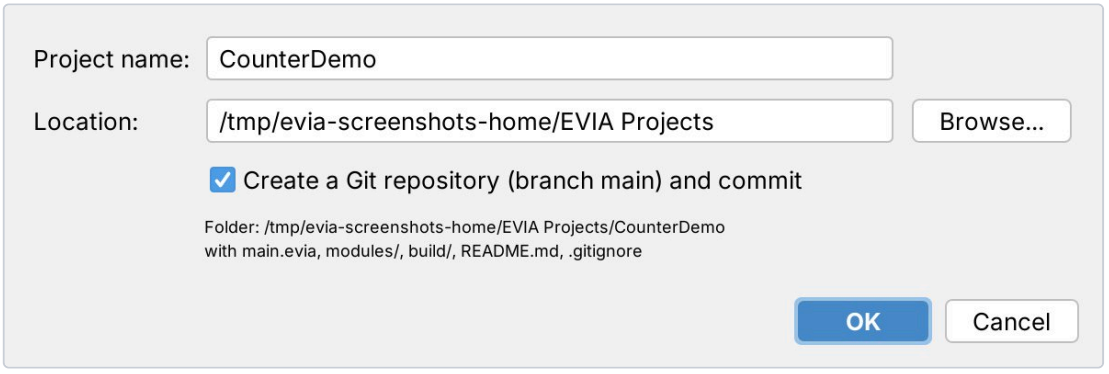
EVIA Paradigma: tutorial

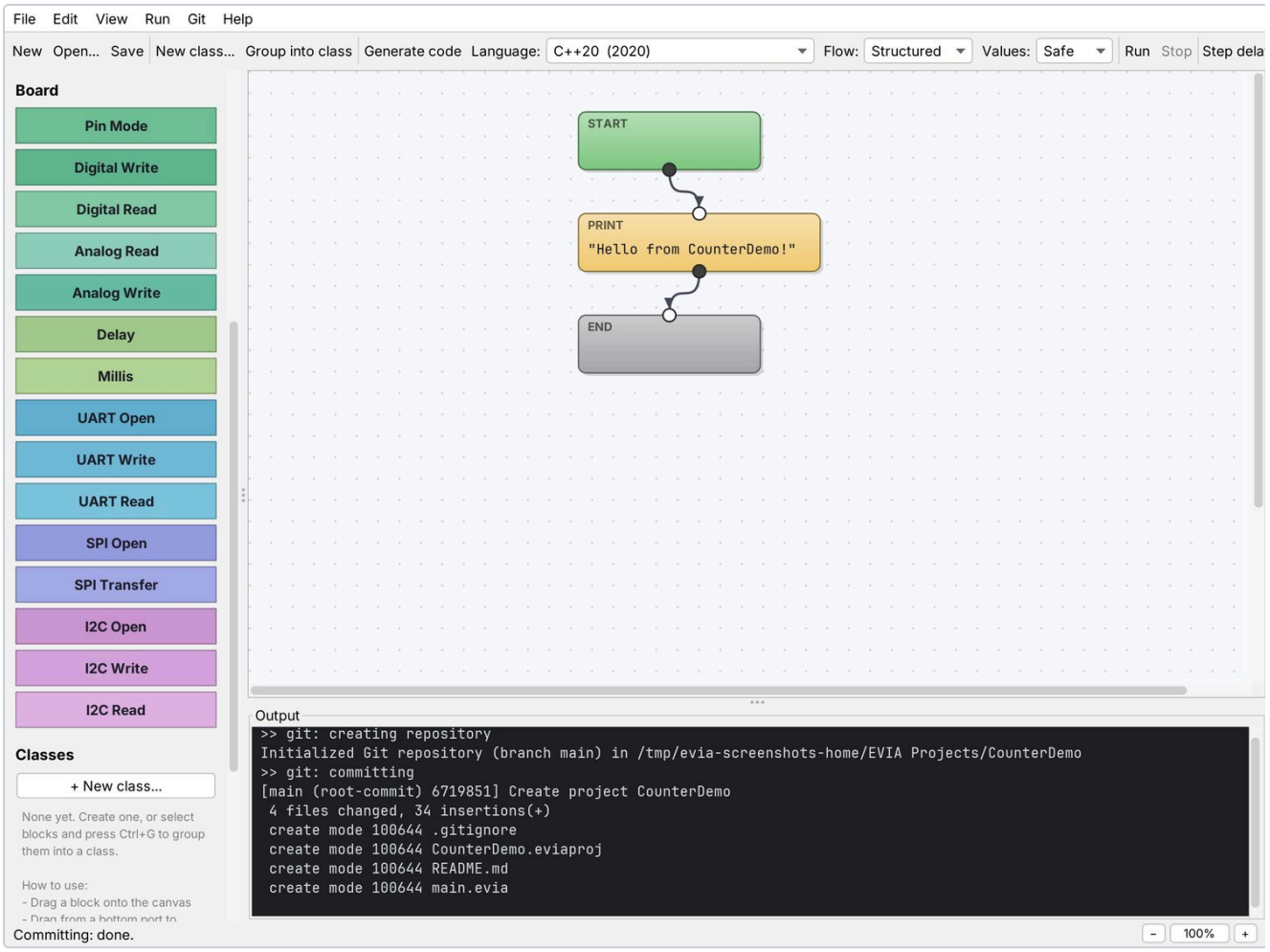
This walk-through builds a small project with a loop and a class, runs it, looks at the C and C++ it becomes, and puts it under Git. Every picture is the real application (they are regenerated with `scripts/take-screenshots.sh`).

1. Create a project

File → New Project... (Ctrl+Shift+N). Enter a name and a location; keep *Create a Git repository* checked if git is installed.



The project opens with a starter program, *Start → Print → End*. The output panel shows that git created the repository and the first commit.

