

Debugging using GDB in Eclipse/CLion

Joeri De Koster / Maarten Vandercammen / Carmen Torres Lopez



What is GDB?

- “GNU Debugger”
- A debugger for several languages including C and C++
- It allows you to inspect what the program is doing at a certain point during execution.
- Errors like *segmentation faults* may be easier to find with the help of gdb.

Compiling programs to work with gdb

- Normally, you would compile a program like:

```
$ gcc [flags] <source_files> -o <output>
```

- For debugging add a -g option to add debugging support to the output binary

```
$ gcc [flags] -g <source_files> -o <output>
```

Example (terminal)

include debug info
during compilation

```
$ gcc -g hello.c -o hello
```

```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello World!\n");
5     return 0;
6 }
```



hello.c

```
$ gdb hello
(gdb) target create "hello"
Current executable set to 'hello' (x86_64).
(gdb)
```

On OS X, use lldb
instead of gdb:
lldb hello

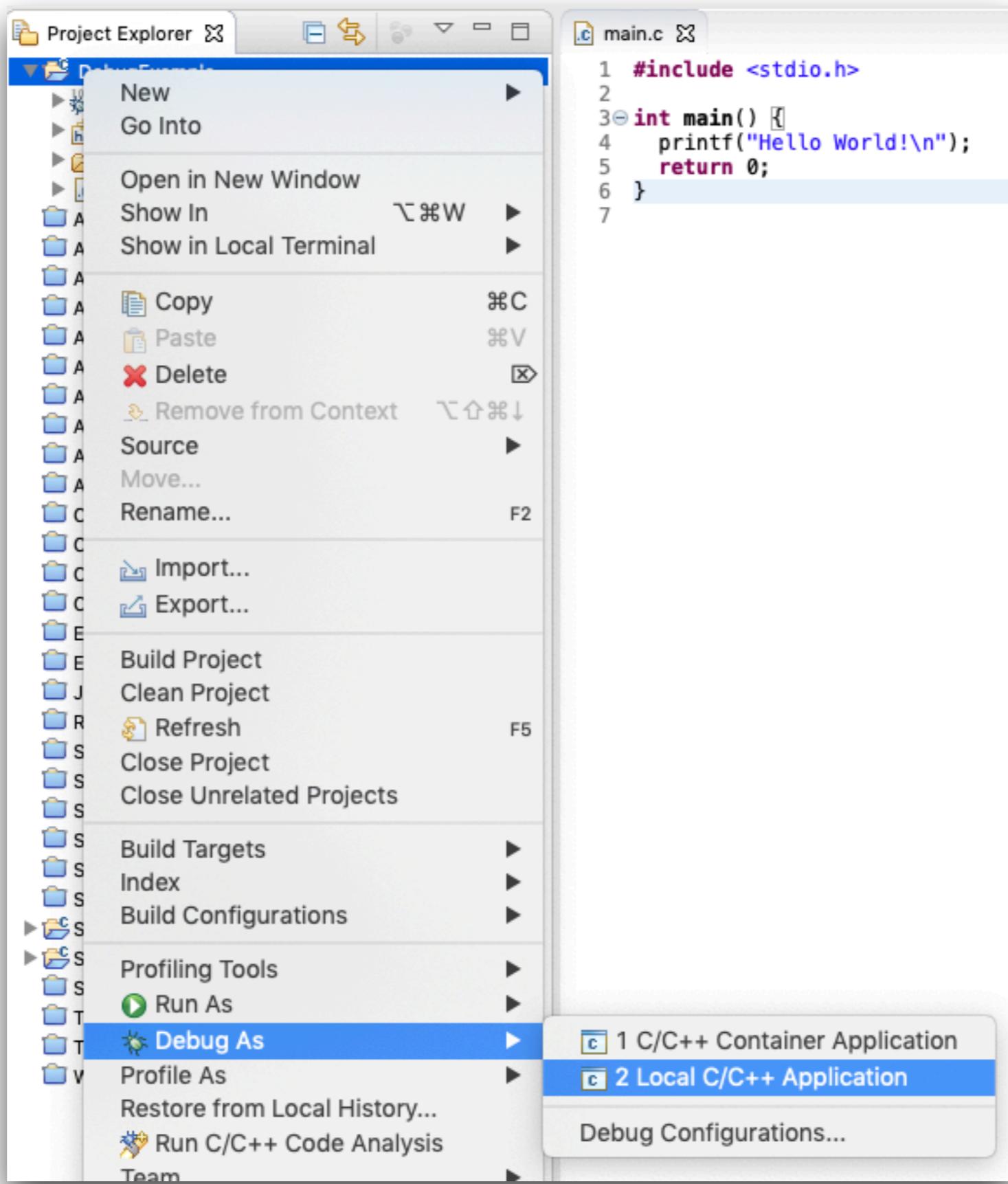
once the binary is
loaded you can run
the program

