

(C) *Vistas*: son componentes visuales de la ventana de trabajo que ayudan a llevar a cabo determinadas tareas. Una vista puede aparecer por sí sola o junto a otras vistas en una barra de pestañas.

(D) *Editor*: permite introducir y modificar el código fuente.

Las vistas se usan típicamente para navegar por una lista o jerarquía de información (como el conjunto de ficheros y carpetas de nuestro proyecto), o mostrar propiedades para el editor activo. Las vistas más utilizadas en la perspectiva de desarrollo de C/C++ son las siguientes:

- `Project Explorer`: situada a la izquierda de la pantalla, proporciona una visión jerárquica de los recursos (carpetas y ficheros) de nuestro espacio de trabajo.
- `Console`: situada en la parte inferior de la pantalla, muestra la salida del programa y del compilador. También permite la entrada de texto por teclado al programa. Dependiendo del tipo de texto (salida estándar, salida de error o entrada estándar), el color del texto que se muestra es diferente.
- `Outline`. Muestra los elementos del código fuente que estemos editando (clases, variables, funciones, etc.).

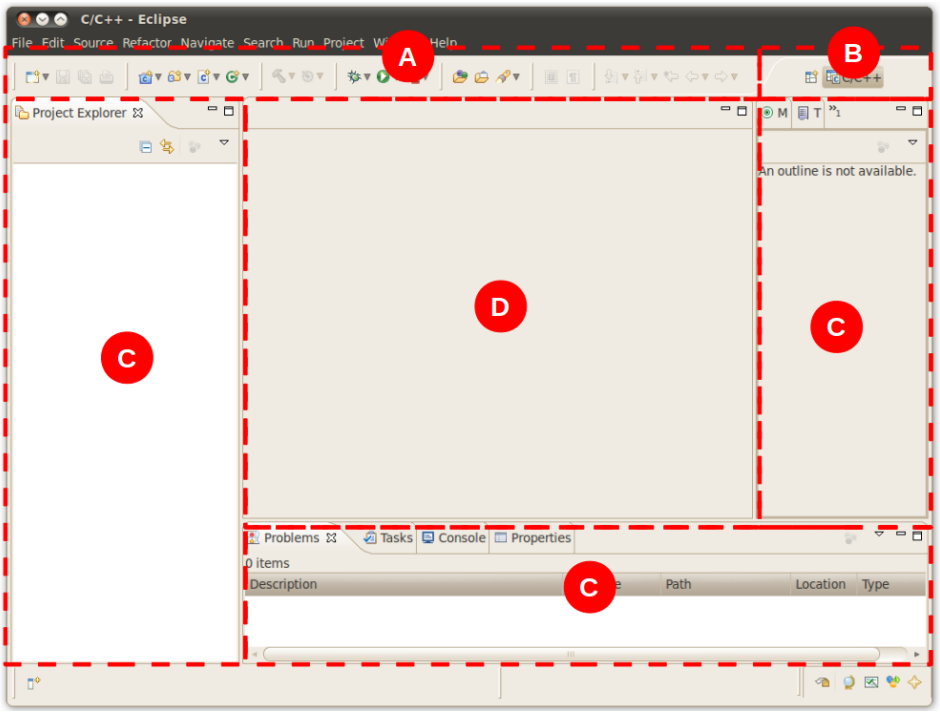

Figura 2: Perspectiva de Eclipse para trabajar en C/C++.

- `Problems`. Muestra los errores generados por el sistema, las advertencias (*warnings*) y la información asociada con un recurso. Esta información la proporciona típicamente el compilador. Si compilamos un programa que contiene errores de sintaxis, éstos se mostrarán automáticamente en esta vista.

Podemos modificar el conjunto de vistas que se muestran en pantalla añadiendo o eliminando en función de nuestras necesidades. Para añadir vistas a una perspectiva elegimos el menú `Window > Show View` y seleccionamos la vista que queramos incorporar. En `Other...` se encuentran las vistas que no están asociadas por defecto a la perspectiva de trabajo actual.

