Universidad de Alcalá

Departamento de Automática

ARQUITECTURA DE REDES
Laboratorio

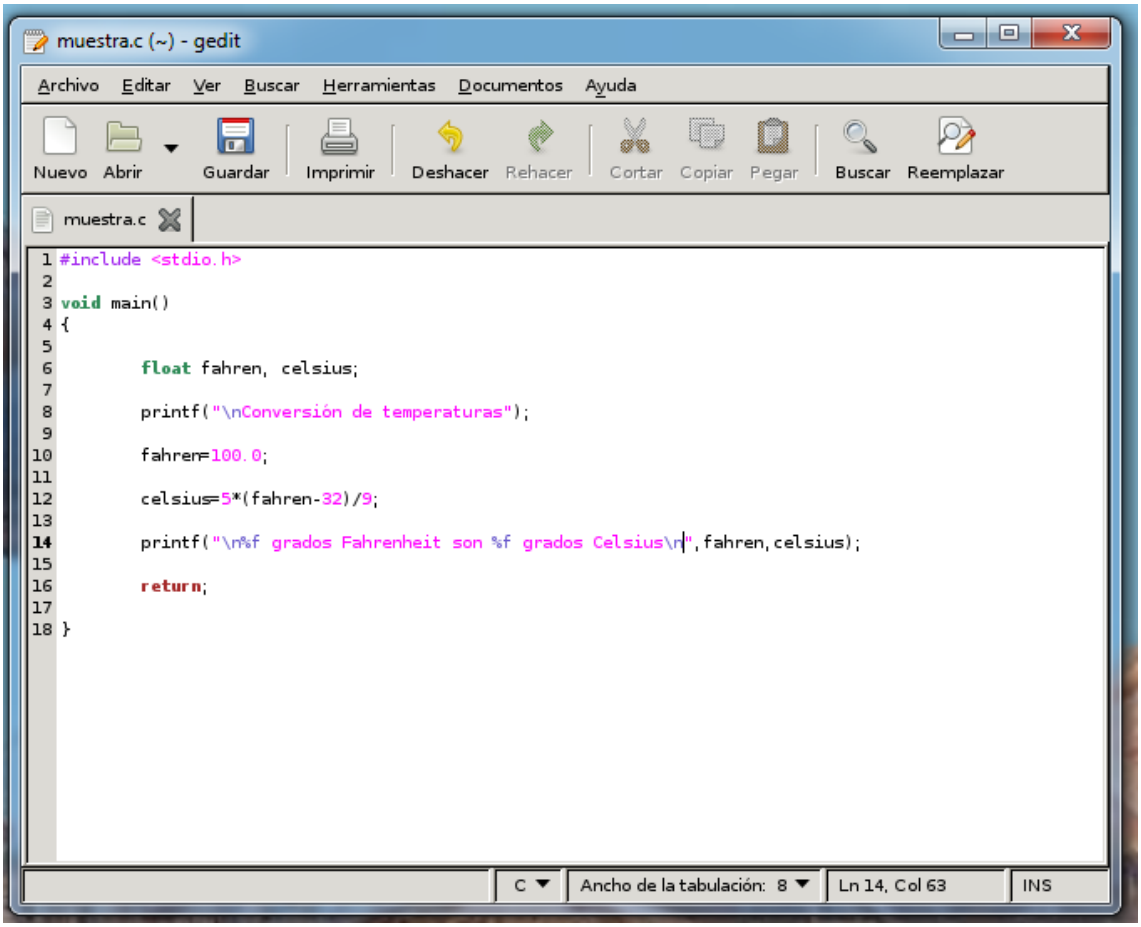
Práctica 1 ANEXO
Introducción a Editor y Depurador.

Grado en Ingeniería Informática

CREACIÓN DE UN PROGRAMA

A la hora de realizar los ejercicios de las prácticas posteriores, la primera labor es el diseño del algoritmo. Una vez se tiene claro el algoritmo, se puede pasar a la creación del programa, que será una mera traslación a código del mismo. En el caso de los ejercicios más sencillos, el programa estará contenido en un único fichero fuente. Para crearlo se puede utilizar cualquier editor de texto sin formato, como `vim` o `emacs`. En el caso de este cuaderno, se va a utilizar un editor gráfico presente en la distribución de Ubuntu que se ha seleccionado llamado `gedit`.

Una vez arrancado el editor, se escribe el programa. Conviene ir salvando el trabajo de vez en cuando para evitar pérdidas si se va la luz o causas similares. El nombre del archivo fuente debe tener la extensión `.c`. En este ejemplo, se usa como nombre `muestra.c`. La siguiente figura muestra el editor de texto con el código del programa de ejemplo.



Una vez escrito el programa, debe compilarse y "*linkarse" (*del verbo inglés *to link*, enlazar, que se utiliza en la terminología inglesa*)*. Se trata de dos operaciones diferentes, pero en el caso de un programa con un único fichero fuente se puede hacer con una única invocación al programa `gcc`. `gcc` es una aplicación que engloba (entre otras cosas) el compilador y el "*linkador*". La forma de invocarlo para este ejemplo es:

```
bash-ln.05$ gcc -g -o muestra muestra.c
```

La opción `-g` le indica que genere información de depuración (ya que, en caso contrario no se genera, y por lo tanto no será posible depurarlo después). La opción `-o nombre` le indica que debe poner el nombre `nombre` al fichero ejecutable resultado de la compilación. En este caso, `muestra`. El último argumento de la línea es el nombre del fichero fuente que hay que compilar.

Si el programa tiene errores de sintaxis, `gcc` indicará las líneas en la que se encuentran y aportará una pequeña descripción del problema. En caso de que no encuentre errores de sintaxis, el resultado será el archivo ejecutable.

DEPURACIÓN. DEPURADOR DDD.

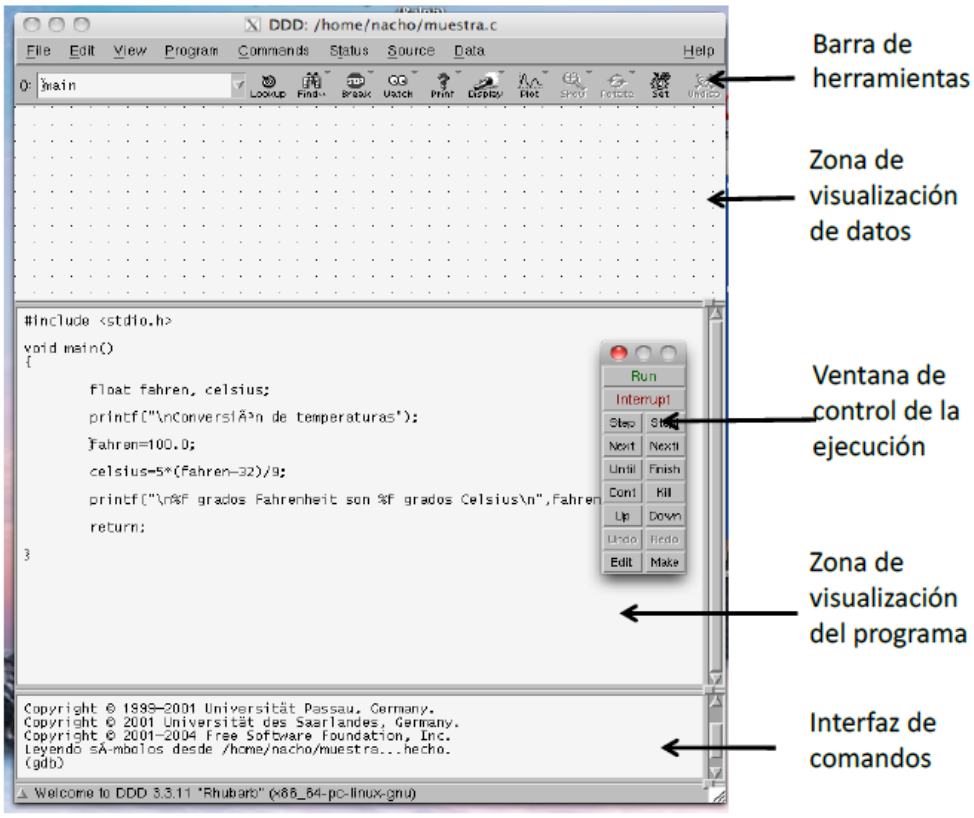
Es bastante frecuente que los programas contengan errores de diseño o de programación, por lo que una herramienta imprescindible es el depurador. El depurador permite ejecutar un programa paso a paso (instrucción a instrucción) y analizar el valor de las variables en cada momento, para ver en qué momento la ejecución se separa de lo previsto. Es pertinente recordar aquí la máxima:

"Un programa hace lo que dices, no lo que quieres".

