

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

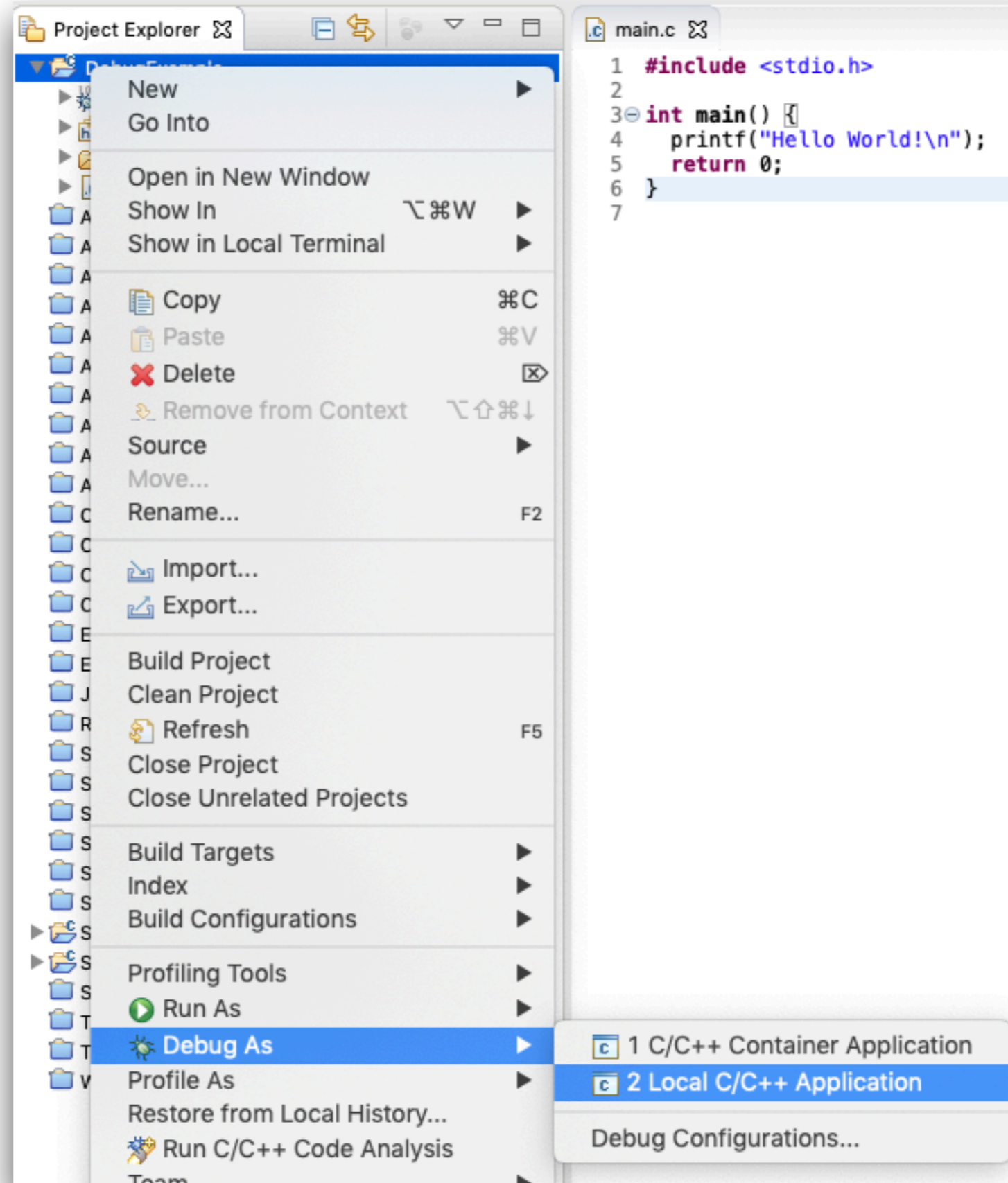
once the binary is
loaded you can run
the program

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)

