DEPURADORES PARA PROGRAMAS EN C

Manel Guerrero <guerrero AT ac.upc.edu>

DEPURANDO con el DDD

DDD Compilar y ejecutar

```
gcc -g -o ejemplo ejemplo.c     // “-g” para que sea debugable
ddd ejemplo &                   // '&' para que la terminal no se quede pillada
                                // si nos olvidamos el &: <Ctrl+Z> fg<Intro>
```

Kill DDD

Si el ddd se queda colgado, desde una terminal teclea: killall -9 ddd
(Esta instrucción significa “mata a todos los ddd”.)

Opciones del DDD

Recomiendo cambiar estas 3 preferencias en la configuración del DDD para que su uso sea mas cómodo:

- Suprimir los molestos warnings del ddd que aparecen en la terminal:
  Edit => Preferences => General => Suppress X warnings.
- Que nos muestre los números de línea en el código fuente:
  Edit => Preferences => Source => Display Source Line Numbers
- Que los nuevos displays se vayan poniendo a la derecha en lugar de abajo:
  Edit => Preferences => Data => Placement: [X] Left to right
- Que no haga falta que el puntero este sobre una area para escribir allí:
  Edit => Preferences => Startup => Keyboard Focus: [X] Click to Type

DDD Shortcuts

- Ctrl + - Muestra la variable seleccionada en el código en un display.

Esto es muy útil cuando el “doble clickar” sobre la variable no va ni a la primera, ni a la segunda. Curiosamente no me va con el '-' del teclado numérico pero si con el '-' del teclado alfanumérico.

- Alt + L Muestra todas las variables locales en la ventana "display".
- Alt + U Muestra los argumentos con los que se ha invocado la función actual.

DDD Options

- Para que no se mezclen los printf() i scanf() con la consola del gdb:
  ddd --exec-window ...
- para desactivarlo:
  ddd --no-exec-window ...
- Para cambiarlo una vez ya hemos abierto el ddd: Alt + 9

Core Dumped

Si un programa da cualquier cosa que genere un "(core dumped)": "Segmentation Fault" o "Floating point exception":

- Ejecutar el ddd con el programa en cuestión y dar a "Run" sin breakpoints.
- Al producirse el “core dumped” aparecerá una flecha roja (en lugar de la típica flecha verde) en la linea donde se ha producido.
- Si es "segmentation fault" buscar un vector (ej: "v[i]") poned el cursor sobre la 'i' y veréis que contiene un valor no permitido (ej: -1 o 22365). (Si no hay nada de vectores en esa linea habrá algún puntero que apunta a una posicion de memoria incorrecta.)
- Si es "floating point exception" probablemente estáis dividiendo por '0' o haciendo el modulo '0'. De nuevo poner el cursor sobre las variables para

ver que valores tenían en el momento del “core dumped”.

Debugando

Si nada mas empezar le damos al botón <STOP> (breakpoint), nos creara un breakpoint en la primera instrucción del main().

- [Run] Para empezar la ejecución:

A partir de aquí:

- [Next] Para ejecutar la siguiente instrucción o
- [Step] Para cuando en la linea actual hay una llamada a función "entrar" dentro del código de la función.
- [Cont] Continua la ejecución hasta el siguiente "Stop" o hasta el final del programa.

Teclas rápidas

Una vez hecho [Run], clickad en el área de código C y podéis usar estas teclas:

- [F5] Step
- [F6] Next
- [F9] Continue

(No hagáis [Step] en printf(), scanf() o otras funciones que no sean vuestras)

Resetear DDD

Resetear ddd como si nunca se hubiera ejecutado:

```
rm -Rf ~/.ddd
```

Ejemplo de uso del DDD

Fichero t4_divisio_pref.c:

```
#include <stdio.h>

void divisio(int n, int d, int *p_q, int *p_r){
      *p_q = n/d;
      *p_r = n%d;
}

main(){
      int n, d, q, r;
      printf("Divisio entera: Intro 'numerador/denominador': ");
      scanf("%d/%d", &n, &d);
      divisio(n, d, &q, &r);
      printf("El quocient es %d i el residu es %d\n", q, r);
}
```

