SESION 8 - DEPURACIÓN DE PROGRAMAS (DANIEL PÉREZ RUIZ)

EJERCICIO 9.1: Compile los archivos main.cpp factorial.cpp hello.cpp y genere un ejecutable con el nombre ejemplo1. Lance gdb con dicho ejemplo y ejecútelo dentro del depurador. Describa la información que ofrece.

```
$ g++ -g main.cpp hello.cpp factorial.cpp -o ejemplo1
GNU gdb (Ubuntu/Linaro 7.2-1ubuntu11) 7.2
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Para las instrucciones de informe de errores, vea:
<http://www.gnu.org/software/gdb/bugs/>...
Leyendo símbolos desde /home/superjes/modulos/ejemplo1...hecho.
(gdb)
```

EJERCICIO 9.2: Usando la orden list muestre el código del programa principal y el de la función factorial utilizados en el ejercicio 1 (para ello utilice la orden help list).

```
(gdb) l main //Lineas del programa principal
(gdb) l factorial //Lineas de la funcion factorial
```

EJERCICIO 9.3: Ponga un punto de ruptura asociado a cada línea del programa fuente mainsesion09.cpp donde aparezca el comentario /* break */. Muestre información de todas las variables que se estén usando cada vez que en la depuración se detenga la ejecución. Muestre la información del contador de programa mediante $pc y el de la pila con $sp.

```
$ g++ -g main.cpp -o ejemploprueba
$ gdb ejemploprueba

(gdb) break 14 //Breakpoint en linea 14
Punto de interrupción 1 at 0x80485d3: file main.cpp, line 14.
(gdb) run
Starting program: /home/usuario/exemple

Breakpoint 1, cuenta (y=0) at main.cpp:17
17 return tmp;
(gdb) info locals //Valor de las variables locales
tmp = 2
(gdb) p/x $pc //Informacion del contador de programa
$2 = 0x80485d3
(gdb) p/x $ps //Informacion del contador de pila
$3 = 0x200202

(gdb) break 30 //Creo un breakpoint en linea 30
Punto de interrupción 1 at 0x80485ee: file main.cpp, line 30.
(gdb) run
Starting program: /home/usuario/exemple
Breakpoint 1, multiplica (x=3, y=2) at main.cpp:31
31 final = final + y;
(gdb) info locals
```

```
final = 0
i = 0
(gdb) p/x $pc
$3 = 0x80485ee
(gdb) p/x $ps
$4 = 0x200202
(gdb) break 43 //Breakpoint en linea 30
Punto de interrupción 1 at 0x8048613: file main.cpp, line 43.
(gdb) run
Starting program: /home/usuario/exemple

Breakpoint 1, main () at main.cpp:44
44 final1 = multiplica(3, 2);
(gdb) info locals
final1 = 134514411
final2 = 1174096
i = 3903476
(gdb) p/x $pc
$4 = 0x8048613
(gdb) p/x $ps
$5 = 0x200282

(gdb) break 47
Punto de interrupción 2 at 0x804862b: file main.cpp, line 47.
(gdb) continue
Continuando.

Breakpoint 2, main () at main.cpp:47
47 for (i = 0; i < 100; i ++)
(gdb) info locals
final1 = 6
final2 = 1174096
i = 3903476
```

