Introduction to Laboratory Classes: “CLion IDE”

Arkadiusz Chrobot, PhD

September 4, 2026

1 Introduction

CLion is an integrated development environment (IDE) that will be used for programming during the Fundamentals of Programming laboratory classes. It is professional proprietary software created by JetBrains. Fortunately, JetBrains allows students to use it free of charge after registering on the company's website and obtaining an educational license.

The purpose of this laboratory guide is to explain how to install, register, configure, and use CLion to write, compile, run, and debug programs. Please note that this is not a detailed description of the IDE. Such a description would require much more space and would probably be less useful than the official documentation available on the JetBrains website.

Section 2 explains how to download CLion. Section 3 describes how to register for a free student license and activate it.

Section 4 presents the configuration of the IDE. Section 5 shows how to create a new project. Section 6 explains how to compile and run a program using CLion.

Section 7 demonstrates how to debug a program. Section 8 contains source code used in several exercises. Finally, Section 9 contains a list of exercises that should be completed during the laboratory classes.

2 Downloading CLion

To download the CLion IDE, first open a web browser and visit the software manufacturer's website:

`https://www.jetbrains.com`

The website is shown in Figure 1.

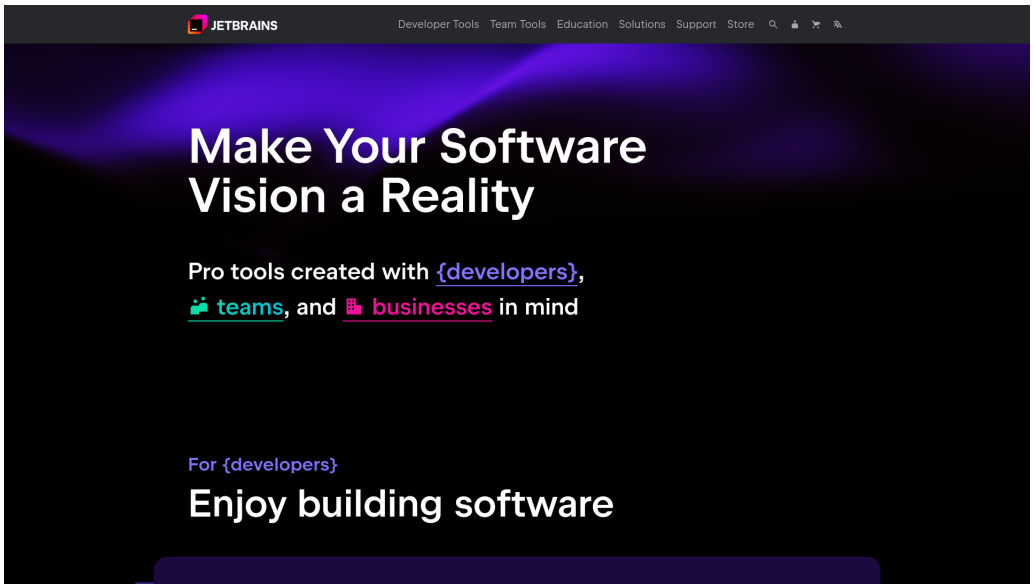


Figure 1: JetBrains website

Next, scroll down until a view similar to the one presented in Figure 2 appears. Click the button marked with a red box in the figure.

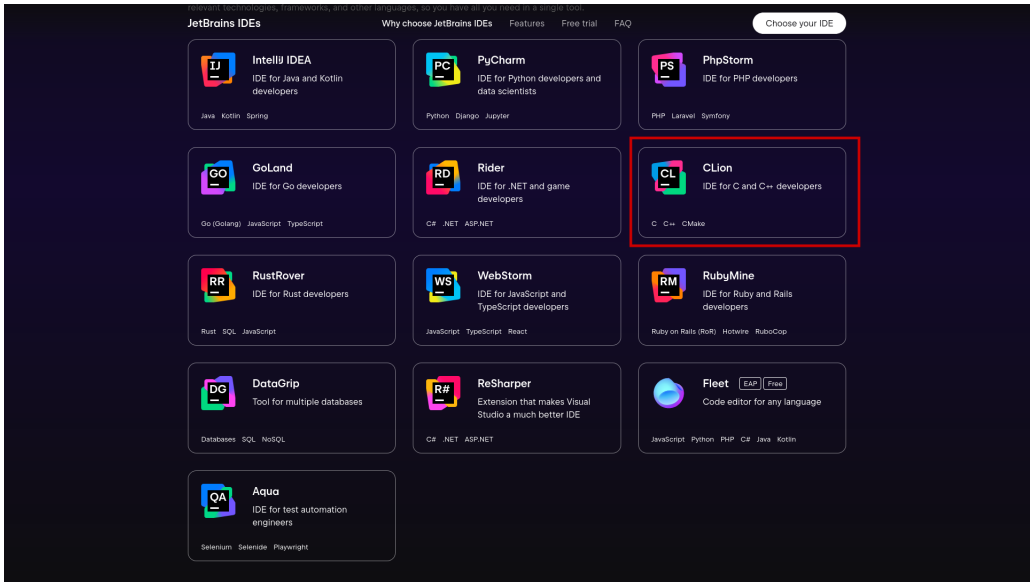


Figure 2: Software available for download from the JetBrains website

After that, a webpage similar to the one presented in Figure 3 should appear.

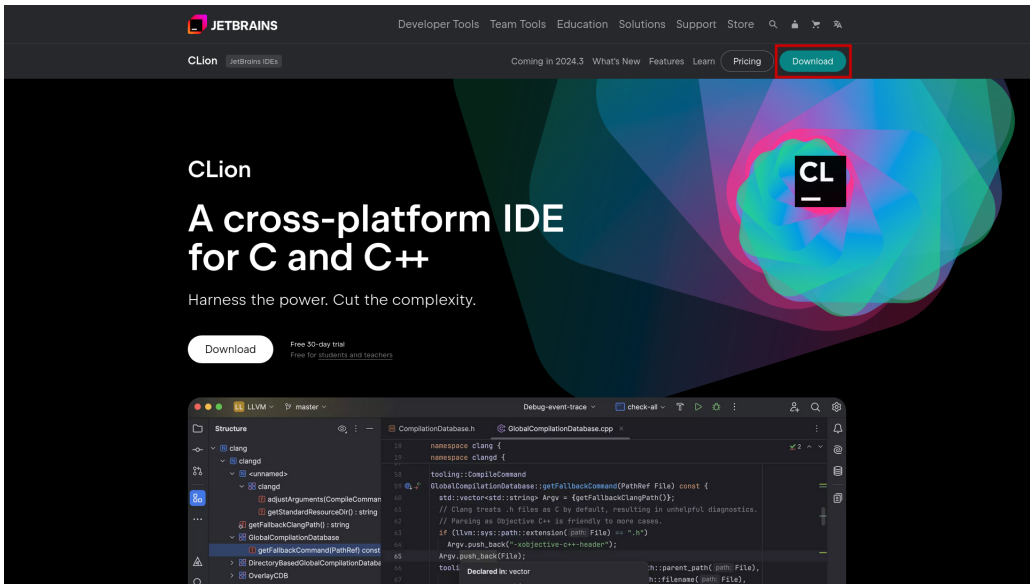


Figure 3: Webpage where CLion can be downloaded

Click the button marked with a red box. This time, the button is labelled *Download*. You should then see a webpage where you can select the installation version of the IDE, as shown in Figure 4.

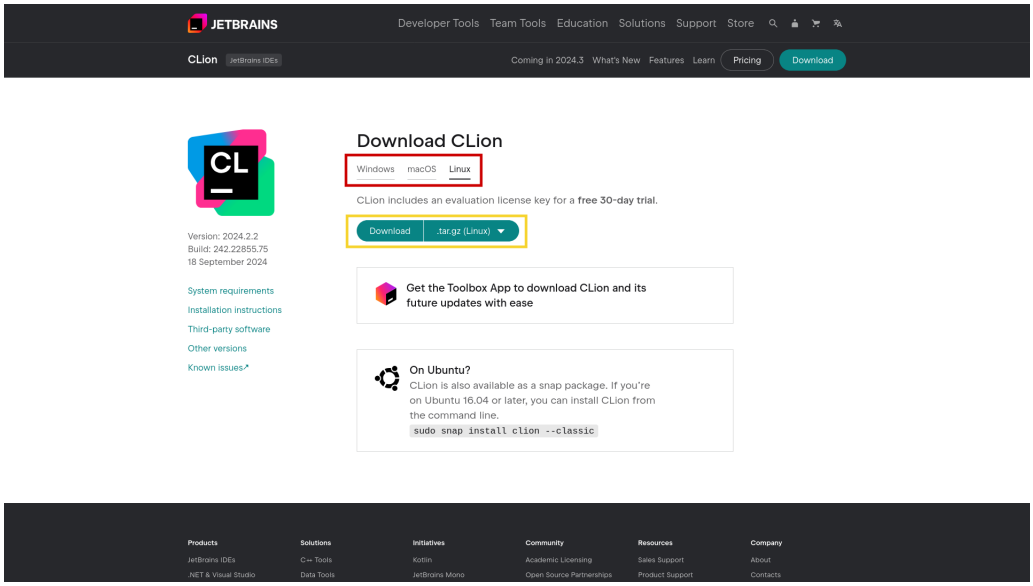


Figure 4: Choosing the CLion installation version

The tabs marked with a red box in Figure 4 allow you to select the version of CLion for your operating system.

Three options are available: Windows, macOS, and Linux.

To download the IDE installer, click the left part of the split button labelled *Download*. This part is marked with a yellow box in Figure 4.

The right part of the split button can be used to change the type of installer file.

Ubuntu Linux users may install CLion either by issuing the command suggested by the website or by using the *Software Center* application.

3 Registering

By default, the IDE will work only for 30 days after installation. However, students may apply for a free license that allows them to use CLion and other JetBrains products while they are enrolled at a university.

To obtain the license, first register using the JetBrains website.

As in the case of downloading the software, go to the company's website and click the *Education* option, marked with a red box in Figure 5.

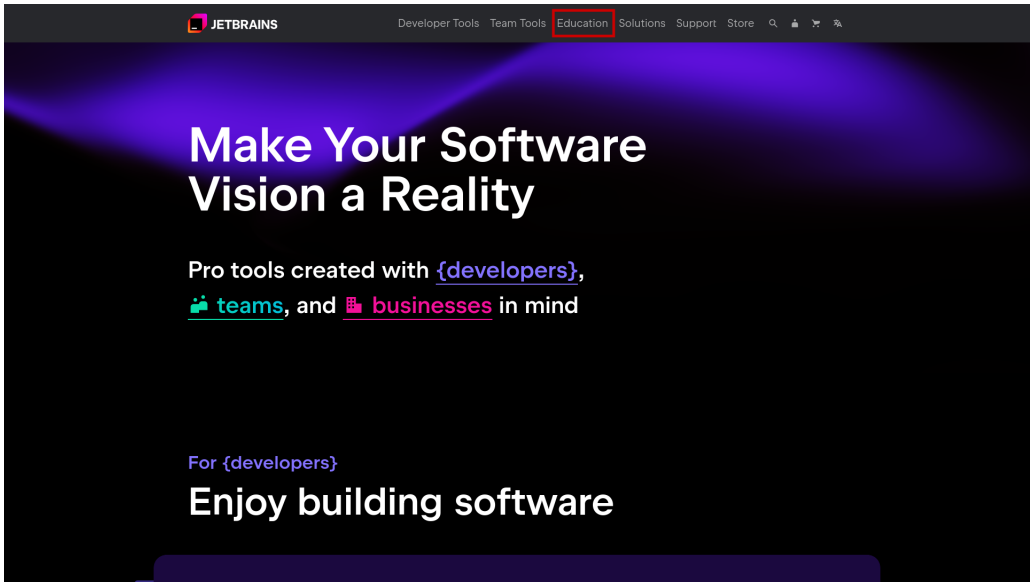


Figure 5: The first step of the registration process

After that, a window similar to the one shown in Figure 6 should appear. Click the link marked with a red box.

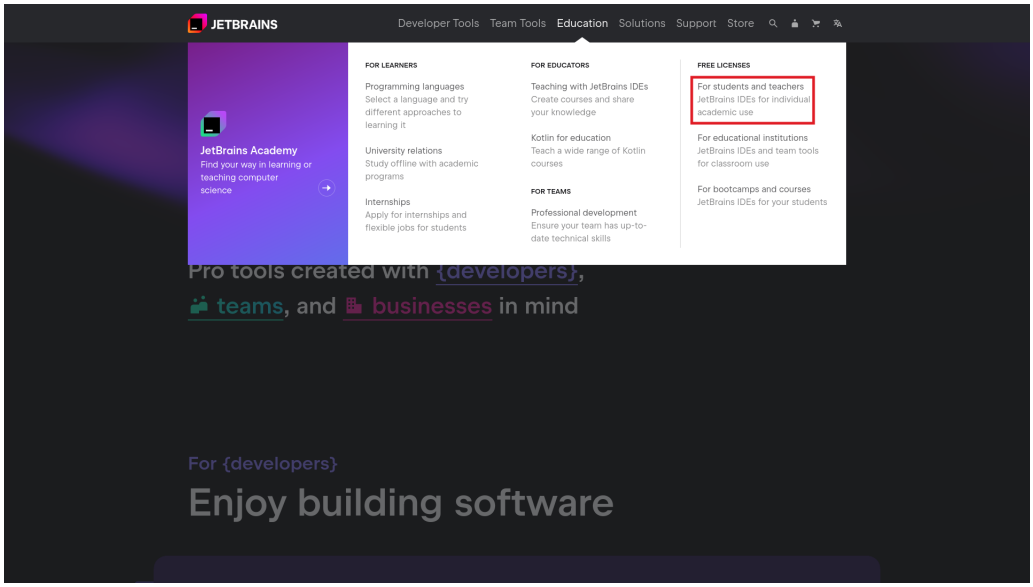


Figure 6: The educational license link

A new page will appear, as shown in Figure 7. Scroll down until you see a view similar to the one presented in Figure 8.