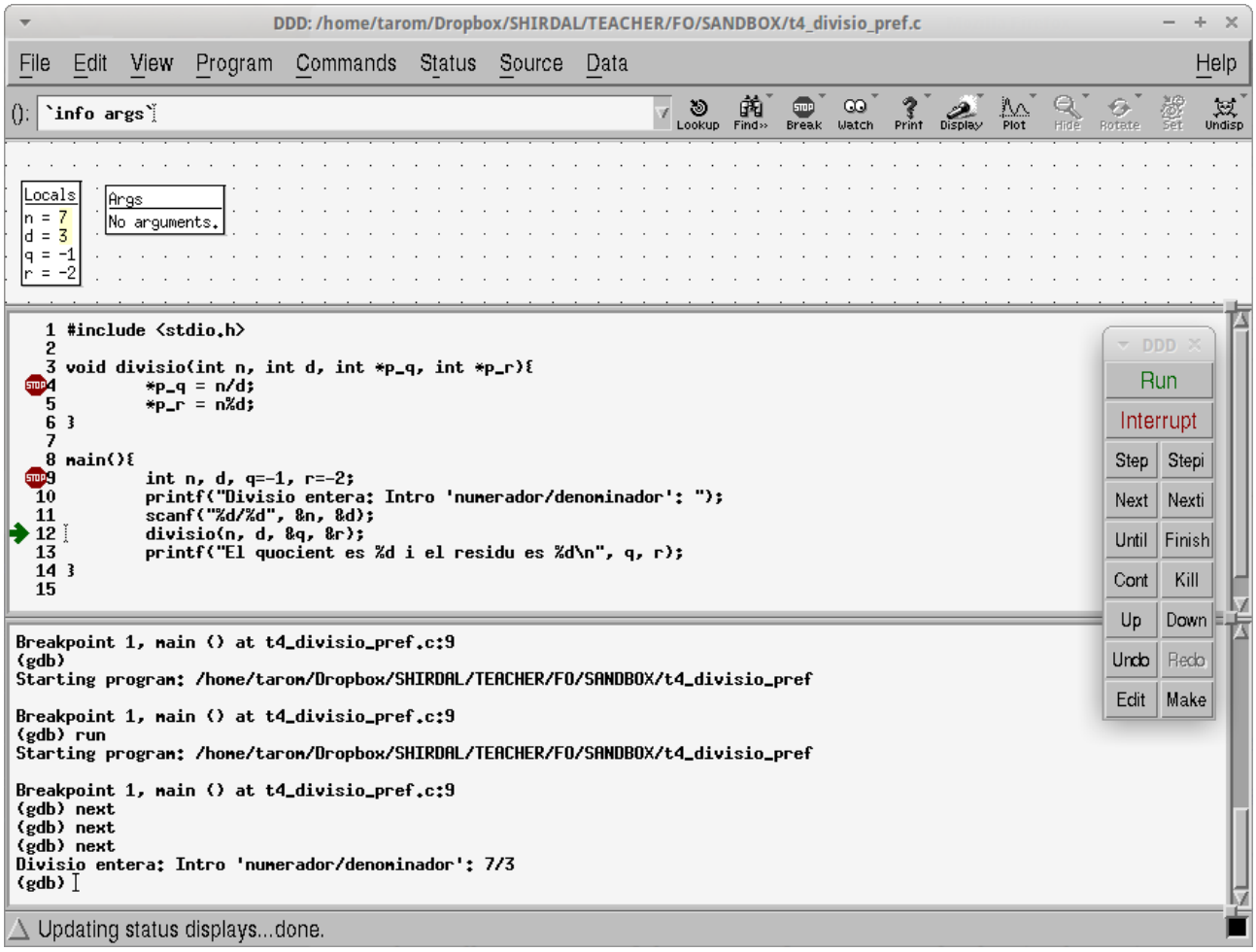
Este programa tiene una función “divisio()” que dado un numerador “n” y un denominador “d”, calcula el cociente y el residuo y los guarda en los parámetros que se le han pasado por referencia (“p_q”: “punter a quocient”; “p_r”: “punter a residu”). La inicialización de “q” y “r” en el “main()” no son necesarias.