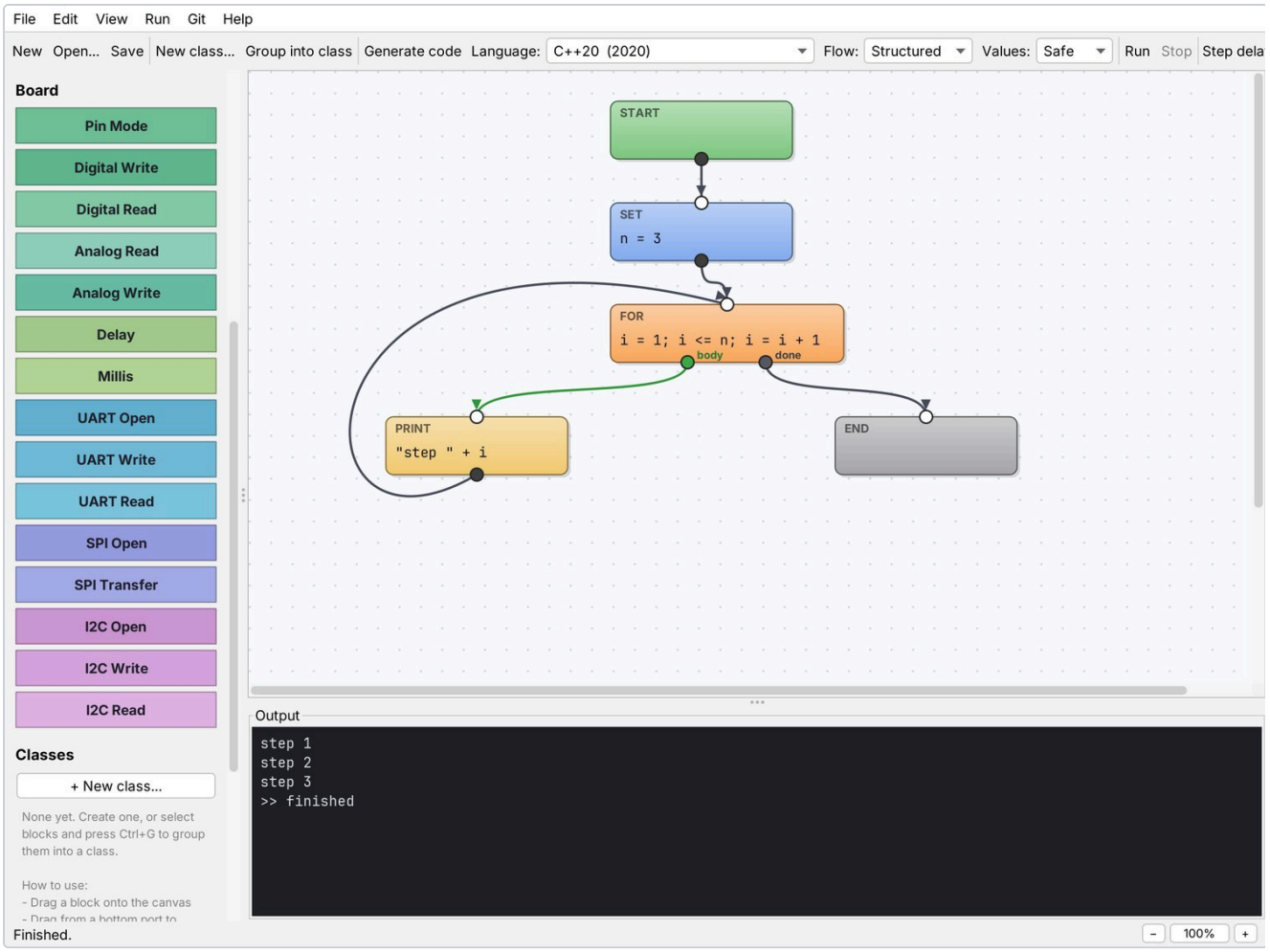


2. Build a loop

Drag blocks from the palette onto the canvas and connect them: drag from a block's bottom port (dark dot) to the next block's top port (white dot). Double-click a block to edit its code.

- **Set**: `n = 3`
- **For**: `i = 1; i ⩽ n; i = i + 1`. Its `body` port goes to **Print** `"step " + i`; wire the end of the body **back into the For block** to start the next pass. Its `done` port goes to **End**.

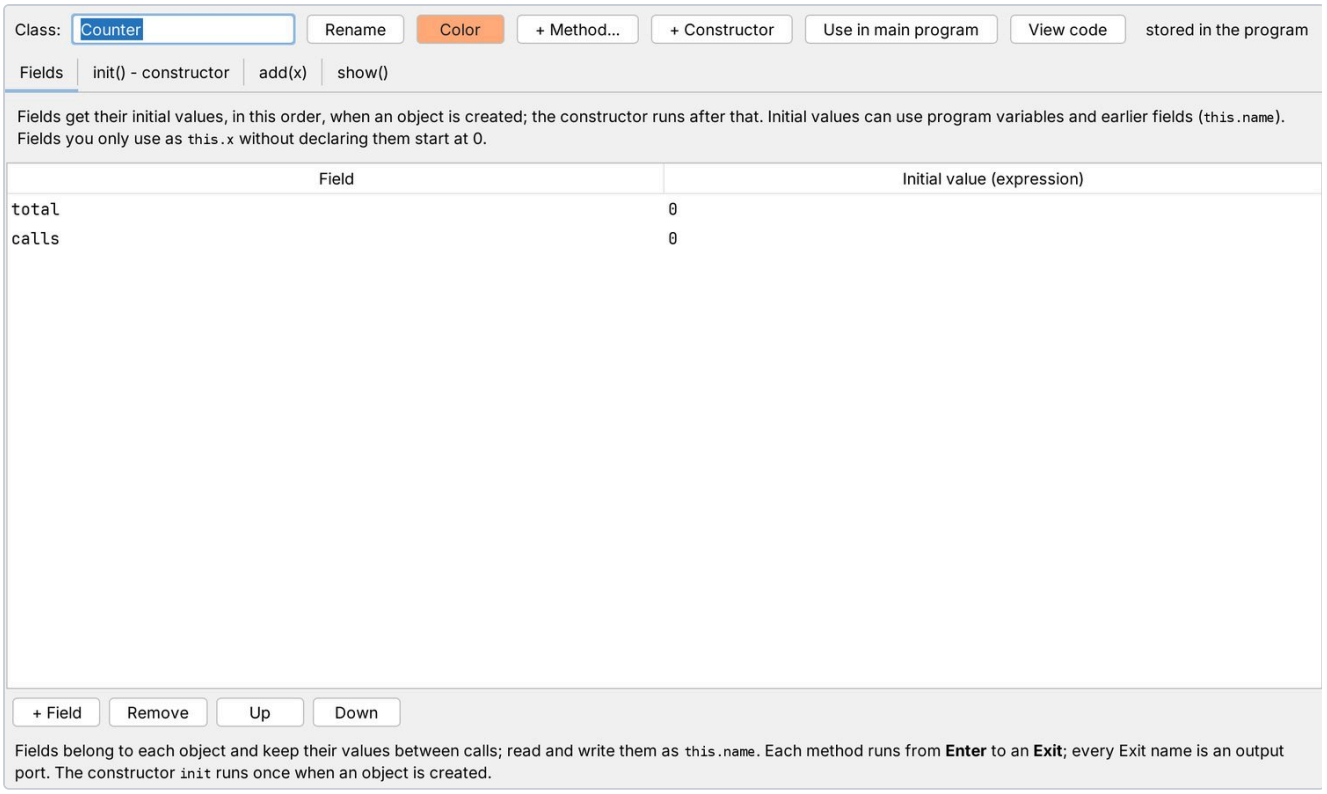
Press **Run** (F5). Set *Step delay* to watch the active block move. Turn the mouse wheel over the diagram to zoom in and out.



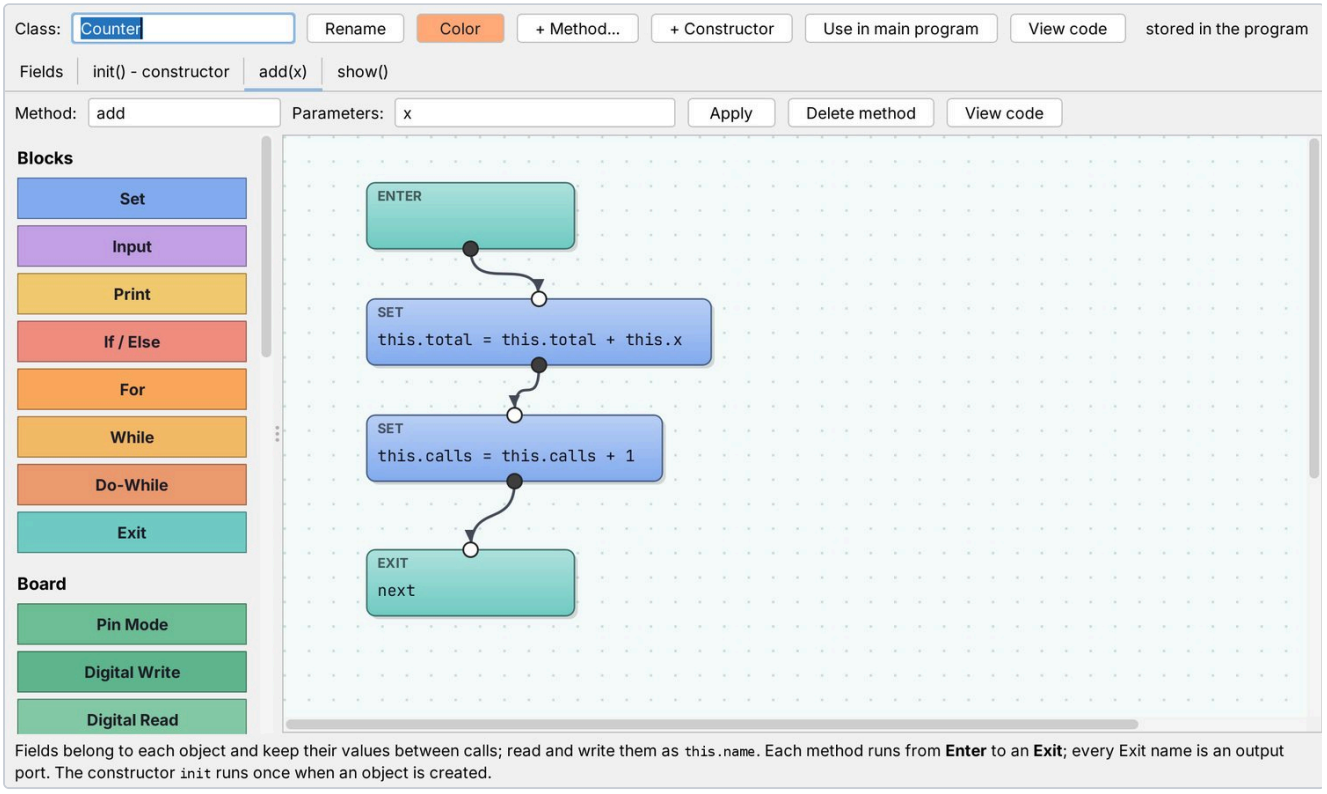
3. Create a class

Edit → New Class... (or *+ New class...* in the palette), name it `Counter`. The class window has a **Fields** tab and one tab per method.