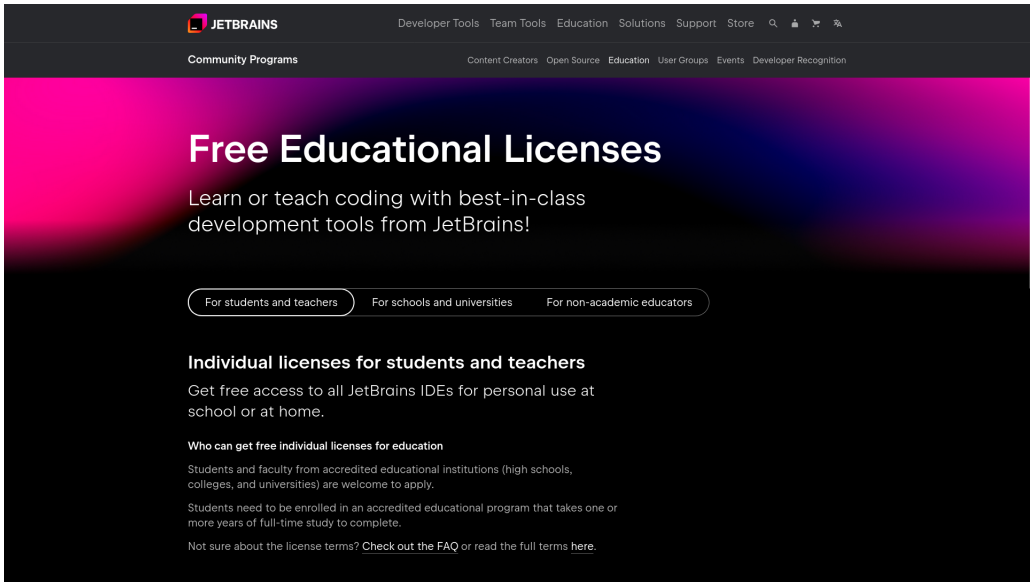


Figure 7: The free licenses page

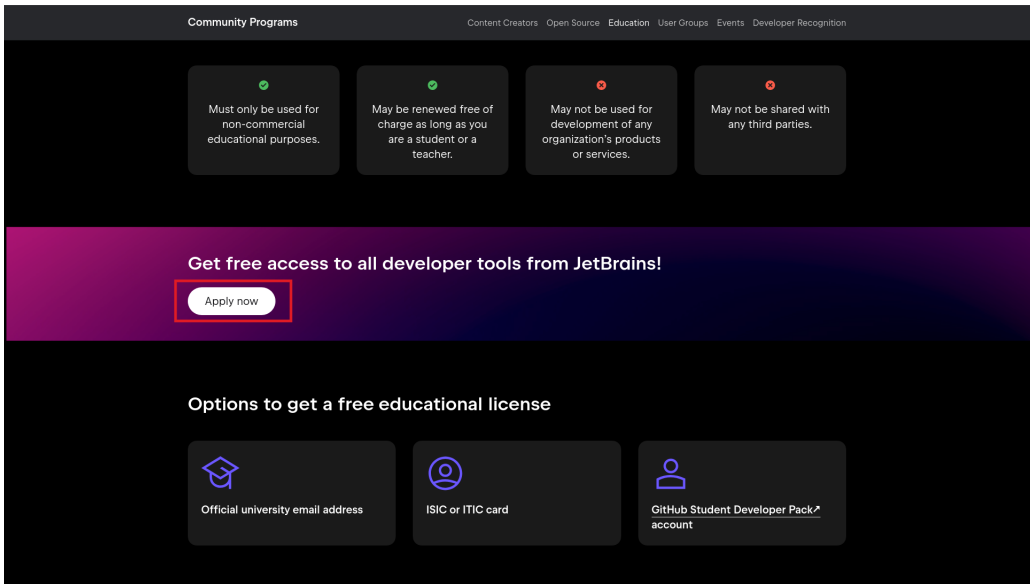


Figure 8: The application button

Click the *Apply now* button.

You will be redirected to the registration form shown in Figure 9. Fill in the form.

Use the email address assigned to you by the university. Other email addresses may not be accepted.

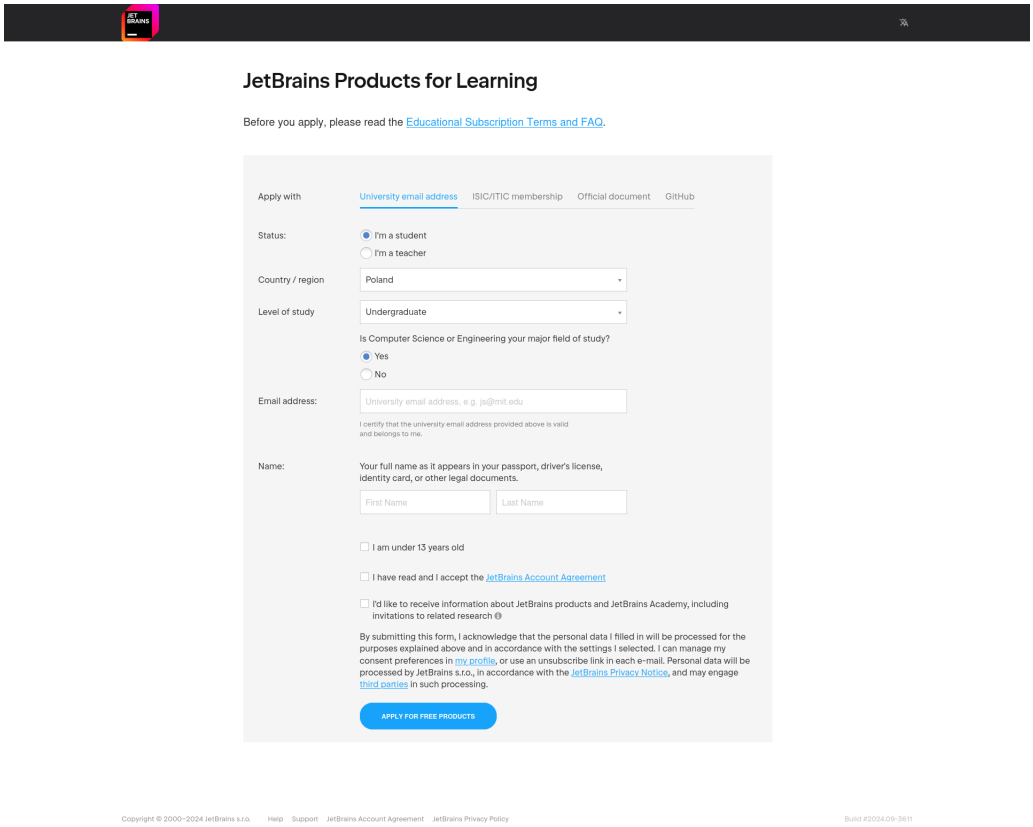


Figure 9: Registration form

After receiving an email from JetBrains, follow the instructions and activate the license in CLion.

To do this, start the IDE. When the main window appears, click the gear icon shown in Figure 10 and select the *Manage Licenses…* option.

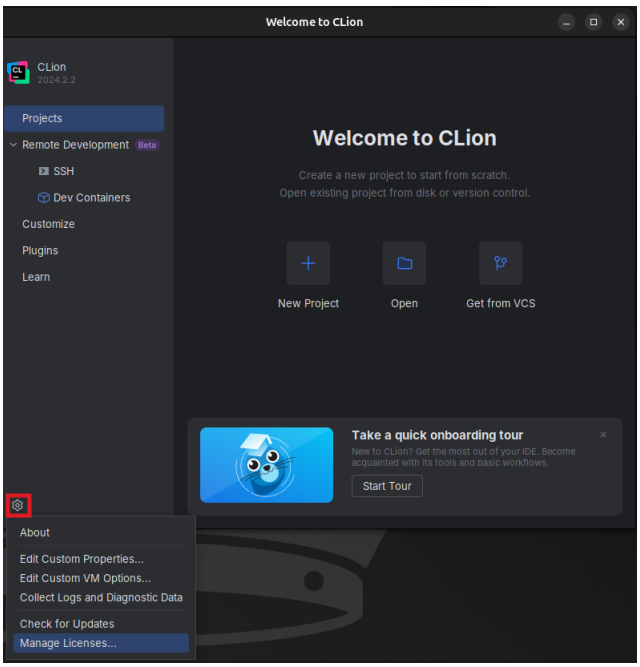


Figure 10: Opening the license manager

In the new window shown in Figure 11, click *Activate New License* and follow the instructions displayed by the software.

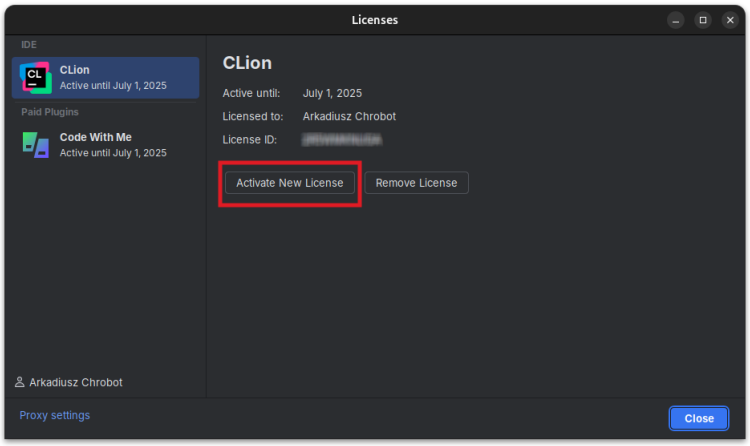


Figure 11: License activation window

At the end of the process, the new license should be activated and remain valid for one year.
After that period, the license can be renewed using the JetBrains account created during registration.

4 Configuration

The CLion IDE includes many tools designed to help experienced software developers create programs.
Unfortunately, some of these features may be distracting or confusing for beginners.
To avoid this, you can download a configuration file called *settings.zip* from the same website from which this laboratory guide was obtained.
This file disables some features, such as machine-learning-based code generation and parameter name hints, and enables others, such as compiler warnings.
To import the configuration, start the IDE and select the *Customize* option on the left side of the main window.
Next, on the right side, set the region to *Europe*, if it is not already selected, and click the *Import Settings…* link shown in Figure 12.

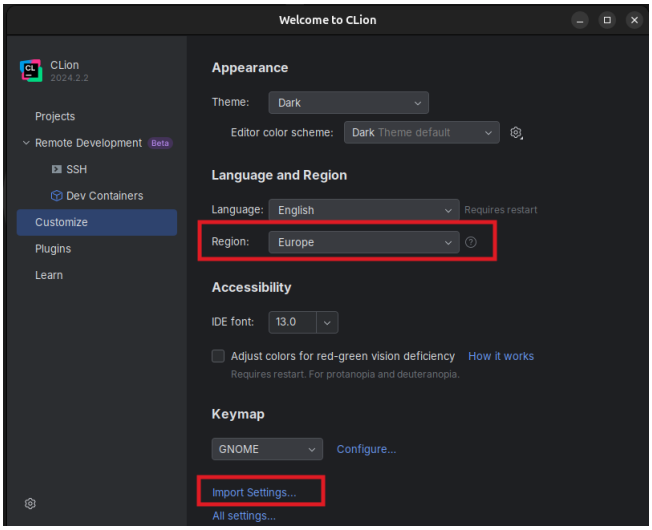


Figure 12: IDE configuration

When the file selection window appears, similar to the one shown in Figure 13, select the *settings.zip* file and click the OK button.

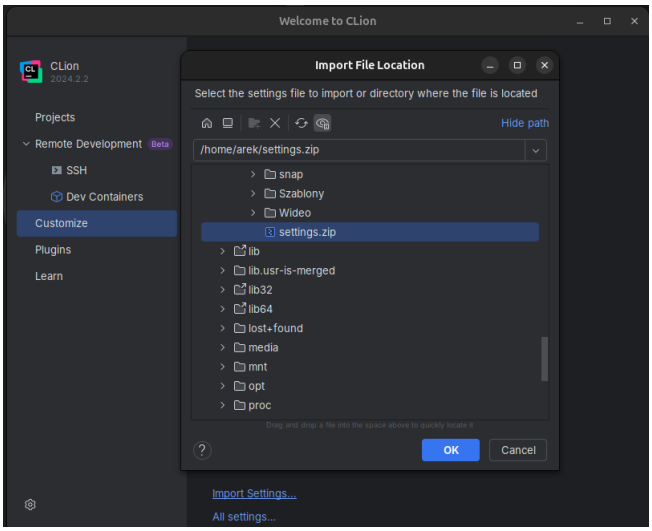


Figure 13: Configuration file selection

In the next window, shown in Figure 14, select all available options if they are not already selected and click the OK button.

