Universidad Carlos III de Madrid

Industrial Informatics Lab Session

Bachelor in Industrial Electronics and Automation

2016-2017

Lab 1: Introduction to Qt Creator development environment

Teaching Staff:

Juan Carlos Gonzalez Victores

Mohamed Abderrahim

Lab 1 - Introduction to Qt Creator development environment

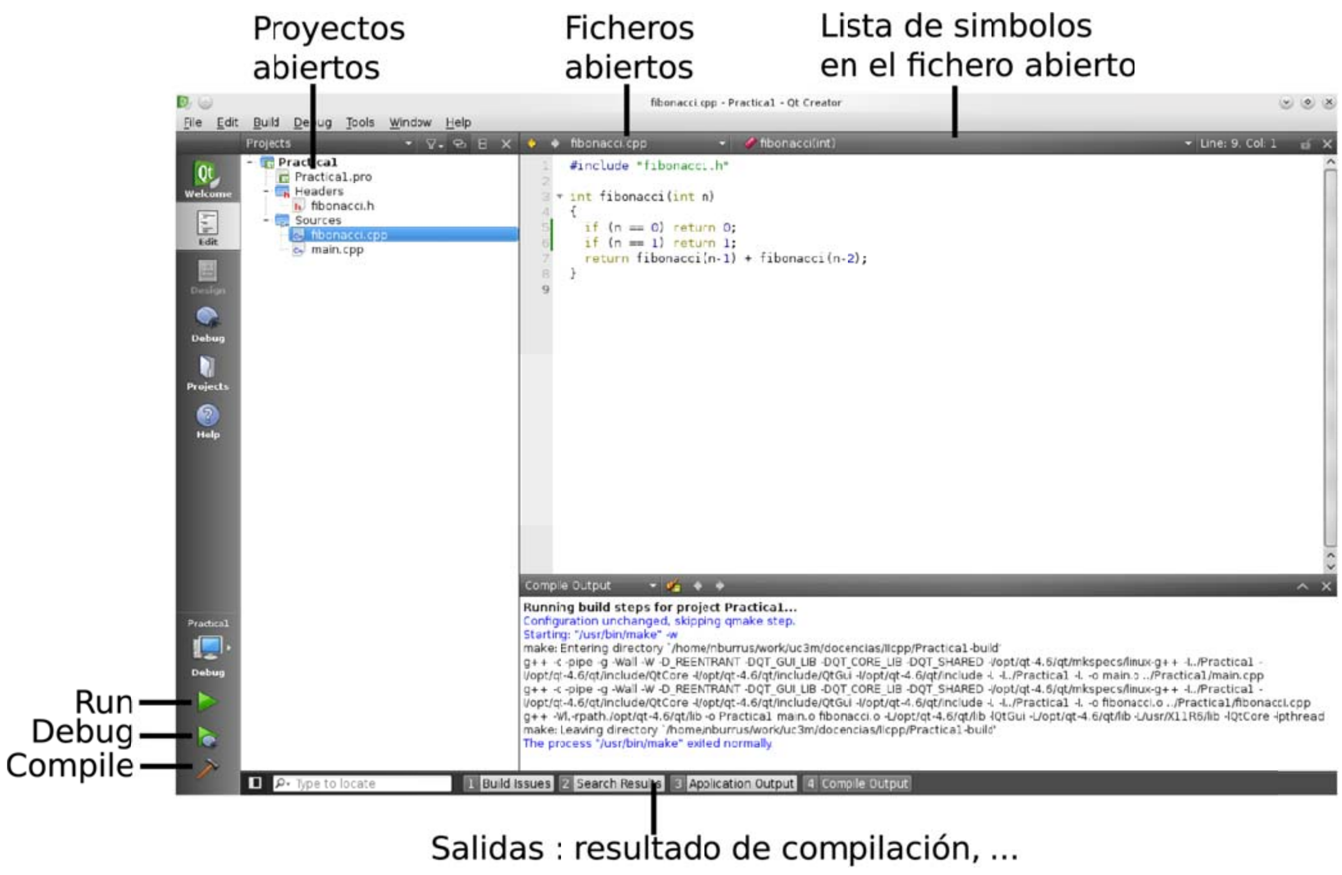
To program effectively in C++, most programmers use IDEs [Integrated Development Environments] that facilitate editing, UI design, compiling (or building), debugging and execution of the developed programs. Among the most popular are Eclipse CDT (open source, Windows/Linux/Mac), QtCreator (open source, Windows/Linux/Mac), Microsoft Visual Studio (Proprietary, paid and for Windows only) and Apple Xcode (Proprietary, paid and for Mac only).

