

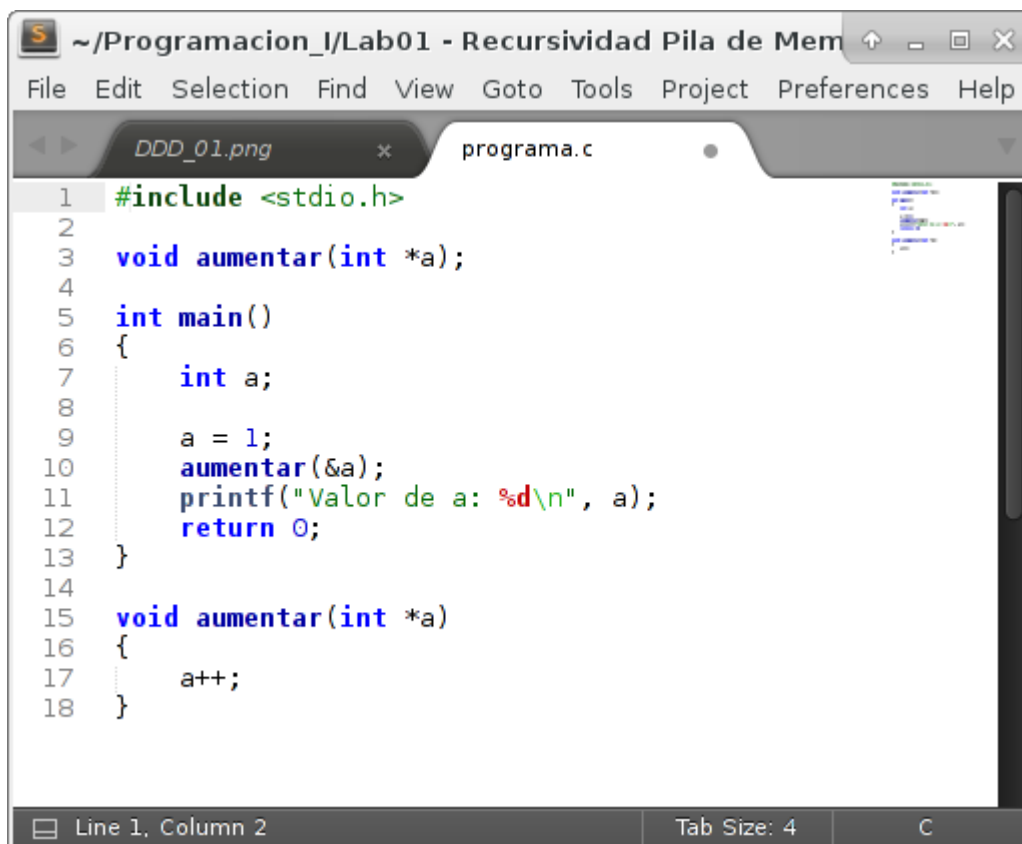
# DATA DISPLAY DEBUGGER

(DDD)

Original: Prof. Kiara Ottogalli

Revisado y modificado: Jorge Castellanos D.

Dado el siguiente programa:



```
1  #include <stdio.h>
2
3  void aumentar(int *a);
4
5  int main()
6  {
7      int a;
8
9      a = 1;
10     aumentar(&a);
11     printf("Valor de a: %d\n", a);
12     return 0;
13 }
14
15 void aumentar(int *a)
16 {
17     a++;
18 }
```

**¿Cuál es el valor impreso de a?**

**¿Cuál valor debería tener?**

En programas sencillos puede ser útil colocar expresiones `printf` para conocer los valores de las variables en un momento determinado de la ejecución de un programa y así determinar donde se encuentran los errores, sin embargo, cuando el programa es complejo y extenso, resulta engorroso depurar el programa realizando el seguimiento de las variables de esa manera.

Para solucionar ese problema se han creado diversas herramientas que permiten

realizar la depuración (debugging) de un programa de manera sencilla. Entre ellas se encuentra el Data Display Debugger (DDD).