El depurador que vamos a utilizar se llama Data Display Debugger (o `ddd`). Se puede arrancar desde el escritorio, o desde un interfaz de comandos, tecleando:

```
bash-ln.05$ ddd muestra&
```

La ventana del depurador será parecida a lo que se ve en la siguiente figura:



Una vez cargado el programa, lo primero es ponerlo en marcha. Para ello, se pone un punto de parada (‘*breakpoint’*) en una línea del mismo (por ejemplo, en el primer `printf`) haciendo click en la línea en la que se encuentra, al principio. Eso llevará el cursor a esa posición. A continuación se pulsa el botón ‘*Break’* de la barra de herramientas. Para desactivarlo, basta con volver a pulsar el botón ‘*Break*’. Una vez puesto el punto de parada, se echa a correr el programa pulsando el botón ‘*Run*’ de la ventana de control de la ejecución. El programa se parará en la línea para la que hemos creado un punto de parada.

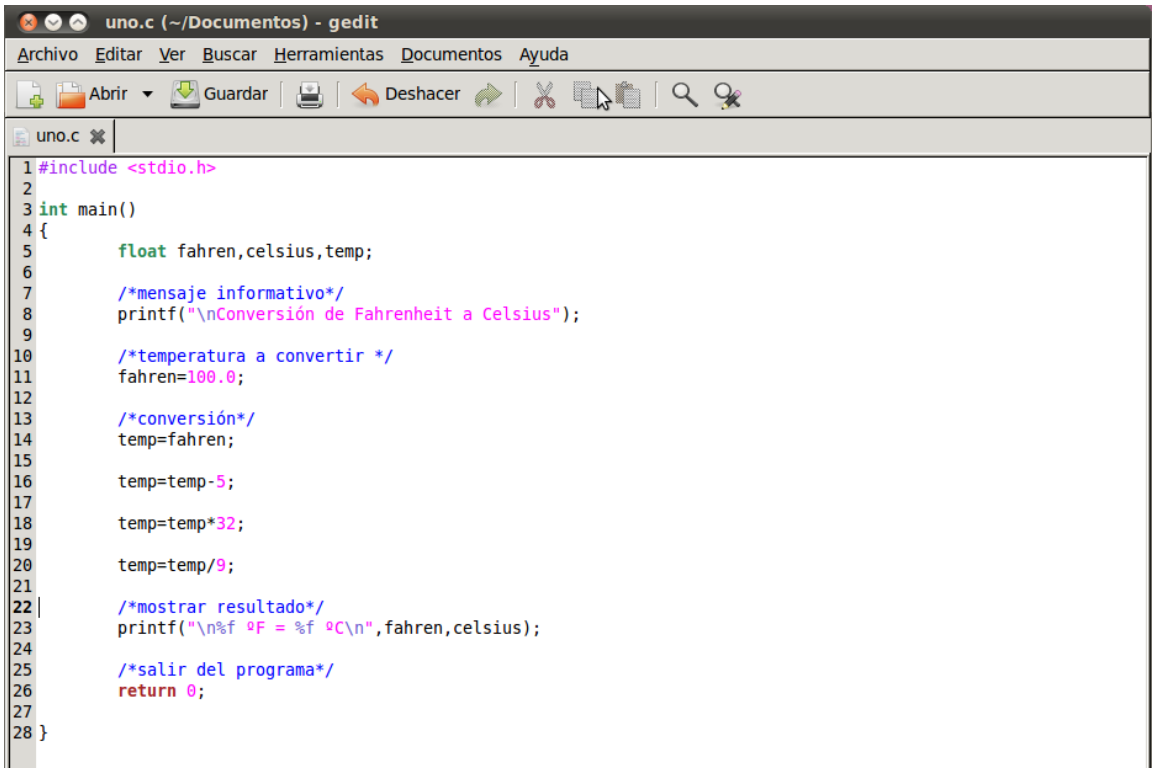
Es posible ver el valor de variables seleccionándolas con el ratón (haciendo click sobre ellas en la zona de visualización de programa) y, a continuación, pulsando el botón ‘*Display*’. Eso hará que aparezcan en la zona de visualización de datos, y se vayan mostrando los valores que van tomando.

Para ejecutar una línea del programa, se pulsa bien el botón ‘*Step*’, bien el botón ‘*Next’*, ambos en la ventana de control de la ejecución. La diferencia entre ambos radica en el tratamiento de las funciones. Cuando se llega a una función, ‘*Next’* ejecuta toda las instrucciones de la función de una vez, es decir, con una única pulsación del botón, mientras que cada pulsación de ‘*Step*’ ejecuta una única instrucción de la función.

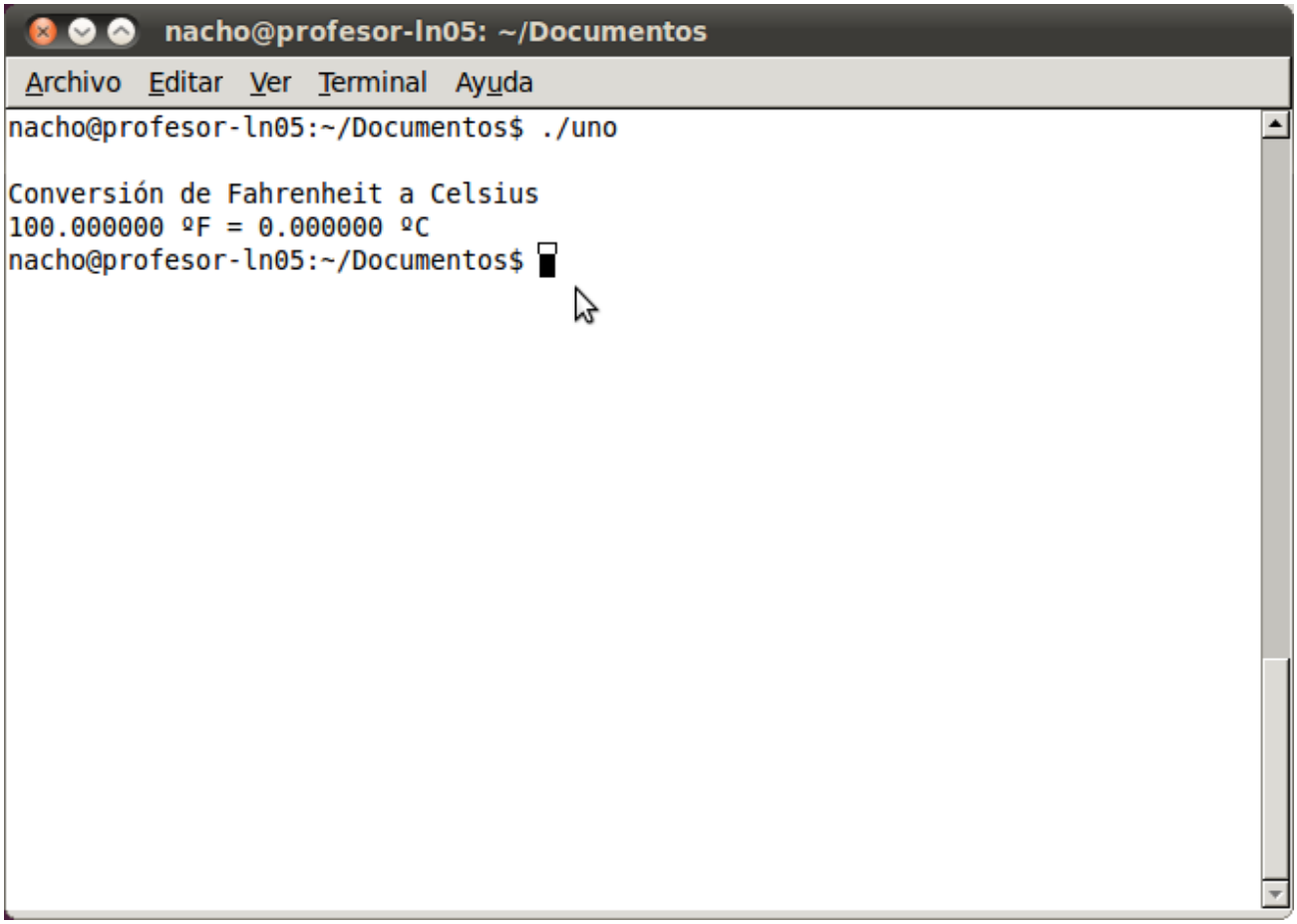
Además, es posible saltarse zonas completas de código simplemente poniendo un punto de parada en la instrucción donde termina la zona que nos queremos saltar, y echando a correr el programa con el botón ‘*Cont’*. La ventana de control de la ejecución tiene otras opciones que es interesante explorar. Para ello, consultar la bibliografía de esta práctica, situada al final de este cuaderno.