Para compilar y debugarlo tecleamos en la terminal (una vez estamos en el directorio que contiene “t4_divisio_pref.c”:

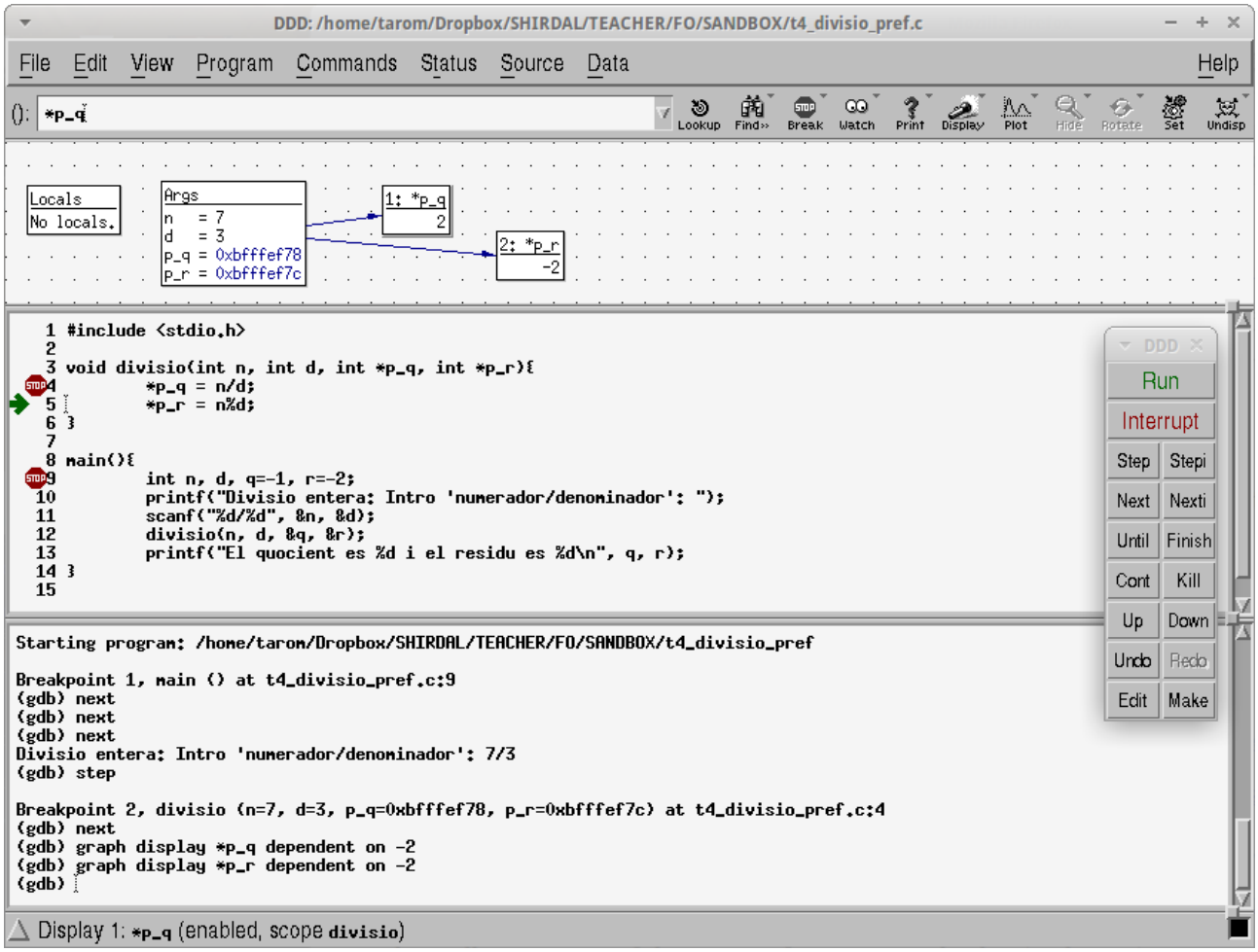
```
gcc -g -o t4_divisio_pref t4_divisio_pref.c
ddd t4_divisio_pref&
```

Se abre el DDD. Si le doy al [STOP] me creara un breakpoint en la línea 9, que es la primera en ejecutarse. Clicko en la línea 4 y le vuelvo a dar al [STOP] para que cree otro breakpoint en esa línea.
Le doy a [Run] para que empieze el programa.