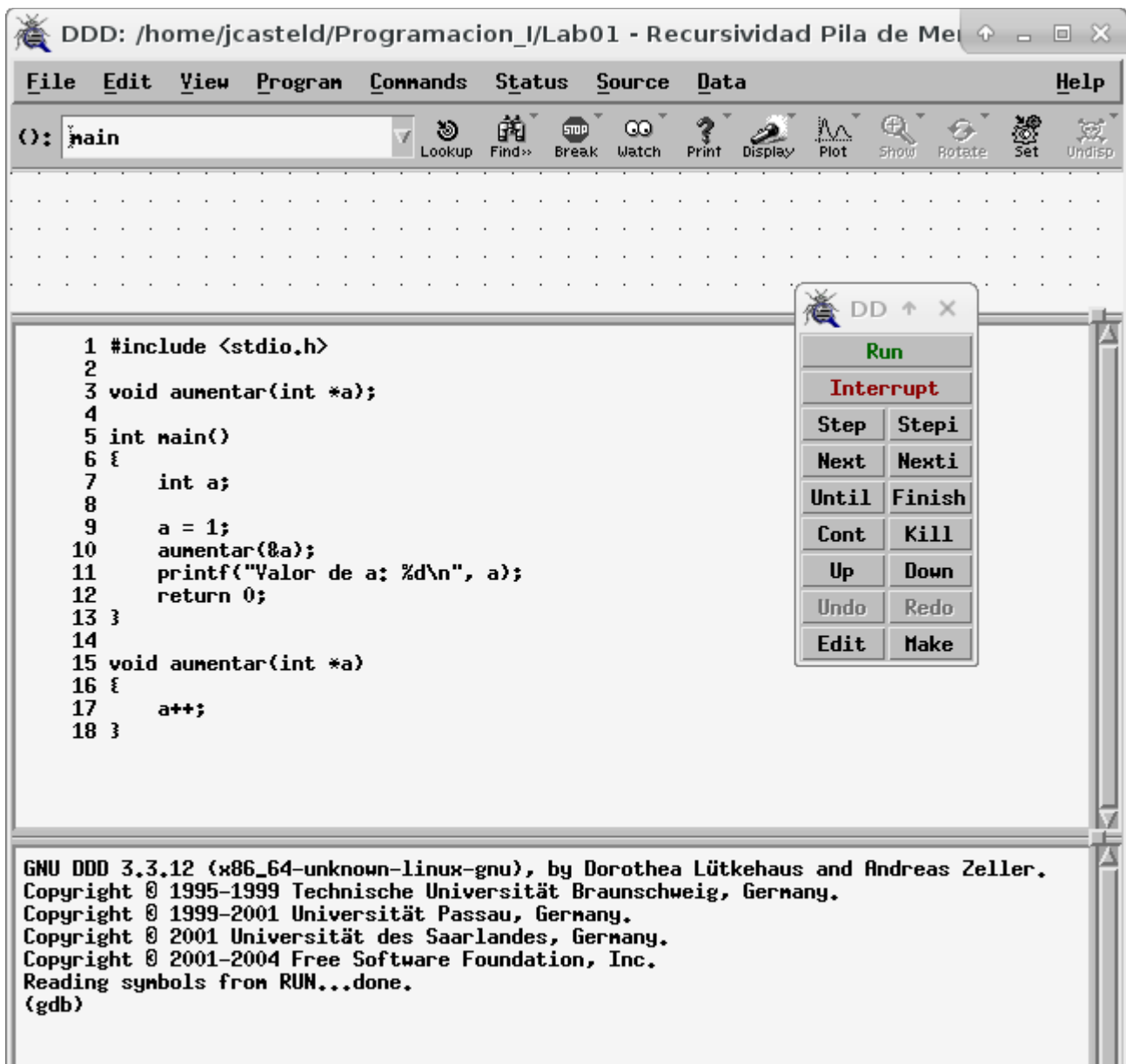
Para depurar un programa con DDD se debe compilar el mismo agregando el flag -g de la siguiente forma:

```
cc -g -o RUN programa.c
```