Utilización de ddd para depurar un programa.

En esta sección se muestra cómo utilizar `ddd` para encontrar los errores de un programa. El punto de partida es el siguiente programa:

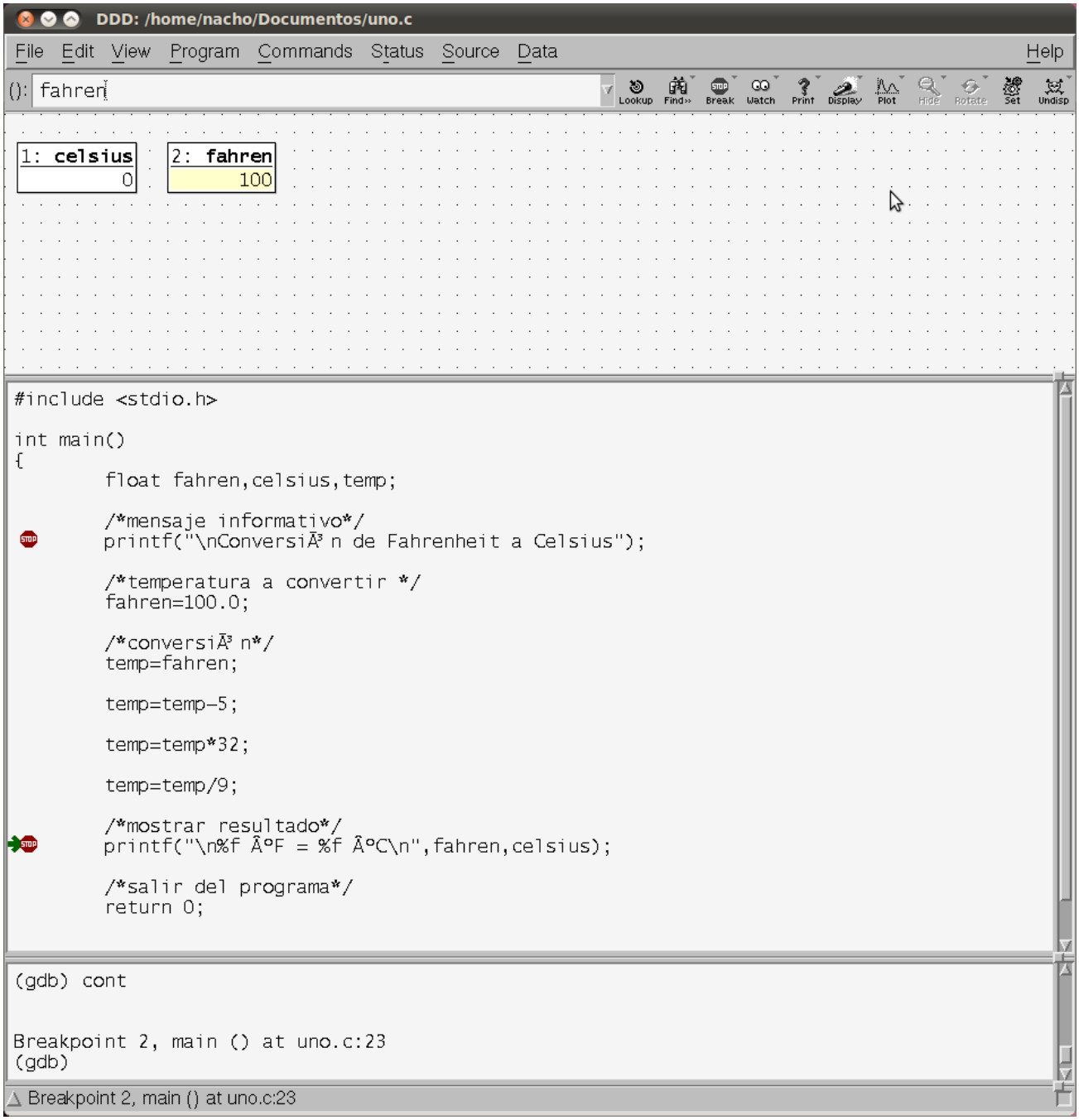


Se pretende que realice la conversión de una temperatura expresada en grados Fahrenheit, contenida en la variable `fahren`, a grados Celsius, y la muestre por pantalla. Si se compila y ejecuta, el resultado es el siguiente:

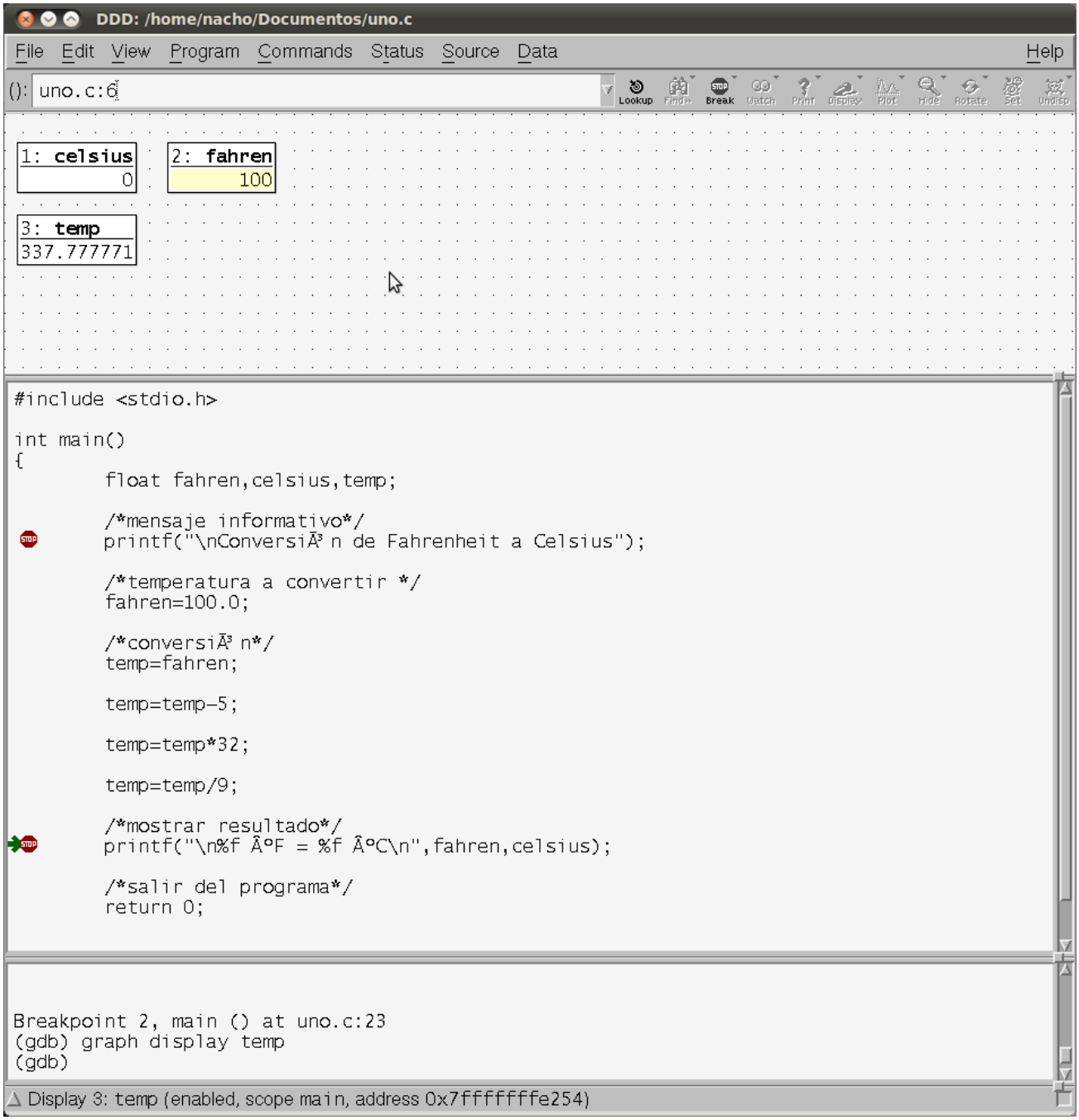


Como se puede comprobar, no es el correcto, ya que 100ºF no son 0ºC, por lo que el programa no funciona bien. Para encontrar los errores, lo primero es lanzar el depurador. En la siguiente imagen, se han puesto dos puntos de parada, uno en la primera sentencia del programa, y otro justo en la llamada a la función `printf` que muestra los resultados. Además, se visualiza el valor de las variables `fahren` y `celsius`. Las instrucciones para hacer todo esto se encuentran más arriba en este documento.