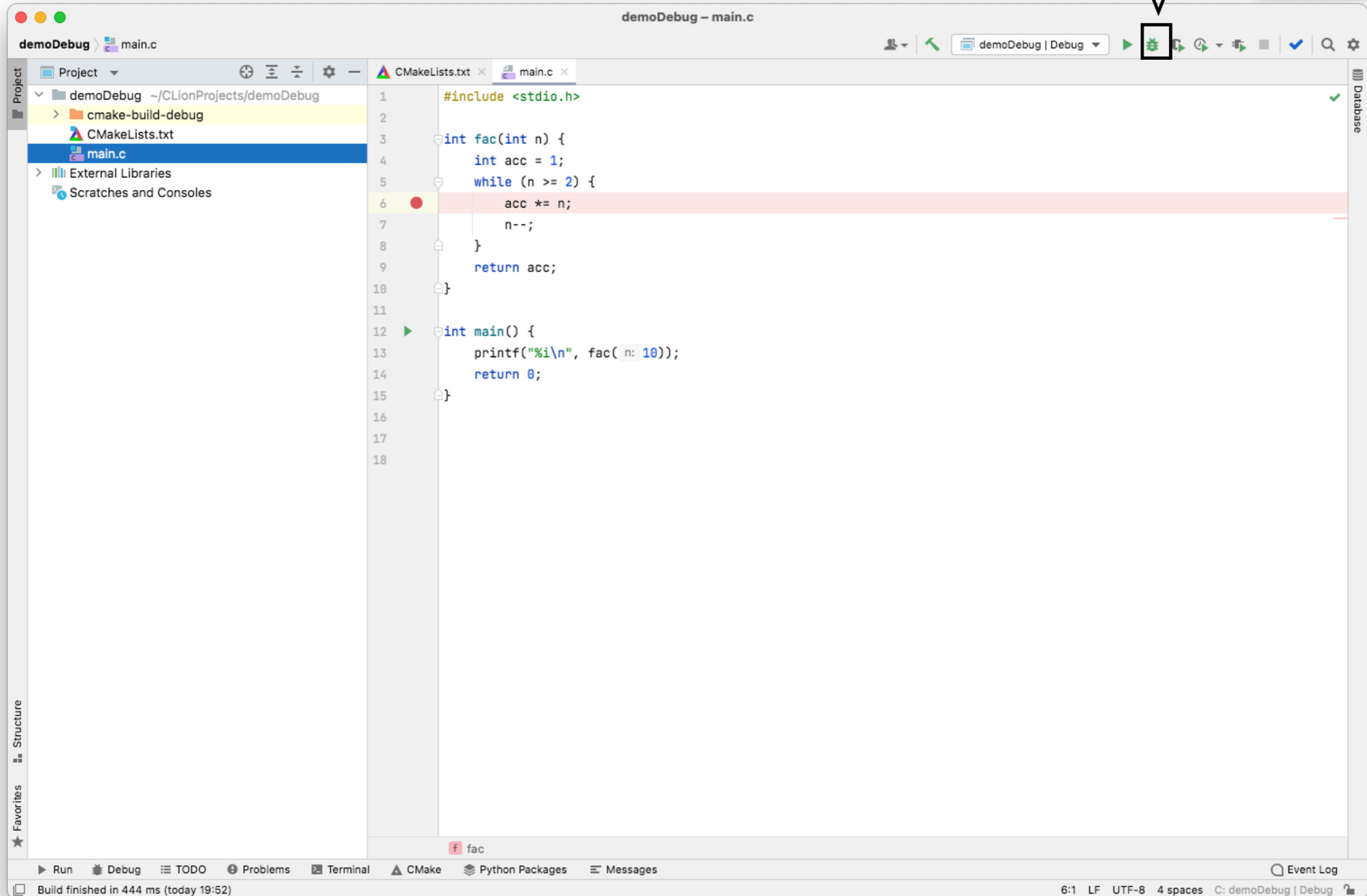


Example (CLion)

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

Debug

1. Run the program in Debug mode



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)