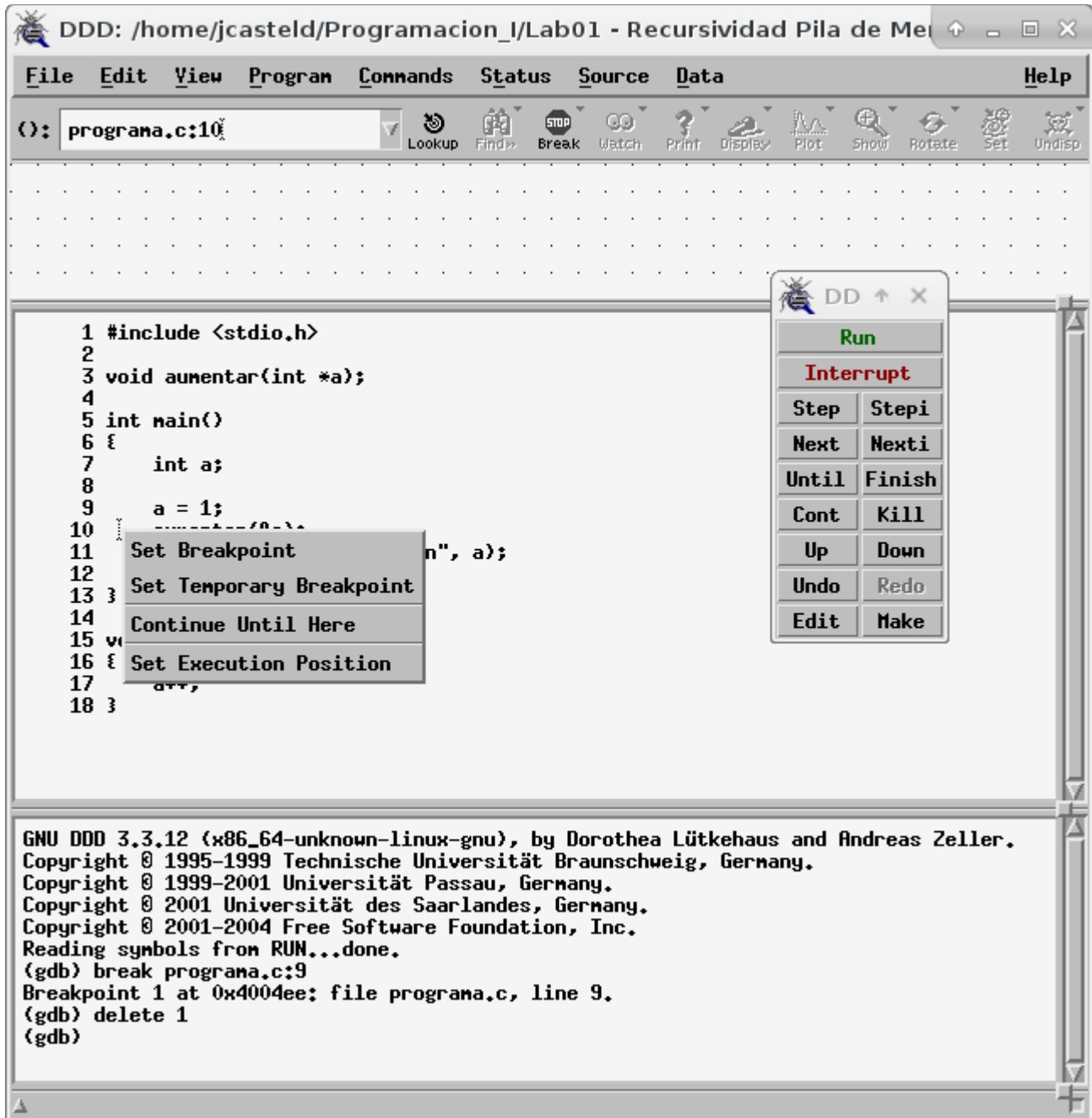
Luego para ejecutar el DDD se coloca el comando:

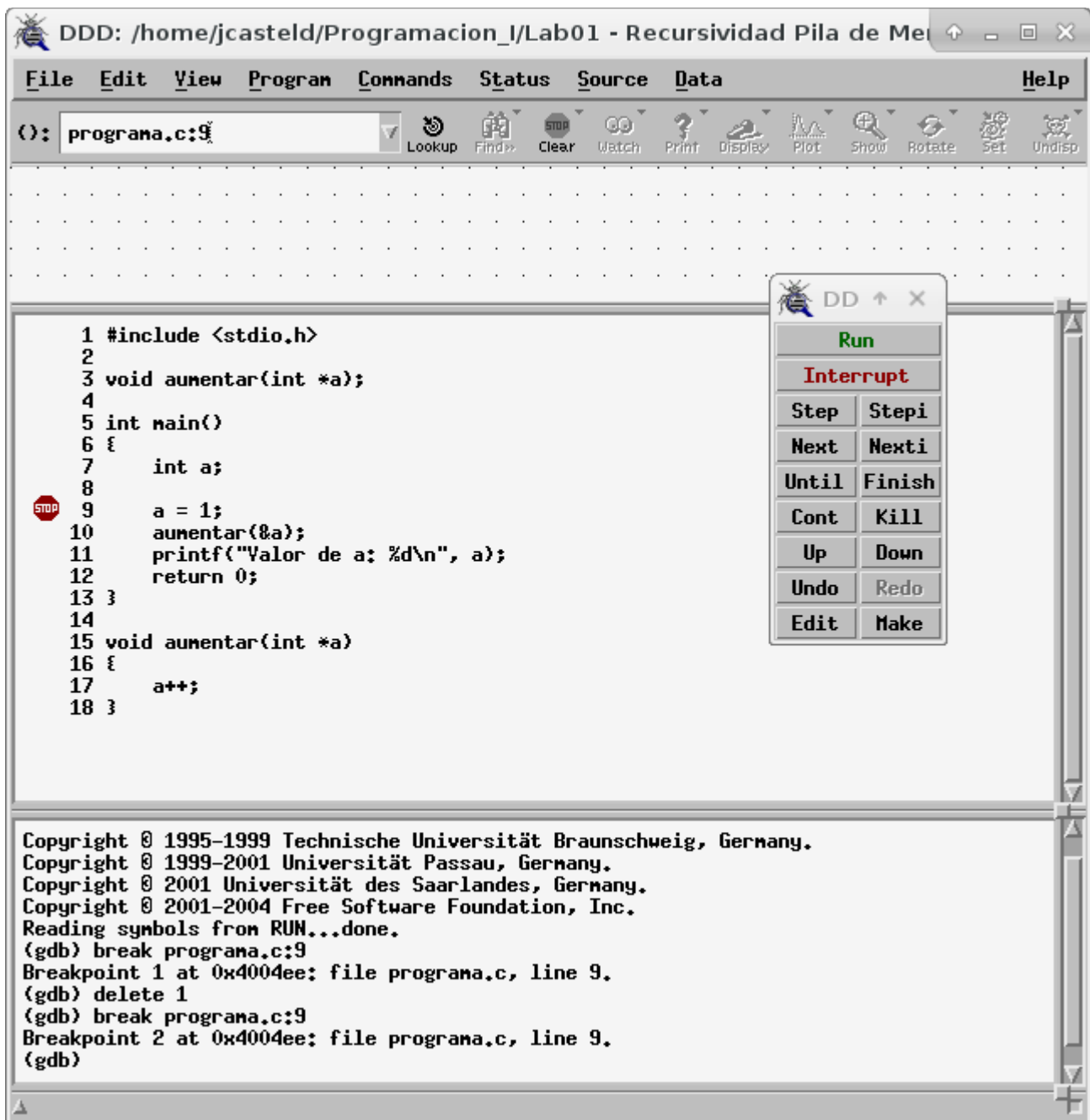
```
ddd ./RUN
```

Al ejecutar DDD aparece la interfaz del programa en la cual se muestra el código a depurar, un menú de comandos y una zona para mostrar los datos.



Se puede colocar un *breakpoint* en cualquier instrucción que se quiera analizar. En este caso se colocará un *breakpoint* en la primera instrucción después de la declaración de variables.





Luego en el panel de comandos se pulsa el botón <Run> para comenzar la ejecución del programa. En este punto se puede pulsar el botón <step> para ejecutar una por una las instrucciones del programa.