During the lab session of this course we will work with QtCreator, as it is quite simple to use and works on Linux OS.

The objective of this lab session is to familiarise yourself with this tool, and to write and implement your first C++ program using QtCreator.

1. Introduction of the main window

1.1. Start QtCreator, select the ***`Edit`*** tab to go to the main window of the environment.



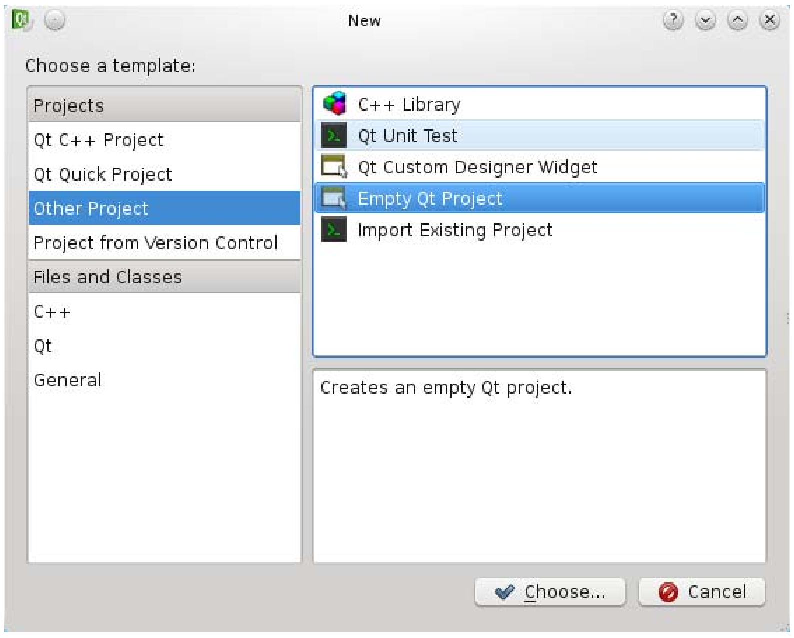
```
#include "fibonacci.h"

int fibonacci(int n)
{
  if (n == 0) return 0;
  if (n == 1) return 1;
  return fibonacci(n-1) + fibonacci(n-2);
}
```

```
Running build steps for project Practical...
Configuration unchanged, skipping qmake step.
Starting: "/usr/bin/make" -w
make: Entering directory `/home/nburrus/work/uc3m/docencias/llcpp/Practical-build'
g++ -c -pipe -g -Wall -W -D_REENTRANT -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/opt/qt-4.6/qt/mkspecs/linux-g++ -I../Practical -
I/opt/qt-4.6/qt/include/QtCore -I/opt/qt-4.6/qt/include/QtGui -I/opt/qt-4.6/qt/include -I. -I../Practical -I. -o main.o ../Practical/main.cpp
g++ -c -pipe -g -Wall -W -D_REENTRANT -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/opt/qt-4.6/qt/mkspecs/linux-g++ -I../Practical -
I/opt/qt-4.6/qt/include/QtCore -I/opt/qt-4.6/qt/include/QtGui -I/opt/qt-4.6/qt/include -I. -I../Practical -I. -o fibonacci.o ../Practical/fibonacci.cpp
g++ -Wl,-rpath,/opt/qt-4.6/qt/lib -o Practical main.o fibonacci.o -L/opt/qt-4.6/qt/lib -lQtGui -L/opt/qt-4.6/qt/lib -L/usr/X11R6/lib -lQtCore -lpthread
make: Leaving directory `/home/nburrus/work/uc3m/docencias/llcpp/Practical-build'
The process "/usr/bin/make" exited normally.
```