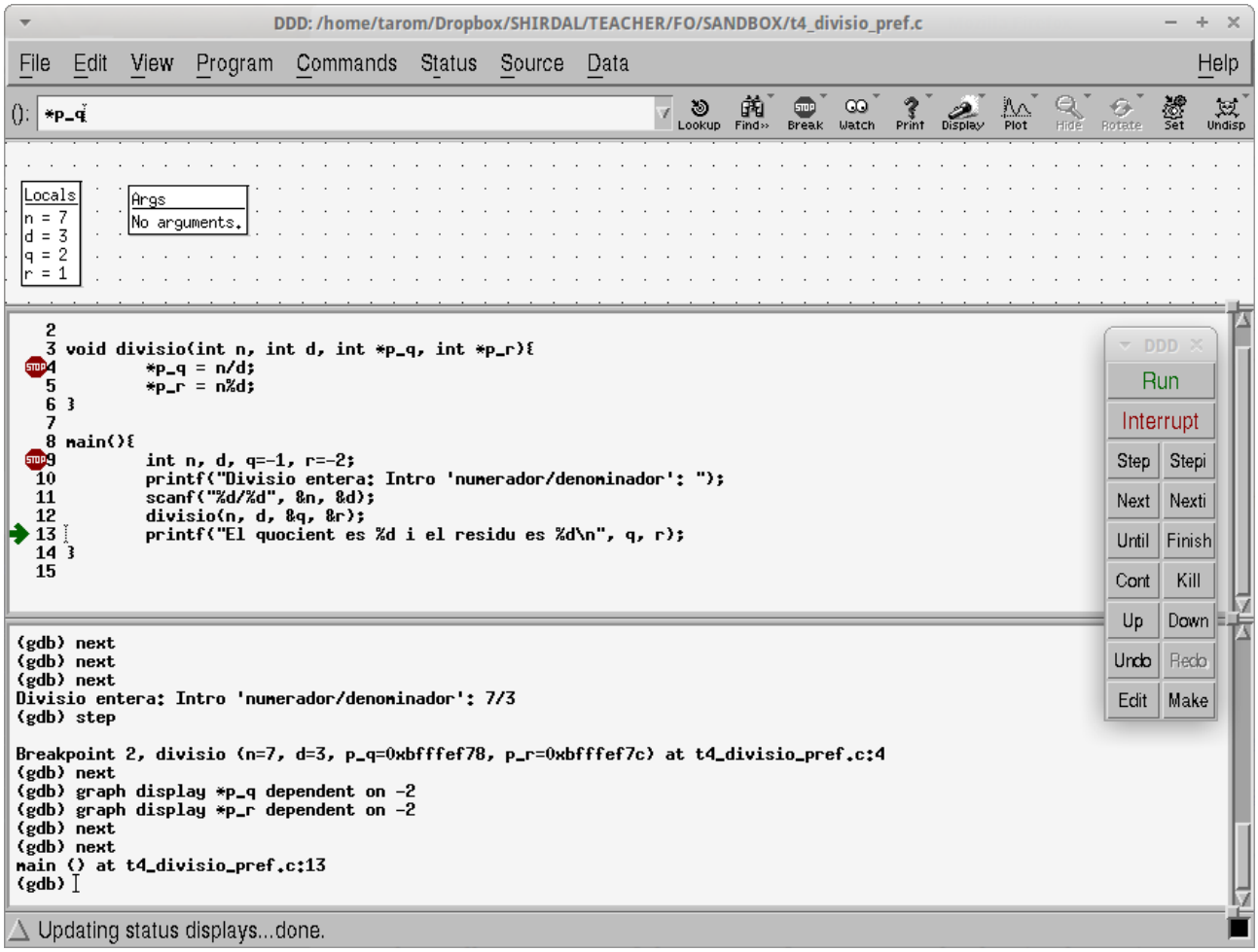
Clicko sobre el código fuente para que el “focus” no esté en la barra de botones de control y tecleo “CTRL+L” y “CTRL+U” para ver las variables locales y los argumentos con los que se ha invocado la función actual.
Veo que las variables tienen valores al azar porque aún no han sido inicializadas. Le doy a [Next] y veo que se inicializan (se marcan en amarillo).
Le doy al [Next] tres veces más hasta que me dice en la barra inferior “Next: waiting until GDB gets ready” (porque hemos llegado al “scanf()”.
Clicko en la ventana inferior y escribo “7/3” [Intro].
Y veo que las variables “n” y “d” se inicializan.



Ahora, si clickase [Next], ejecutaría todo el código de la función “divisio()”.
Para que “entre” dentro de la función clicko [Step].
Doble-clicko sobre “p_q” y “p_r” para que se vea que apuntan a la “q” y a la “r” del “main()”. Podemos ver que las variables que apuntan valen “-1” y “-2” tal y como las inicializamos en el momento de declararlas.
Le doy a [Next] y veo como se cambia el valor de “*p_q” al cociente de la división. “*p_q” es “aquello apuntado por el puntero “p_q”, es decir la “q” del contexto del “main()”.



