

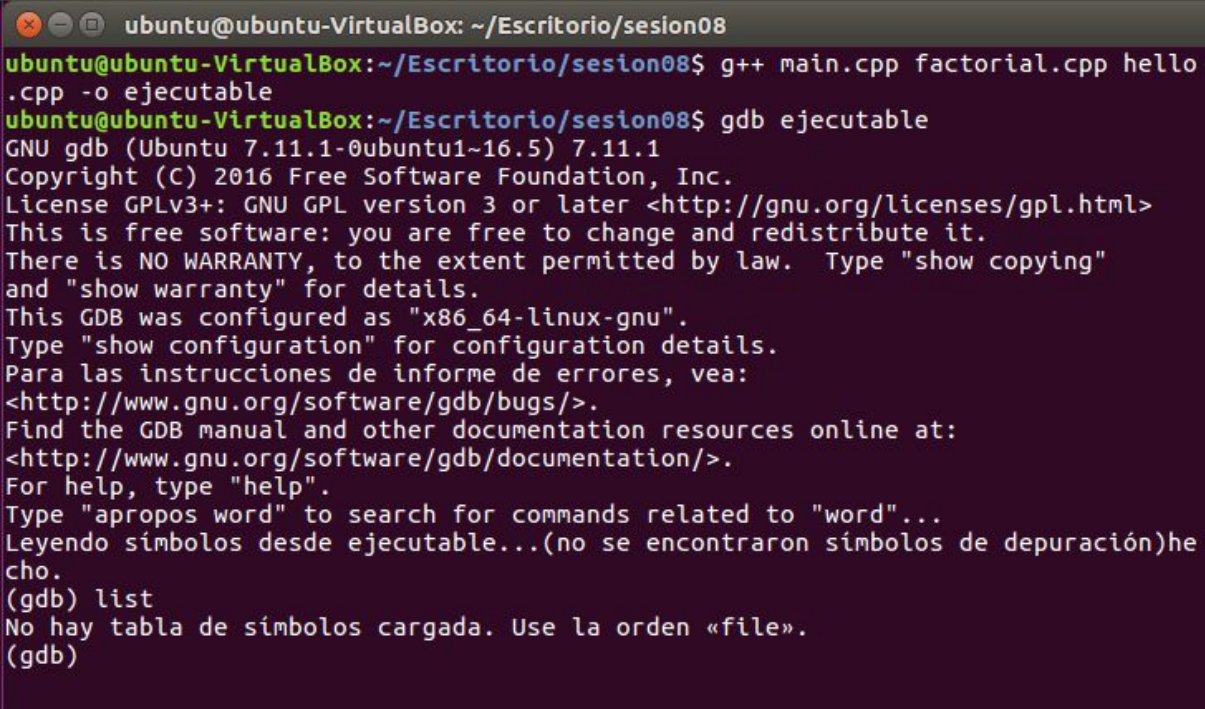
Ejercicio 1

Utilizando los archivos disponibles en la carpeta de la sesión 8 (main.cpp, factorial.cpp y hello.cpp). Ejecuta la siguiente orden:

```
g++ main.cpp factorial.cpp hello.cpp -o ejecutable
```