EJERCICIO 9.4: Indique las órdenes necesarias para ver el valor de las variables final1 y final2 del programa generado en el ejercicio anterior en los puntos de ruptura correspondientes tras un par de iteraciones en el bucle for. Indique la orden para obtener el código ensamblador de la zona depurada

```
(gdb) break 43 //Breakpoint en linea 30
Punto de interrupción 1 at 0x8048613: file main.cpp, line 43.
(gdb) run
Starting program: /home/usuario/exemple

Breakpoint 1, main () at main.cpp:44
44 final1 = multiplica(3, 2);

(gdb) print final1
$1 = 134514411
(gdb) print final2
$2 = 1174096
(gdb) disassemble
Dump of assembler code for function main():
0x0804860a <+0>: push %ebp
```

```
0x0804860b <+1>: mov %esp,%ebp
0x0804860d <+3>: and $0xfffffff0,%esp
0x08048610 <+6>: sub $0x20,%esp
=> 0x08048613 <+9>: movl $0x2,0x4(%esp)
0x0804861b <+17>: movl $0x3,(%esp)
0x08048622 <+24>: call 0x80485d8 <multiplica(int, int)>
0x08048627 <+29>: mov %eax,0x18(%esp)
0x0804862b <+33>: movl $0x0,0x1c(%esp)
0x08048633 <+41>: jmp 0x804864a <main()+64>
0x08048635 <+43>: mov 0x1c(%esp),%eax
0x08048639 <+47>: mov %eax,(%esp)
0x0804863c <+50>: call 0x80485c4 <cuenta(int)>
0x08048641 <+55>: mov %eax,0x14(%esp)
0x08048645 <+59>: addl $0x1,0x1c(%esp)
0x0804864a <+64>: cmpl $0x63,0x1c(%esp)
0x0804864f <+69>: setle %al
0x08048652 <+72>: test %al,%al
0x08048654 <+74>: jne 0x8048635 <main()+43>
0x08048656 <+76>: mov 0x18(%esp),%eax
0x0804865a <+80>: mov %eax,0x4(%esp)
0x0804865e <+84>: movl $0x804a040,(%esp)
---Type <return> to continue, or q <return> to quit---
0x08048665 <+91>: call 0x8048498 <_ZNSolsEi@plt>
0x0804866a <+96>: movl $0x80487a0,0x4(%esp)
0x08048672 <+104>: mov %eax,(%esp)

0x08048675 <+107>: call 0x80484f8
<_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@plt>
0x0804867a <+112>: mov $0x0,%eax
0x0804867f <+117>: leave
0x08048680 <+118>: ret
End of assembler dump.

(gdb) continue
Continuando.
Breakpoint 2, main () at main.cpp:47
47 for (i = 0; i < 100; i ++)
(gdb) print final1
$3 = 6
(gdb) print final2
$4 = 1174096
(gdb) disassemble
Dump of assembler code for function main():
0x0804860a <+0>: push %ebp
0x0804860b <+1>: mov %esp,%ebp
0x0804860d <+3>: and $0xfffffff0,%esp
0x08048610 <+6>: sub $0x20,%esp
0x08048613 <+9>: movl $0x2,0x4(%esp)
0x0804861b <+17>: movl $0x3,(%esp)
0x08048622 <+24>: call 0x80485d8 <multiplica(int, int)>
0x08048627 <+29>: mov %eax,0x18(%esp)
=> 0x0804862b <+33>: movl $0x0,0x1c(%esp)
0x08048633 <+41>: jmp 0x804864a <main()+64>
```

```
0x08048635 <+43>: mov 0x1c(%esp),%eax
0x08048639 <+47>: mov %eax,(%esp)
0x0804863c <+50>: call 0x80485c4 <cuenta(int)>
0x08048641 <+55>: mov %eax,0x14(%esp)
0x08048645 <+59>: addl $0x1,0x1c(%esp)
0x0804864a <+64>: cmpl $0x63,0x1c(%esp)
0x0804864f <+69>: setle %al
0x08048652 <+72>: test %al,%al
0x08048654 <+74>: jne 0x8048635 <main()+43>
0x08048656 <+76>: mov 0x18(%esp),%eax
0x0804865a <+80>: mov %eax,0x4(%esp)
0x0804865e <+84>: movl $0x804a040,(%esp)
---Type <return> to continue, or q <return> to quit---
0x08048665 <+91>: call 0x8048498 <_ZNSolsEi@plt>
0x0804866a <+96>: movl $0x80487a0,0x4(%esp)
0x08048672 <+104>: mov %eax,(%esp)
0x08048675 <+107>: call 0x80484f8
<_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@plt>
0x0804867a <+112>: mov $0x0,%eax
0x0804867f <+117>: leave
0x08048680 <+118>: ret
End of assembler dump.
```

EJERCICIO 9.5: Considerando la depuración de los ejercicios anteriores, elimine todos los puntos de ruptura salvo el primero.

```
(gdb) delete breakpoint 2 3 4
```

EJERCICIO 9.6: Realice las acciones del ejercicio 3 y las del ejercicio 5 en un guion y ejecútelas de nuevo mediante la opción -x de gdb. ¿Sabría decir qué hace este programa con la variable final2?

```
/*guion.mdb*/
break 14
run
info locals
p/x $pc
p/x $ps
break 30
n
info locals
p/x $pc
p/x $ps
break 43
n
info locals
p/x $pc
p/x $ps
break 47
n
info locals
p/x $pc
```

```
p/x $ps
delete breakpoint 2 3 4
gdb -x guion.gdb ejemplo1
```

EJERCICIO 9.7: Realice la depuración del programa ejecutable obtenido a partir del archivo fuente ejsesion09.cpp. Utilizando gdb, trate de averiguar qué sucede y por qué no funciona.
Intente arreglar el programa.

```
#include <stdlib.h>
#include <stdio.h>
/*
Este programa trata de sumar una lista de números almacenados en la variable "vector" y
almacena el resultado en la variable "final".
*/

float suma (float x, float y) //Han de ser float para que no se pierda la
precisión
{
float tmp;
tmp = x + y;
return tmp;
}

int sumatoria (float vector[], int n)
{
int i;
float tmp; //Ha de ser float para que no se pierda la precisión

tmp = 0;
for (i = 0; i < n; i ++)
tmp = suma(tmp, vector[i]);
devuelto de la función suma

printf ("Suma = %d\n", tmp);

return tmp;
}

int main (void)
{
float final;
float vector[] = {0, 1, 2.3, 3.7, 4.10, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4};

final = sumatoria(vector, 15);

return 0;
}
```