2. Create a new project

The first step is to create a new project. To do this:

1. Select ***`File/New File or Project`***.
2. Select ***`Other Project → Empty Qt Project`***.

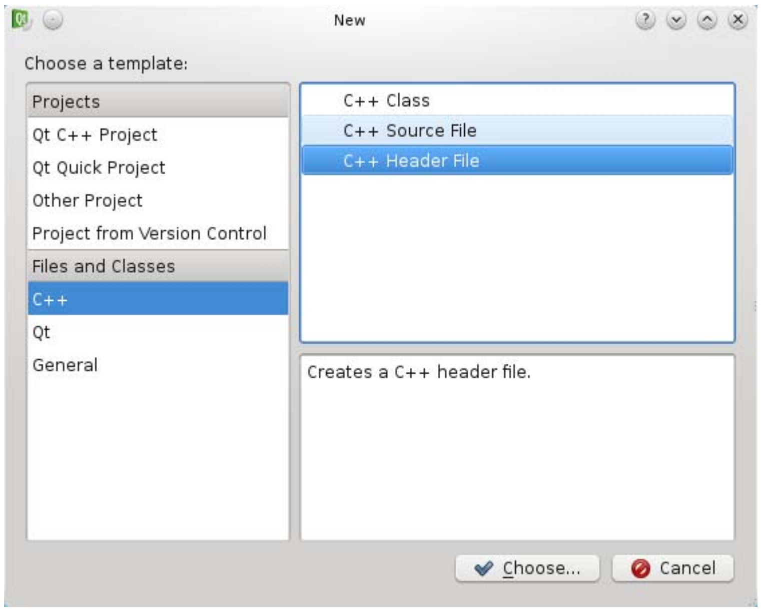


Creates an empty Qt project.

3. Provide a name for the project as ***Lab1*** and choose a destination folder for your project. Click next until finish.

We now have an empty project. We will add two files (one for definition and one for implementation) to calculate the Fibonacci series of a given integer.