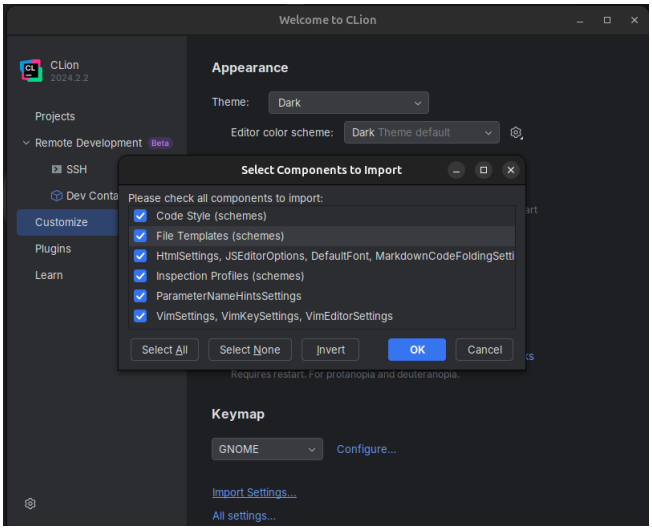


Figure 14: Selecting the import options

Finally, when the next window appears, click the *Import and Restart* button shown in Figure 15 and wait until the IDE restarts.

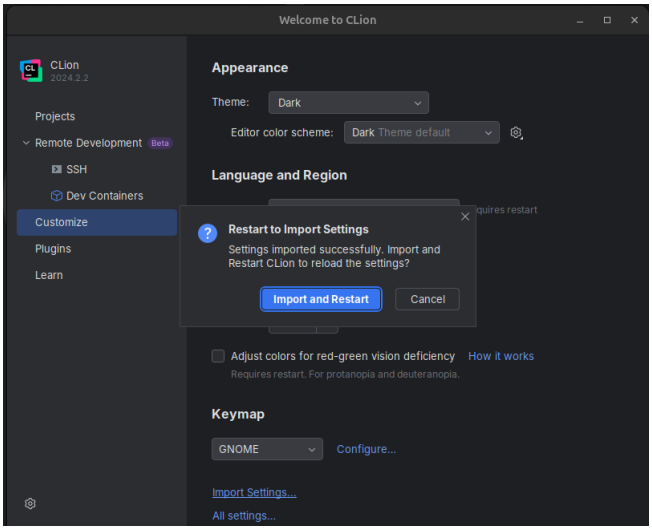


Figure 15: Configuration file import

5 Creating a New Project

One of the reasons CLion makes programming convenient is its integration with many external tools, such as the *GCC* compiler and the *CMake* build system.

These tools may also require additional configuration files.

CLion groups all files associated with the program being developed into a *project*. By default, projects are stored in the *CLionProjects* directory.

The most important file is the one with the `.c` extension because it contains the program source code. By default, CLion names this file `main.c`.

Before writing a program in CLion, you should therefore create a new project.

If CLion has not been used before, its main window should look similar to the one shown in Figure 16.

To create a new project, click the button with the plus sign, labelled *New Project*.

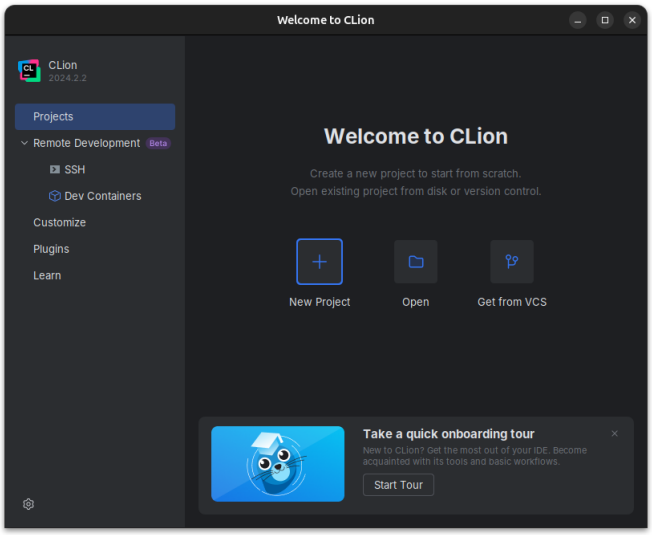


Figure 16: CLion IDE main window

When the new window appears, as shown in Figure 17, select the *C Executable* option on the left.

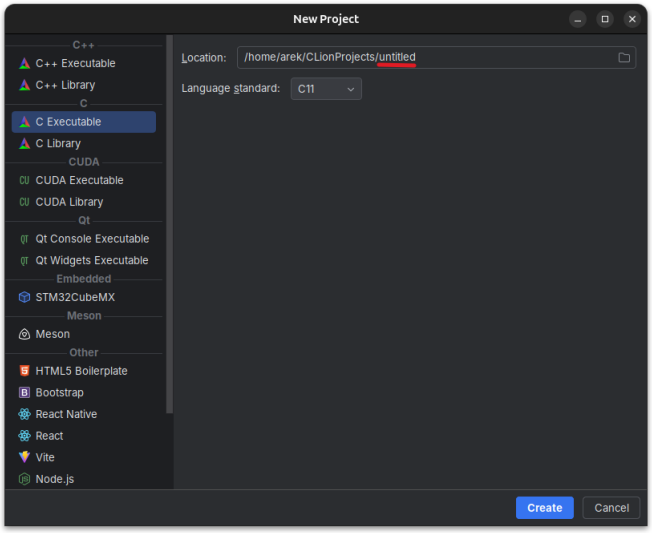


Figure 17: Window for creating a new project

Change the last part of the path at the top-right side of the window unless you want the project to be named *untitled*.

This is the default project name assigned by CLion and also the name of the subdirectory inside *CLionProjects* where the project will be stored.

It is recommended to change it to something more meaningful, for example *first*, as shown in Figure 18.

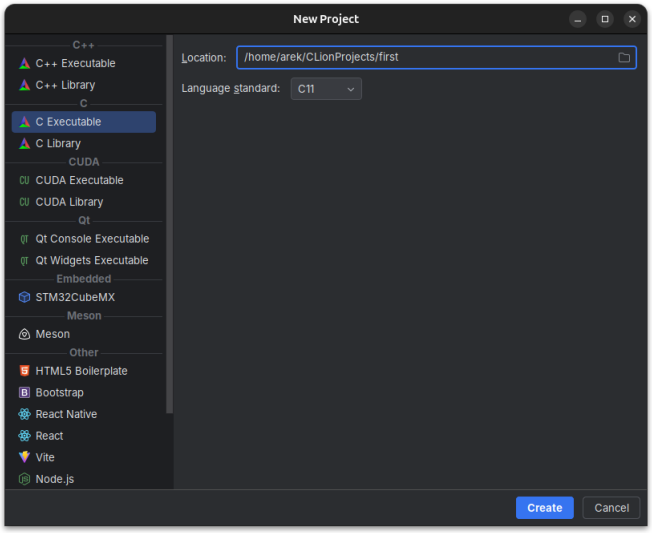


Figure 18: Defining the project name

After clicking the *Create* button, a window similar to the one shown in Figure 19 should appear.

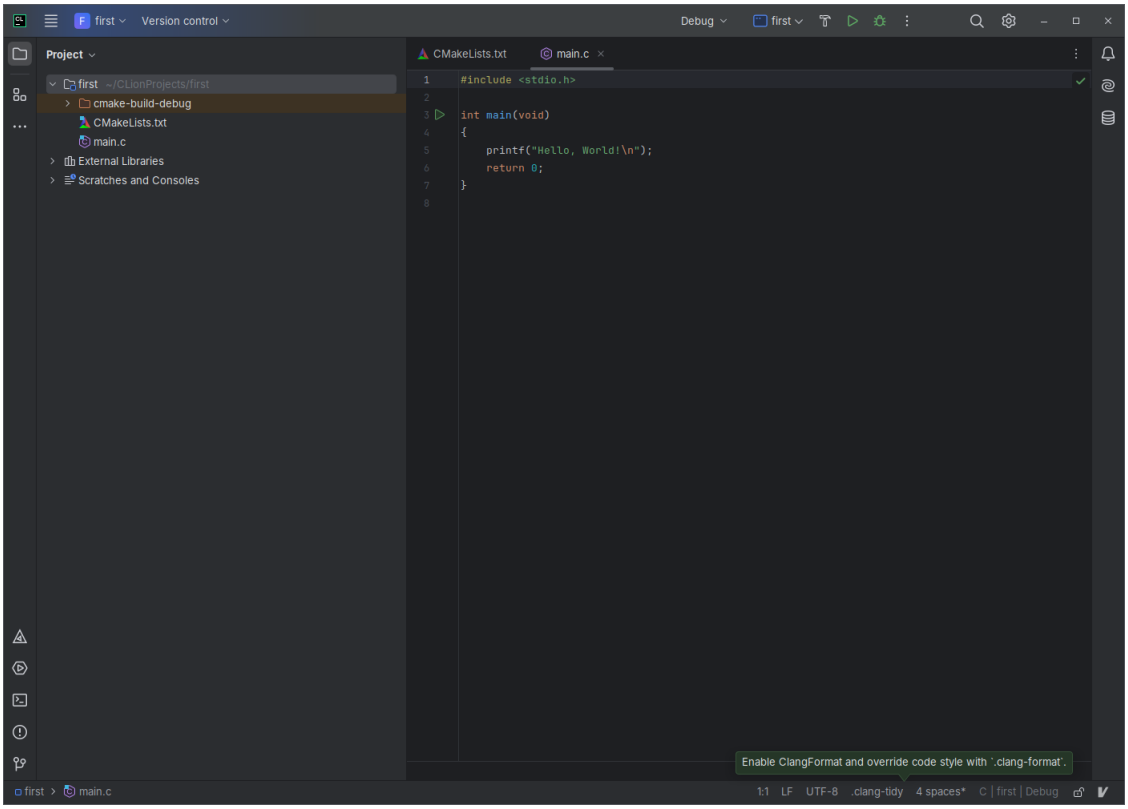


Figure 19: The CLion working window

If this window is partially obscured by a modal window containing IDE settings, close the modal window by clicking OK.

Similarly, if a *Tip of the Day* window appears, it can be closed. You may also select the option that prevents it from appearing the next time the IDE is started.

If the AI Assistant plugin window appears, as shown in Figure 20, it can be closed using the button on the right marked with a red box.

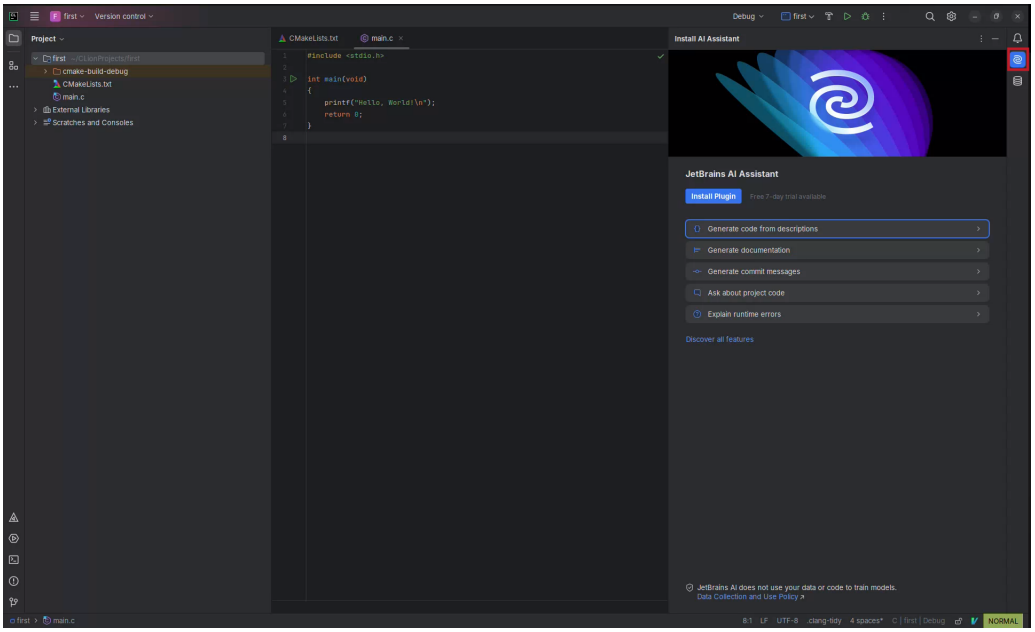


Figure 20: The AI Assistant window

In the default CLion configuration, the IdeaVim plugin may be enabled.

It allows programmers to use Vim editor keyboard shortcuts. These shortcuts can be very useful, but if you are not familiar with the Vim editor, you may want to disable the plugin.

To do this, click the colourful V icon at the bottom-right corner of the CLion working window, shown in Figure 21, and then select the *Enable* option in the plugin menu.