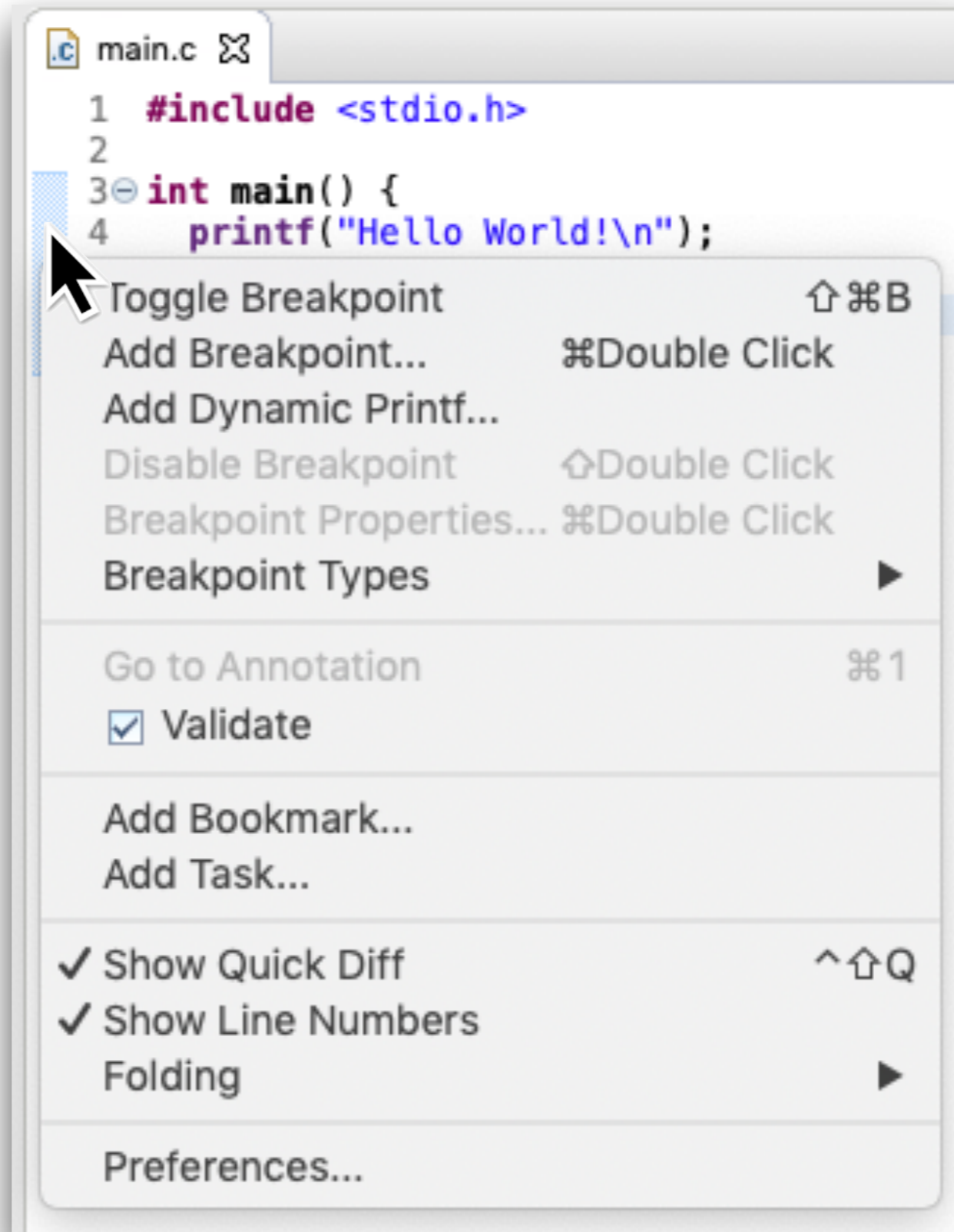
- Once you've set a breakpoint, you can try using the **run** command again. This time, it should stop where you tell it to (unless a fatal error occurs before reaching that point).
- You can proceed onto the next breakpoint by typing **"continue"** (Typing **run** again would restart the program from 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)