4. Select ***File/New File or Project***.

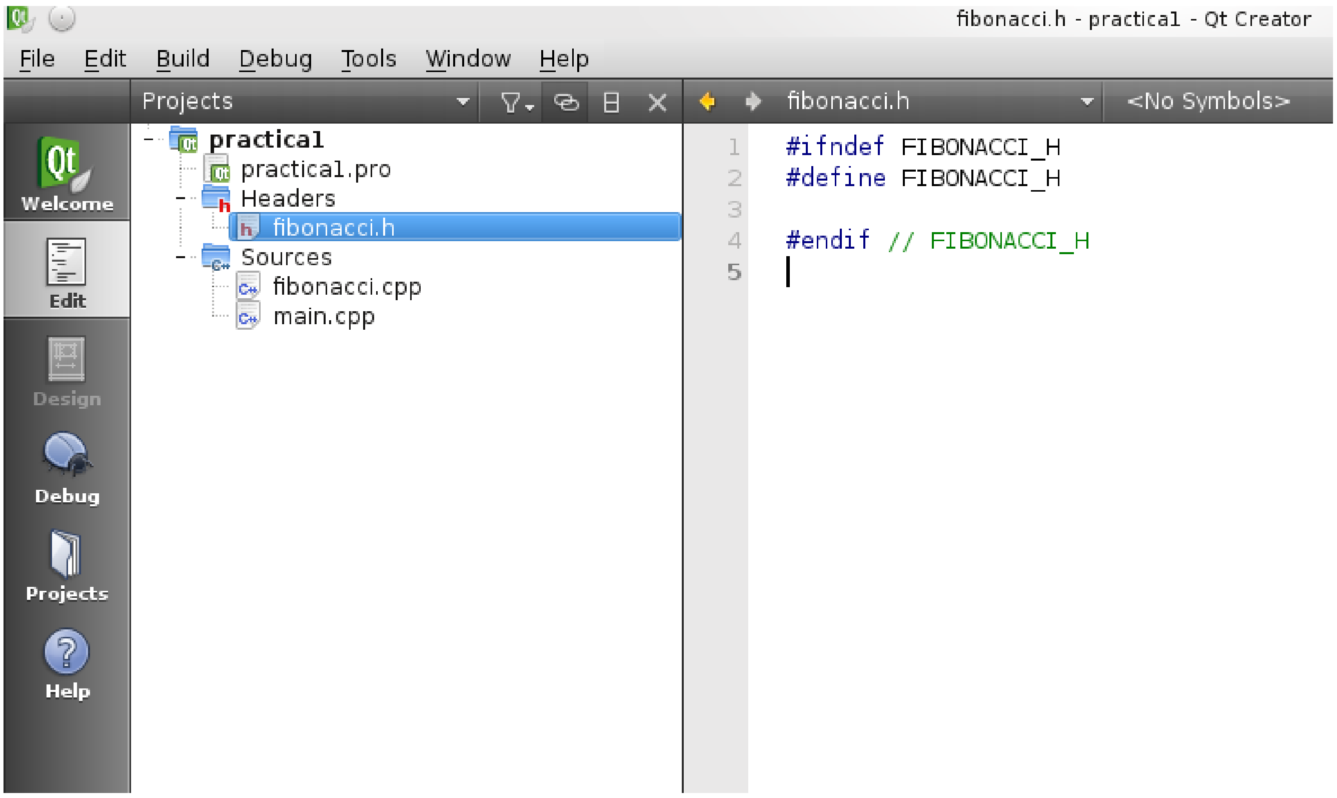


Creates a C++ header file.

5. Select ***C++ / C++ Header File***, and name its as ***fibonacci.h***.
6. Do the same with ***C++ / C++ Source File*** to add a source file named ***fibonacci.cpp***.

7. Add another, and the final, C++ source file, and call it ***main.cpp***.

   You should then get an environment similar to the screen-shot below:



```
#ifndef FIBONACCI_H
#define FIBONACCI_H

#endif // FIBONACCI_H
```

3. Editing the code

Now let’s edit the code. To do this:

1. Select the file ***fibonacci.h*** and add the following function prototype declaration:

```
int fibonacci(int n);
```

2. Implement this function in the file ***fibonacci.cpp***.
3. Also, implement the ***main*** function in the file ***main.cpp***. The program should request a number from the user, entered via the keyboard, and it should display the Fibonacci series of the entered number.
4. While editing the code, there are several keyboard short-cuts that are very useful for editing:

   4.1. ***Ctrl+Space***: can be used to automatically completes the current word. For example, by typing "fib + Ctrl-Space" in ***main.cpp***, the editor would propose "fibonacci" and prompt the function arguments that needs to be provided.

   4.2. ***F2***: to jump to the definition of a function or any symbol.

   4.3. ***F4***: to move from a header file to the corresponding source file, for example from *fibonacci.h* to *fibonacci.cpp*.