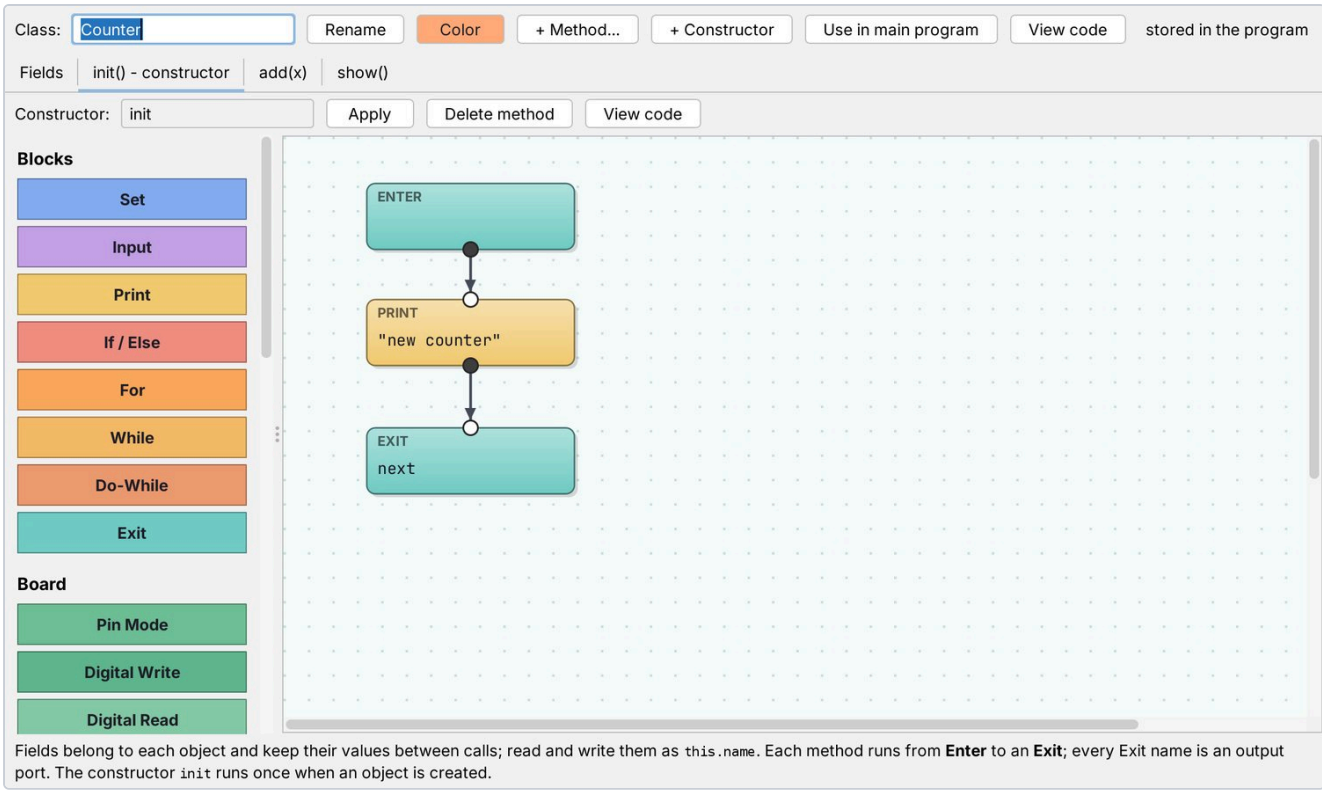
Fields: add `total` and `calls`, both starting at `0`. Every object gets its own copy.



Methods: rename the first method to `add` with parameter `x` (press **Apply**), then build its body from **Enter** to **Exit**. Inside a method, fields and parameters are written as `this.name`:

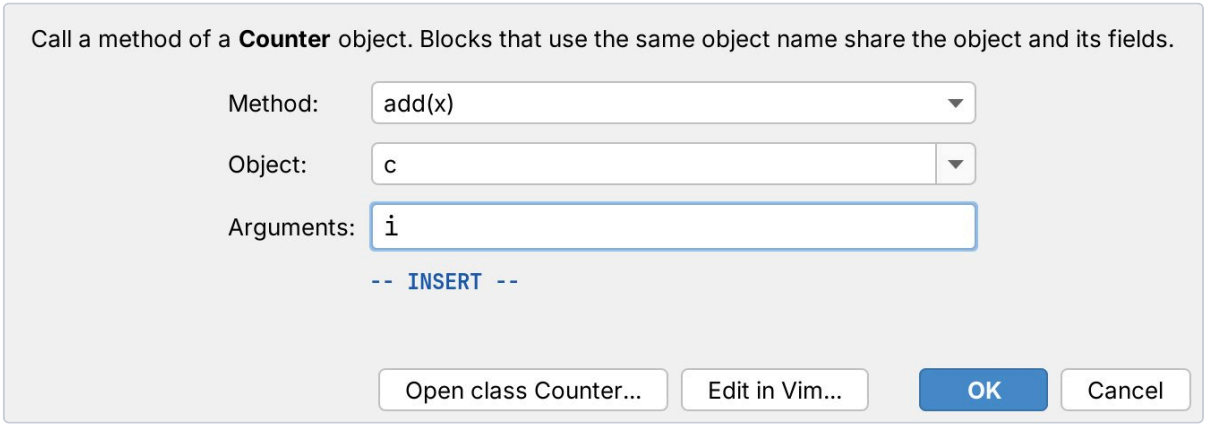


+ Constructor adds `init`, which runs once when an object is created (after the fields get their initial values). Here it prints `"new counter"`. **+ Method...** adds `show` that prints `"total = " + this.total + " after " + this.calls + " calls"`.