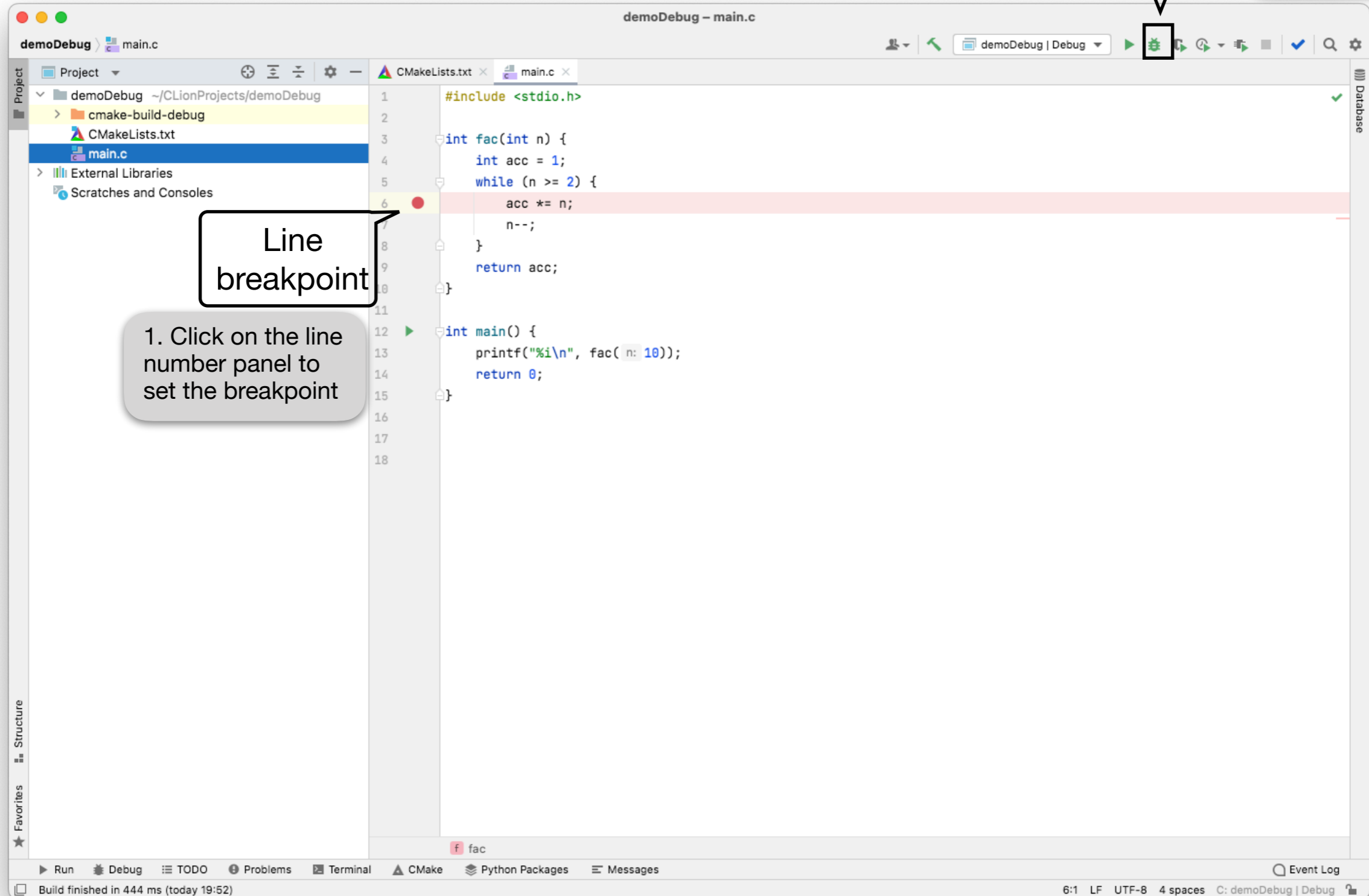


Setting breakpoints (CLion)