Mientras se trabaja en Eclipse es habitual abrir, mover, redimensionar y cerrar vistas. Podemos volver al aspecto original de la perspectiva seleccionando `Window > Reset Perspective`.

Crear un proyecto

Antes de empezar a escribir el código de nuestra aplicación debemos crear un proyecto para almacenar el código fuente, los binarios y otros ficheros relacionados que necesitemos. Para crear un nuevo proyecto en C++, seleccionamos `File > New > C++ Project`. El asistente de creación de proyectos nos permitirá seleccionar el tipo de proyecto que queremos crear. La Figura 3 muestra el primer paso de este asistente.

Los dos tipos de proyecto (`Project type`) que más nos interesan son `Executable` y `Makefile Project`. En el primer caso, Eclipse generará un fichero makefile

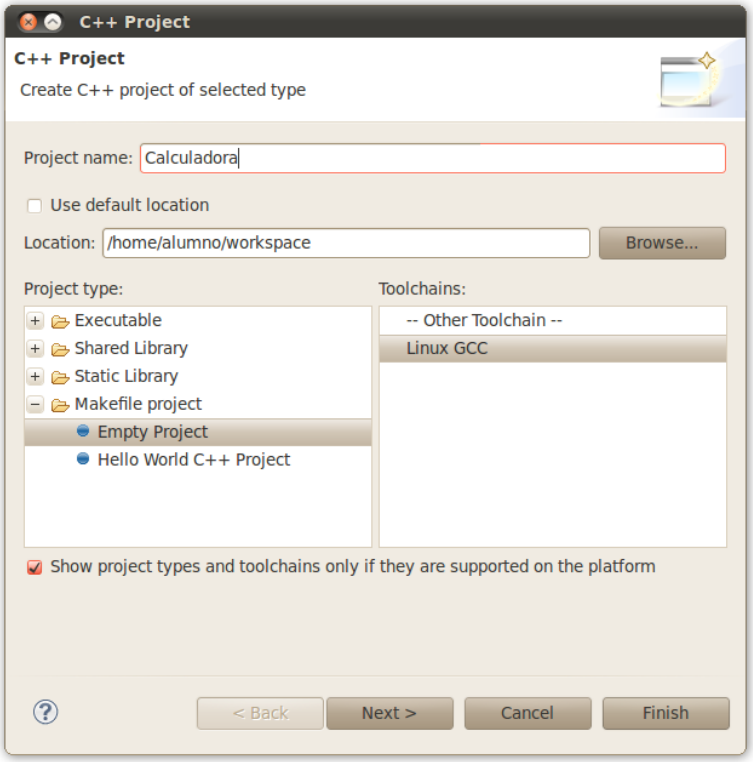

Figura 3: Asistente para la creación de proyectos C++. Primer paso.

automáticamente para construir nuestro programa. En el segundo caso, seremos nosotros los encargados de crear el makefile.

A modo de ejemplo, vamos a desarrollar un pequeño programa que nos permita sumar dos números introducidos por teclado. Para ello crearemos un proyecto llamado `Calculadora`. Los pasos a seguir son los siguientes:

- Introducir el nombre del proyecto (`Project name`).
- Seleccionar como tipo de proyecto `Makefile project > Empty Project` (`Project type`).
- Seleccionar como conjunto de herramientas `Linux GCC` (`Toolchains`)

Haciendo click en `Next >` aparece la pantalla siguiente (ver Figura 4), donde se muestran las configuraciones posibles en función del tipo de proyecto y el conjunto de herramientas seleccionadas. Dejamos las opciones por defecto y finalizamos. Podemos modificar los ajustes por defecto del proyecto a través del botón de funciones avanzadas (`Advanced Settings`). Esta opción abre una ventana que permite cambiar ajustes específicos, como las opciones del compilador o las rutas donde se encuentran los ficheros de cabecera y las librerías.[6]

[6]**Nota**: al crear un proyecto de tipo `Makefile project` nos aparecerá un error en la vista `Problems`, advirtiéndonos de problemas con el makefile. Este error desaparece cuando creamos nuestro makefile y lo incorporamos al proyecto.