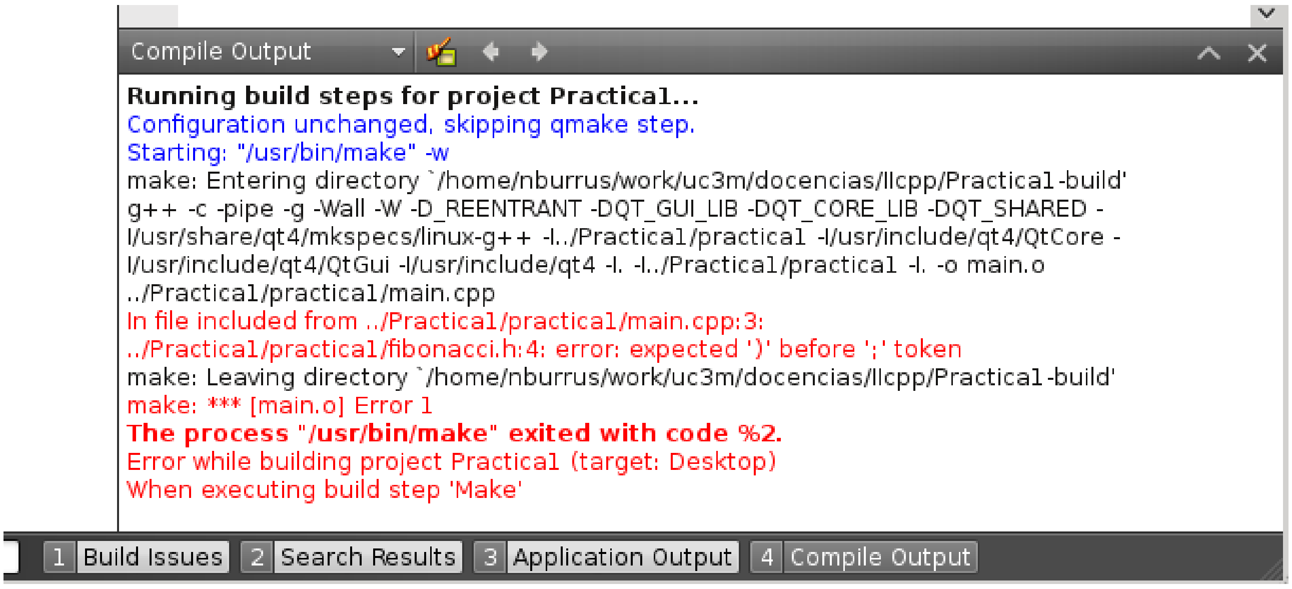
   4.4. ***Ctrl+I***: for automatic code indentation.

4. Compiler

To compile the program you can either click on the hammer button (Bottom left corner) or use the short-cut key ***`Ctrl+B`*** or via the menu option ***`Build→Build All`***.

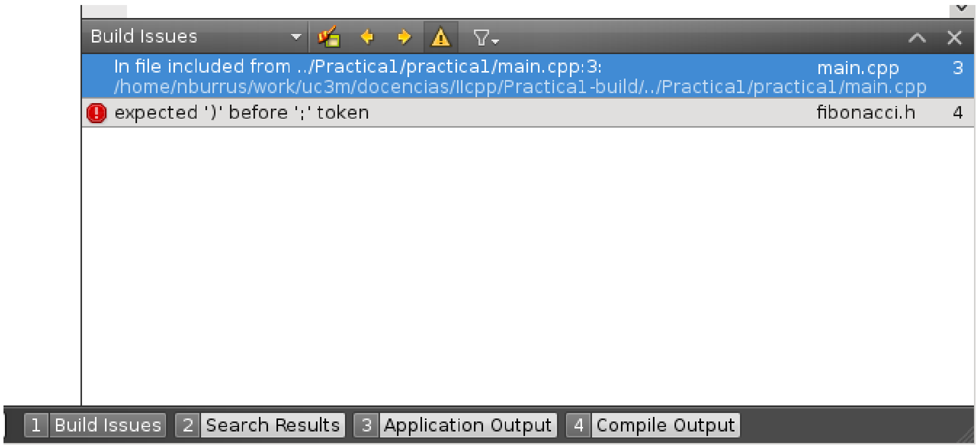
To view the result of the compilation and to see if the compilations has completed successfully, open the *Compile Output* (Bottom right, option 4) clicking on the icon or use the keyboard short-cut ***`Alt+4`***.