run the debugger
and load the binary

You might have to follow the
steps described at:
[https://sourceware.org/gdb/
wiki/PermissionsDarwin](https://sourceware.org/gdb/wiki/PermissionsDarwin)

```
(gdb) run
Process 9348 launched: 'hello' (x86_64)
Hello World!
Process 9348 exited with status = 0 (0x00000000)
```

Example (Eclipse)



Example (CLion)

Documentation: <https://www.jetbrains.com/help/clion/using-breakpoints.html#breakpoint-properties>

Debug

1. Run the program in Debug mode

The screenshot shows the CLion IDE interface with the following details:

- Project View:** Shows the project structure with files: CMakeLists.txt, main.c, and cmake-build-debug.
- Editor:** The main code editor displays the file `main.c`. A red circular breakpoint marker is placed on line 6, which contains the assignment `acc *= n;`.
- Toolbars and Buttons:** The toolbar at the top right includes a "Run" button (green triangle), a "Debug" button (highlighted with a black box and a callout bubble), and other build-related icons.
- Status Bar:** The bottom status bar shows the message "Build finished in 444 ms (today 19:52)" and the current directory "C: demoDebug | Debug".

Setting breakpoints (terminal)

```
(gdb) b hello.c:4  
Breakpoint 1: where = test`main + 39 at hello.c:4, address =  
0x0000000100000f77
```

```
(gdb) b hello.c:4
```

```
(gdb) b 4
```

```
(gdb) b main
```

use the ‘help’ command for more information

```
(gdb) help [command]
```

Running until a breakpoint (terminal)