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

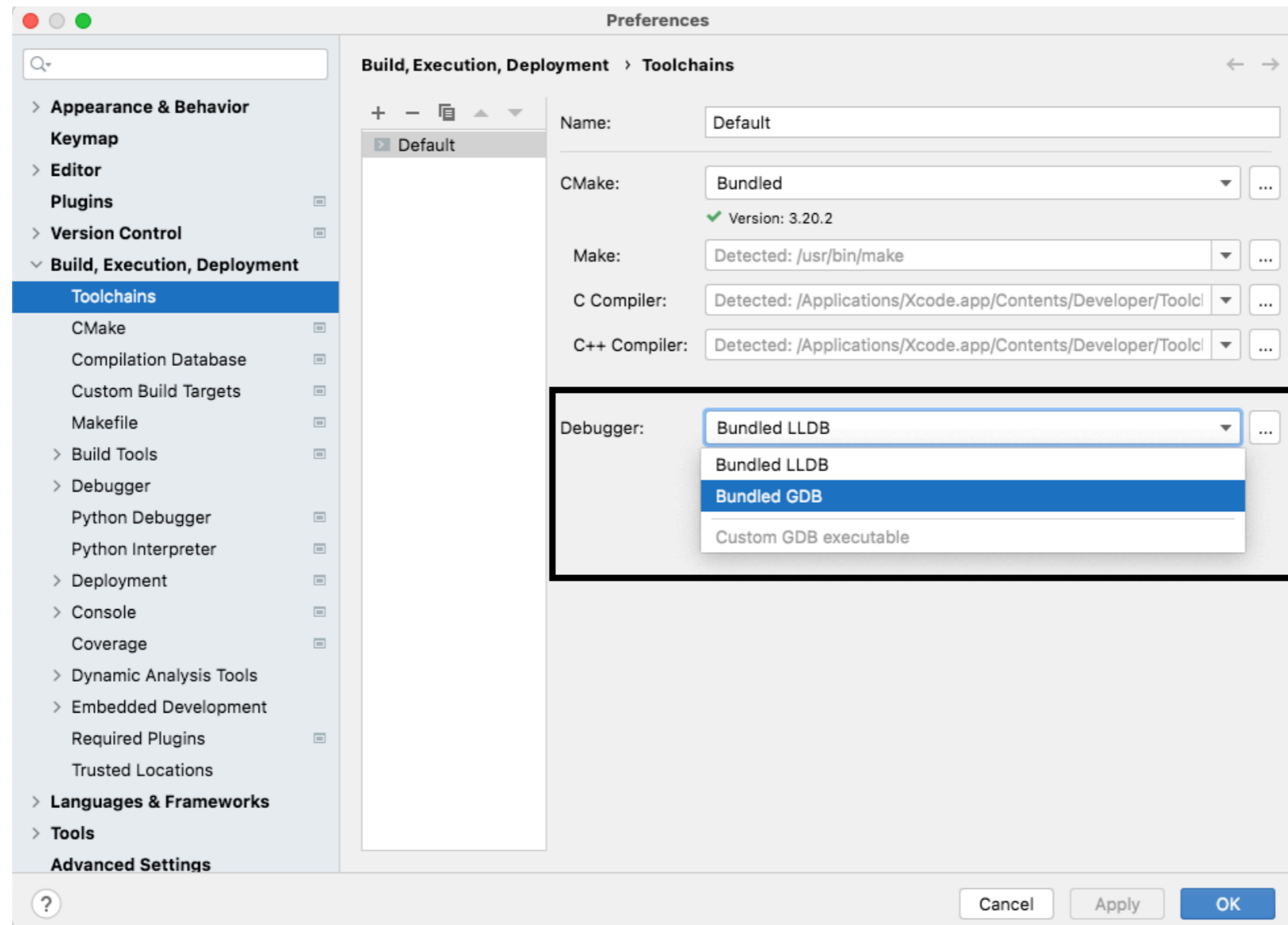
Debug

2. Run the program in Debug mode



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)

Continue to
next breakpoint

Abort

Step - Next - Finish

The screenshot shows the Eclipse IDE in the Debug perspective. The top toolbar contains several buttons, with callouts pointing to specific ones: 'Continue to next breakpoint' (a green play button), 'Abort' (a red square button), and 'Step - Next - Finish' (a blue play button with a right arrow). The main interface is divided into several panes:

- Debug Console:** Shows the current state of the debugged application, including the thread name, the current function, and the line of code being executed.
- Variables:** A table showing the current values of variables in the scope. The table has columns for Name, Type, and Value.
- Backtrace:** A list of the current stack frames, showing the function names and the line numbers where the program was stopped.
- Source code:** A text editor showing the source code of the program being debugged. The current line of code is highlighted.
- Disassembly:** A text editor showing the assembly code corresponding to the source code. The current instruction is highlighted.

Name	Type	Value
y	int	10
z	float	3.1400001

```
1 #include <stdio.h>
2
3 int f() {
4     int y = 10;
5     float z = 3.14;
6     return y;
7 }
8
9 int main() {
10    char x = 'X';
11    int y = f();
12    printf("Hello World!\n");
13    return 0;
14 }
15
```

```
0000000100000f33: movss %xmm0,-0x8(%rbp)
6      return y;
0000000100000f38: mov -0x4(%rbp),%eax
0000000100000f3b: pop %rbp
0000000100000f3c: retq
0000000100000f3d: nopl (%rax)
9      int main() {
main:
0000000100000f40: push %rbp
0000000100000f41: mov %rsp,%rbp
0000000100000f44: sub $0x10,%rsp
0000000100000f48: movl $0x0,-0x4(%rbp)
10     char x = 'X';
0000000100000f4f: movb $0x58,-0x5(%rbp)
11     int y = f();
0000000100000f53: callq 0x100000f20 <f>
0000000100000f58: mov %eax,-0xc(%rbp)
```

Debugging (CLion)

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

Stop

demoDebug - main.c

Project: demoDebug ~/CLionProjects/demoDebug

- cmake-build-debug
- CMakeLists.txt
- main.c
- External Libraries
- Scratches and Consoles

```
1 #include <stdio.h>
2
3 int fac(int n) {  n: 10
4     int acc = 1;  acc: 1
5     while (n >= 2) {
6         acc *= n;  acc: 1  n: 10
7         n--;
8     }
9     return acc;
10 }
11
12 int main() {
13     printf("%i\n", fac( n: 10));
14     return 0;
15 }
```

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

Stepping commands

Resume

Stop

View breakpoints

Stack trace

4. Inspect program state using the Frames and Variables view

Variables

Variable	Value
n	{int} 10
acc	{int} 1

Build finished in 444 ms (moments ago)

6:1 LF UTF-8 4 spaces C: demoDebug | Debug

Using the GDB/LLDB console in CLion

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