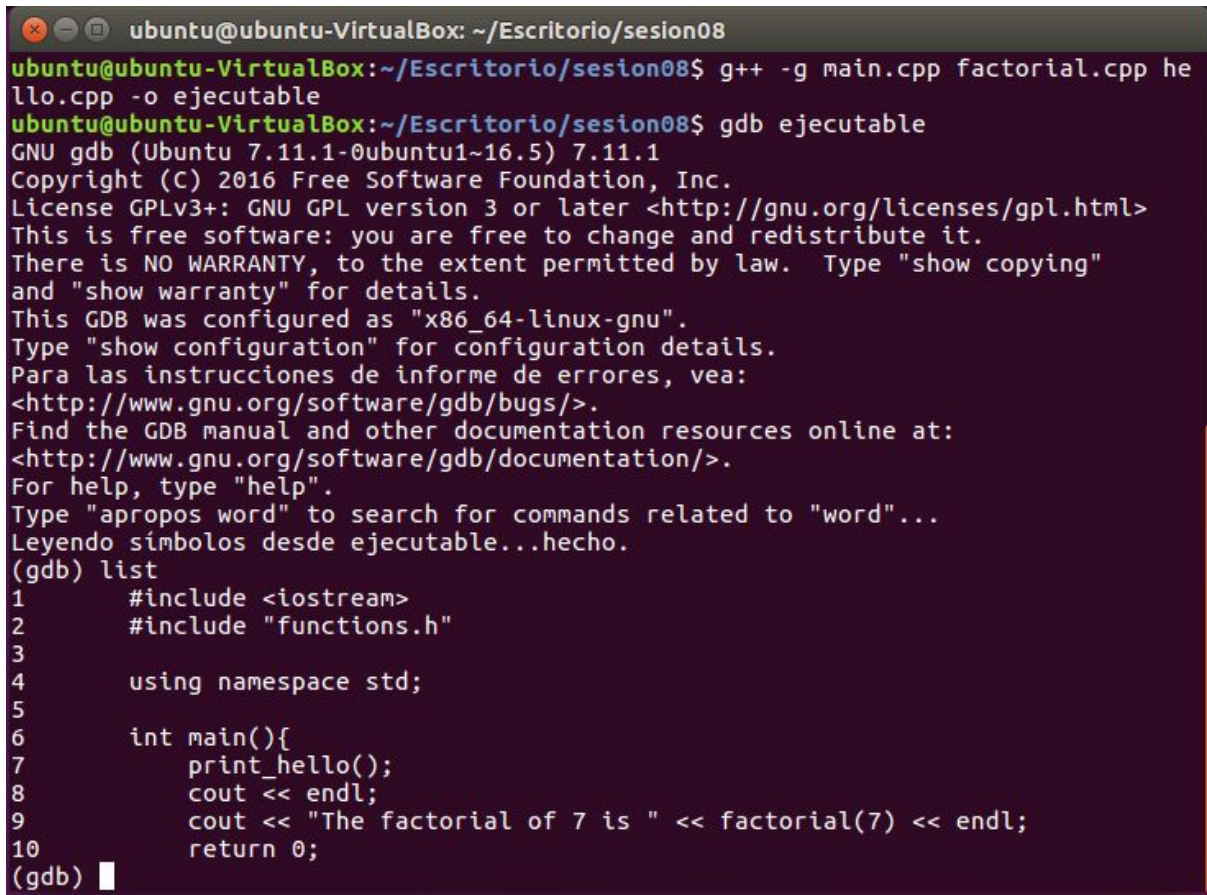
Una vez compilado el código, utiliza la herramienta gdb para ver el código del programa principal. Describe qué sucede al intentar ver las líneas del código del main y cómo harías para ver el código por medio del depurador. Indica cómo se puede salir del depurador.

Si ejecutamos la orden tal cual viene escrita en el enunciado, podemos observar en las imágenes más abajo adjuntadas como no se puede realizar la depuración. Sin embargo, para solucionarlo tal y como viene en la imagen (2) utilizamos la orden descrita en ella que varía por el “-g” lo cual nos permite depurar nuestro ejecutable. Para ver el código de nuestro main utilizamos la orden “list”. Cabe recalcar que para realizar la orden correcta hemos borrado el archivo “ejecutable” primero.



```
ubuntu@ubuntu-VirtualBox: ~/Escritorio/sesion08
ubuntu@ubuntu-VirtualBox:~/Escritorio/sesion08$ g++ main.cpp factorial.cpp hello
.cpp -o ejecutable
ubuntu@ubuntu-VirtualBox:~/Escritorio/sesion08$ gdb ejecutable
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 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 "x86_64-linux-gnu".
Type "show configuration" for configuration details.
Para las instrucciones de informe de errores, vea:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Leyendo símbolos desde ejecutable...(no se encontraron símbolos de depuración)he
cho.
(gdb) list
No hay tabla de símbolos cargada. Use la orden «file».
(gdb)
```