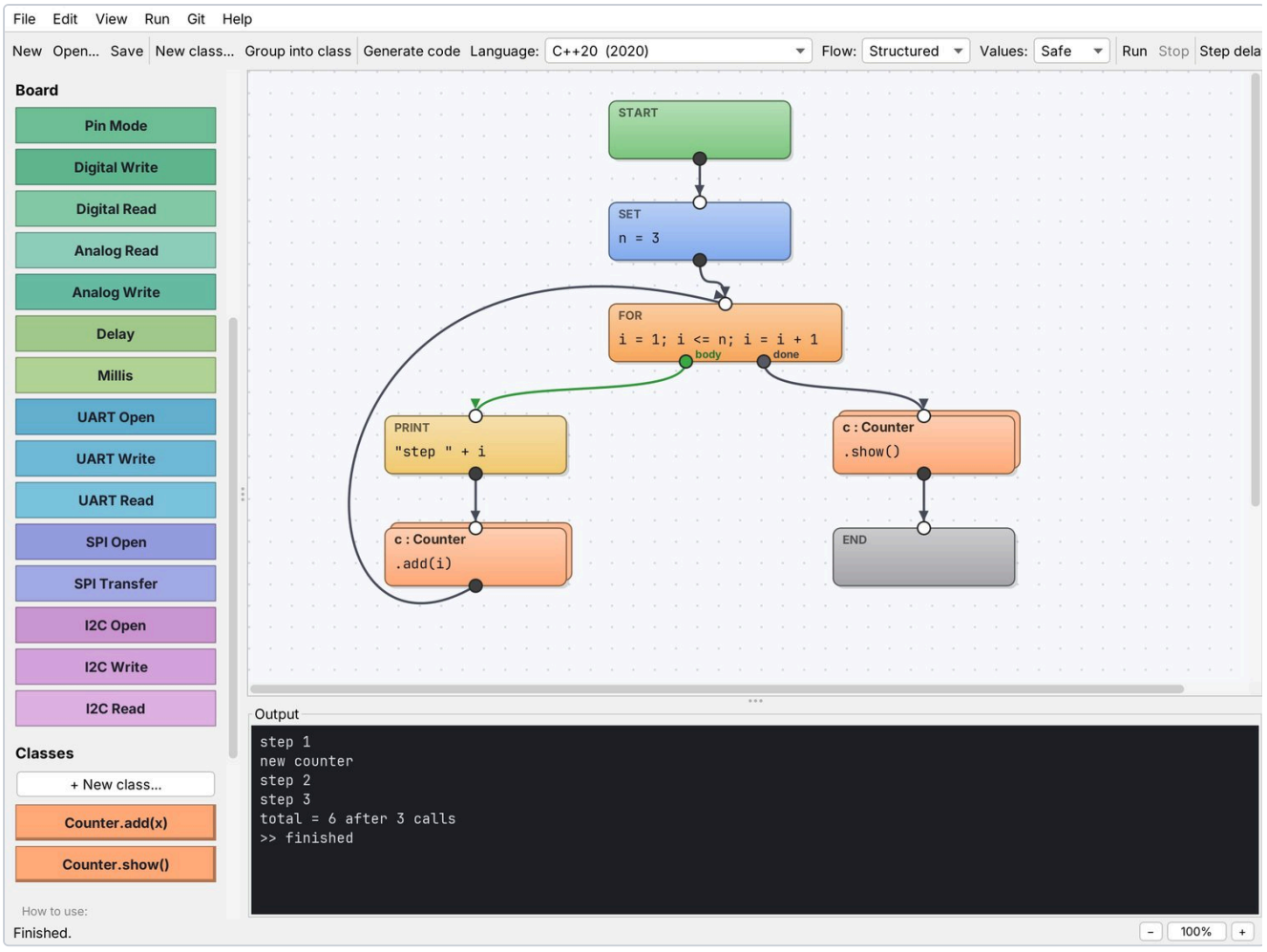


4. Use the class

The palette now lists `Counter.add(x)` and `Counter.show()`. Drag `Counter.add(x)` into the loop body after **Print** and `Counter.show()` between `done` and **End**. Double-click a call block to choose the method, the **object** and the arguments. Give both blocks the object name `c` so they share one object:

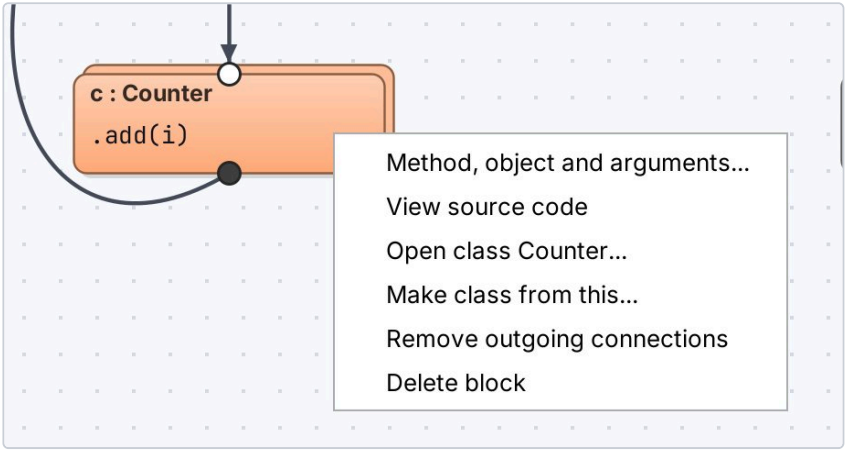


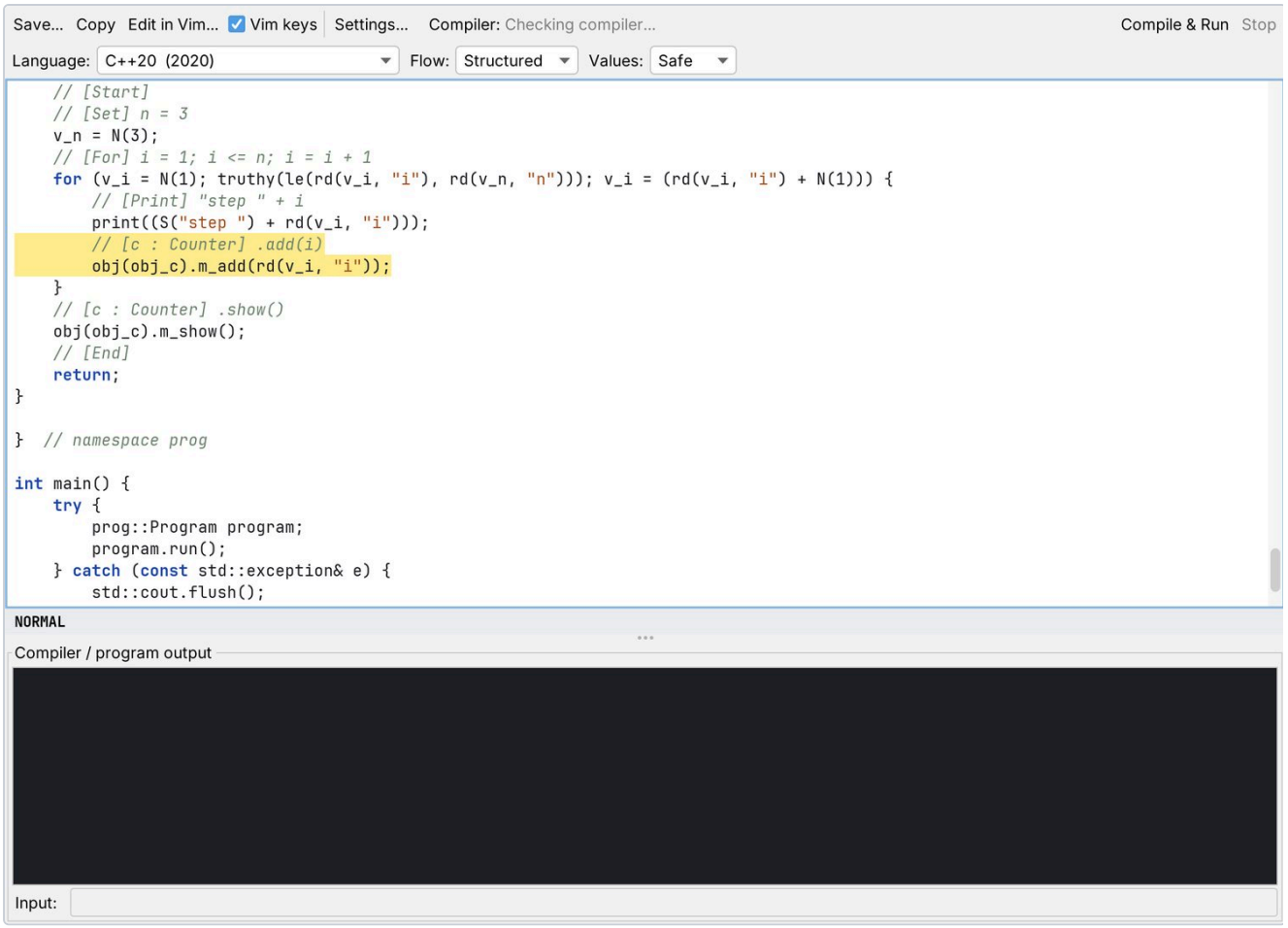
Run it. `c` is created on its first use (so `new counter` is printed then), `add` runs three times, and `show` prints the total:



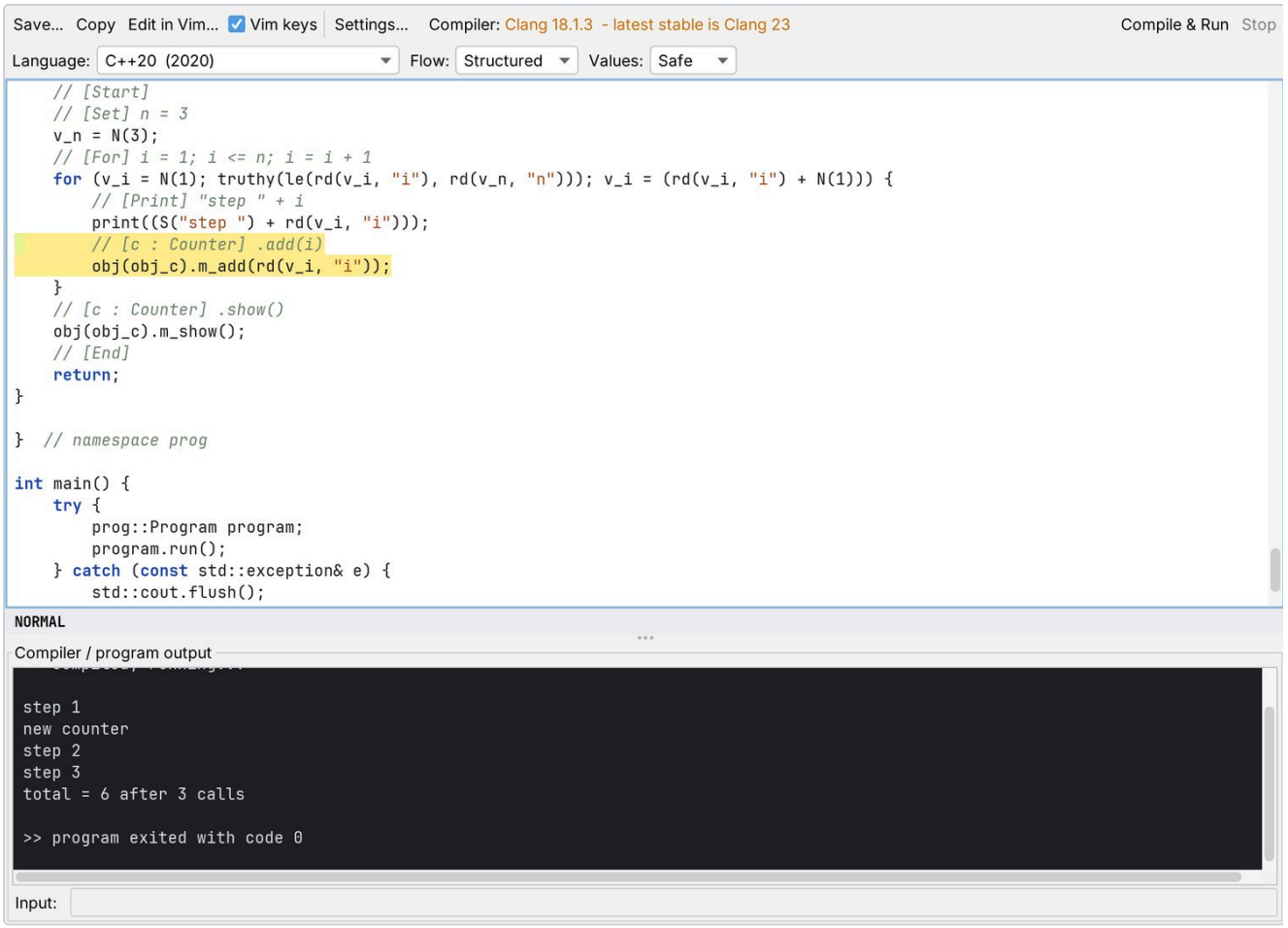
5. See the code for any block

Right-click a block → **View source code**. The generated code opens with exactly that block's lines highlighted (here C++20, the default):



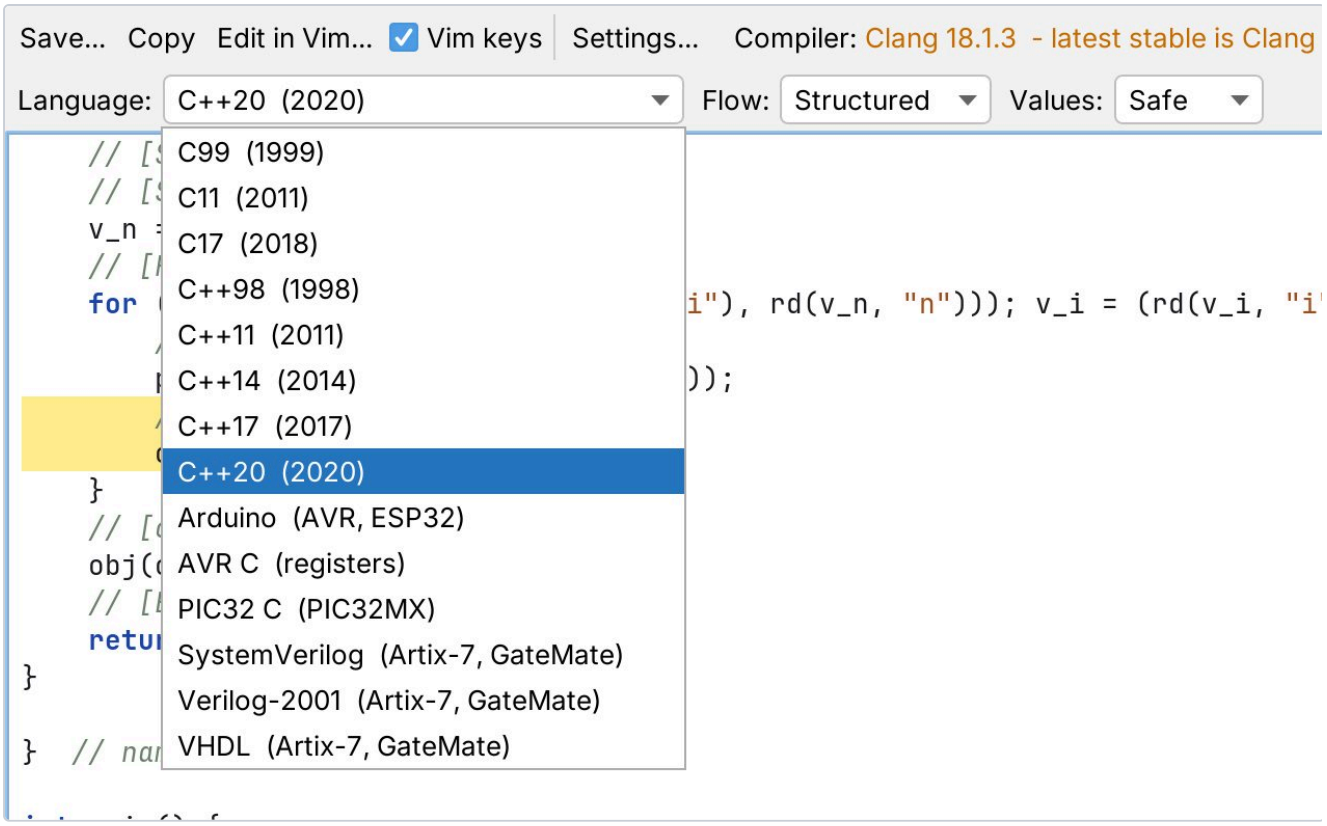


Compile & Run builds it with Clang and runs it. The compiled program prints the same as the interpreter. On Windows the app installs the llvm-mingw Clang for you (no Visual Studio needed).