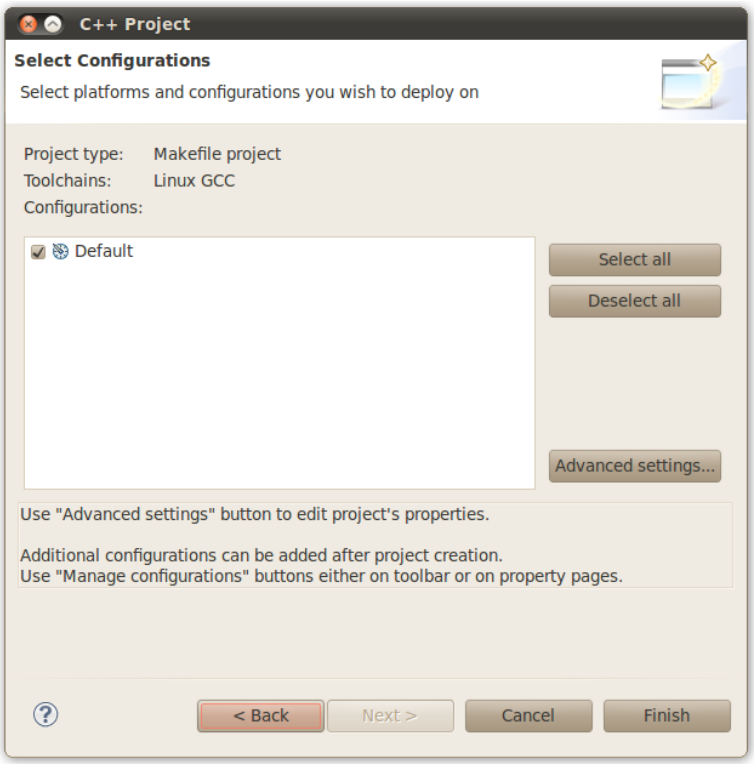

Figura 4: Asistente para la creación de proyectos C++. Segundo paso.

Crear el código fuente

Para crear un nuevo fichero de código fuente seleccionamos `File > New > Source File`. Aparecerá una ventana como la mostrada en la Figura 5.

El código se escribe en el editor C/C++ localizado en el centro de la pantalla. El margen izquierdo del editor contiene la barra de marcadores. Esta barra muestra información sobre marcas (*bookmarks*), puntos de parada (*breakpoints*) y errores y advertencias del compilador. Siguiendo el ejemplo anterior, vamos a crear el fichero de código para nuestra calculadora. Los pasos a seguir son:

- Introducir el nombre del fichero (`Source file`).
- Indicar la carpeta (`Source folder`) donde se guardará el fichero. Por defecto será la carpeta del propio proyecto.
- Como plantilla (`Template`) seleccionar `Default C/C++ Source Template`. Esta plantilla crea un fichero de código vacío con un pequeño comentario inicial con la fecha y el autor del fichero.

A continuación introducimos el código fuente del programa en el editor. Este pequeño programa solicita al usuario que introduzca dos números a través de teclado, devolviendo por pantalla el resultado de su suma:

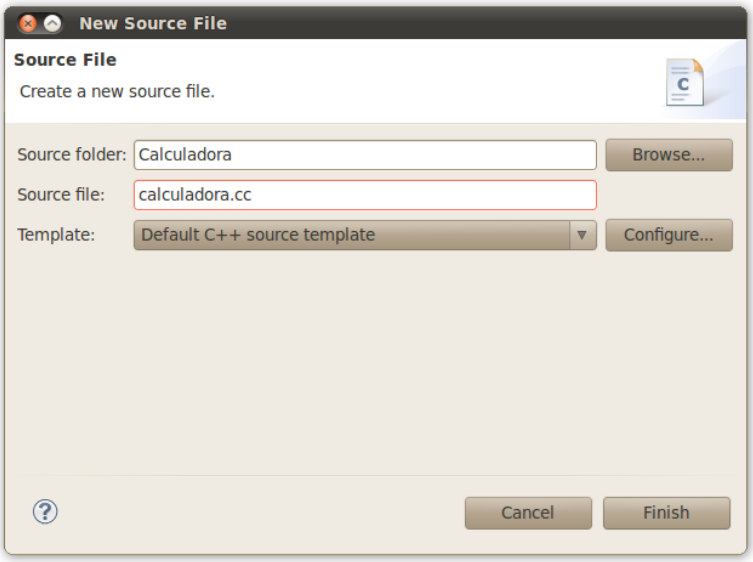

Figura 5: Creación de un fichero de código fuente.

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

int main()
{
  int a=0,b=0,resultado=0;

  cout << "Introduzca el primer sumando: ";
  cin >> a;
  cout << "Introduzca el segundo sumando: ";
  cin >> b;
  resultado=a+b;
  cout << a << " + " << b << " = " << resultado << endl;

  return 0;
}
```

Una vez guardado, el fichero se mostrará en la vista `Project Explorer`. Cuando se crea un fichero dentro de un proyecto, Eclipse crea un registro de todos los cambios llevados a cabo en él. Cada vez que se edita y se guarda un fichero, se crea una copia del mismo. Esto permite comparar la versión actual del fichero con versiones previas, o reemplazar el fichero con una versión anterior. Cada versión se identifica por la fecha y la hora en la que fue guardada. Se puede acceder a esta información a través de la vista de historial (*History View*). Para mostrar esta vista, accedemos al menú `Window > Show View > Other... > Team > History`. Alternativamente se puede hacer click con el botón derecho sobre el fichero en la vista `Project Explorer` y seleccionar `Team > Show Local History`.

Crear un makefile

Antes de poder generar el ejecutable debemos crear un fichero makefile. Para ello, dentro de la vista `Project Explorer` hacemos click con el botón derecho sobre el proyecto actual y seleccionamos la opción `New > File` del menú emergente. Como nombre de fichero (`File name`) escribimos `makefile` y finalizamos. Siguiendo nuestro ejemplo, introducimos el siguiente código para el makefile:

```
all : calculadora
calculadora : calculadora.o
	g++ -Wall -g calculadora.o -o calculadora
calculadora.o : calculadora.cc
	g++ -Wall -g -c calculadora.cc
clean :
	rm calculadora.o
```

El nuevo makefile aparece ahora en la vista `Project Explorer`. La configuración por defecto de Eclipse hace que la herramienta `make` busque un objetivo `all` en el makefile para compilar los ficheros del proyecto. Por esta razón hemos introducido el objetivo `all : calculadora` en nuestro ejemplo. Podemos eliminar esta restricción en el menú `Project > Properties > C/C++ Build > Behaviour`, borrando `all` en `Build (Incremental build)`, tal y como se muestra en la Figura 6. Por motivos similares hay un objetivo `clean` en el ejemplo anterior. Éste se dispara cuando el usuario selecciona la opción `Project > Clean...` que comentaremos en la siguiente sección.

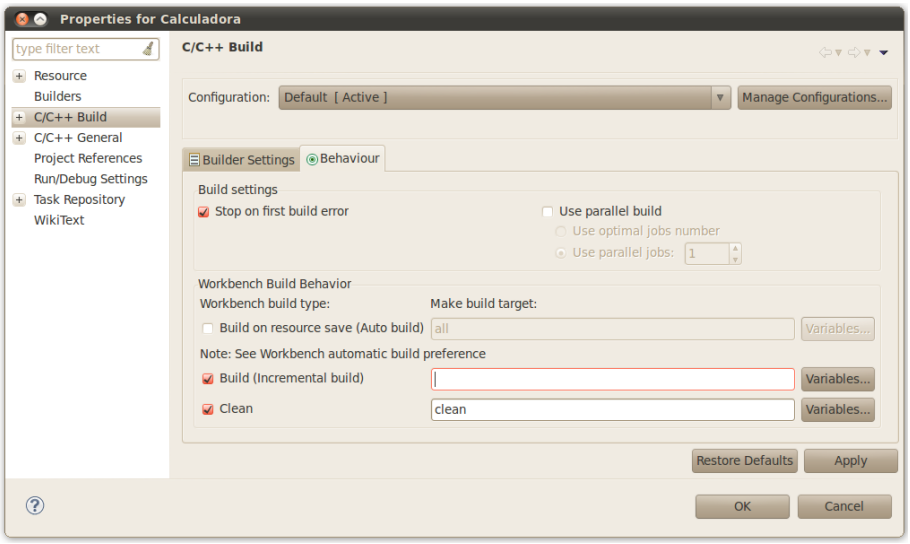

Figura 6: Opciones del compilador.

Una vez definido el código fuente y el fichero makefile, estamos en disposición de construir nuestro programa.

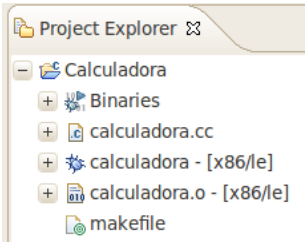

Figura 7: Vista del explorador de proyecto una vez creado el fichero fuente y el makefile.

Construir el programa

Construir el programa implica compilar y enlazar todo el código fuente de nuestra aplicación para obtener un fichero binario ejecutable. Por defecto, la compilación del programa se lleva a cabo de forma automática cuando guardamos los cambios realizados en el fichero fuente. Esta opción puede desactivarse a través del menú `Project > Build Automatically`. Resulta conveniente desactivar esta opción para tener más control sobre la compilación, sobre todo en los casos en los que sabemos que no se debe compilar hasta haber realizado un buen número de cambios. Algunas de las opciones que se comentan más abajo sólo están disponibles cuando la compilación automática está desactivada.

Para realizar una compilación manual del proyecto, seleccionamos `Project > Build Project`. Esto produce una compilación incremental, donde sólo se compilan los recursos que hayan cambiado desde la última vez que se compiló. Para compilar todos los ficheros, incluso aquellos que no hayan cambiado desde la última vez, seleccionamos `Project > Clean...` y luego `Project > Build Project`. Si tenemos varios proyectos en nuestro espacio de trabajo y los queremos compilar todos, debemos seleccionar `Project > Build All`. La vista `Console` mostrará la salida del compilador. Si se produce algún problema, los errores o advertencias se mostrarán en la vista `Problems`.