Imagen (1)



```
ubuntu@ubuntu-VirtualBox: ~/Escritorio/sesion08
ubuntu@ubuntu-VirtualBox:~/Escritorio/sesion08$ g++ -g main.cpp factorial.cpp he
llo.cpp -o ejecutable
ubuntu@ubuntu-VirtualBox:~/Escritorio/sesion08$ gdb ejecutable
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 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 "x86_64-linux-gnu".
Type "show configuration" for configuration details.
Para las instrucciones de informe de errores, vea:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Leyendo símbolos desde ejecutable...hecho.
(gdb) list
1      #include <iostream>
2      #include "functions.h"
3
4      using namespace std;
5
6      int main(){
7          print_hello();
8          cout << endl;
9          cout << "The factorial of 7 is " << factorial(7) << endl;
10         return 0;
(gdb) █
```

Imagen(2)

Ejercicio 2

¿Cómo harías para ejecutar la función factorial del ejercicio anterior?
Muestra una captura de pantalla indicando el resultado y cómo has llegado a él.

Para ejecutarlo hemos puesto un punto de ruptura en nuestra función factorial que es llamada por el main con el argumento 7 (factorial(7)). Posteriormente hemos utilizado la orden run para depurar del programa hasta el punto de ruptura encontrado en la línea 9 de nuestro main. Una vez llegado hasta este punto, utilizamos la orden step (o la next que es similar para nuestro caso) para avanzar paso a paso la depuración del programa. Como podemos observar cada vez que se realiza un step la depuración avanza en nuestro factorial.cpp hasta que se cumple por completo para nuestro argumento 7 y aunque sigamos utilizando step, solo nos devuelve la llave final de la función puesto que ya ha terminado de ejecutarse factorial(7). Todo esto se puede observar en la imagen (3).

```
ubuntu@ubuntu-VirtualBox: ~/Escritorio/sesion08
(gdb) break factorial
Punto de interrupción 1 at 0x40094f: file factorial.cpp, line 4.
(gdb) run
Starting program: /home/ubuntu/Escritorio/sesion08/ejecutable
Hello World!

Breakpoint 1, factorial (n=7) at factorial.cpp:4
4         if(n!=1){
(gdb) step
5         return(n * factorial(n-1));
(gdb) step

Breakpoint 1, factorial (n=6) at factorial.cpp:4
4         if(n!=1){
(gdb) step
5         return(n * factorial(n-1));
(gdb) step

Breakpoint 1, factorial (n=5) at factorial.cpp:4
4         if(n!=1){
(gdb) step
5         return(n * factorial(n-1));
(gdb) step

Breakpoint 1, factorial (n=4) at factorial.cpp:4
4         if(n!=1){
(gdb) step
5         return(n * factorial(n-1));
(gdb) step

Breakpoint 1, factorial (n=3) at factorial.cpp:4
4         if(n!=1){
(gdb) step
5         return(n * factorial(n-1));
(gdb) step

Breakpoint 1, factorial (n=2) at factorial.cpp:4
4         if(n!=1){
(gdb) step
5         return(n * factorial(n-1));
(gdb) step

Breakpoint 1, factorial (n=1) at factorial.cpp:4
4         if(n!=1){
(gdb) step
7         else return 1;
(gdb) step
8     }
(gdb) step
8     }
(gdb) step
8     }
```

Imagen (3)

Ejercicio 3

Compila el código del siguiente programa y haz todas las configuraciones necesarias con gdb para mostrar el valor de la variable final2 justo antes de que se le asigne ningún valor dentro del bucle for. Describe todos los pasos que has seguido e incluye una captura de pantalla de los mismos.