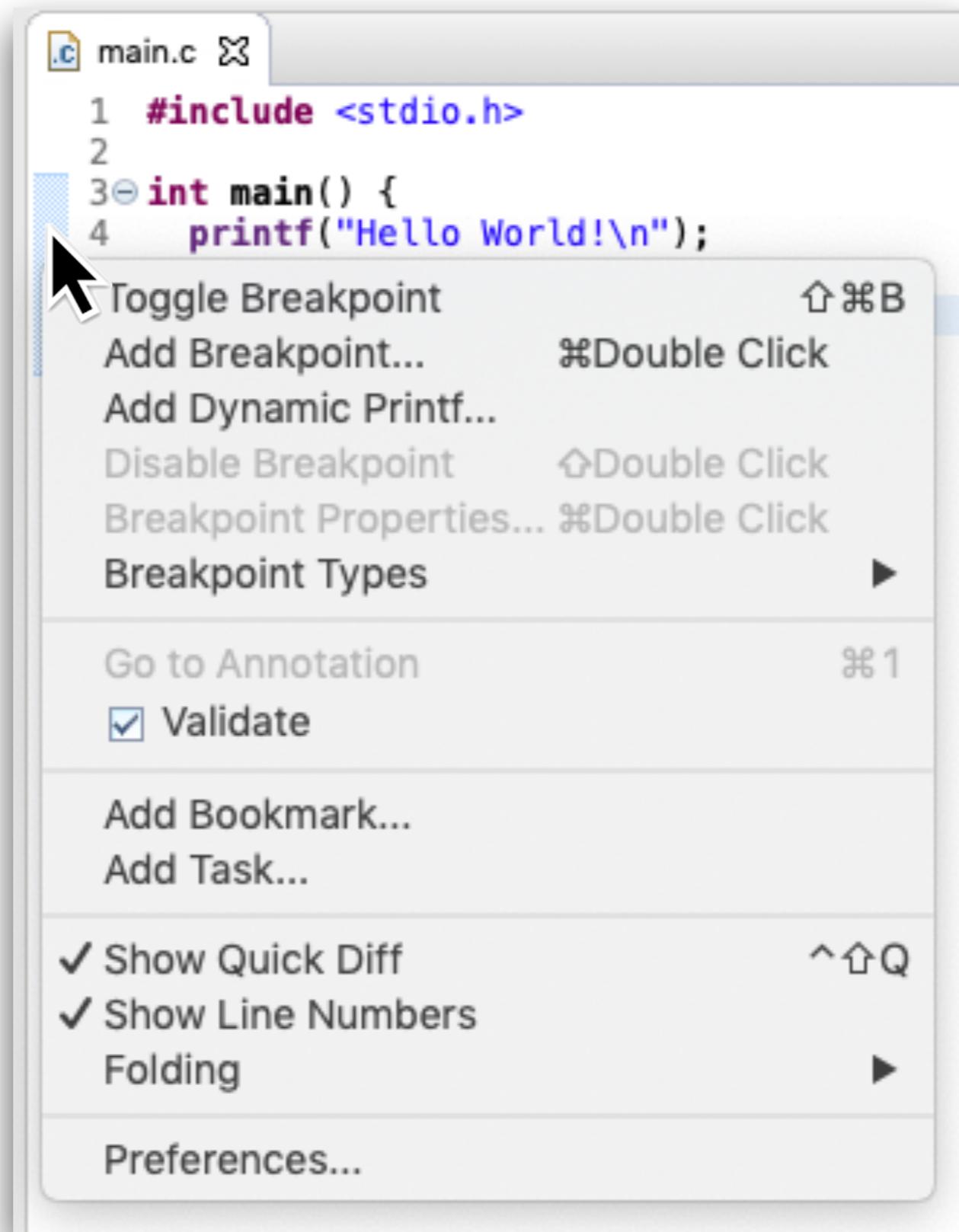
- Once you've set a breakpoint, you can try using the **run** command again. This time, it should stop where you tell it to (unless a fatal error occurs before reaching that point).
- You can proceed onto the next breakpoint by typing “**continue**” (Typing **run** again would restart the program from the beginning, which isn't very useful.) It allows you to inspect what the program is doing at a certain point during execution.

```
(gdb) run
Process 9348 launched: 'hello' (x86_64)
(gdb) continue
Hello World!
Process 9348 exited with status = 0 (0x00000000)
```

```
(gdb) r
```