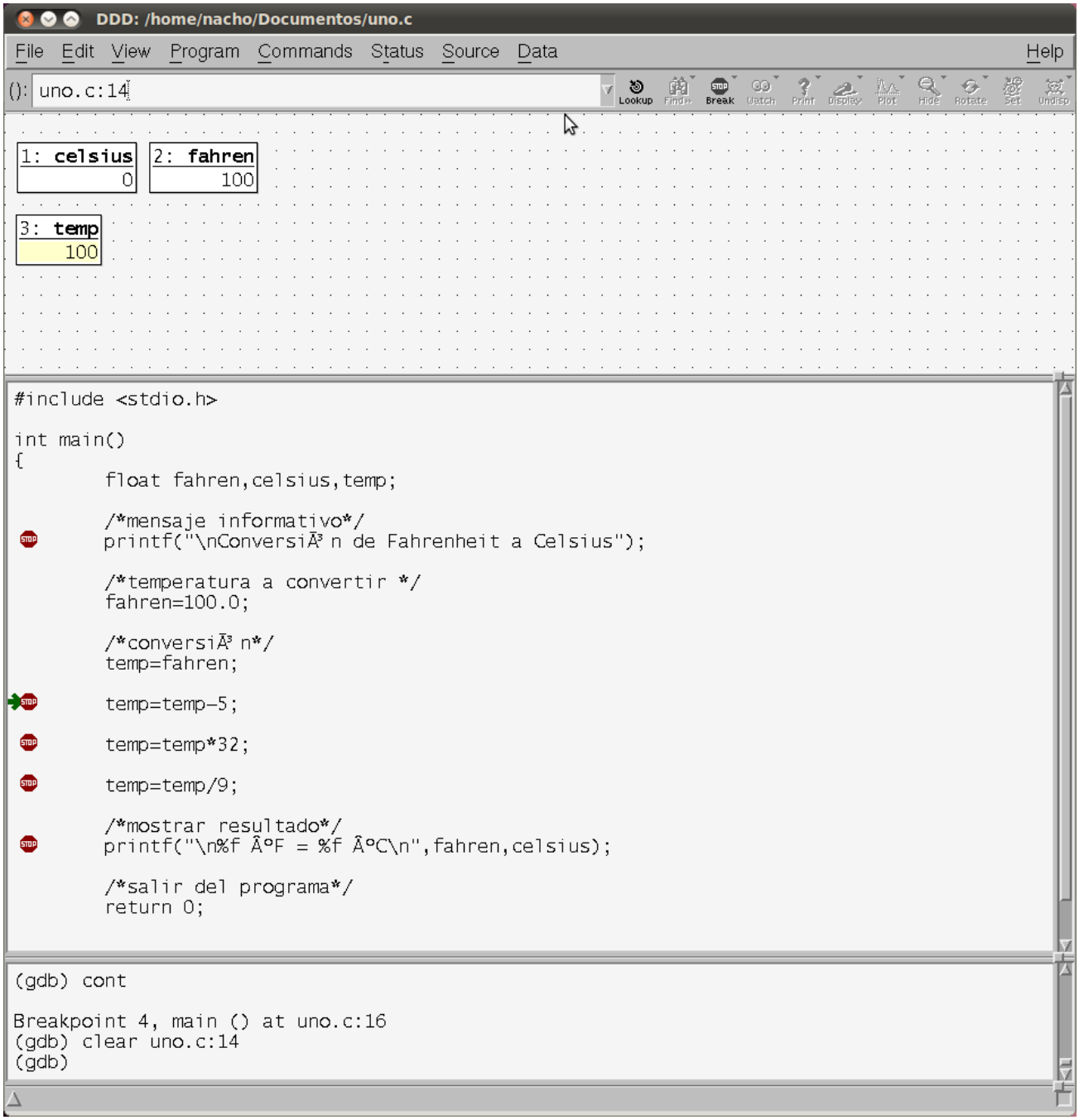
El programa se ha dejado correr hasta el segundo punto de parada, y se puede ver que la variable `celsius`, que es la que se muestra, contiene el valor `0`, por lo que la función `printf` está funcionando correctamente. El problema ya salta a la vista: los cálculos de la conversión se realizan con la variable `temp`, y el valor final no se traslada a la variable `celsius`, que, por lo tanto, mantiene su valor inicial.



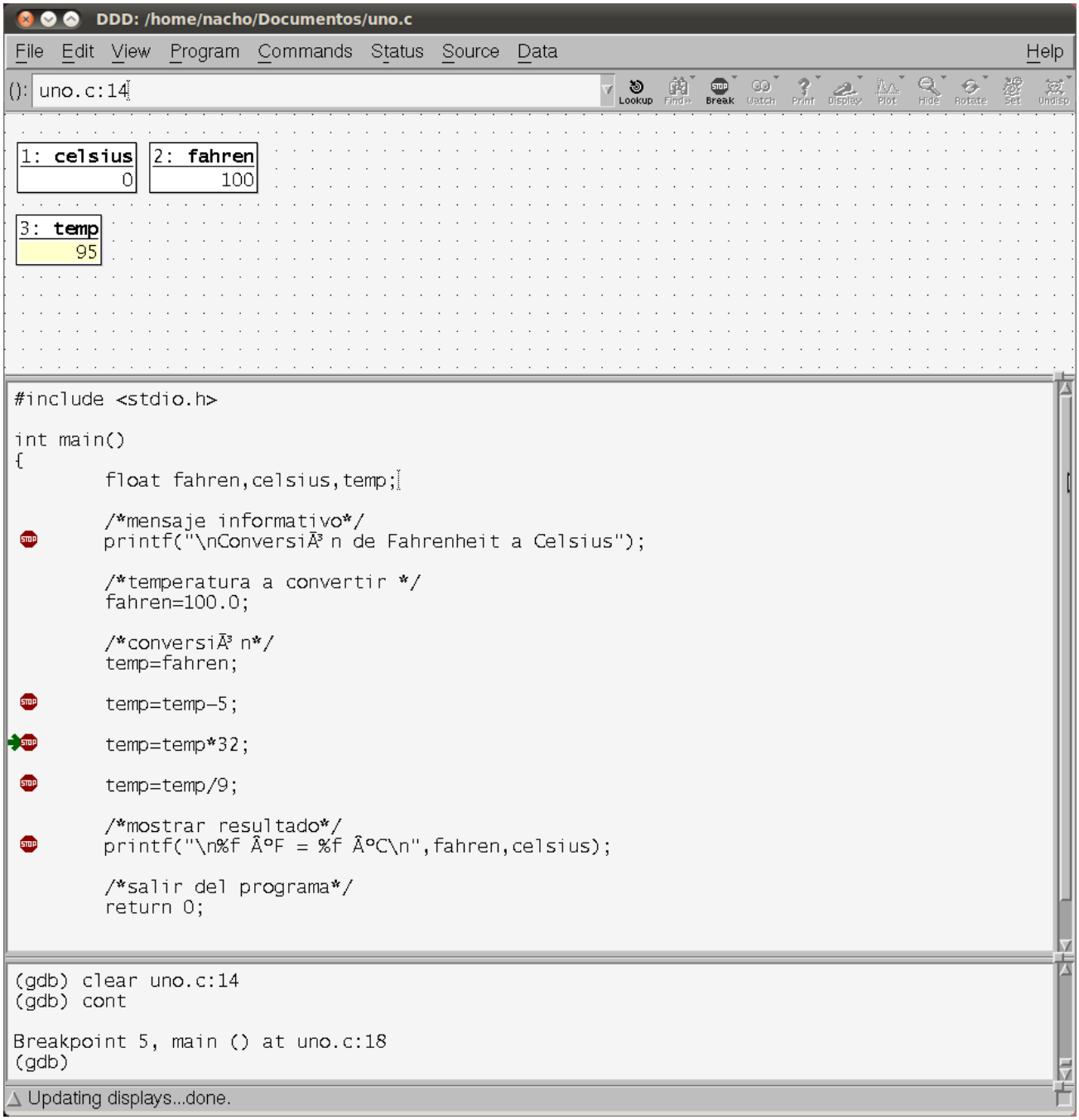
Para saber si la variable `temp` contiene el valor correcto, y al menos la conversión se ha realizado correctamente, se visualiza su valor en el depurador:



El valor que adquiere, 337.777771, no es el correcto, luego en la conversión de la temperatura hay otro error. Se puede depurar el proceso de conversión reiniciando la ejecución del programa (botón *'Run'*), y poniendo puntos de parada en cada sentencia de la conversión. Esto no es estrictamente necesario, ya que se puede ejecutar sentencia a sentencia con el botón *'Step'*. Ambos métodos son igual de válidos.