In this window you can see the details of the compilation commands executed by QtCreator.



```
Running build steps for project Practical...
Configuration unchanged, skipping qmake step.
Starting: "/usr/bin/make" -w
make: Entering directory `/home/nburrus/work/uc3m/docencias/llcpp/Practical-build'
g++ -c -pipe -g -Wall -W -D_REENTRANT -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -
I/usr/share/qt4/mkspecs/linux-g++ -I../Practical/practical -I/usr/include/qt4/QtCore -
I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. -I../Practical/practical -I. -o main.o
../Practical/practical/main.cpp
In file included from ../Practical/practical/main.cpp:3:
../Practical/practical/fibonacci.h:4: error: expected ')' before ';' token
make: Leaving directory `/home/nburrus/work/uc3m/docencias/llcpp/Practical-build'
make: *** [main.o] Error 1
The process "/usr/bin/make" exited with code %2.
Error while building project Practical (target: Desktop)
When executing build step 'Make'
```

If there is a problem with the compilation, you can see the error messages here, but QtCreator also offers a special view for errors in the *Build Issues tab (Bottom left, option 1)*.



```
In file included from ../Practical/practical/main.cpp:3:                 main.cpp     3
/home/nburrus/work/uc3m/docencias/llcpp/Practical-build/../Practical/practical/main.cpp
expected ')' before ';' token                                            fibonacci.h  4
```

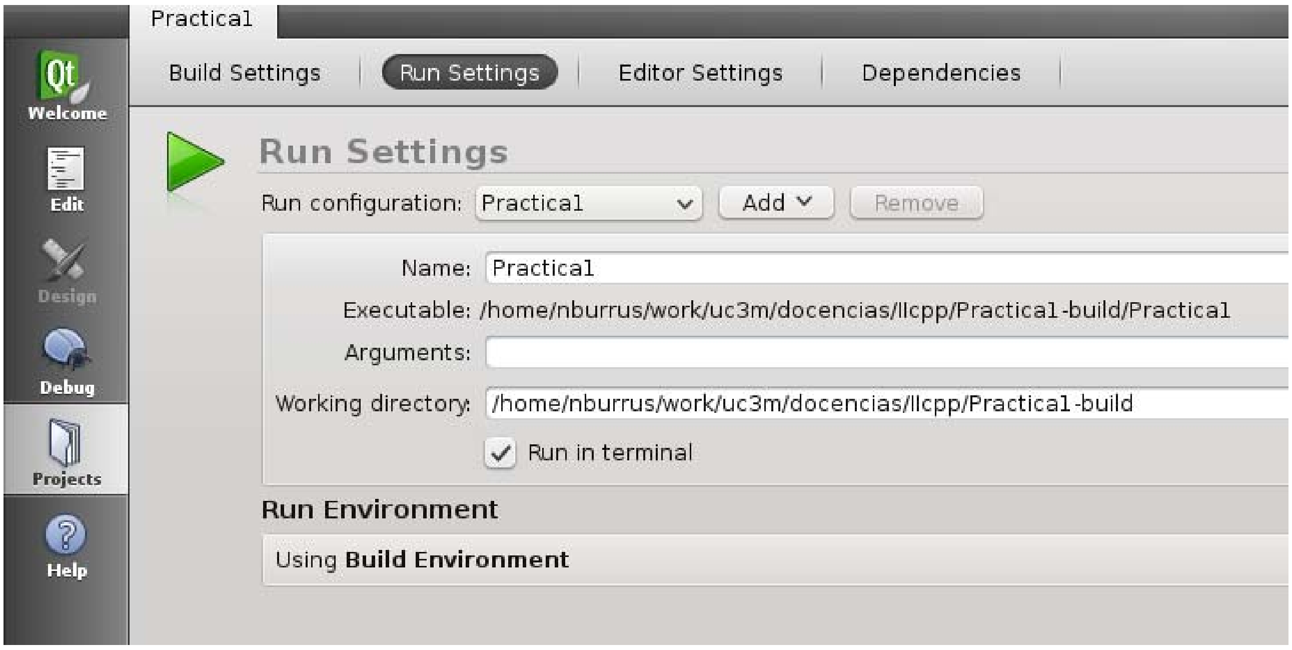
Clicking on an error directly opens the file and points to the line of the code where the detected error is at.