Para ver el valor de la variable final2 justo antes del que se le asigne ningún valor en el bucle for lo que he hecho es copiar el código adjuntado en el enunciado en un editor de texto y lo he declarado con la correspondiente extensión .cpp. Posteriormente lo he compilado con el comando “-g” para poder realizar la depuración del ejecutable correspondiente (ejecutable2). Utilizo la orden list para ver el código del programa, he utilizado la función break para añadir un punto de ruptura concretamente en la línea 31 que es la primera en la que aún el bucle no le ha asignado ningún valor a la variable en cuestión. Utilizo la orden “run” para ver depurar el programa y por último la orden print final2 para mostrar el valor de final2 que deseamos ver. Todo ello viene realizado en las capturas de pantalla adjuntadas correspondientes a las imágenes (4 y 5).



```
ubuntu@ubuntu-VirtualBox: ~/Escritorio/sesion08
ubuntu@ubuntu-VirtualBox:~/Escritorio/sesion08$ g++ -g main2.cpp -o ejecutable2
ubuntu@ubuntu-VirtualBox:~/Escritorio/sesion08$ gdb ejecutable2
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 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 "x86_64-linux-gnu".
Type "show configuration" for configuration details.
Para las instrucciones de informe de errores, vea:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Leyendo símbolos desde ejecutable2...hecho.
(gdb) list 1,36
1      #include <iostream>
2
3      /* Incrementa en 2 una variable */
4      int cuenta (int y) {
5          int tmp;
6          tmp = y + 2;
7          return tmp;
8      }
9
10     /* Calcula la multiplicación de dos números */
11     int multiplica (int x, int y) {
12         int final;
13         int i;
14
15         final = 0;
16         for (i = 0; i < x; i++) {
17             final = final + y;
18         }
19
20         return final;
21     }
22
23     int main (void) {
24         int final1;
25         int final2;
26         int i;
27
28         final1 = multiplica(3, 2);
29
30         for (i = 0; i < 100; i++)
31             final2 = cuenta(i);
32
33     ---Type <return> to continue, or q <return> to quit---<return>
33         std::cout << final1 << "\n";
```

Imagen(4)

```
ubuntu@ubuntu-VirtualBox: ~/Escritorio/sesion08
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Leyendo símbolos desde ejecutable2...hecho.
(gdb) list 1,36
1      #include <iostream>
2
3      /* Incrementa en 2 una variable */
4      int cuenta (int y) {
5          int tmp;
6          tmp = y + 2;
7          return tmp;
8      }
9
10     /* Calcula la multiplicación de dos números */
11     int multiplica (int x, int y) {
12         int final;
13         int i;
14
15         final = 0;
16         for (i = 0; i < x; i ++) {
17             final = final + y;
18         }
19
20         return final;
21     }
22
23     int main (void) {
24         int final1;
25         int final2;
26         int i;
27
28         final1 = multiplica(3, 2);
29
30         for (i = 0; i < 100; i ++)
31             final2 = cuenta(i);
32
33         ---Type <return> to continue, or q <return> to quit---<return>
34         std::cout << final1 << "\n";
35
36         return 0;
37     }
(gdb) break 31
Punto de interrupción 1 at 0x400823: file main2.cpp, line 31.
(gdb) run
Starting program: /home/ubuntu/Escritorio/sesion08/ejecutable2

Breakpoint 1, main () at main2.cpp:31
31         final2 = cuenta(i);
(gdb) print final2
$1 = 0
(gdb) █
```

Imagen(5)

Ejercicio 4

Siguiendo el ejercicio anterior, haz todas las configuraciones necesarias utilizando el depurador gdb para obtener el valor de la variable final2 cuando i vale 50. Muestra todos los comandos utilizados y el valor de las variables final2 e i.

Para realizar este ejercicio he vuelto a utilizar la orden “list” para previsualizar el código. Posteriormente he utilizado un punto de ruptura condicional en la línea 31, imponiéndole como condición la puesta por el enunciado, que i valga 50. Por último he utilizado las órdenes “print i y print final2” para mostrar el valor que toman ambas variables justo en ese preciso momento. Todo esto se puede observar en la Imagen(6).

```
ubuntu@ubuntu-VirtualBox: ~/Escritorio/sesion08
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Leyendo símbolos desde ejecutable2...hecho.
(gdb) list 1,36
1      #include <iostream>
2
3      /* Incrementa en 2 una variable */
4      int cuenta (int y) {
5          int tmp;
6          tmp = y + 2;
7          return tmp;
8      }
9
10     /* Calcula la multiplicación de dos números */
11     int multiplica (int x, int y) {
12         int final;
13         int i;
14
15         final = 0;
16         for (i = 0; i < x; i ++) {
17             final = final + y;
18         }
19
20         return final;
21     }
22
23     int main (void) {
24         int final1;
25         int final2;
26         int i;
27
28         final1 = multiplica(3, 2);
29
30         for (i = 0; i < 100; i ++)
31             final2 = cuenta(i);
32
33         std::cout << final1 << "\n";
34
35         return 0;
36     }
(gdb) break 31 if i == 50
Punto de interrupción 1 at 0x400823: file main2.cpp, line 31.
(gdb) run
Starting program: /home/ubuntu/Escritorio/sesion08/ejecutable2

Breakpoint 1, main () at main2.cpp:31
31     final2 = cuenta(i);
(gdb) print i
$1 = 50
(gdb) print final2
$2 = 51
(gdb) █
```

Imagen(6)

Ejercicio 5

Con el mismo código del ejercicio anterior no parece que la variable `final2` pueda ser mayor de 101. Utilizando el depurador gdb haz, que sin tocar la variable `final2`, esta variable tenga un valor por encima de 1000 al final del programa.