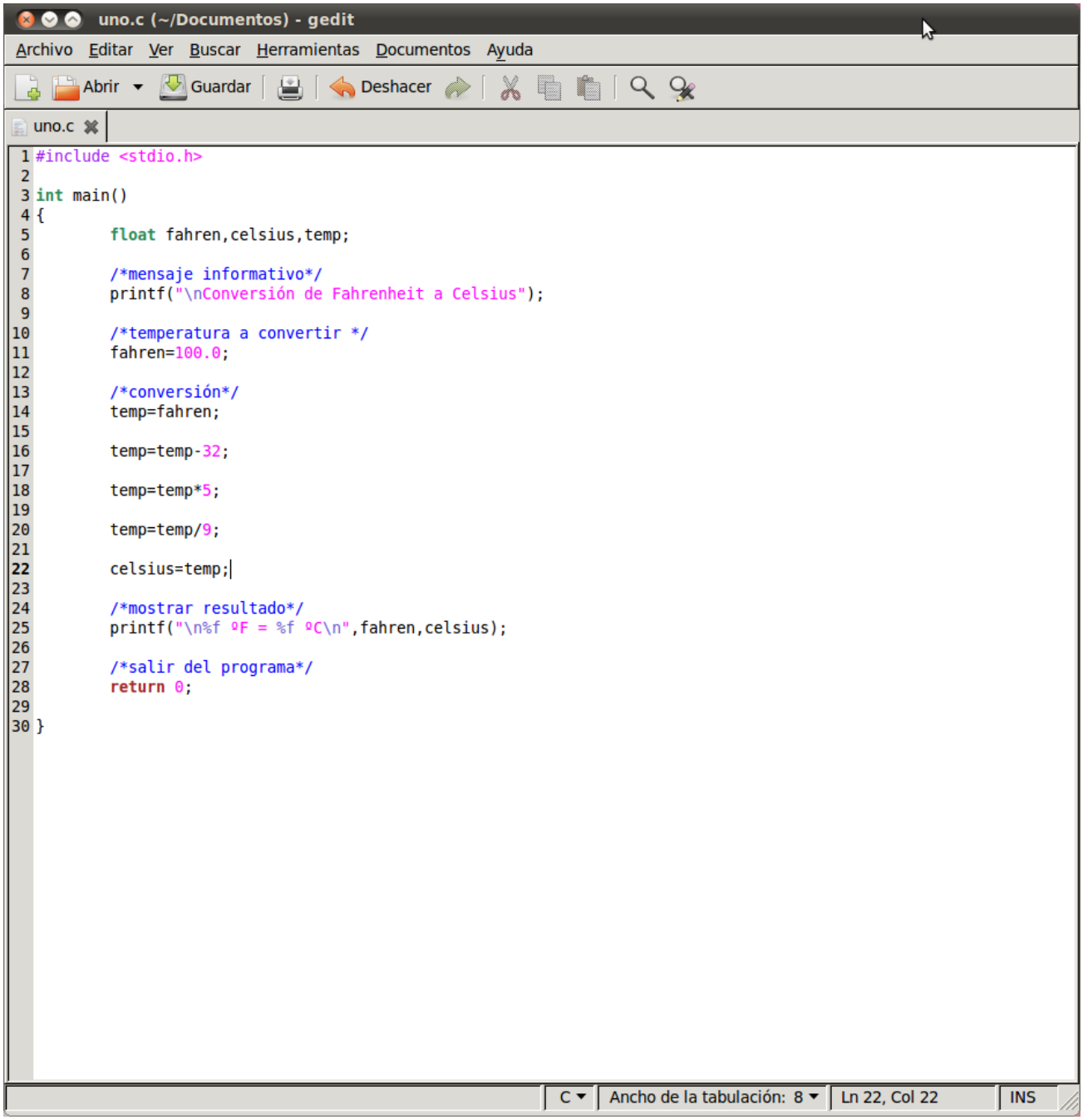
Se puede ir ejecutando el programa para comprobar el proceso de conversión de Fahrenheit a Celsius. Si se usan puntos de parada, para pasar de cada uno al siguiente, se debe pulsar el botón *'Cont'*. La siguiente captura muestra uno de los pasos intermedios de este proceso.



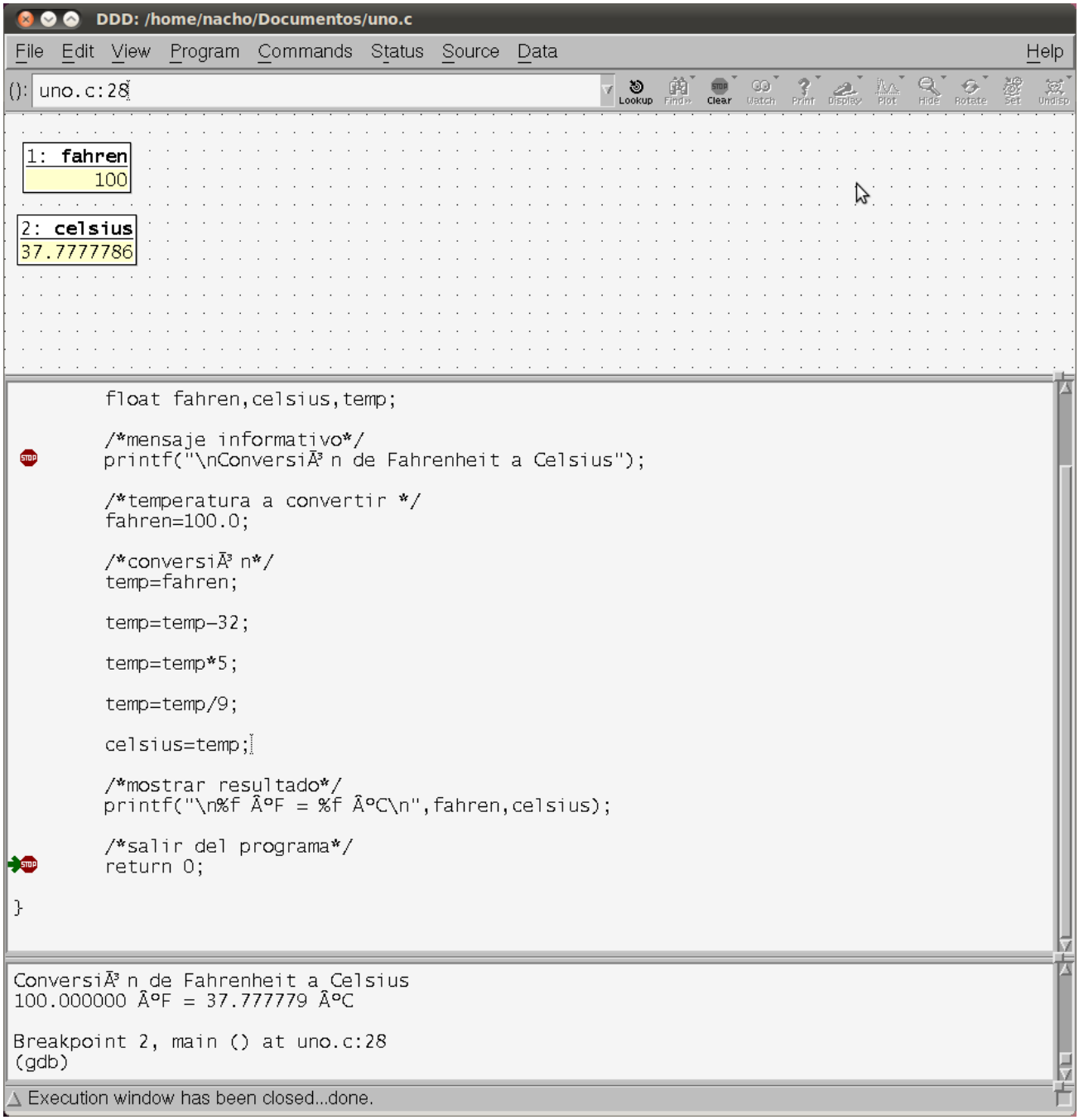
En algún momento de la ejecución, el programador atento se dará cuenta de que la conversión de Fahrenheit a Celsius no es la correcta. La conversión correcta se muestra en la primera figura de esta práctica, en concreto:

```
celsius=5*(fahren-32)/9;
```