DDD: /home/jcasteld/Programacion\_I/Lab01 - Recursividad Pila de Memoria

File Edit View Program Commands Status Source Data Help

( ): programa.c:9

Lookup Find Clear Watch Print Display Plot Show Rotate Set Undisp

```
1 #include <stdio.h>
2
3 void aumentar(int *a);
4
5 int main()
6 {
7     int a;
8
9     a = 1;
10    aumentar(&a);
11    printf("Valor de a: %d\n", a);
12    return 0;
13 }
14
15 void aumentar(int *a)
16 {
17     a++;
18 }
```

DD

Run

Interrupt

Step StepI

Next NextI

Until Finish

Cont Kill

Up Down

Undo Redo

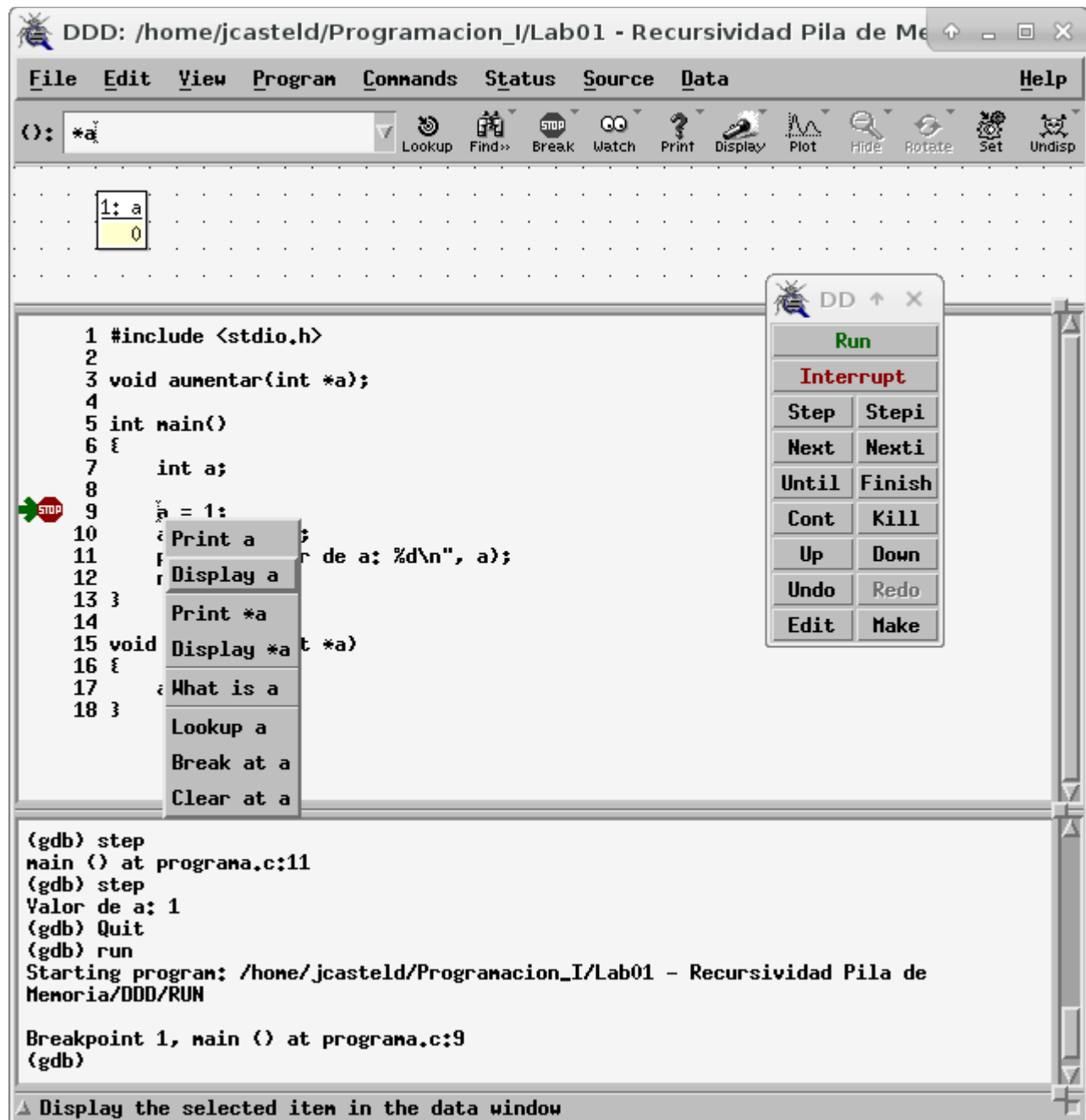
Edit Make

Copyright © 2001 Universität des Saarlandes, Germany.  
Copyright © 2001-2004 Free Software Foundation, Inc.  
Reading symbols from RUN...done.  
(gdb) break programa.c:9  
Breakpoint 1 at 0x4004ee: file programa.c, line 9.  
(gdb) run  
Starting program: /home/jcasteld/Programacion\_I/Lab01 - Recursividad Pila de Memoria/ddd/RUN

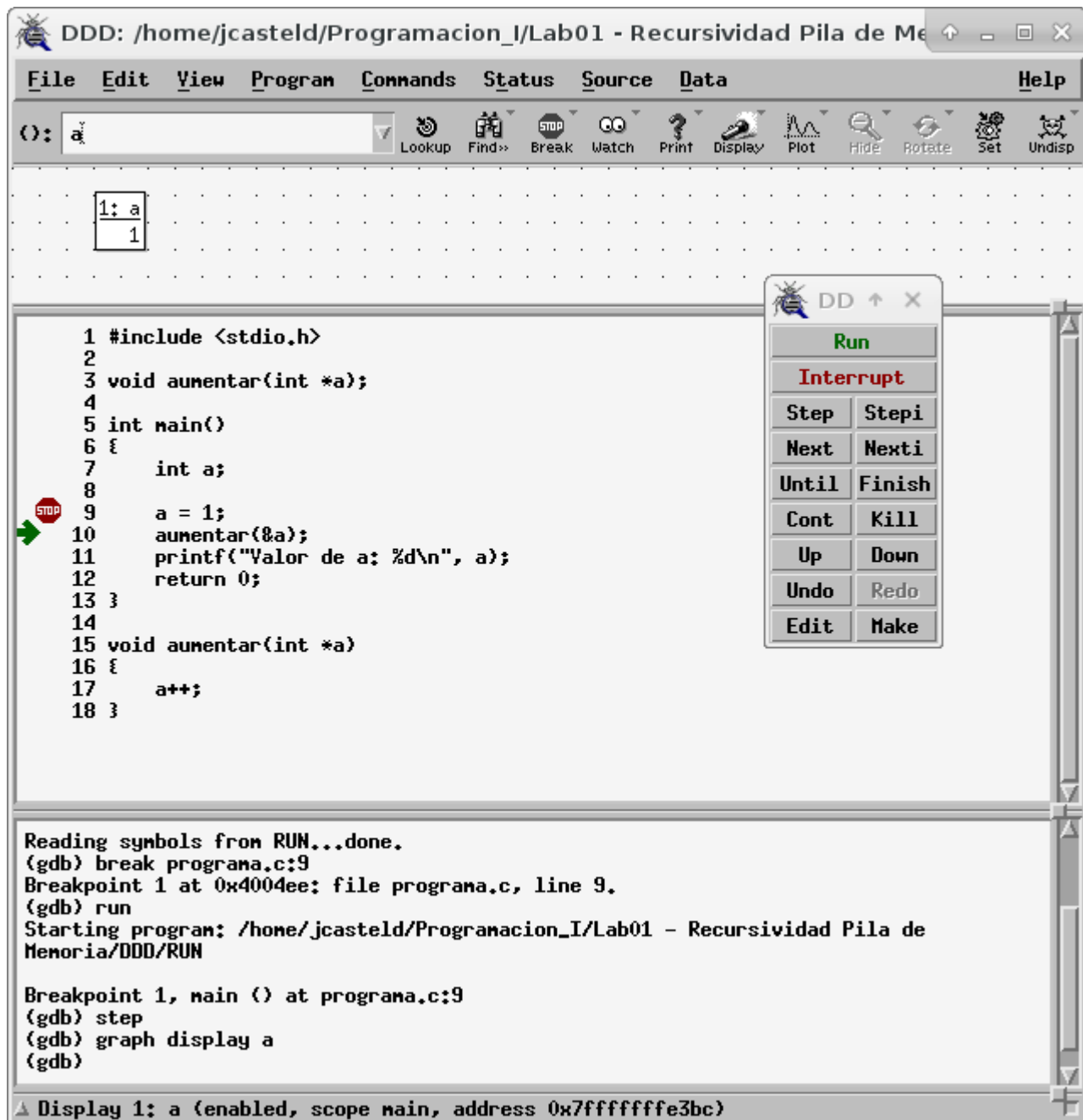
Breakpoint 1, main () at programa.c:9  
(gdb)