Describe todos los pasos que has seguido para conseguirlo e incluye la captura de pantalla correspondiente.

Para ello he utilizado la orden "List" para previsualizar el código. Como queremos que la variable `final2` tome al final de programa un valor superior a 1000, lo que he hecho es poner un punto de ruptura condicional en la línea 7 que se de si y solo si, el argumento de entrada para la función "cuenta" es igual a 99, lo cual solo ocurre cuando el valor "i" en el bucle "for" del main toma su último valor, es decir el 99 ($i < 100$). Posteriormente he realizado la depuración. Y justo antes de realizar el "return tmp" le he asignado a la variable tmp el valor 1001 con la orden "set variable tmp = 1001". Posteriormente he utilizado la orden "next" para que avance la depuración y justo al final del programa he utilizado la orden "print final2" para poder visualizar el valor que toma final2 en este caso en concreto cuando acaba el programa. Como se puede ver en las imagenes (7 y 8) final2 toma el valor 1001 al final del programa como se describía en el enunciado.



```
ubuntu@ubuntu-VirtualBox: ~/Escritorio/sesion08
ubuntu@ubuntu-VirtualBox:~/Escritorio/sesion08$ gdb ejecutable2
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 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 "x86_64-linux-gnu".
Type "show configuration" for configuration details.
Para las instrucciones de informe de errores, vea:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Leyendo símbolos desde ejecutable2...hecho.
(gdb) list 1,36
1      #include <iostream>
2
3      /* Incrementa en 2 una variable */
4      int cuenta (int y) {
5          int tmp;
6          tmp = y + 2;
7          return tmp;
8      }
9
10     /* Calcula la multiplicación de dos números */
11     int multiplica (int x, int y) {
12         int final;
13         int i;
14
15         final = 0;
16         for (i = 0; i < x; i++) {
17             final = final + y;
18         }
19
20         return final;
21     }
22
23     int main (void) {
24         int final1;
25         int final2;
26         int i;
27
28         final1 = multiplica(3, 2);
29
30         for (i = 0; i < 100; i++)
31             final2 = cuenta(i);
32
33         std::cout << final1 << "\n";
34
35         return 0;
```

Imagen(7)

```
ubuntu@ubuntu-VirtualBox: ~/Escritorio/sesion08
13     int i;
14
15     final = 0;
16     for (i = 0; i < x; i ++) {
17         final = final + y;
18     }
19
20     return final;
21 }
22
23 int main (void) {
24     int final1;
25     int final2;
26     int i;
27
28     final1 = multiplica(3, 2);
29
30     for (i = 0; i < 100; i ++)
31         final2 = cuenta(i);
32
33     std::cout << final1 << "\n";
34
35     return 0;
36 }
(gdb) break 7 if y== 99
Punto de interrupción 1 at 0x4007c6: file main2.cpp, line 7.
(gdb) run
Starting program: /home/ubuntu/Escritorio/sesion08/ejecutable2

Breakpoint 1, cuenta (y=99) at main2.cpp:7
7     return tmp;
(gdb) set variable tmp = 1001
(gdb) next
8     }
(gdb) next
main () at main2.cpp:30
30     for (i = 0; i < 100; i ++)
(gdb) next
33     std::cout << final1 << "\n";
(gdb) print final2
$1 = 1001
(gdb) next
6
35     return 0;
(gdb) next
36     }
(gdb) next
__libc_start_main (main=0x4007fc <main()>, argc=1, argv=0x7fffffffdf38,
init=<optimized out>, fini=<optimized out>, rtld_fini=<optimized out>,
stack_end=0x7fffffffdf28) at ../csu/libc-start.c:325
325     ../csu/libc-start.c: No existe el archivo o el directorio.
(gdb) █
```

Imagen(8)