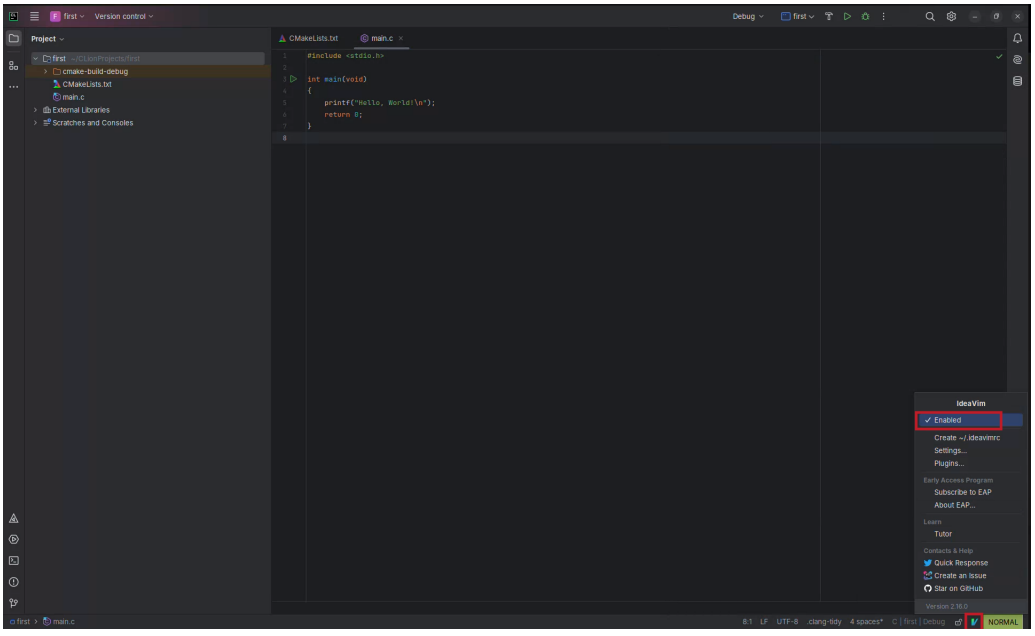


Figure 21: The IdeaVim plugin menu

The main part of the CLion working window is occupied by the source code stored in the `main.c` file.

It contains the *Hello, World!* program, one of the simplest programs that can be written in the C language.

The project can be closed by clicking the main menu button shown in Figure 22 and then selecting the *File* → *Close Project* option shown in Figure 23.

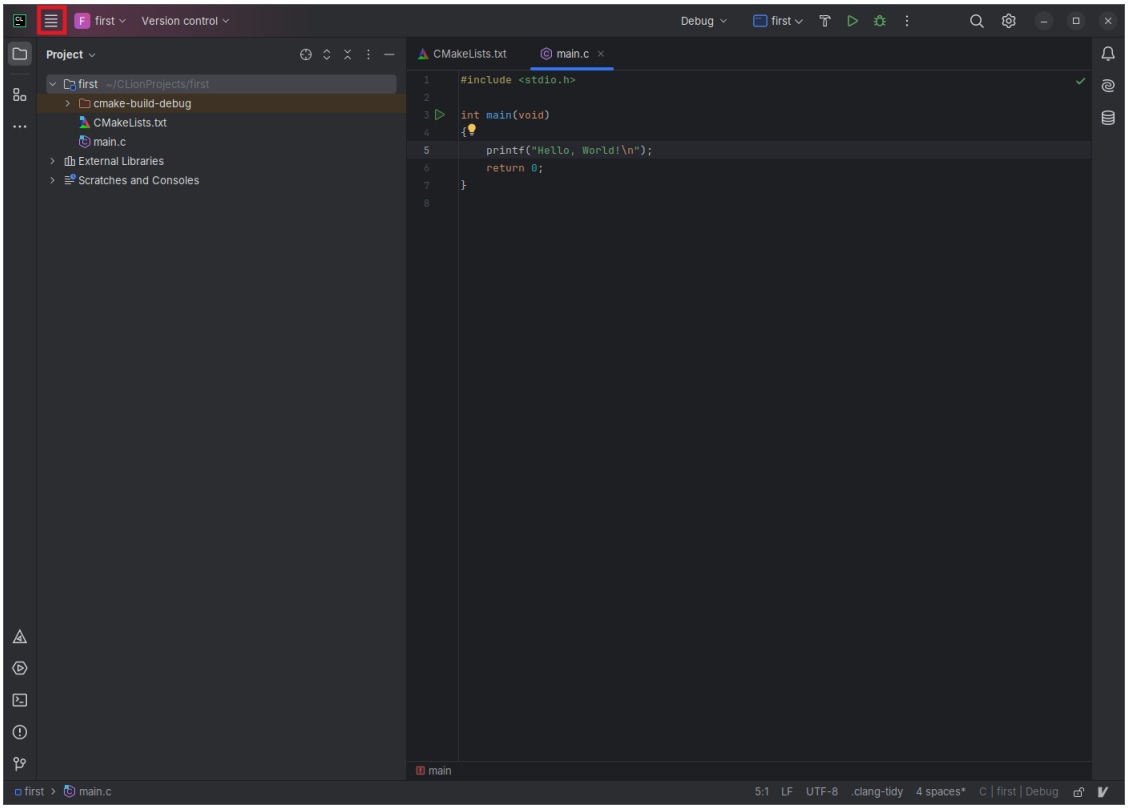


Figure 22: Main menu button

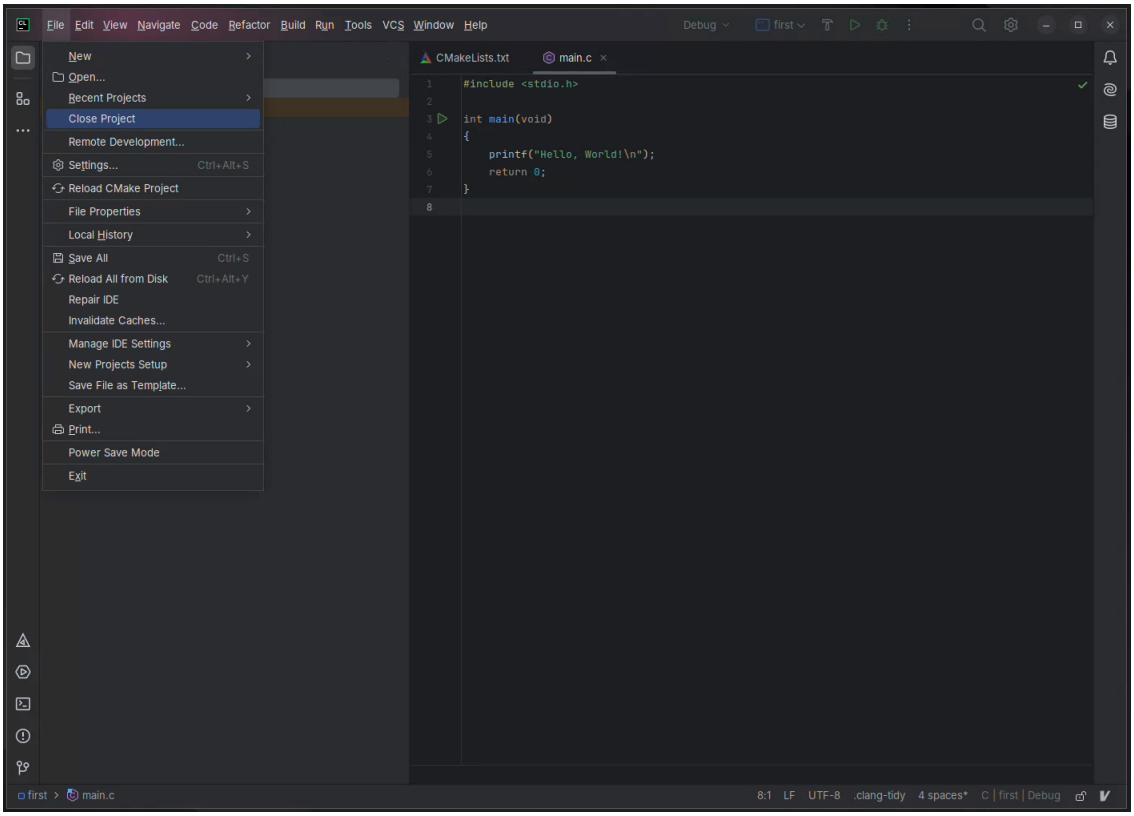


Figure 23: Close Project option in the main menu

After the project has been closed, the main IDE window should appear, similar to the one presented in Figure 24.

The window now displays a list of recently accessed projects, allowing one of them to be reopened quickly.

The *New Project* button is also available, although it may look slightly different.

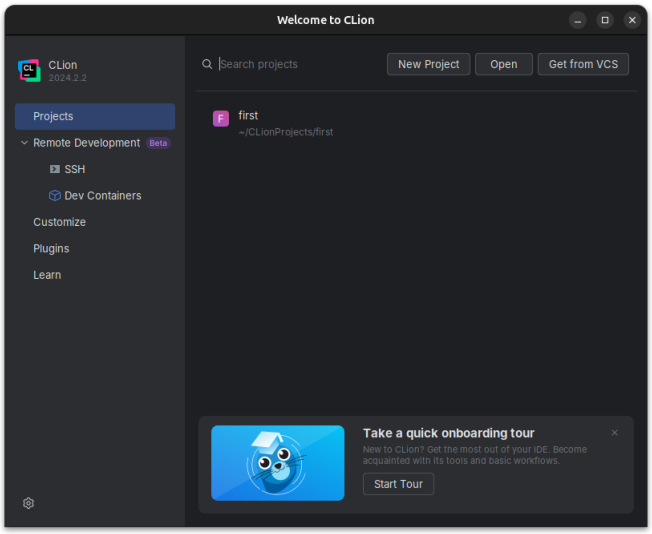


Figure 24: CLion main window after closing a project

6 Compilation and Running

The text visible in the central part of the IDE working window shown in Figures 19 and 25 is the *Hello, World!* program written in the C language.

It is easy to understand for a programmer who knows the language, but it is not directly executable by the computer.

The source code must first be translated into machine code.

CLion makes this process easy.

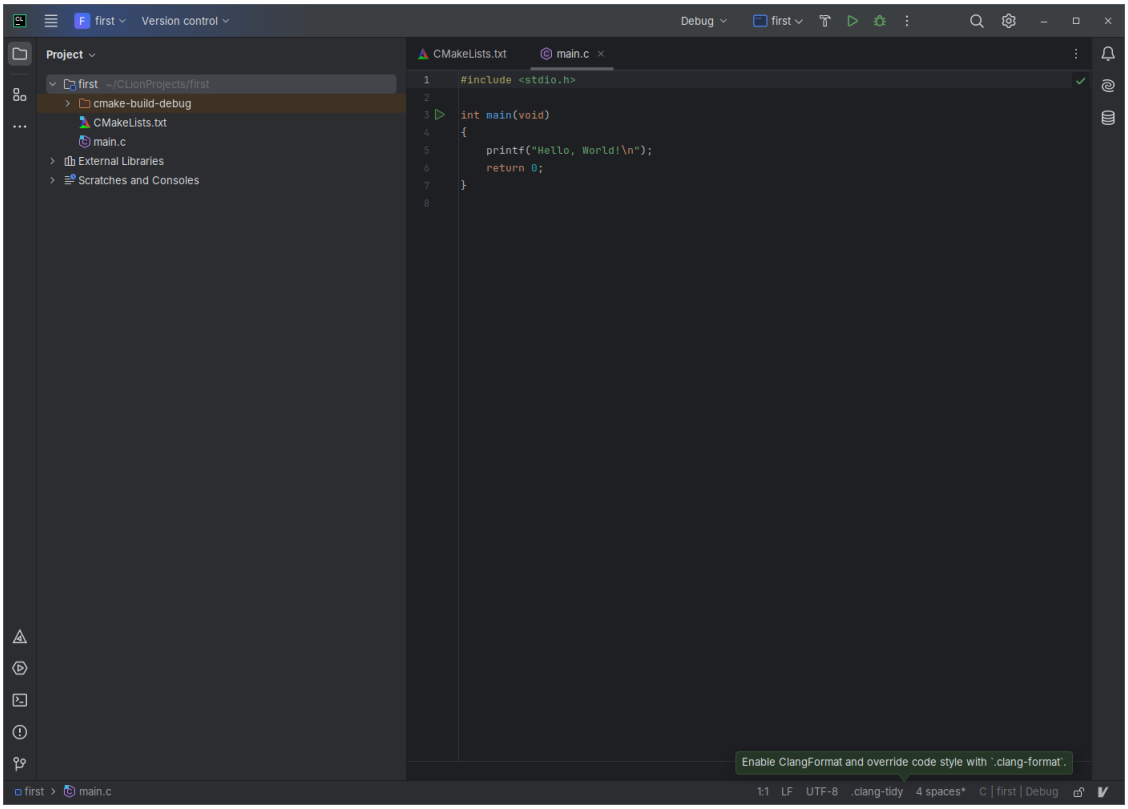


Figure 25: Program source code in CLion

To compile the program, click the button with the hammer icon shown in Figure 26.