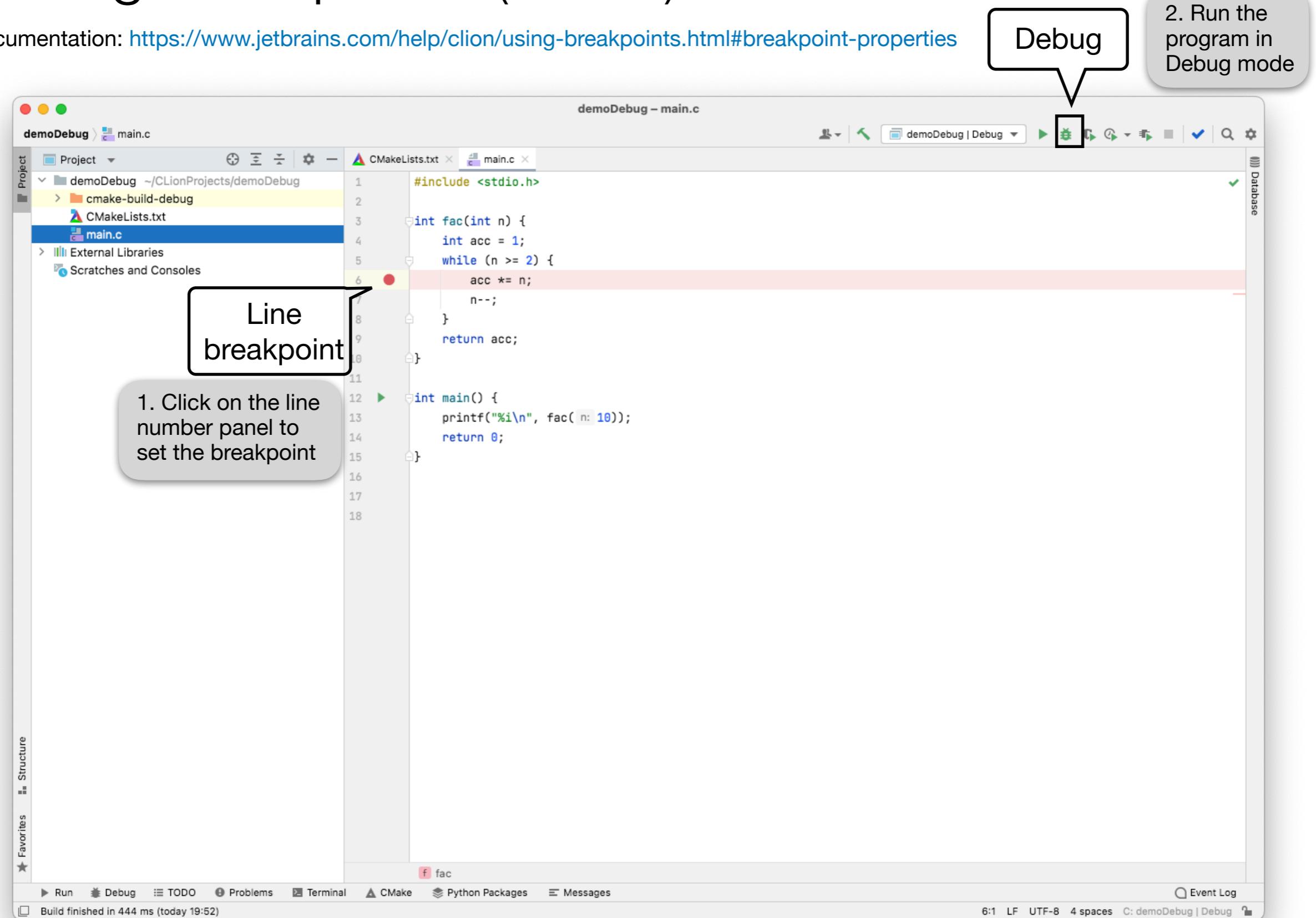
```
(gdb) c
```

Setting breakpoints (Eclipse)