Ejecutar el programa

Una vez tengamos compilada nuestra aplicación y se haya creado el fichero ejecutable (ver Figura 7), estamos en disposición de ejecutar nuestro programa. Para ello utilizaremos la opción `Run > Run`.

Se pueden realizar diversas modificaciones sobre la configuración de ejecución del programa a través del menú `Run > Run Configurations...`. Una de las modificaciones más interesantes consiste en especificar los argumentos que se quieren pasar al programa por línea de comando. Estos argumentos se pueden introducir en la ventana de configuración (Figura 8), seleccionando la pestaña `Arguments` y escribiendo en la ventana `Program arguments` un argumento en cada línea. La Figura 8 muestra un ejemplo donde se han incluido dos argumentos de entrada al programa (el número entero `7` y el número entero `8`).

Depuración del programa

El depurador (*debugger*) es una herramienta que permite ver qué está ocurriendo dentro de nuestro programa mientras se está ejecutando. Para poder

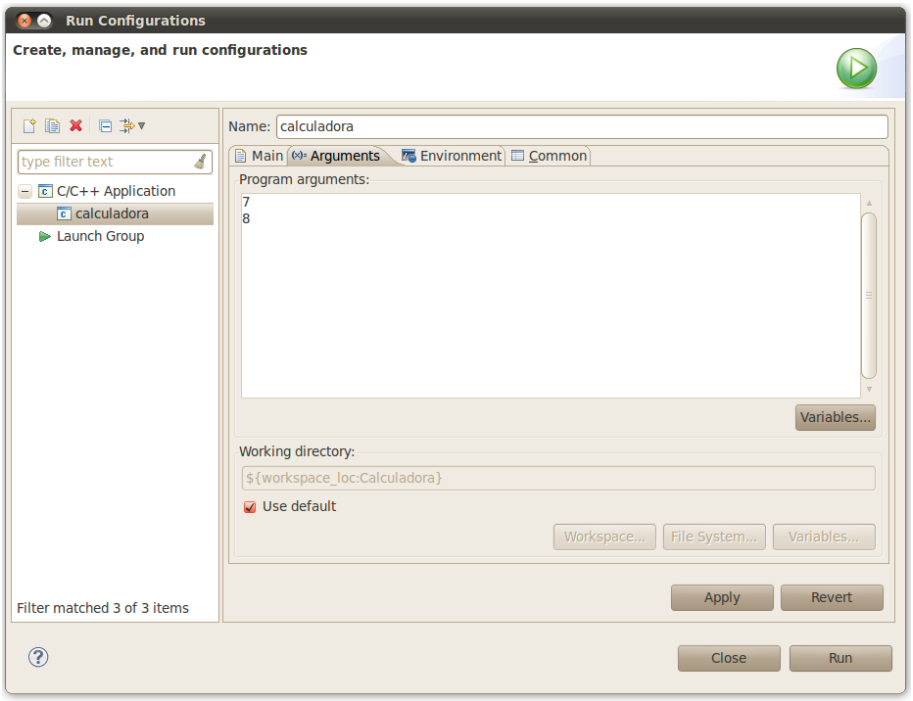

Figura 8: Ventana para la modificación de los argumentos de entrada del programa.

depurar un programa es necesario usar ejecutables que hayan sido compilados para poder ser depurados. Estos ejecutables contienen información adicional de depuración que permiten que el depurador pueda hacer asociaciones directas entre el código fuente y el fichero ejecutable binario generado a partir de éste. Con el compilador `g++`, esto se consigue utilizando la opción `-g` (tal y como se puede ver en el ejemplo de makefile dado más arriba). Como se dijo al inicio de este documento, usaremos la herramienta `gdb` de GNU para depurar nuestro código.[7]

El depurador permite controlar la ejecución del programa estableciendo puntos de parada (*breakpoints*), suspendiendo la ejecución, recorriendo paso a paso el código y examinando el contenido de las variables en un determinado momento.

Para insertar un punto de parada en el código fuente, hacemos doble click en el margen izquierdo del editor de código a la altura de la línea que nos interese inspeccionar (aparecerá una pequeña marca azul). Podemos eliminar un punto de parada haciendo nuevamente doble click sobre él. Adicionalmente, podemos eliminar todos los puntos de parada del código mediante el menú `Run > Remove All Breakpoints`.