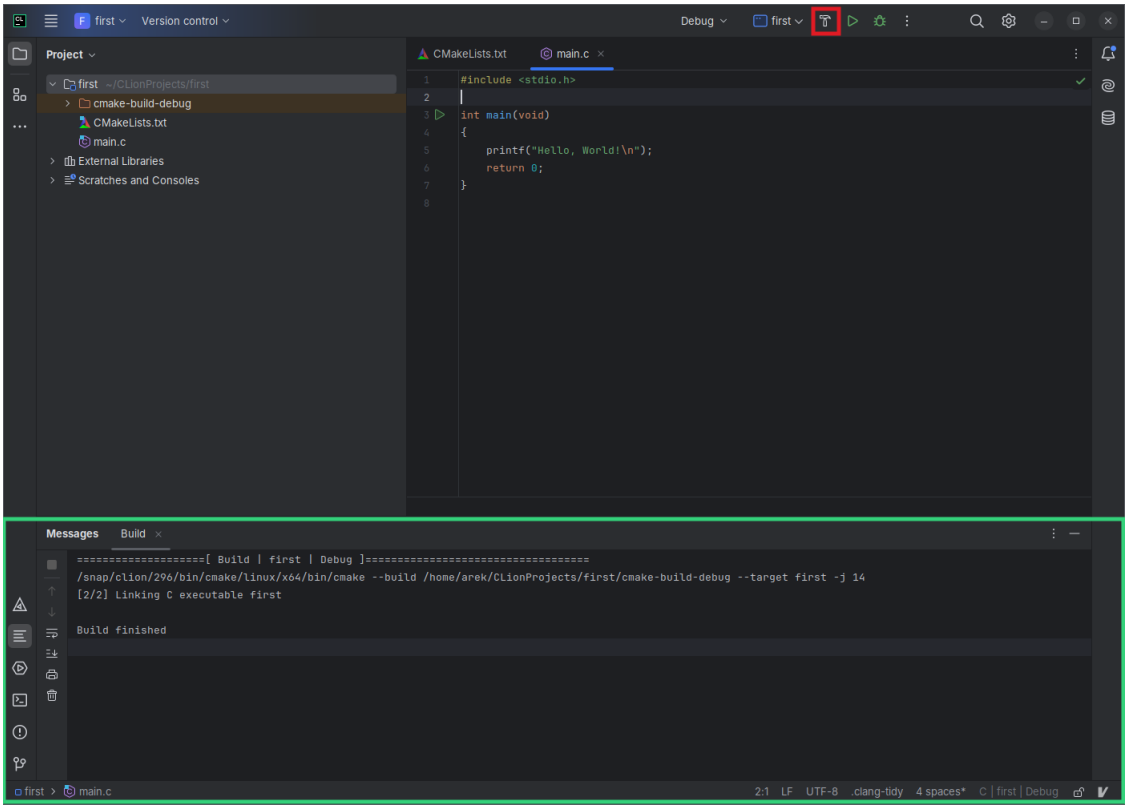


Figure 26: Compilation of a program in CLion

The result of the compilation is displayed in the window located at the bottom of the screen.

In this example, the compilation has been successful. This means that the compiler was able to translate the source code into machine code and create an executable file.

Compilation does not always end successfully.

The compiler may generate two main types of messages about problems detected in the source code.

When it detects a non-critical problem but is still able to translate the program into machine code, it generates a *warning*.

A warning usually indicates that part of the source code is ambiguous or potentially incorrect.

The compiled program may still run, but it may not behave as expected.

Another kind of defect is a *syntax error*.

A syntax error means that part of the source code does not comply with the grammar of the programming language.

As a result, the compiler cannot translate the program into machine code.

Compilation is then interrupted, the executable file is not created, and the program cannot be run.

The configuration of the IDE described in Section 4 instructs the compiler to display basic warnings.

Figures 27 and 28 show examples of a warning and a syntax error.

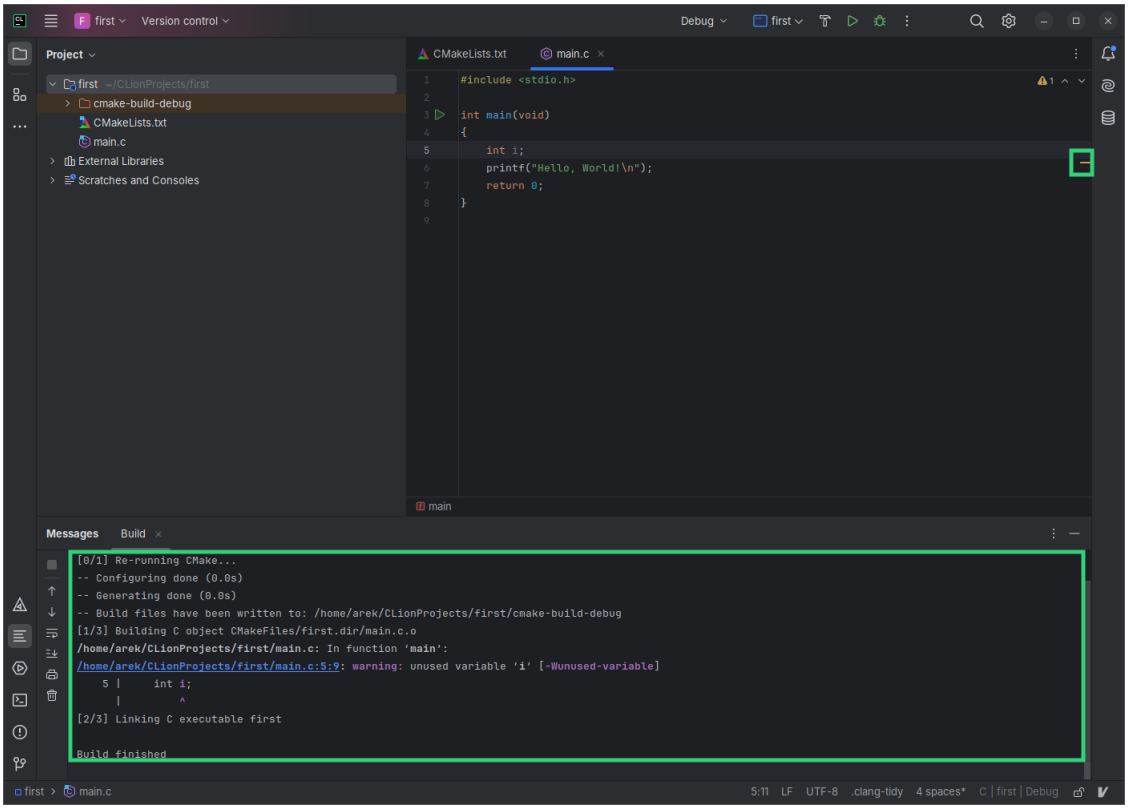


Figure 27: A warning message

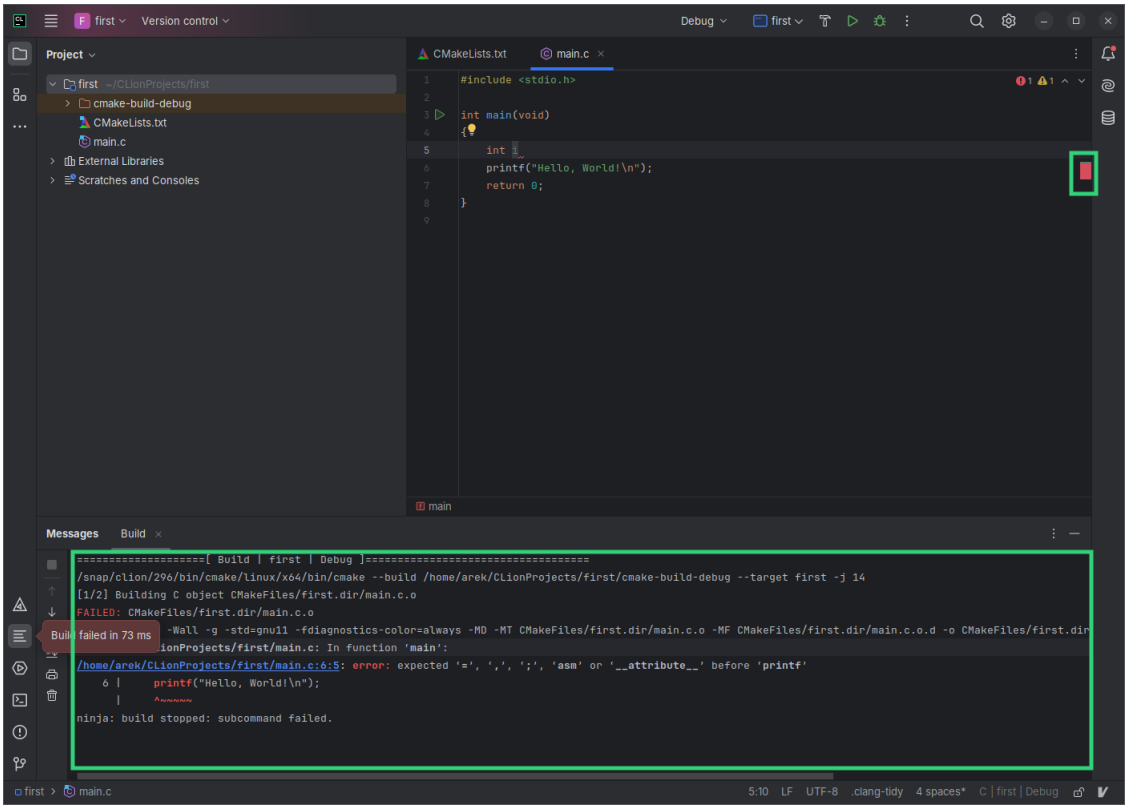


Figure 28: A syntax error message

To make the compiler generate a warning, a local variable named `i` was added in line 5.

The variable can store integer values but is not used anywhere in the program. Therefore, the compiler reports an unused variable.

The word “warning” can be seen in the compiler message.

A yellow marker also appears in the editor window near line 5 and a yellow warning icon appears at the top of the window, as shown in Figure 27.

[1]

If the mouse pointer is placed over the marker, the IDE displays a message similar to the one produced by the compiler.

CLion includes a built-in tool that analyses source code while it is being written. This allows programmers to detect and remove some defects even before compilation begins.

In Figure 28, a syntax error has been introduced by removing the semicolon from the declaration of a local variable.

This character is required by the grammar of the C language.

If it is missing, the compilation is interrupted and the compiler displays an error message.

In this case, the executable file is not created.

The code editor also detects the syntax error. It displays an error icon at the top of the window and a red marker next to the line where the defect occurs.

If the mouse pointer is placed over the marker, the IDE displays a message similar to the one generated by the compiler.

The compiler messages can therefore be summarised as follows:

For a program to work correctly, all defects reported by the compiler should be removed. Compiler warnings should not be ignored.

After restoring the source code to its original form, as shown in Figure 25, the program can be compiled and executed.

To run the program, click the green arrow above the editor window or use the `Shift + F10` keyboard shortcut.

The program output is shown in Figure 29.

[1] Line numbers are not part of the source code. They are added by CLion to make the programmer’s work easier.

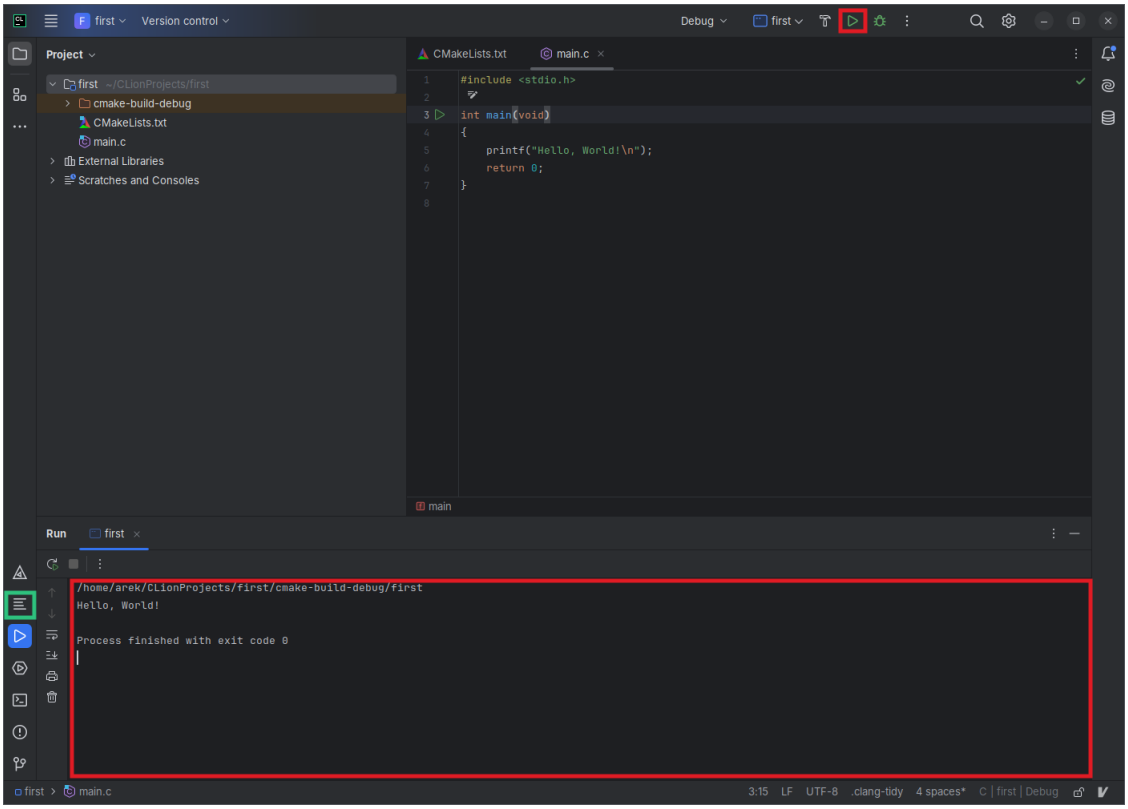


Figure 29: The program output

In this example, the program displays the message:

Hello, World!

If you try to run the program before compiling it, CLion detects that the executable file is missing and automatically starts compilation.

This is convenient, but it has one important drawback: the programmer may fail to notice warnings generated by the compiler.

For this reason, it is recommended to inspect the compiler messages whenever the program is built automatically.

After compilation and execution, the executable file remains on the storage device.

Running the program again therefore does not require recompilation unless the source code has changed.

However, the executable file and other generated files occupy disk space.

CLion provides a cleaning operation that removes files created during the build process.

To use it, select the *Build* → *Clean* option shown in Figure 30.