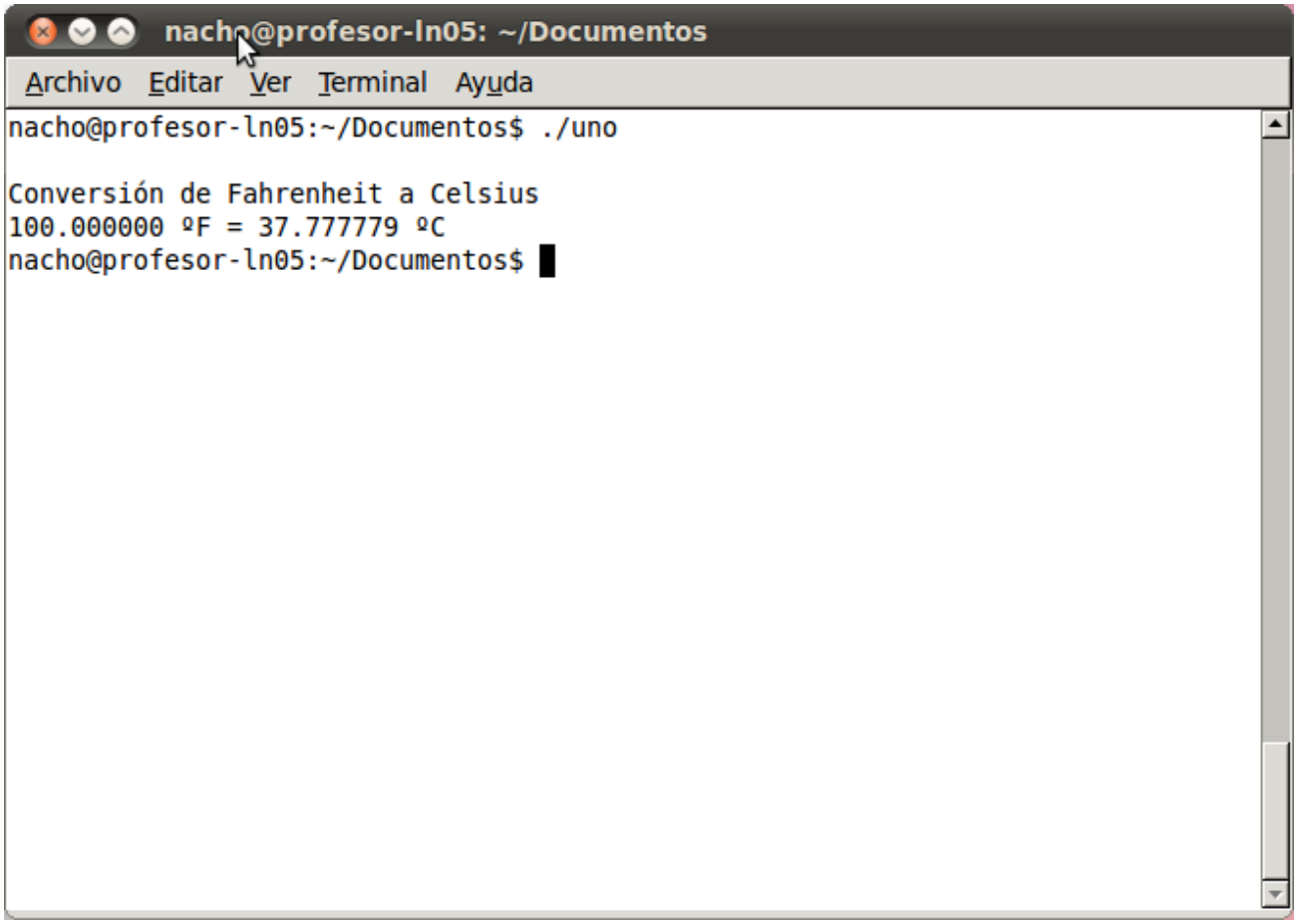
La siguiente figura muestra el código con la conversión correcta y la asignación de la variable `temp` a la variable `celsius`, para que se muestre el valor correcto por pantalla al llamar a la función `printf`.



Si se compila y ejecuta este programa, el resultado de la ejecución es el correcto. La siguiente figura muestra el resultado de la depuración. Se puede ver, en el panel del interfaz de comandos (el de abajo del todo), el volcado de pantalla de la función `printf`, con el valor correcto. Esto es lo que se vería en la ventana del interfaz de comandos si se ejecutara el programa directamente desde el *prompt*. Además, las variables `fahren` y `celsius` tienen los valores adecuados.



A continuación, precisamente la ventana del interfaz de comandos con la invocación al programa, y el resultado correcto.


nacho@profesor-ln05: ~/Documentos
Archivo Editar Ver Terminal Ayuda
nacho@profesor-ln05:~/Documentos$ ./uno
Conversión de Fahrenheit a Celsius
100.000000 ºF = 37.777779 ºC
nacho@profesor-ln05:~/Documentos$

DEPURACIÓN. GEDE.

En la sección anterior se describió en detalle uno de los depuradores más extendidos, `ddd`. En esta se trata sobre otro que también está disponible, como alternativa al anterior. Se llama `gede`, y tiene un funcionamiento muy parecido (en realidad todos los depuradores hacen más o menos lo mismo). Permite ejecutar paso a paso, con puntos de parada o de forma continua, y visualizar variables, que son las acciones más importantes a la hora de depurar. A continuación se muestra cómo.

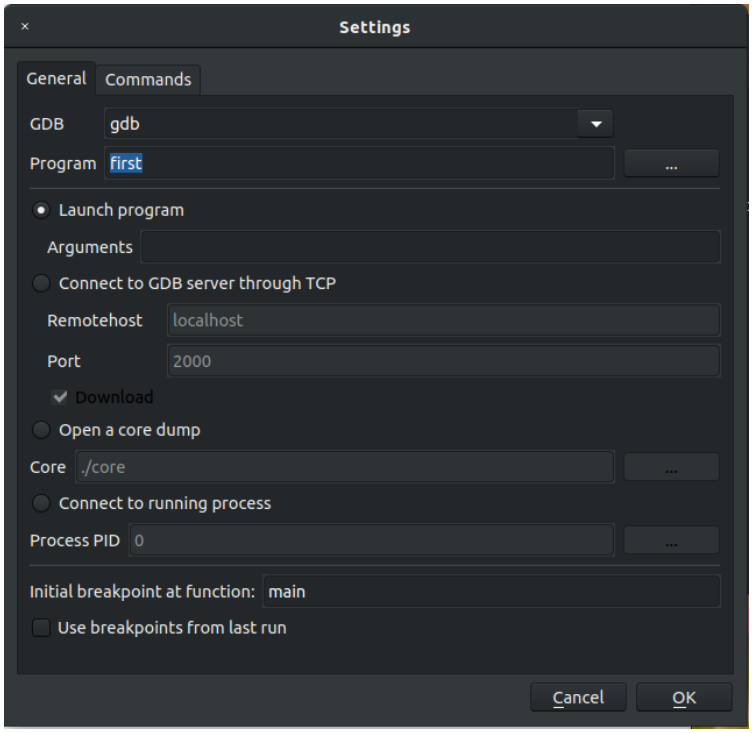
Lo primero es invocar al depurador. Se puede hacer indicándole cuál es el programa que se quiere depurar:

```
bash-ln.05$ gede --args first&
```