Clicko otra vez en [Next] y ocurrirá lo mismo con “*p_r” (el residuo de la división).
Un [Next] más y retorno al main().



Otro [Next] y hago el printf() que imprime el cociente y el residuo. (Importante no darle al [Step] aquí porque no queremos entrar dentro del código del printf().)

DDD Ensamblador

Para ver el valor de los registros. Ejecuto (con [Run]) el programa. Antes le habré incluido algún breakpoint. Y cuando se pare puedo ir al menú “Status” y clicar a la opción “Registers”. Esto me enseñará todos los registros.
Si lo que quiero es ver uno sólo (por ejemplo %ecx), como si fuera una variable, clicaré en el “(gdb)” de la ventana inferior y teclearé:

```
graph display $ecx
```

Para ver el stack:

```
graph display `x /20xw $esp`
```

Esto muestra los 20 primeros long empezando por la cima de la pila (apuntada por %esp). Lo que no nos indica es donde está la base de la pila.

Si estamos dentro de una subrutina podemos hacer:

```
graph display `x /24xw $ebp-48`
```

Esto nos muestra los primeros 12 longs de la pila hasta la base de la pila (apuntada por %ebp) y los 12 longs de la pila por debajo de la base. En el ddd las 3 primeras lineas son los longs por encima de (%ebp).

Esto no se puede hacer en el main. Porque en el main %ebp vale 0. Habría que hacerlo con el valor de %esp antes de que se ejecute la primera instrucción. Si %esp vale 0xbfffef8c:

```
graph display `x /24xw 0xbfffef8c-48`
```

DEPURANDO con NEMIVER

Nemiver: Compilar y ejecutar

Primero compila un archivo fuente en lenguaje C (nombre_archivo.c en el ejemplo):
gcc -g -o nombre_archivo nombre_archivo.c

Depurar con nemiver

nemiver nombre_archivo &

- Si no tienes nemiver instalado en tu Linux, puedes instalarlo en Ubuntu y otras distribuciones de Linux basadas en Debian con:

sudo apt-get install nemiver

- Una vez que nemiver esté funcionando, recomiendo la siguiente configuración:
Editar -> Preferencias -> Diseño -> Panel de dos estados
Alt+1 // Inferior: Contexto
// tiene tres subventanas que puedes cambiar de tamaño
Alt+2 // Derecha: terminal de destino

- Cuando inicies nemiver ya tendrá un punto de interrupción al principio y puede presionar F6 (Siguiente) para ejecutar el programa línea por línea.
- Si olvidamos compilar con el "-g" aparecerá el código en ensamblador en lugar del código C. Sal y compila con "-g".
- printf() solo aparece en la terminal si termina en '\n' o si se invoca un scanf().
- Para escribir en la terminal tienes que hacer clic en ella para que el cursor quede lleno y parpadee.
- Puede agregar/eliminar puntos de interrupción (breakpoints) haciendo clic a la derecha de la línea número.

Comandos principales

F5: [Continuar] Continúa hasta el siguiente punto de interrupción.
Shift + F5: [Ejecutar o reiniciar] Se reinicia desde el principio
F6: [Siguiente] Ejecuta la siguiente línea.
F7: [Paso (entrada)] Si la siguiente línea es una llamada de función, ingrésela.
Shift + F7: [Salir] Se ejecuta hasta el final de la función actual.

Controles avanzados
F12: [Inspeccionar una expresión] Puedes escribir cosas como "a==b" y agrégarlos al monitor de expresiones.
Alt + 6: Muestra el monitor de expresiones.

Problemas con distribuciones recientes

No hay package de Nemiver en las distribuciones recientes de Ubuntu. Tienes que descargarte los siguientes packages:

https://packages.ubuntu.com/jammy/nemiver
https://packages.ubuntu.com/jammy/libgtkhex-3-0
https://packages.ubuntu.com/jammy/libgail-3-0

Instalarlos:

```
sudo dpkg -i libgtkhex-3-0_3.41.1-1_amd64.deb
sudo dpkg -i libgail-3-0_3.24.33-1ubuntu1_amd64.deb
sudo dpkg -i nemiver_0.9.6-1.2build1_amd64.deb
```

Problemas: Al empezar tienes que darle al botón de “Stop” para que la flecha amarilla aparezca. Y, al final del programa se queda colgado. No le dés al “step into” al final de una función o se quedará colgado.