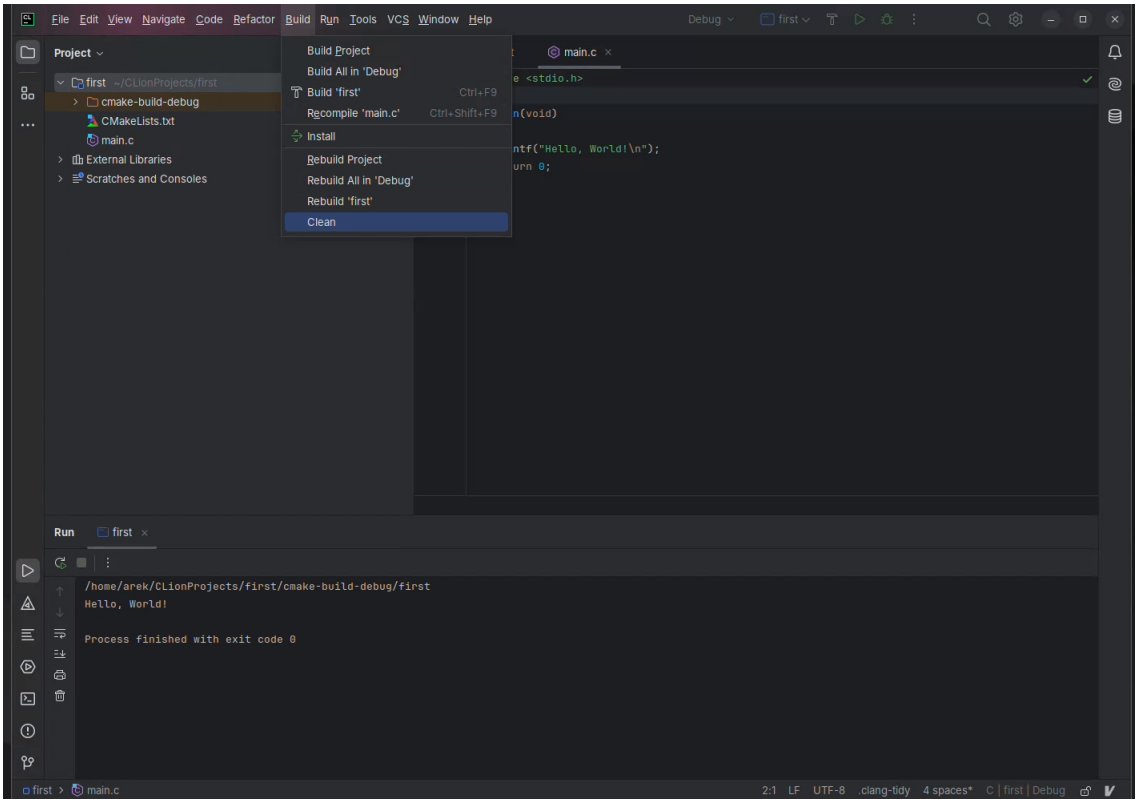


Figure 30: The Clean option in the main menu

When the operation finishes, a message is displayed in the bottom window, as shown in Figure 31.

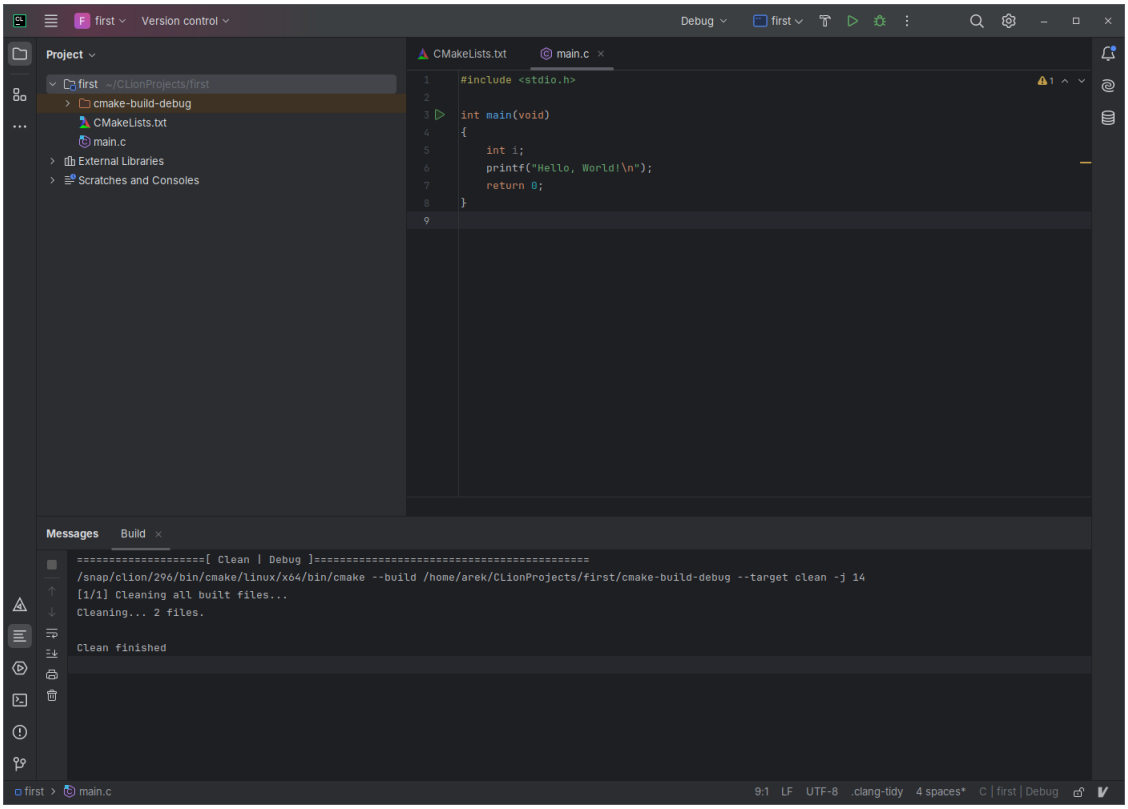


Figure 31: The message displayed after cleaning

7 Debugging

If you want your program to work correctly, all syntax errors and warnings reported by the compiler should be addressed.

However, this is necessary but not sufficient.

A program may also contain *logic errors*, commonly referred to as *bugs*, which cause it to behave incorrectly.

Logic errors are much harder to locate than syntax errors because the compiler usually cannot detect them.

Fortunately, IDEs such as CLion provide tools that help programmers find such defects.

One of them is a *debugger*.

A debugger allows the programmer to execute a program line by line and observe how its state changes, including variable values and control flow.

This laboratory guide demonstrates the debugger using a program that calculates the area of a square whose side length is entered by the user.

The source code is shown in Figure 32.

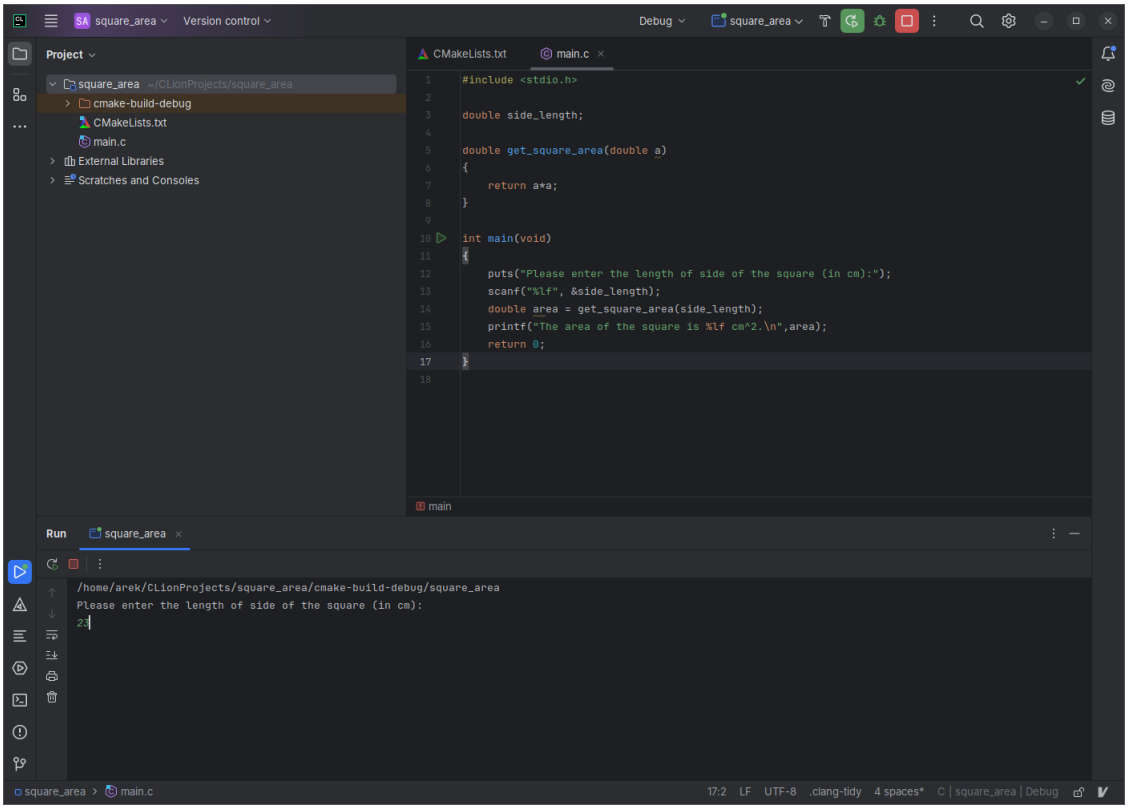


Figure 32: Source code of the example program

This program is slightly more advanced than the one used in Section 6.

It contains two functions:

`get_square_area()`

and

`main()`.

In line 12 of the `main()` function, the `puts()` function is called. It displays a message asking the user to enter the length of the side of the square.

In the next line, another function is called. It reads the number entered using the keyboard and stores it in a global variable named `side_length`, declared in line 3.

The name of this function is `scanf()`.

The program then calls the `get_square_area()` function defined in lines 5–8.

The function squares the value of its parameter `a` and returns the result.

When the function is called, the parameter receives the value of the function argument. In this case, the argument is the `side_length` variable.

The value returned by the function is stored in the local variable `area`.

In line 15, the value of `area` is displayed on the screen.

The program then terminates.

Figure 32 shows the program when it is executed normally, without debugging.

More precisely, it shows the moment when the program calls `scanf()` and waits for the user to enter a number and confirm it by pressing the `Enter` key.

The entered value is visible in the bottom window.

The result calculated by the program is displayed in the same window, as shown in Figure 33.

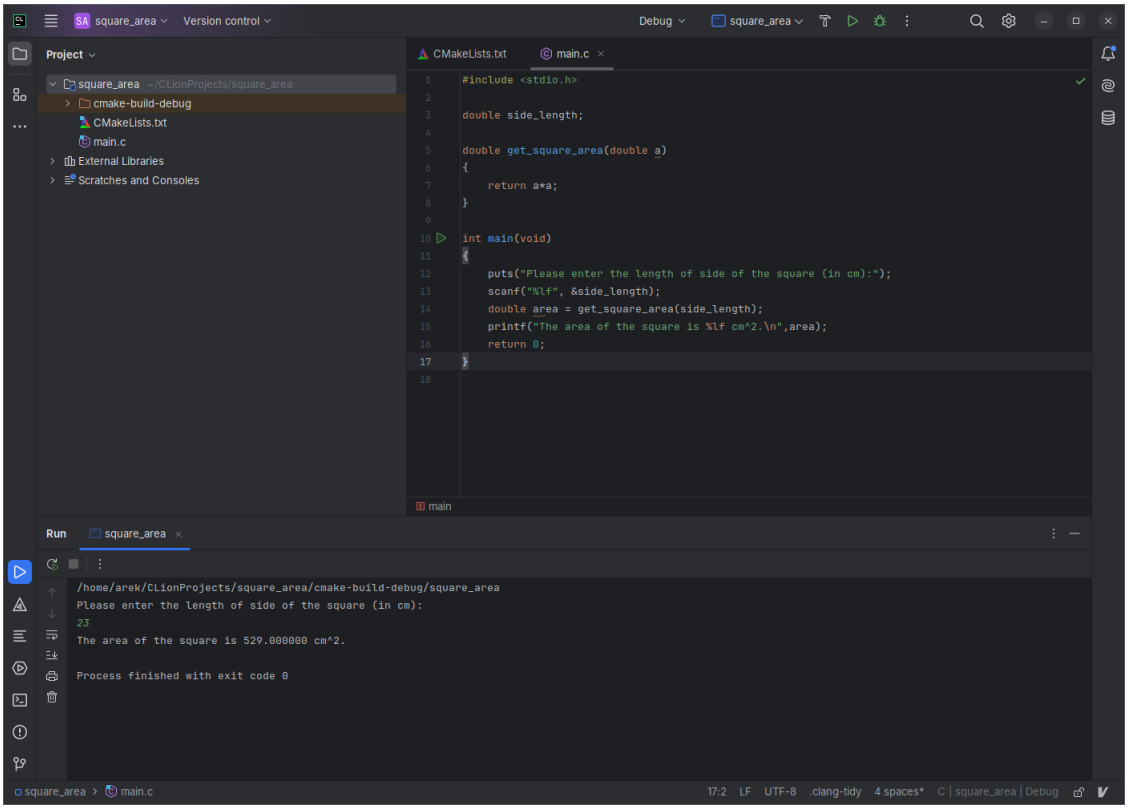


Figure 33: The program output

Before debugging the program, select the place in the source code from which the debugger should begin executing the program line by line.

This place is called a *breakpoint*.

In CLion, click the line number where the breakpoint should be placed.

A red dot appears next to that line, as shown in Figure 34.