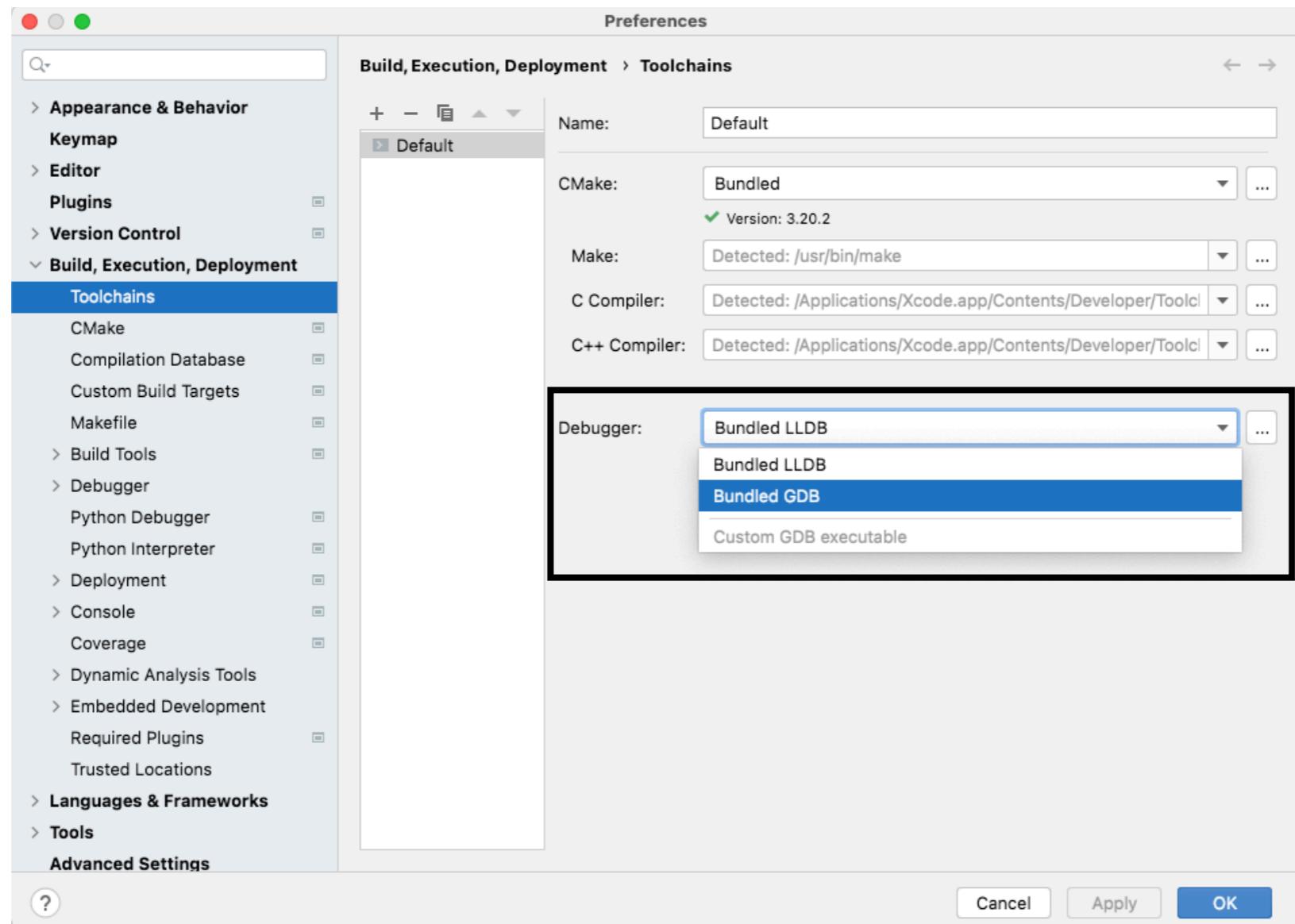
Setting breakpoints (CLion)

Documentation: <https://www.jetbrains.com/help/clion/using-breakpoints.html#breakpoint-properties>



CLion Optional: GDB/LLDB configuration

Menu CLion/Preferences



Stepping through the code (terminal)

- You can single-step (execute just the next line of code) by typing “**step**”. This gives you really fine-grained control over how the program proceeds. You can do this a lot...

```
(gdb) step
```

```
(gdb) s
```

- Similar to “**step**,” the “**next**” command single-steps as well, except this one doesn’t execute each line of a sub-routine, it just treats it as one instruction.

```
(gdb) next
```

```
(gdb) n
```

press ‘Enter’ to repeat
a command

- To run until the current function is finished, you can use the **finish** command

```
(gdb) finish
```

```
(gdb) fin
```

Printing and inspecting (terminal)

- The **print** command prints the value of the variable or expression specified

```
(gdb) print my_var
```

```
(gdb) p my_var
```

- The **backtrace** command prints out a stack trace of the current execution

```
(gdb) backtrace
```

```
(gdb) bt
```

- The **list** command prints out ten lines after or around a line number

```
(gdb) list 4
```