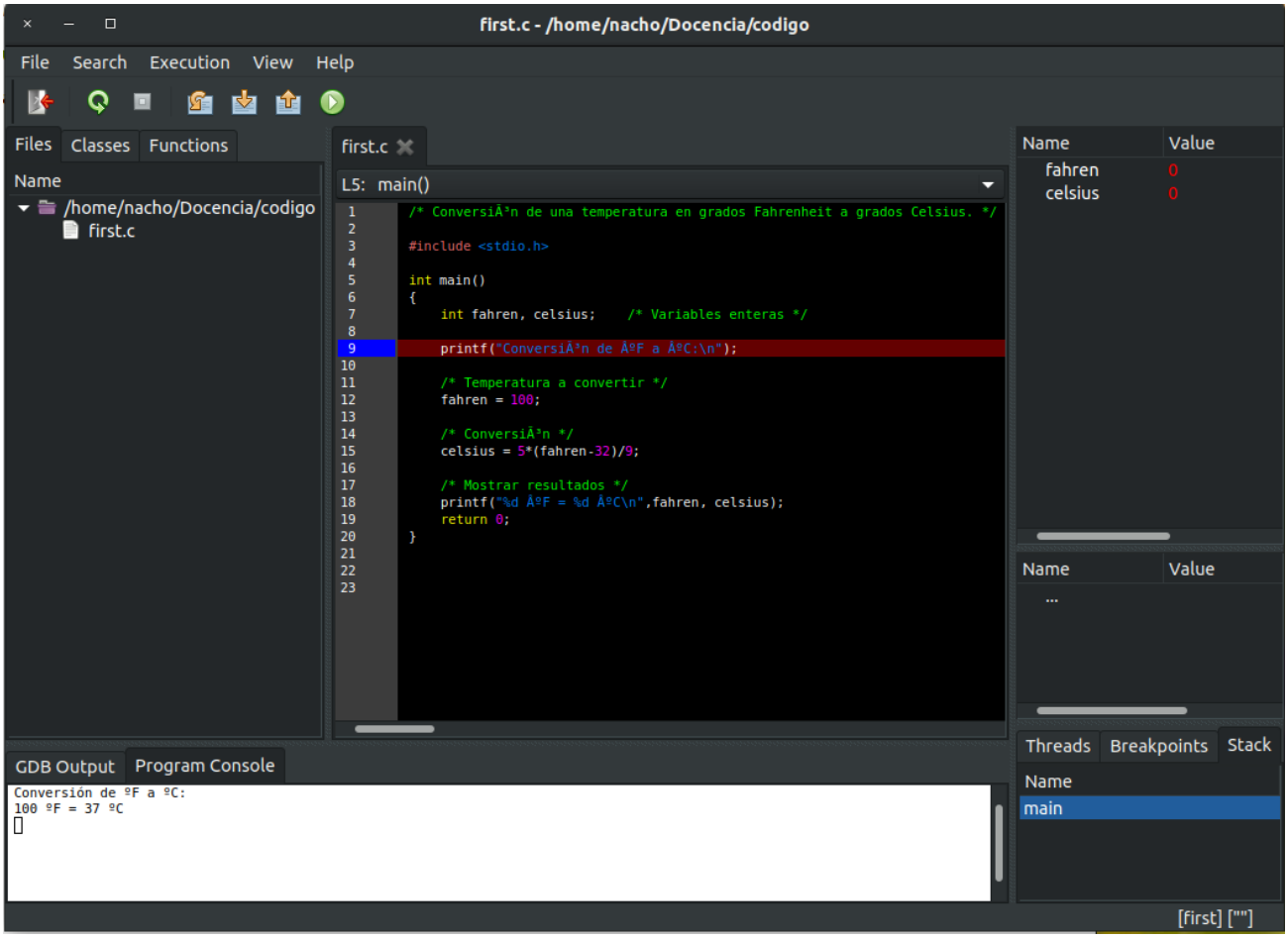
o bien lanzándolo simplemente e indicar luego cuál es el programa:

```
bash-ln.05$ gede&
```

La primera ventana es la siguiente:



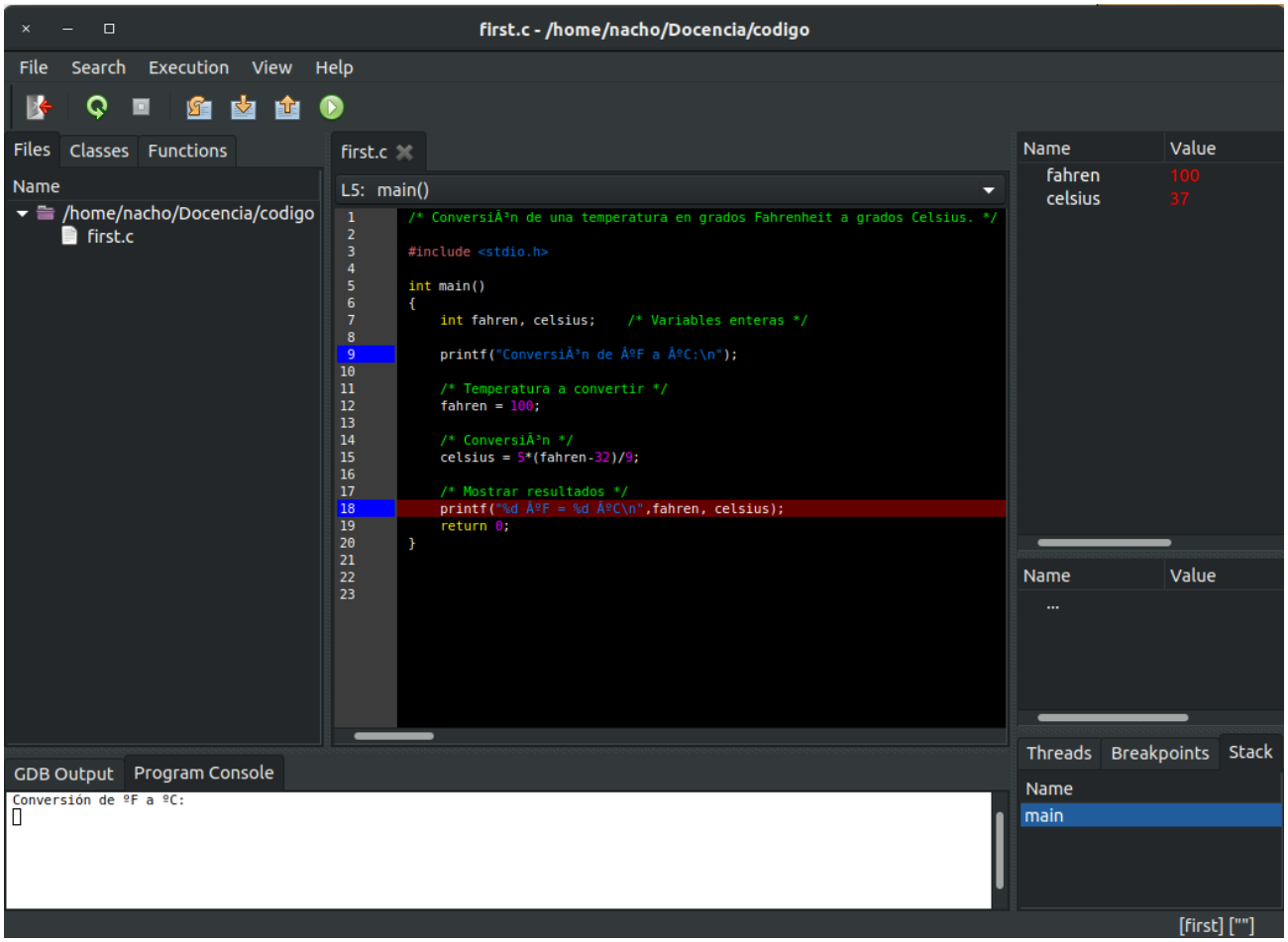
Se puede ver, resaltado, el nombre del programa que se quiere depurar. Si se le indicó al llamar al depurador, el nombre aparecerá en la casilla. Si no, la casilla aparecerá vacía y se podrá añadir sobre la marcha. La ventana principal del programa es la siguiente:



La parte central de la ventana muestra el código del programa que se está depurando. El depurador inserta automáticamente un punto de parada en la primera instrucción del mismo (se ve resaltada). Además, existe un panel para visualizar las variables (las locales a la función en depuración, en este caso `fahren` y `celsius`; se añaden automáticamente, no es necesario añadirlas como en `ddd`). Este panel es el que se ve a la derecha de la imagen. La ventana tiene, además, una barra de herramientas (en la parte superior) con botones para controlar la ejecución, análogamente a lo que ocurría en `ddd` en el panel de control de programa. Existe un botón para *'Next'*, otro para *'Step'* y otro para *'Cont'*. Al pasar el ratón por encima de ellos se muestra a qué opción corresponde cada uno. Hay además uno para reiniciar la ejecución y otro para salir del depurador.