Para depurar un programa lo haremos desde el menú `Run > Debug`. Eclipse nos alertará de que este tipo de acción está asociada con la perspectiva de depuración (*Debug perspective*), dando la posibilidad de cambiar a dicha perspectiva. Aceptamos y automáticamente Eclipse alterará los elementos de la ventana

[7] **Nota**: editar el código fuente después de compilar puede causar que la numeración de las líneas varíe con respecto a la información de depuración contenida en el ejecutable. De forma similar, depurar binarios que han sido optimizados puede causar saltos inesperados en la traza de ejecución.

ofreciendo las vistas más adecuadas para el proceso de depuración (ver Figura 9). Podemos volver en cualquier momento a la perspectiva de C/C++ seleccionando `Window > Open Perspective > C/C++` o a través del botón `C/C++ Perspective` en la parte superior derecha de la pantalla.

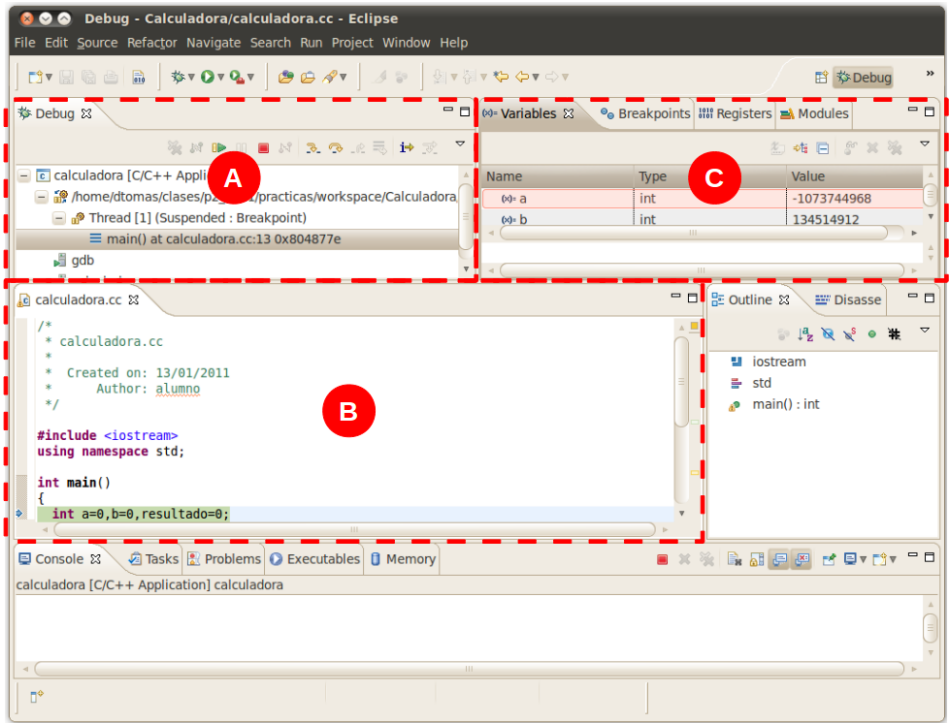

Figura 9: Perspectiva de depuración de código.

Los principales elementos que componen esta perspectiva son:

(A) `Vista de depuración`: permite controlar el proceso de depuración. Tenemos las siguientes opciones:

- `Terminate`: finaliza el proceso de depuración
- `Resume`: continúa la ejecución hasta el siguiente punto de control (o hasta el final del programa, si no hubiera punto de control).
- `Step Into`: ejecuta una instrucción del programa. Si la instrucción llama a una función, la depuración continuará dentro de dicha función.
- `Step Over`: ejecuta una instrucción del programa. A diferencia de la opción anterior, si la instrucción llama a una función, pasa a la instrucción siguiente sin entrar en la función.

(B) *Editor*: muestra el código fuente, resaltando en cada paso de depuración la línea que se está ejecutando.

(C) *Vista de variables*: muestra las variables del programa, su tipo de dato asociado y el valor que poseen en el punto de ejecución actual.

A modo de ejercicio, introduce un punto de parada en la penúltima línea del ejemplo anterior (`cout << a << ...`) y depura el programa utilizando la opción `Resume` para avanzar en la ejecución. Una vez haya terminado la ejecución, repite el proceso utilizando en esta ocasión la opción `Step Into` para avanzar. Observa en ambos casos como van cambiado los valores de las variables del programa (`a`, `b` y `resultado`) en la vista `Variables`.