5. Execution

To run the program you can either click on the *green triangle* button (Bottom left corner), or use the short-cut key ***`Ctrl+R`***, or through the menu option ***`Build→Run`***.

The output of the program is shows in the *Application Output* tab *(Bottom left, option 3).*

If the program needs interaction with the keyboard, then the output should be displayed on a terminal. For which you need to configure by selecting the *'Run in terminal'* option under, ***`Projects→Run Settings`*** (*Left*) tab.

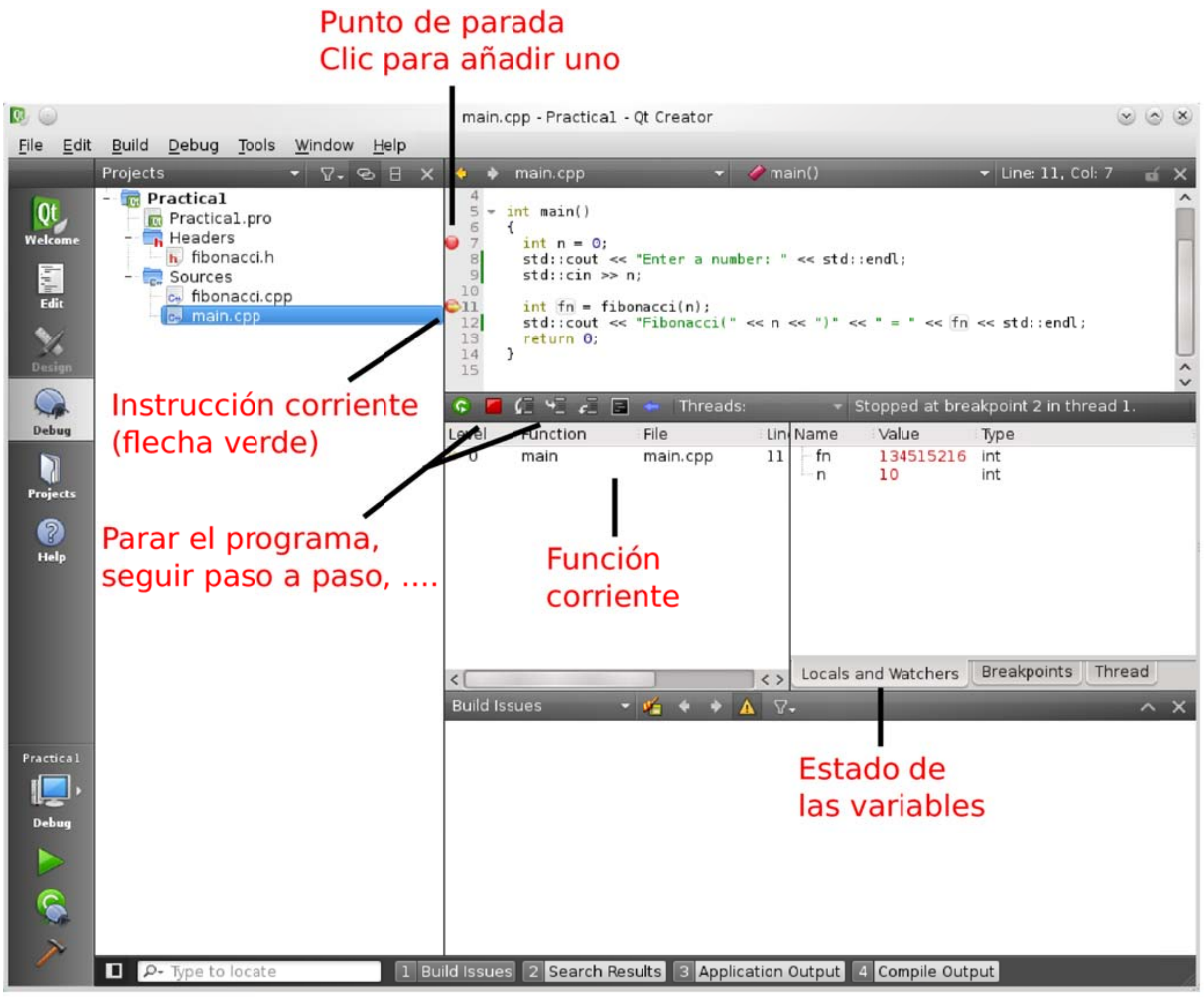


Run configuration: Practical
Name: Practical
Executable: /home/nburrus/work/uc3m/docencias/llcpp/Practical-build/Practical
Arguments:
Working directory: /home/nburrus/work/uc3m/docencias/llcpp/Practical-build
✓ Run in terminal
Using **Build Environment**

6. For advanced. Code debugging

QtCreator has an integrated **GDB** debugger interface. You can enable the debug mode either by clicking the green triangle button with a small bug icon (icon for debugging), or through the short-cut key **F5** or via the menu option ***`Debug→Start debugging→Start debugging`***.

To open the debug window, select the *Debug* tab (Bottom left), and it must appear similar to the setting shown in the figure below.



```
int main()
{
  int n = 0;
  std::cout << "Enter a number: " << std::endl;
  std::cin >> n;

  int fn = fibonacci(n);
  std::cout << "Fibonacci(" << n << ")" << " = " << fn << std::endl;
  return 0;
}
```

Stopped at breakpoint 2 in thread 1.

Level	Function	File	Line
0	main	main.cpp	11

Name	Value	Type
fn	134515216	int
n	10	int

7. Exercise

So far we have programmed in C++, but by using C member functions. This is possible because C++ is compatible with C. However, C++ has its own standard library, which is more powerful and complete. In particular, the Input/Output operations in C++ is handled differently by using flow operators "<<" and ">>".

7.1. **Exercise 1**

The following program is an example of the "Hello world" program in C++ (left) and its equivalent in C (right):

```
#include <iostream> // std::cout        #include <stdio.h> /* printf */

int main() {                            int main() {
      std::cout << “Hello world\n”;           printf(“Hello world\n”);