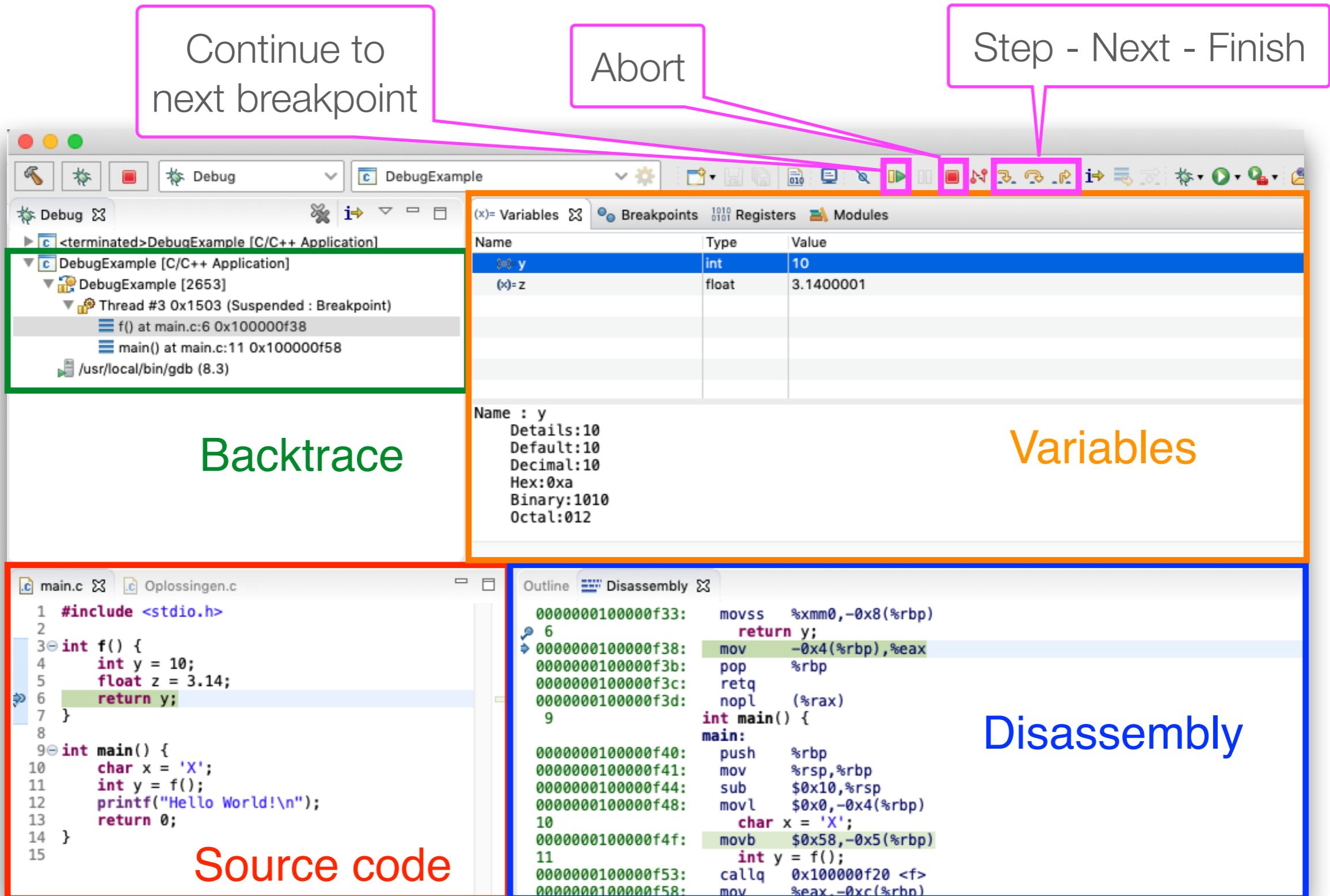
```
(gdb) l 4
```

Conditional breakpoints (terminal)

- Just like regular breakpoints, except that you get to specify some criterion that must be met for the breakpoint to trigger. We use the same break command as before:

```
(gdb) b hello.c:4 if my_var >= 10
```

Debugging (Eclipse)



Debugging (CLion)