Disassembling location 0x4004ee...done.

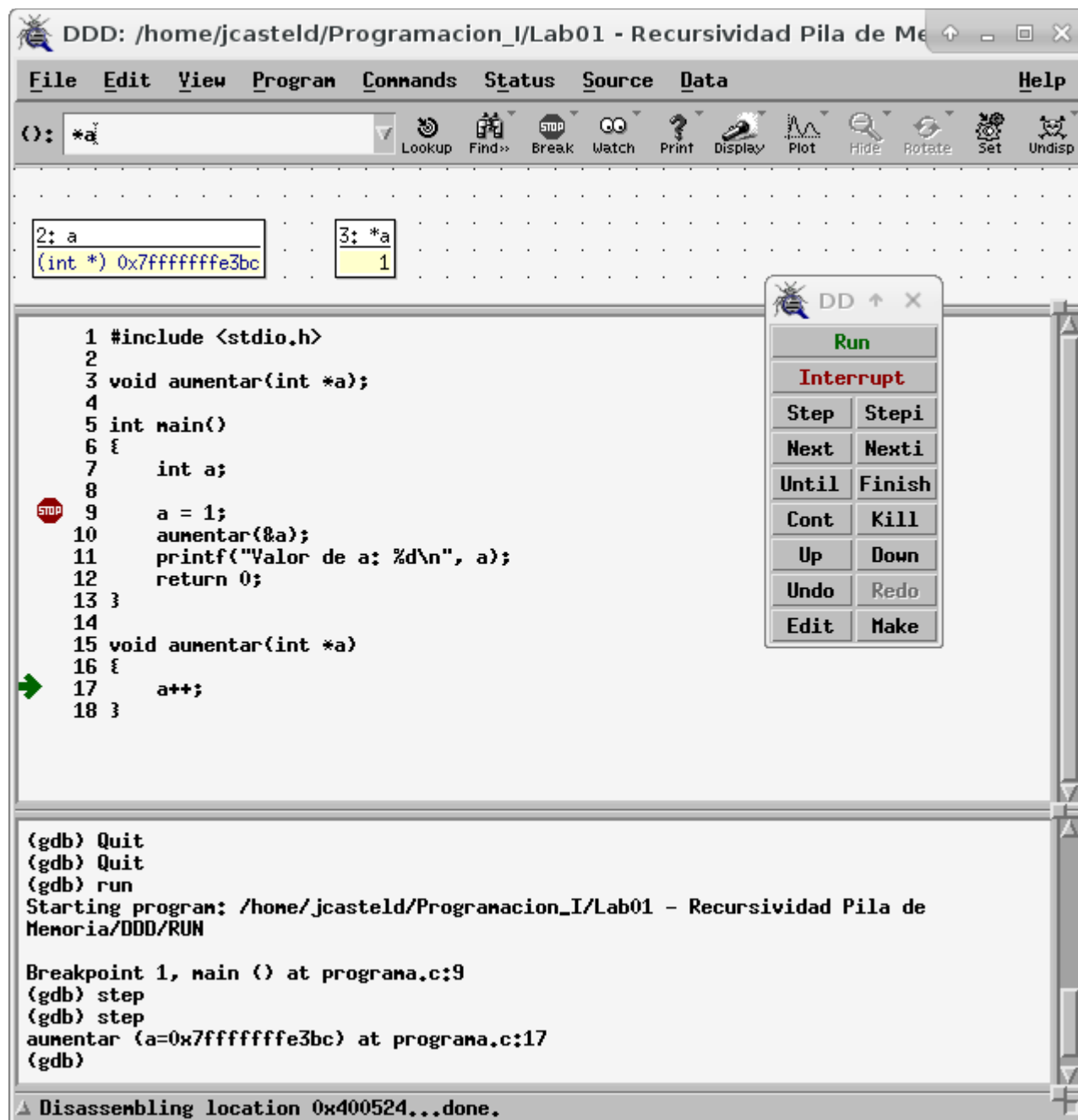
Si se quiere ver el valor de alguna variable se hace click derecho sobre la misma y se pulsa <display>. Se puede ver el valor de la variable `a` antes de ser inicializada.