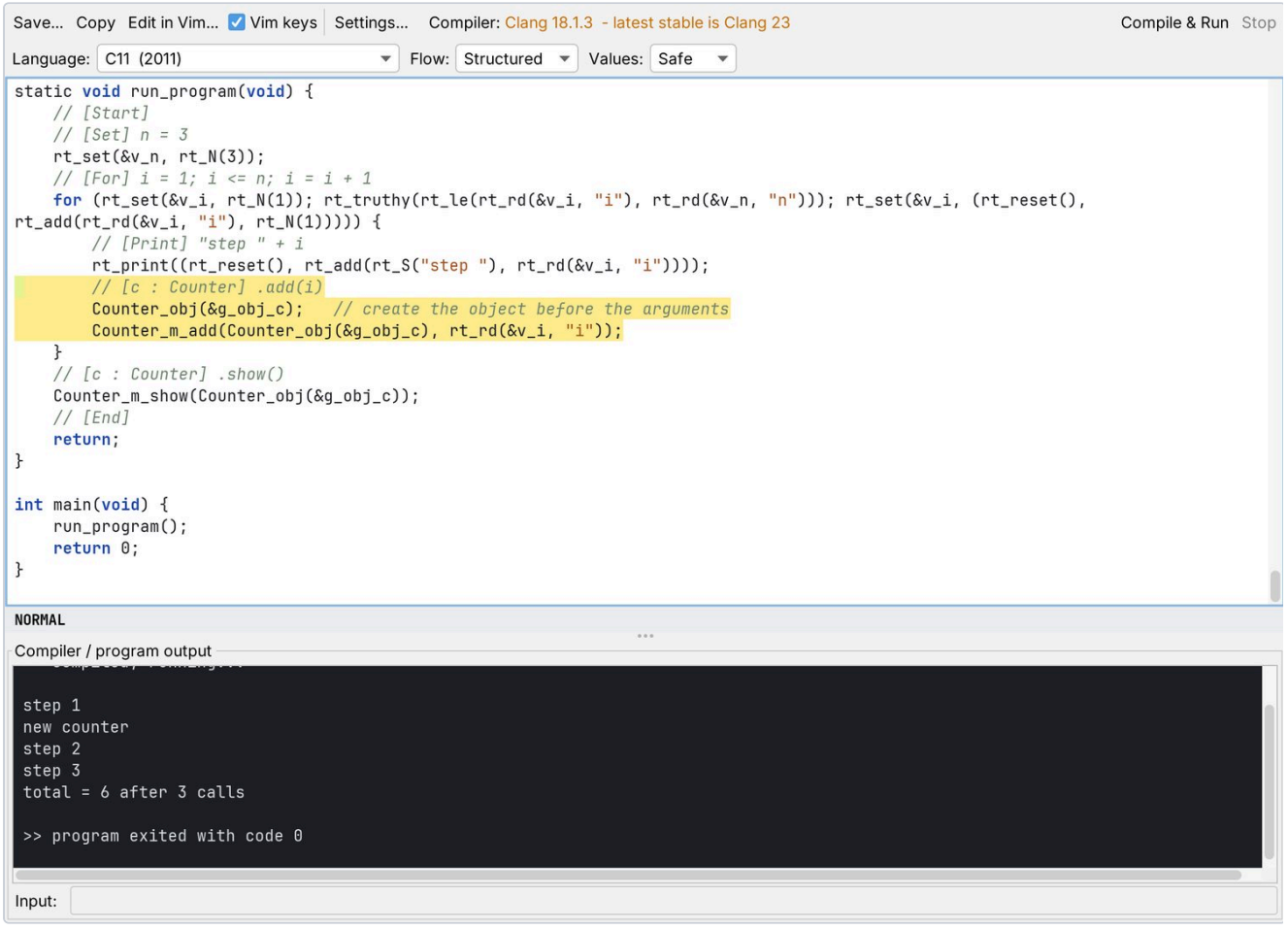


6. Choose C or another C++ standard

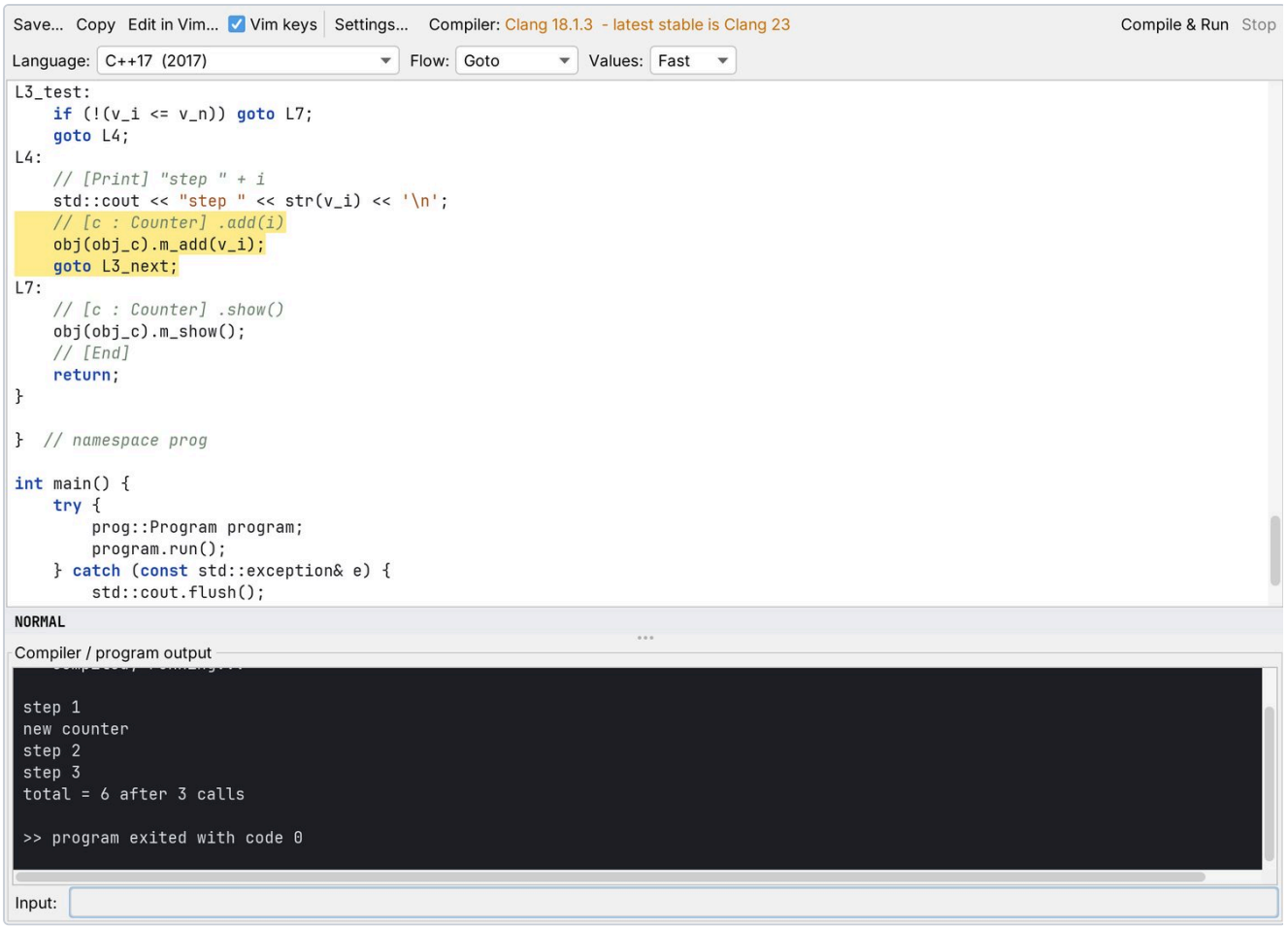
Pick a **Language** in the drop-down list (in the code window or the main toolbar); the code is generated again at once, and the same block stays highlighted. **Flow** and **Values** choose how it is written (see C and C++ code generation).



The same program in **C11**, compiled with `clang -std=c11` and run: the output is identical.

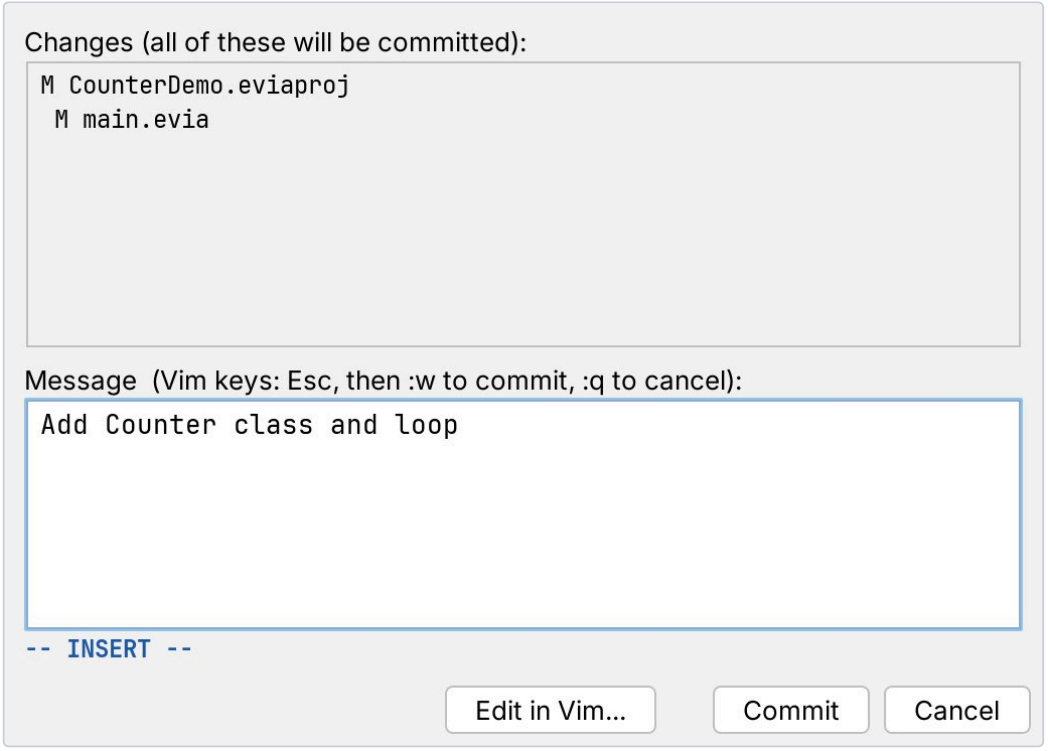


C++17 with **Goto** flow (every block a label, every wire a jump) and **Fast** values (plain `double`):