}                                       }
```

7.2. **Exercise 2**

This program displays a message on the terminal using ***std::cout*** which represents the standard output of the program. Then we use the operator "<<" to send data to the output stream. The data can be sent as a chain, irrespective of the data type.

```
#include <iostream> // Header for std::cout

int main() {
      int i = 42;
      float f = 42.5;

      std::cout << “Integer: “ << i << “ Float: ” << f << endl;
      // 'endl' is the as “\n” in C
}
```

Reading data from standard input is done using ***std::cin***:

```
#include <iostream> // Header for std::cout, std::cin

int main() {
      int n;

      std::cout << “Enter a number: “ << endl;
      std::cin >> n;
      std::cout << “The number is “ << n << “\n”;
}
```

Reading an input can be done by using the inverse flow operator ">>". It can also be used as a chain and can be used for reading values of various data types:

```
#include <iostream> // Header for std::cout, std::cin

int main() {
      int n;
      float f;

      std::cout << “Enter an integer number and a floating point number: “ << 
endl;
      std::cin >> n >> f;
      std::cout << “Integer: “ << n << “ Float: “ << f << endl;
}
```

7.3. **Exercise 3**

Rewrite the fibonacci program using standard C++ functions, where ever possible.

7.4. Exercise 4. Namespace

The prefix ***std::*** preceding ***cout*** and ***cin*** comes from a C++ functionality called *namespace*. It allows setting multiple functions with the same name, placing them in different spaces.

All the standard library functions are in the namespace ***std***. To avoid typing the prefix always, you can add the following line in the beginning of the program, which enables the compiler to automatically search for the appropriate functions.

```
#include <iostream>

using namespace std; // This line allows omitting the prefix std::

int main() {
      cout << “Hello without the prefix!“ << endl;
}
```

7.5. Exercise 5. Proposed

Write a program that reads 10 integer values from the keyboard, store them in a table and display them on the screen. The integers must satisfy the condition of being a positive even number. 0 is not permitted.