Documentation: <https://www.jetbrains.com/help/clion/debugging-code.html>

Stop

3. Use stepping commands to navigate the program's execution

Stepping commands

Resume

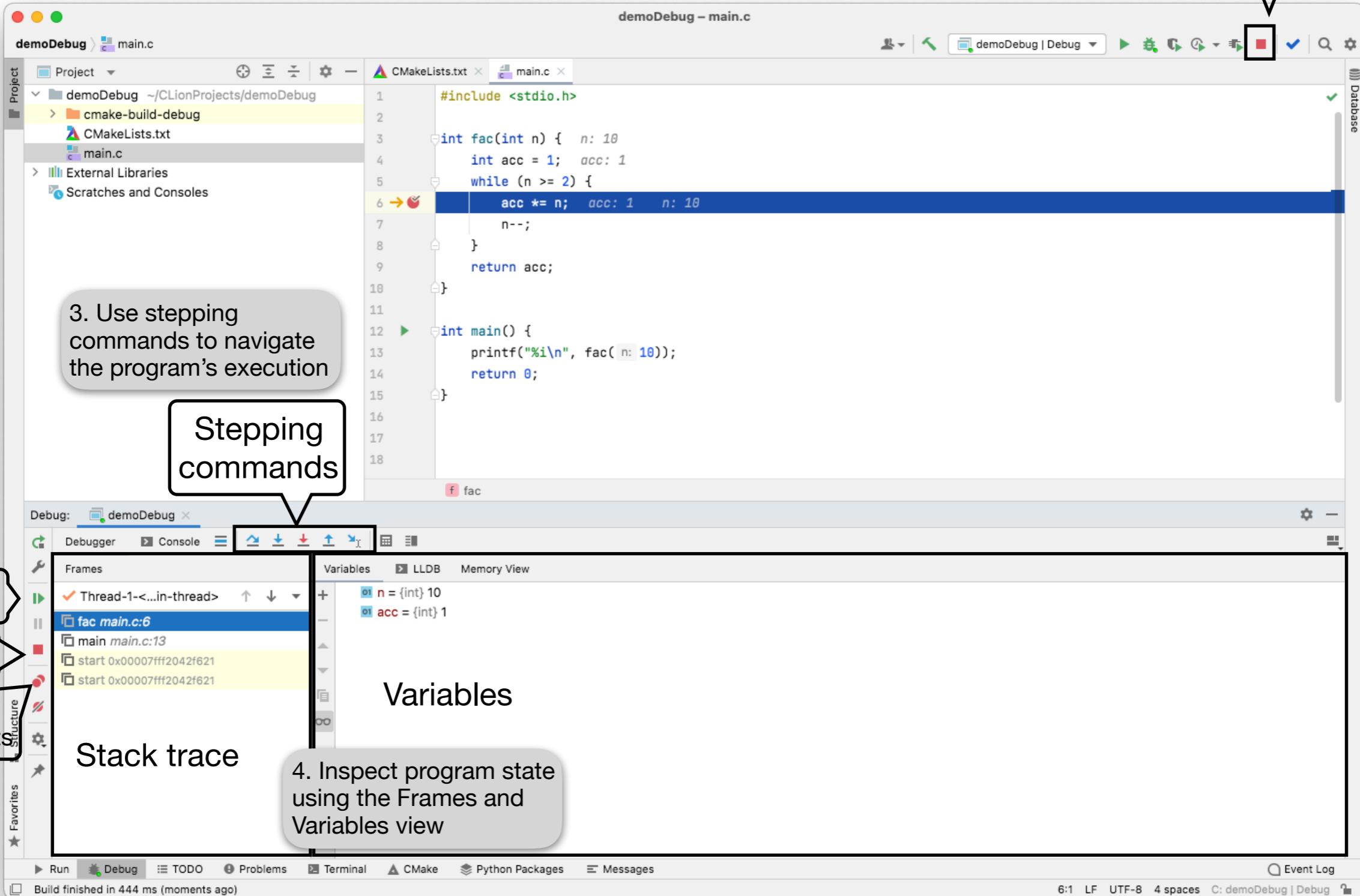
Stop

View
breakpoints

Variables

Stack trace

4. Inspect program state using the Frames and Variables view



Using the GDB/LLDB console in CLion

Documentation: <https://www.jetbrains.com/help/clion/debugger-console.html>