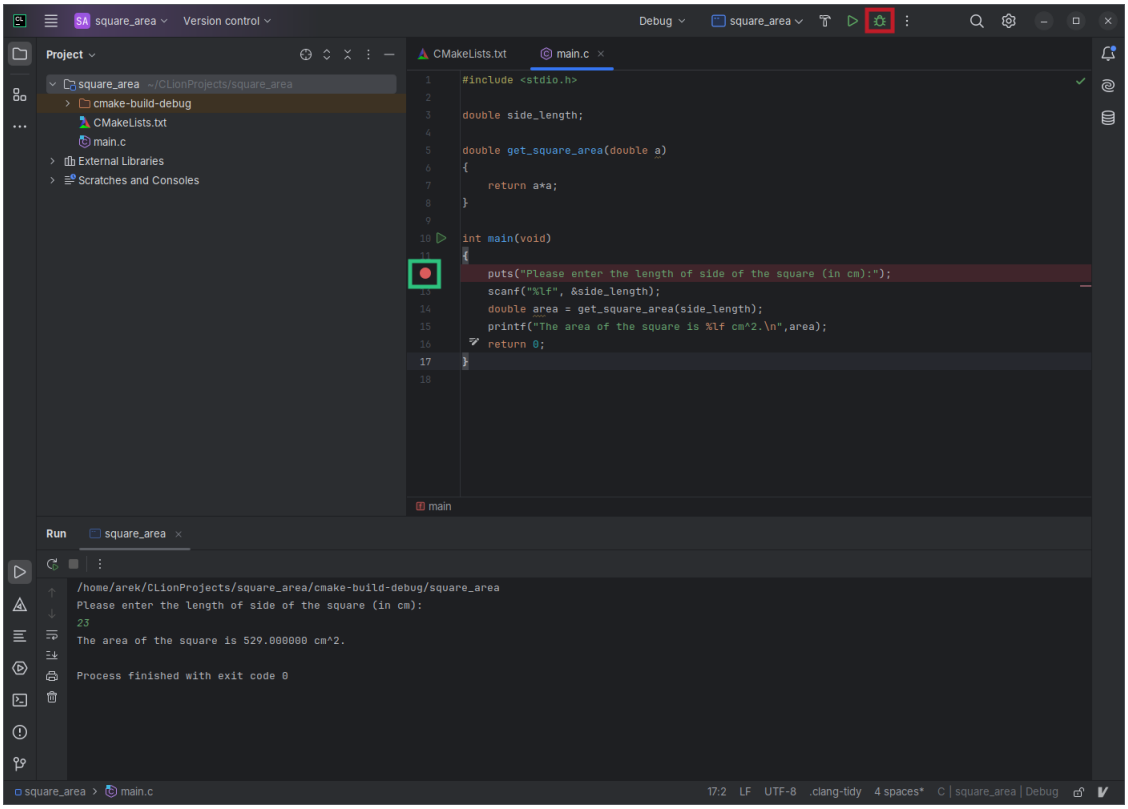


Figure 34: Setting a breakpoint

To start debugging, click the green insect icon shown in Figure 35, or use the `Shift + F9` keyboard shortcut.

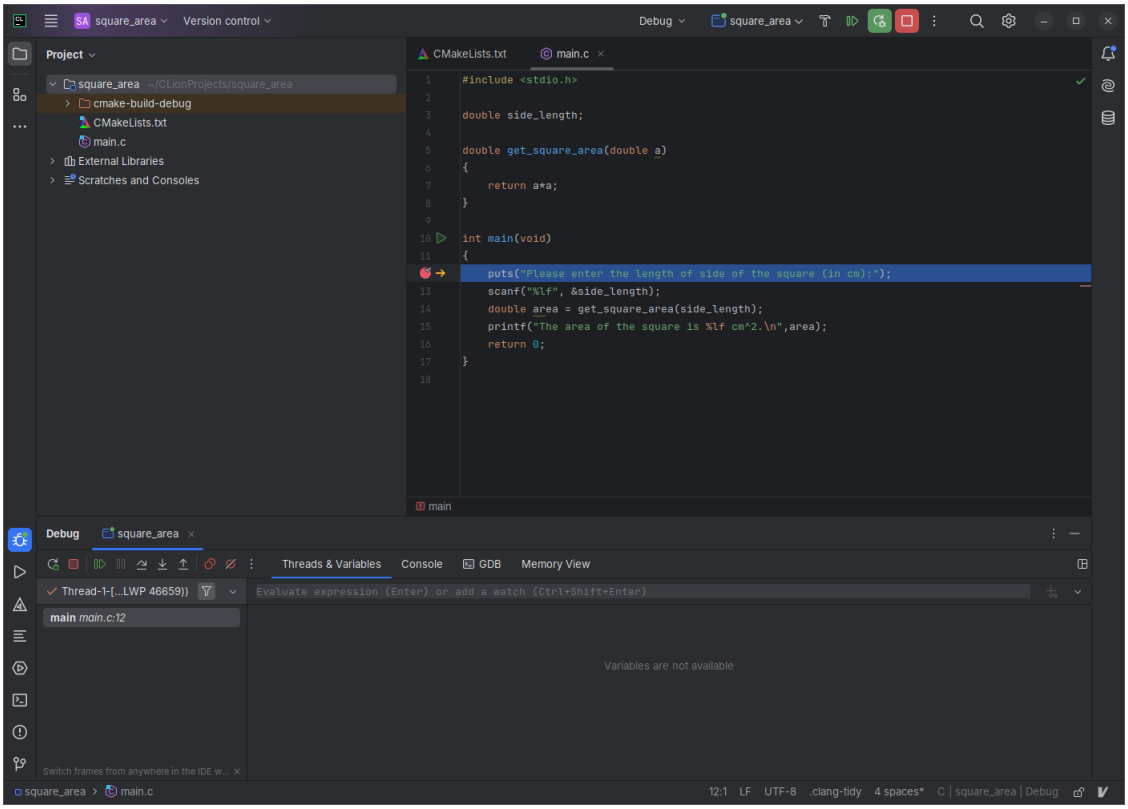


Figure 35: Starting program debugging

The program execution pauses at the line where the breakpoint was set.

In the bottom window, variables or expressions whose values should be observed during debugging can be added to the watch list.

For example, the variables may be:

`side_length`

and

`area`.

Enter the name of each variable separately in the field located at the top of the window and confirm it using the button with the plus sign, as shown in Figure 36.

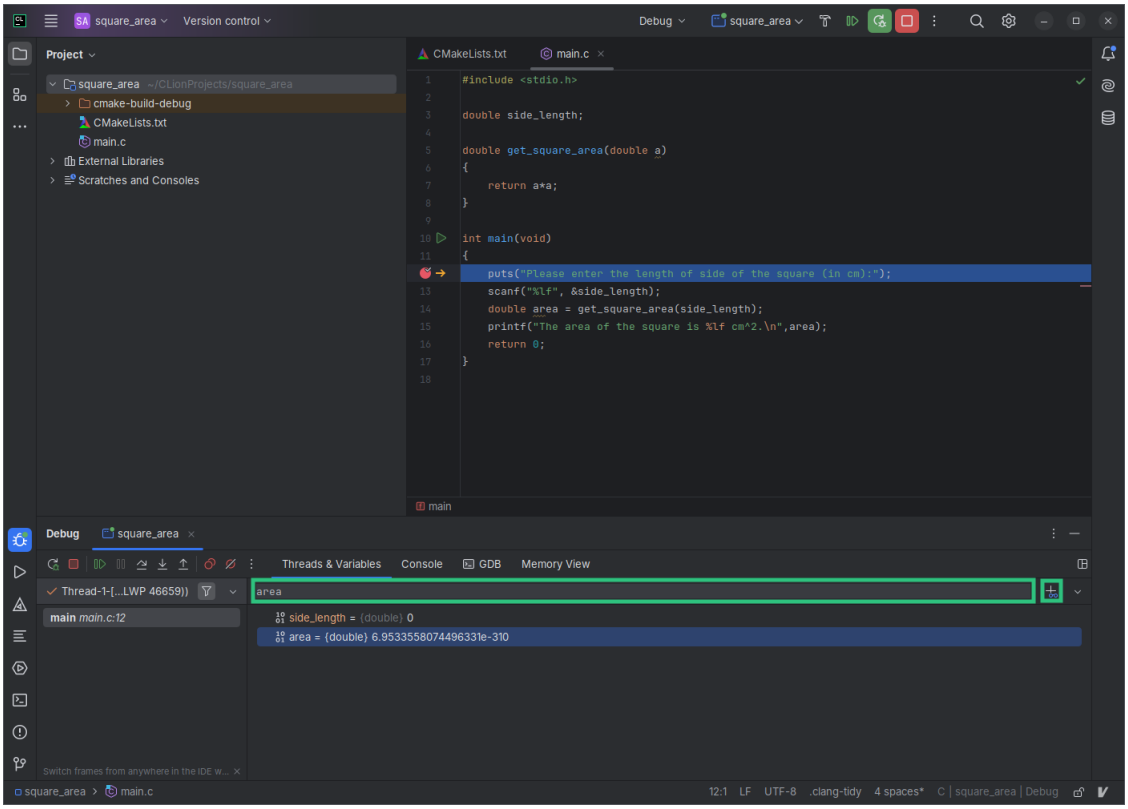


Figure 36: Adding variables to the watch list

Notice that the initial value of the global variable `side_length` is `0.0` by default.
The local variable `area` may initially contain a value that appears random.
It is not necessary to add local variables manually to the watch list.
CLion automatically displays them when the debugger enters a function where they are declared.
This is why the `area` variable may appear twice in the debugger view.
To execute the program line by line, press the `F8` key.
The debugger executes the next source-code line.
In this example, the next instruction calls the `scanf()` function, which waits for input from the keyboard.
Switch to the *Console* tab in the bottom window, as shown in Figure 37, and enter a number.

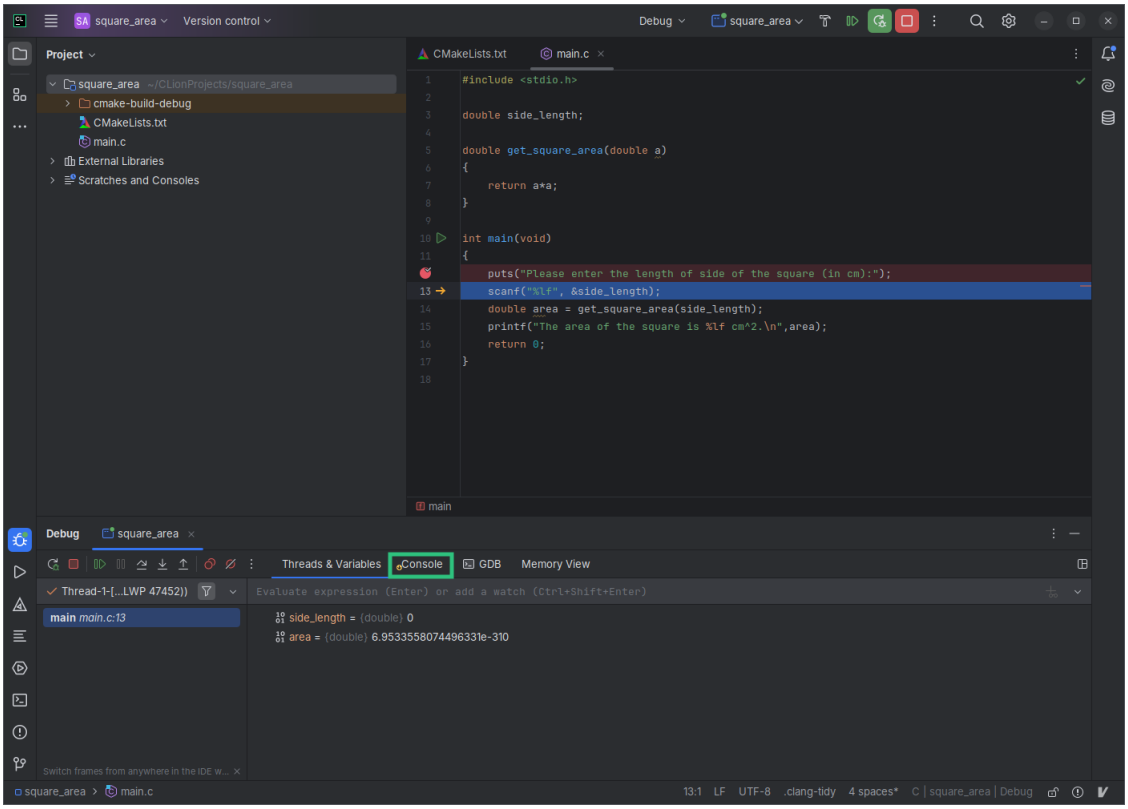


Figure 37: Switching to the console while debugging

After entering the value, switch back to the *Threads & Variables* tab shown in Figure 38.