Al pulsar <step> se ejecuta la instrucción `a = 1`; inicializando así la variable `a`. Este cambio en la variable se visualiza inmediatamente. Si se sigue pulsando <step> se ejecuta la operación *aumentar*. Nótese que dentro de la operación, sólo se muestran las variables locales de la misma.



Se puede ver el valor de  $a$  ( $*a$ ) dentro del procedimiento aumentar es un apuntador a entero. Igualmente se puede ver el espacio de memoria al cual apunta  $a$ , es decir,  $a$ .



DDD: /home/jcasteld/Programacion\_I/Lab01 - Recursividad Pila de Memoria

File Edit View Program Commands Status Source Data Help

(gdb) \*a

2: a (int \*) 0x7fffffff3bc

3: \*a 1

```
1 #include <stdio.h>
2
3 void aumentar(int *a);
4
5 int main()
6 {
7     int a;
8
9     a = 1;
10    aumentar(&a);
11    printf("Valor de a: %d\n", a);
12    return 0;
13 }
14
15 void aumentar(int *a)
16 {
17     a++;
18 }
```

Run Interrupt Step Step Next Next Until Finish Cont Kill Up Down Undo Redo Edit Make

(gdb) Quit  
(gdb) Quit  
(gdb) run  
Starting program: /home/jcasteld/Programacion\_I/Lab01 - Recursividad Pila de Memoria/DDD/RUN

Breakpoint 1, main () at programa.c:9  
(gdb) step  
(gdb) step  
aumentar (a=0x7fffffff3bc) at programa.c:17  
(gdb)

Disassembling location 0x400524...done.



Al ejecutar la operación `a++`; se puede ver que cambia la posición de memoria (se incrementa) porque `a` es un apuntador (dirección de memoria), mientras que `*a` es el contenido de la posición de memoria cuya dirección es `a`. Esto nos indica que la operación que debería aumentar la variable `a` se encuentra mal planteada. En lugar de `a++` debería estar `(*a)++`, es decir, en lugar de incrementar la posición de memoria `a`, se debería incrementar el contenido de la posición de memoria cuya dirección es `a`.

DDD: /home/jcasteld/Programacion\_I/Lab01 - Recursividad Pila de Memoria

File Edit View Program Commands Status Source Data Help

( ): \*a

2: a  
(int \*) 0x7ffffffe3c0

3: \*a  
4195632

```
1 #include <stdio.h>
2
3 void aumentar(int *a);
4
5 int main()
6 {
7     int a;
8
9     a = 1;
10    aumentar(&a);
11    printf("Valor de a: %d\n", a);
12    return 0;
13 }
14
15 void aumentar(int *a)
16 {
17     a++;
18 }
```

Run Interrupt Step Step Next Next Until Finish Cont Kill Up Down Undo Redo Edit Make

(gdb) Quit  
(gdb) run  
Starting program: /home/jcasteld/Programacion\_I/Lab01 - Recursividad Pila de Memoria/DDD/RUN

Breakpoint 1, main () at programa.c:9  
(gdb) step  
(gdb) step  
aumentar (a=0x7ffffffe3bc) at programa.c:17  
(gdb) step  
(gdb)

Updating displays...done.

Finalmente al ejecutar la instrucción printf, se puede ver en la parte inferior el valor de a que ha permanecido sin cambios.