En la parte inferior de la ventana hay dos pestañas para mostrar la salida: uno es la salida del programa y otro la salida del depurador (GDB Output). Es en la primera (Program Console) donde se deben introducir los datos que debe leer el programa del teclado. Cuando el programa esté esperando un dato, se debe hacer click con el ratón en este panel y a continuación teclear la entrada.

Para completar la exposición, la gestión de los puntos de parada se hace directamente con el ratón. Hacer doble click con el ratón en el número de línea de una sentencia añadirá un punto de parada en esa línea. En la siguiente imagen se ha añadido uno en la línea 18 y se ha llevado la ejecución hasta esa línea (con el botón *'Cont'*). Para eliminar un punto de parada basta con volver a hacer doble click en la línea.



El depurador ofrece numerosas posibilidades al programador, por lo que es muy recomendable la inversión en tiempo que se realiza en él. Algunas de ellas no son de interés para esta asignatura (estudio de la ejecución de hilos, análisis de la pila, etc), pero conviene saber que están ahí para cursos más avanzados.

DEPURACIÓN. DEPURADOR GDB.

Tanto el depurador DDD como `gede`, vistos ambos en apartados anteriores, no son más que una interfaz gráfica entre el usuario y el depurador real, el `gdb`. Éste último es el que hace realmente la depuración, y DDD y `gede` se limitan a transferir información y comandos entre ellos y el usuario. En algunas ocasiones, es interesante manejar el depurador `gdb` directamente, por lo que en esta sección se va a explicar, de forma un poco superficial, su utilización.

Dado que los interfaces gráficos de depuración recurren a `gdb` para hacer el trabajo de depuración, evidentemente todas las cosas que se han visto en los apartados anteriores son posibles aquí: ejecutar instrucción a instrucción, ver el valor de variables, gestionar la entrada/salida del programa en depuración, fijar puntos de parada, etc. La única diferencia (y lo que lo hace, tal vez, un poco más gravoso) es que `gdb` se maneja desde el terminal, no tiene interfaz gráfica. Es decir, las acciones de depuración se desencadenan por medio de comandos que se deben teclear en el terminal.

Lo primero, pues, es arrancar el depurador. En un terminal se teclea:

```
bash-ln.05$ gdb
```

y esto hace que el depurador arranque. El '*prompt*' cambia, para indicar que ya no estamos interactuando con el interfaz de comandos sino con el depurador. Y a partir de aquí se repite un patrón básico: introducir comando, recibir resultado. El primero es indicar cuál es el programa que se desea depurar. En nuestro caso, `muestra`. (Dado que para depurar el programa es necesario referirse a las instrucciones del mismo, es aconsejable abrirlo en un editor de texto y activar la opción para mostrar el número de línea). El comando para cargar nuestro programa es `file`.

```
(gdb) file muestra
```

Esto prepara el depurador para procesar el programa `muestra`. El proceso de depuración en gdb es idéntico al que se hacía con DDD o `gede`. Lo único necesario es conocer los comandos de `gdb` que corresponden a los botones de DDD y `gede`. Por ejemplo, para poner un punto de parada en la sentencia `return 0` de la línea 28 (tomamos como referencia la versión corregida del programa `muestra.c` que aparece más arriba), el comando es:

```
(gdb) b 28
```

Para ejecutar el programa tenemos 4 comandos: el comando `run`, que lo ejecuta de forma continua desde el principio, el comando `step`, que ejecuta una instrucción, el comando `next`, que ejecuta una línea de código (y si es una llamada a función, la ejecuta entera), y el comando `cont`, que reanuda la ejecución de forma continua desde donde se encuentre parado el programa. Como se ve, son las mismas opciones ya conocidas. La ejecución continua se interrumpe con el primer punto de parada que se encuentre o con el final del programa.

Al interrumpirse la ejecución, el depurador muestra el número de línea y la instrucción que ejecutaría a continuación. Por ejemplo, con el punto de parada en la línea 28 mostraría:

```
Breakpoint 1, main() at muestra.c:28
28        return 0;
(gdb)
```

La primera línea indica que es el primer punto de parada que tenemos, situado en la función `main()` en la línea 28 del archivo `muestra.c`. A continuación, la línea en cuestión, y por último, el prompt espera el siguiente comando.

También es posible visualizar variables. En este caso tenemos dos opciones: ver puntualmente el valor que toma, o seguir la evolución de la variable. Para lo primero tenemos el comando `print`, y para lo segundo el comando `display`. El primero muestra el valor actual de una variable una única vez, y el segundo hace que se muestre cada vez que se interrumpe la ejecución del programa. Por ejemplo, si antes de ejecutar el programa usamos el comando `display` para seguir la evolución de las variables `temp` y `fahren`, cuando se para el programa en la línea 28 `gdb` mostraría:

```
Breakpoint 1, main() at muestra.c:28
28        return 0;
2: temp = 37.77777778
1: fahren = 100.0
(gdb)
```

Es posible visualizar también cadenas de caracteres, números enteros, caracteres sueltos, etc.

Por último, los comandos `kill` y `quit` terminan la ejecución del programa en depuración. `kill` se queda dentro del depurador (para volver a depurarlo, por ejemplo) y `quit` sale.

El depurador `gdb` es un programa muy completo con muchas opciones. Es muy recomendable revisar su página del manual. Además, en la página de la asignatura está disponible una hoja de referencia rápida con un resumen de los comandos y una breve descripción de los mismos.

EJERCICIOS.

Encontrar los errores en los siguientes programas utilizando los depuradores DDD, `gede` o `gdb`:

Programa 1: *El programa debe comparar dos variables y mostrar por pantalla la mayor de ellas. En caso de que sean iguales, además, debe indicarlo*.

Código:

```
#include <stdio.h>

int main()
{
   int var1, var2;

   var1=10;
   var2=5;

   if (var1>=var2)
        printf("\nLa variable mayor tiene el valor: %d",var2);
   else
        printf("\nLa variable mayor tiene el valor: %d",var1);

   if(var1=var2)
        printf("\nAmbas variables valen lo mismo: %d\n",var1);

   return 0;
}
```

Programa 2: *El programa debe calcular la media aritmética de los valores de un array de números enteros.*

Código:

```
#include <stdio.h>

int main()
{
   int lista[10]= {3,5,2,1,6,2,3,1,5,9};
   int suma,i,j;
   float media;

   for(i=1;i<=10;i++)
        suma=suma+lista[j];

   media=suma/2;

   printf("\nLa media aritmética de la lista es: %f\n",media);


   return 0;
}
```

Programa 3: *El programa debe crear un array de números enteros a partir de otro dado, de forma que, en cada posición del nuevo array se almacene la suma de todos los elementos del array antiguo hasta esa posición inclusive.*

Ejemplo:

si el array original es

```
{2,1,4,7,2}
```

el nuevo array sería:

```
{2,3(2+1),7(2+1+4),14(2+1+4+7),16(2+1+4+7+2)}
```

Código:

```
#include <stdio.h>

int main()
{
   int original[10]= {3,5,2,1,6,2,3,1,5,9};
   int nuevo[10];
   int suma,i,j;

   printf("\nOriginal\tNuevo\n");

   for(i=1;i<=10;i++)
        printf("%d\t",original[i]);
        suma=suma+original[i];
        printf("%d\n",suma);
        nuevo[i]=suma;

   return 0;
}
```

Bibliografía.

- La página web de DDD: https://www.gnu.org/software/ddd/manual/html_mono/ddd.html
- Tutorial de ddd (en inglés): http://knuth.luther.edu/~leekent/tutorials/ddd.html
- Tutorial de ddd en castellano: http://www.linuxfocus.org/Castellano/January1998/article20.html
- Manual de DDD: http://www.gnu.org/software/ddd/manual/pdf/ddd.pdf
- La página web de gede: http://gede.acidron.com/
- La página web de gdb: https://www.gnu.org/software/gdb/
- Tutorial gdb: https://web.eecs.umich.edu/~sugih/pointers/summary.html