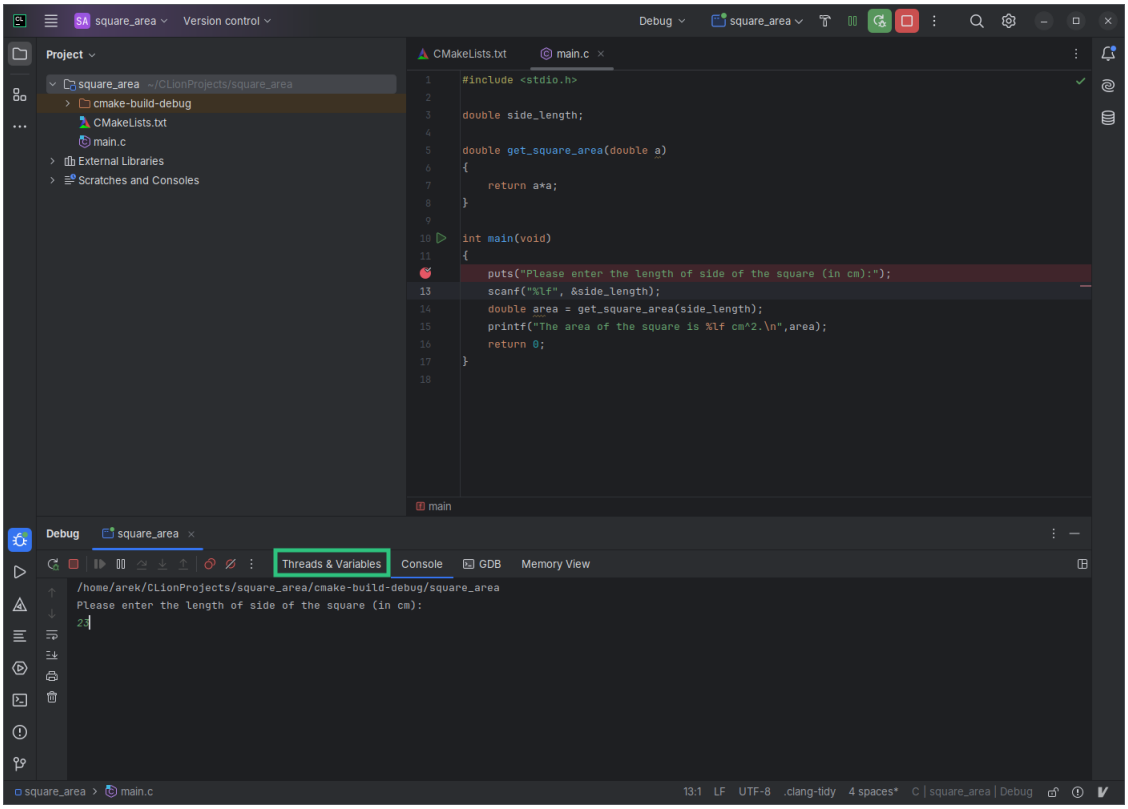


Figure 38: Entering a value while debugging

When the program pauses at the line where `get_square_area()` is called, press the `F7` key, as shown in Figure 39.

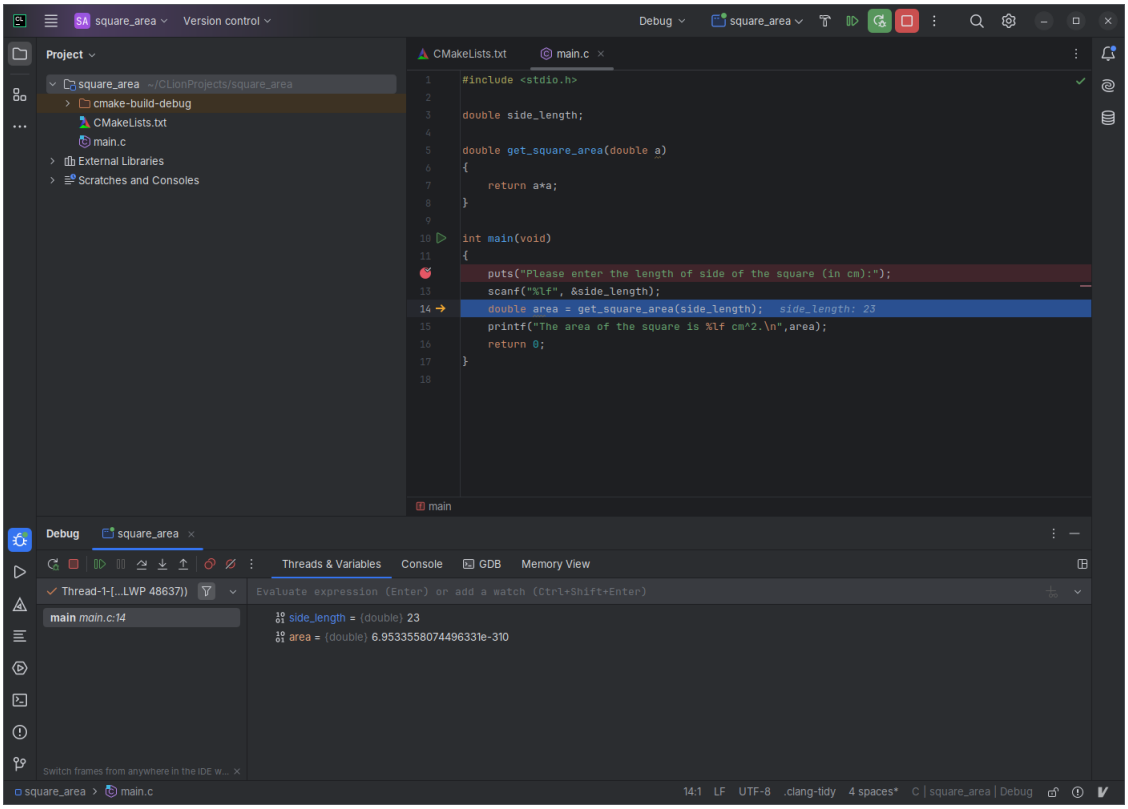


Figure 39: Invocation of the `get_square_area()` function

Unlike `F8`, the `F7` key allows the programmer to enter the called function and observe how its instructions are executed.

This is shown in Figure 40.

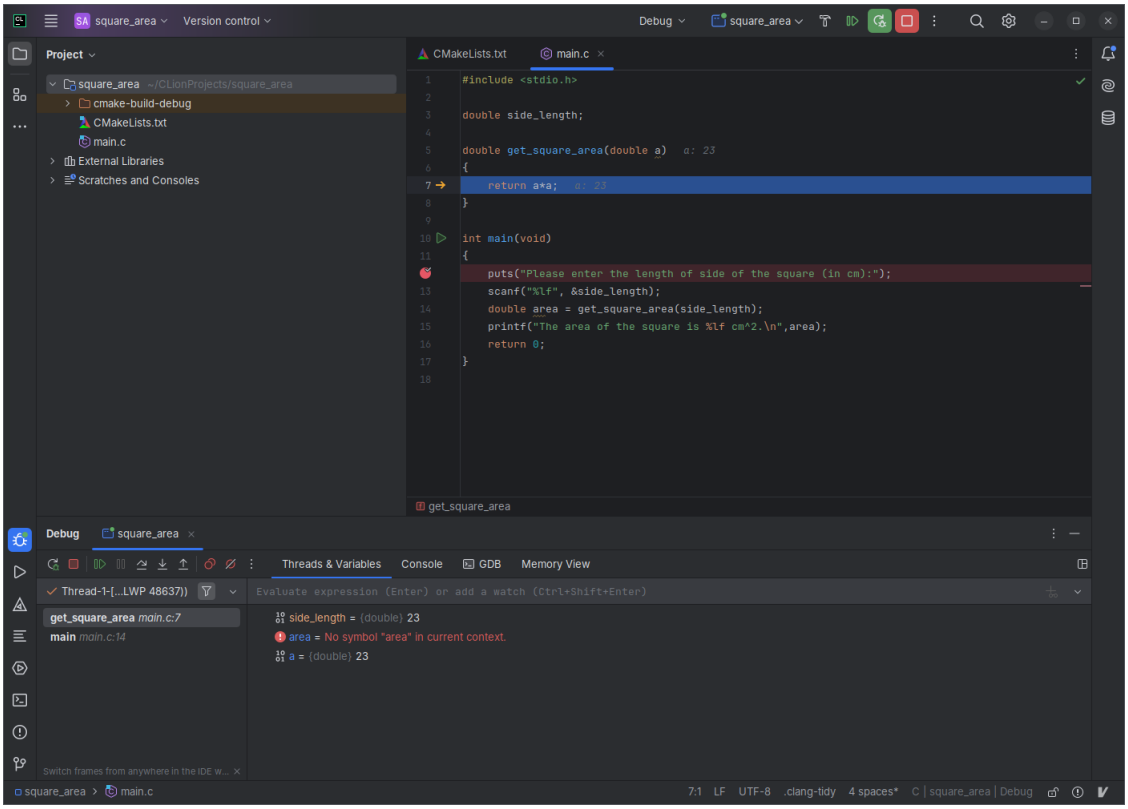


Figure 40: Execution of the `get_square_area()` function

Pressing `F8` instead would execute the entire function call as one debugging step.

The `F8` key can then be used to continue executing the program until the `return 0;` statement in `main()` is reached.

If CLion is running on Linux, debugging can be stopped using the red stop button at the top of the editor window, shown in Figure 41.

On Windows, it is also possible to continue pressing `F8` until the program finishes.

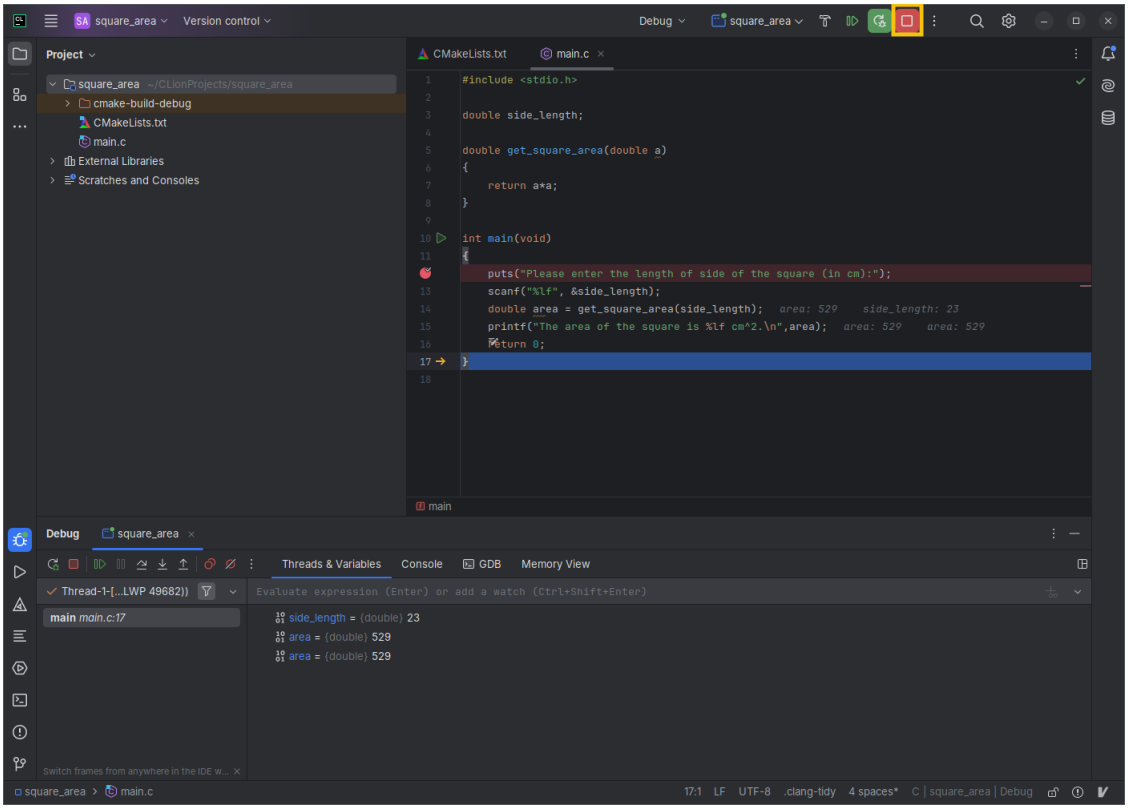


Figure 41: Finishing program debugging

The program may appear to work correctly, but complete Exercise 5 to verify whether this is really the case.

8 Example

Listing 1 presents the source code of a program used in several exercises in Section 9.

The program defines a function named `calculate_area()` that calculates the area of a circle.

It has one input parameter of type `double`, which is used to pass the radius of the circle.

The function returns the area of the circle.

The returned value is also of type `double`, so both the function parameter and its result are floating-point numbers.
2

The `calculate_area()` function is called from the `main()` function.

The program asks the user to enter the radius of the circle and then calls `scanf()` to store the entered value in a variable named `radius`.

If the radius is not negative, it is passed to the `calculate_area()` function, which calculates and returns the area of the circle.

The result is displayed using `printf()`.

If the radius entered by the user is negative, the program displays an error message.

The two cases are handled using an `if` statement.

Important! The source code contains two defects.

One is a syntax error reported by the compiler.

The other is a logic error that causes the program to behave incorrectly.

Listing 1: Source code of a program that calculates the area of a circle

[2] Floating-point numbers are representations of real numbers in computer memory.

```c
#include <stdio.h>

#define PI 3.1415

double calculate_area(double r)
{
    return PI * r * r;
}

double radius;

int main(void)
{
    puts("Please enter the length of the radius of the circle");

    scanf("%lf", &radius);

    // The program verifies whether the radius is greater than or equal to zero.
    if (radius <= 0.0)
        printf(
            "The area of the circle with radius %f cm is %f cm^2.\n",
            radius,
            calculate_area(radius)
        )
    else
        puts("The given radius is incorrect.");

    return 0;
}
```

9 Exercises

1. Read the laboratory guide and learn how to use the CLion IDE.
2. Find and correct the syntax error in the program presented in Listing 1 in Section 8.
3. Find and correct the logic error that occurs when the program from Listing 1 is executed. Use the debugger.
4. Find out what other facilities CLion provides for programmers. Try to learn how to use some of them.
5. Use debugging to determine whether the program calculating the area of a square produces a mathematically correct result when the user enters a negative value for the length of the square's side.

   If it does not, try to fix the problem using the program that calculates the area of a circle as an example.