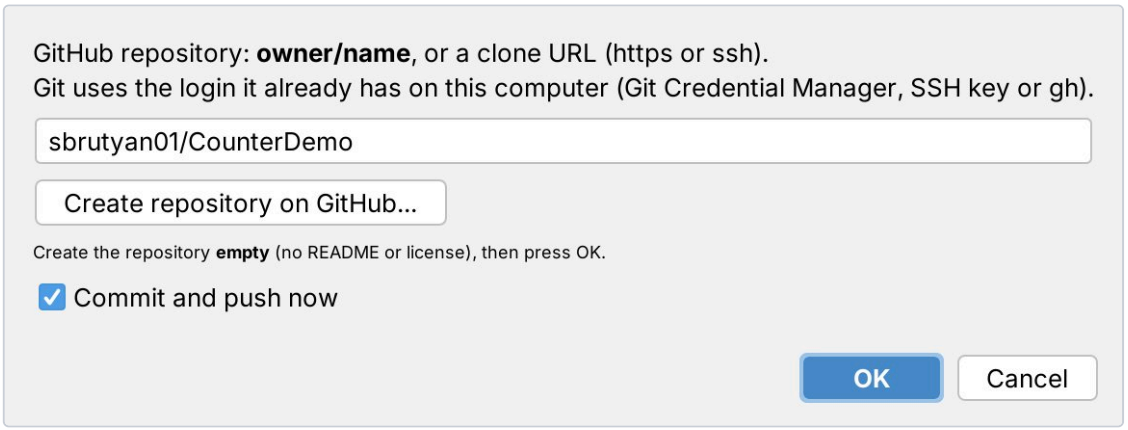


7. Commit and publish on GitHub

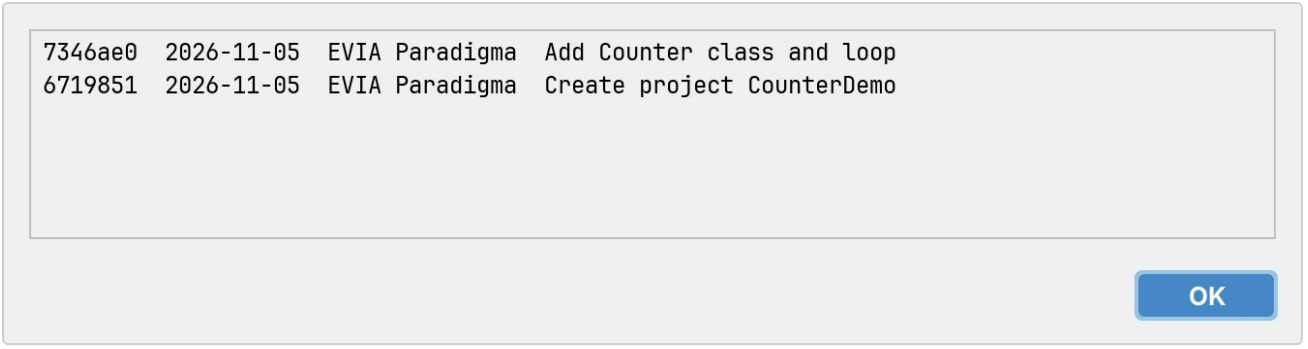
Git → Commit... (Ctrl+K) saves and commits all changes. The message box has Vim keys (`Esc`, then `:w` commits).



Git → Connect to GitHub...: type `owner/repo`. *Create repository on GitHub...* opens github.com with the name filled in; create it **empty**, press **OK**, and the project is pushed. Git uses the login it already has on your computer (Git Credential Manager, an SSH key or `gh auth login`).

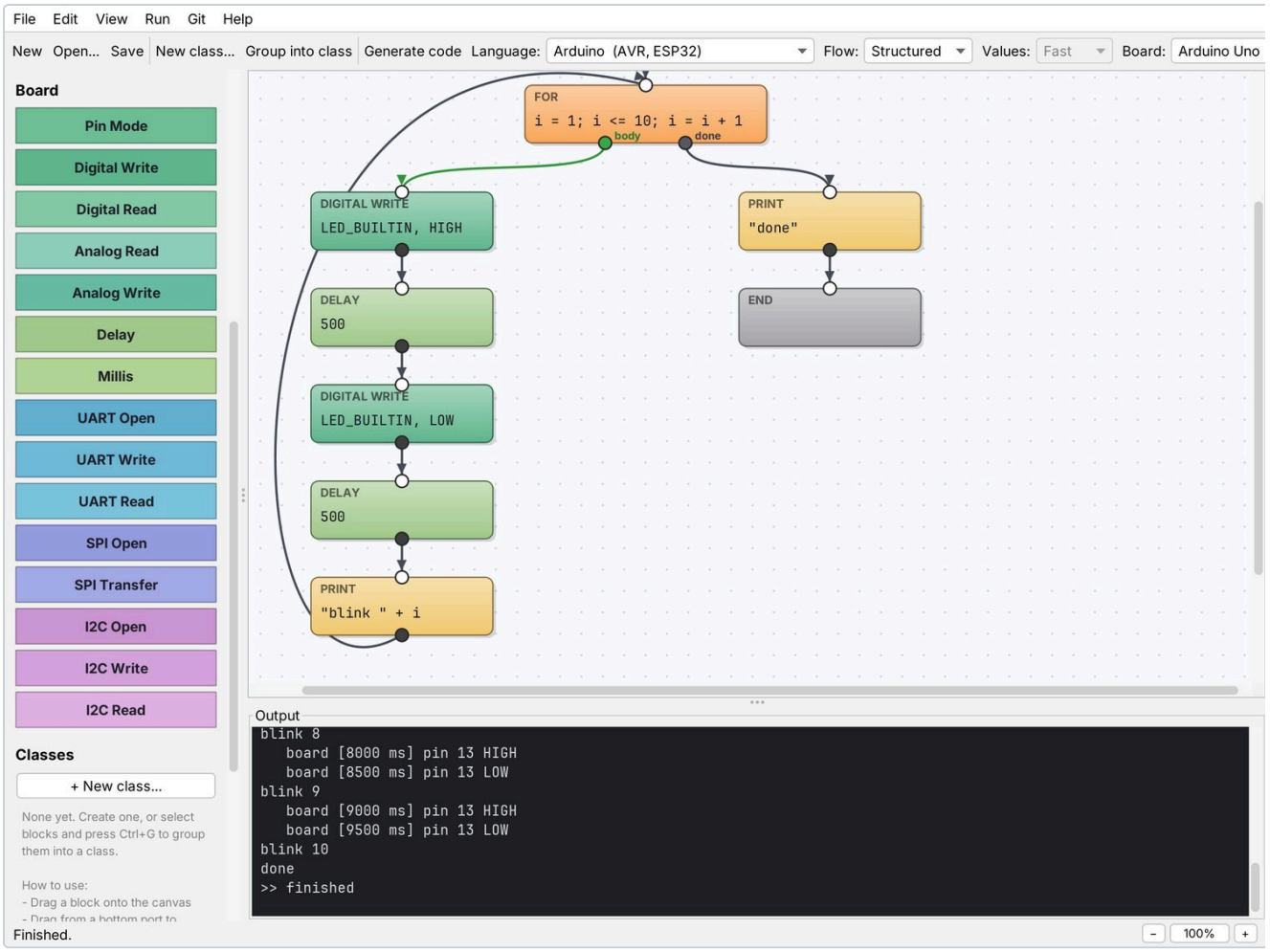


Git → History lists the commits:

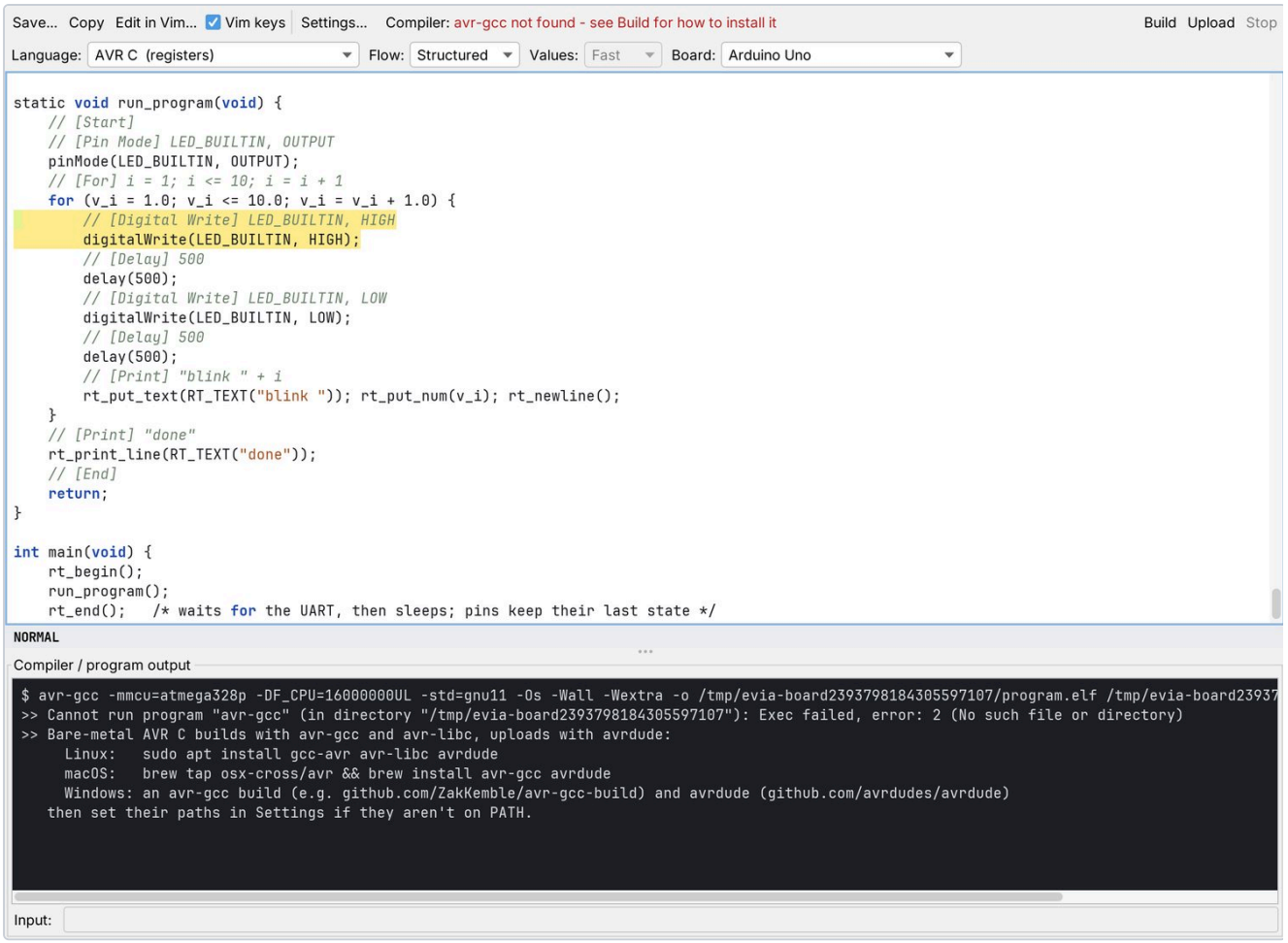


8. Program an Arduino or ESP32 board

File → Open Example → Arduino and ESP32 Boards → Blink the built-in LED. The **Board** blocks (green, under *Board* in the palette) set pins, read buttons and sensors, and wait. **Run** (F5) simulates an Arduino Uno: the output shows every pin change with its time.



Choose **Language: Arduino** and a **Board**, then **Generate code** (F7): one sketch for any of the 12 boards. **Build** compiles it with arduino-cli, **Upload** sends it to the board; Print and Input use the Serial Monitor at 115200 baud. **AVR C** writes the same program without the Arduino core, straight to the chip's registers, built with avr-gcc. See Arduino and ESP32 boards.



9. Talk to other chips: UART, SPI and I2C, and PIC32

Three more programs under **File → Open Example → Arduino and ESP32 Boards** use the bus blocks: *UART: send a byte, read it back*, *SPI: transfer a byte* and *I2C: write and read a device register*. The blocks are in the *Board* group:

Block	Code	What it does
UART Open	`9600`	starts the serial port at this baud rate
UART Write	`65`	sends one byte (0..255)
UART Read	`c`	next received byte, or -1 if none
SPI Open	`1000000`	SPI master at this clock in Hz (mode 0, MSB first)
SPI Transfer	`reply = 66`	sends a byte, stores the byte received at the same time
I2C Open	`100000`	I2C master at this clock in Hz
I2C Write	`80, 0, 42`	address (7-bit), register, value
I2C Read	`v = 80, 0`	stores the register's value, or -1 if nothing answers

Open a bus before using it. **Run** simulates them: UART Write bytes come back from UART Read, SPI's input is wired to its output, and one I2C device with 256 registers answers at address 80 (80 is 0x50; numbers are decimal).

For a Microchip PIC32MX chip choose **Language: PIC32 C** and the chip in the **Board** list (nine chips from the MX1xx/2xx, MX3xx/4xx and MX5xx/6xx/7xx families), then **Generate code** (F7). The result is bare-metal C for Microchip's free XC32 compiler: the clock, pins, PWM, ADC, UART, SPI and I2C are set up through the registers. **Build** runs `xc32-gcc` and `xc32-bin2hex`; **Upload** runs the command set in **Settings** (`pic32prog {hex}` by default). The pins of the chosen chip family are listed at the top of the generated file.

Please note: the PIC32 code is checked against Microchip's real device headers, but it has not yet been built with XC32 or run on a chip. Treat the first build on your board as a test.

10. Build hardware for an FPGA

Choose **Language: SystemVerilog**, **Verilog-2001** or **VHDL**, then a board in the **Board** list: Digilent Nexys A7-100T, Arty A7-35 or A7-100, Basys 3 (Artix-7), or the Olimex GateMateA1-EVB and Cologne Chip GateMate A1 evaluation board. Open **File → Open Example → FPGA Boards → Blink** and press **Generate code** (F7).

Every block becomes one state of a state machine and takes one clock cycle. Pins are fixed when the design is built, so write them as names: `LED0`, `SW3`, `BTNC`, `JA1`. The first comment of the file holds the pin constraints; with `-o` on the command line they are also written as `.xdc` (Vivado) or `.ccf` (GateMate).

- **Build** and **Upload** use Vivado or the open-source flow for Artix-7 (choose in **Settings**), and yosys, nextpnr-himbaechel, gmpack and openFPGALoader for GateMate.
- Numbers are whole 32-bit values, so `5 / 2` is 2 on the FPGA but 2.5 in **Run**.
- Classes, Input, Analog Read, SPI, I2C and text in variables are not supported yet; the error names the block.

Please note: the output was simulated and synthesised with open-source tools, but not run through Vivado or loaded onto a board.

11. Send the numbers on with MQTT

Open **File → Open Example → Arduino and ESP32 Boards → MQTT sensor**. It publishes a temperature to a broker, sends itself a command and reads it back.

The five **MQTT** blocks are in the *Board* group: *MQTT Connect*, *MQTT Publish*, *MQTT Subscribe*, *MQTT Has Message* and *MQTT Read*. The network name, password, broker and topics are fixed texts, because they are written into the program when it is built.

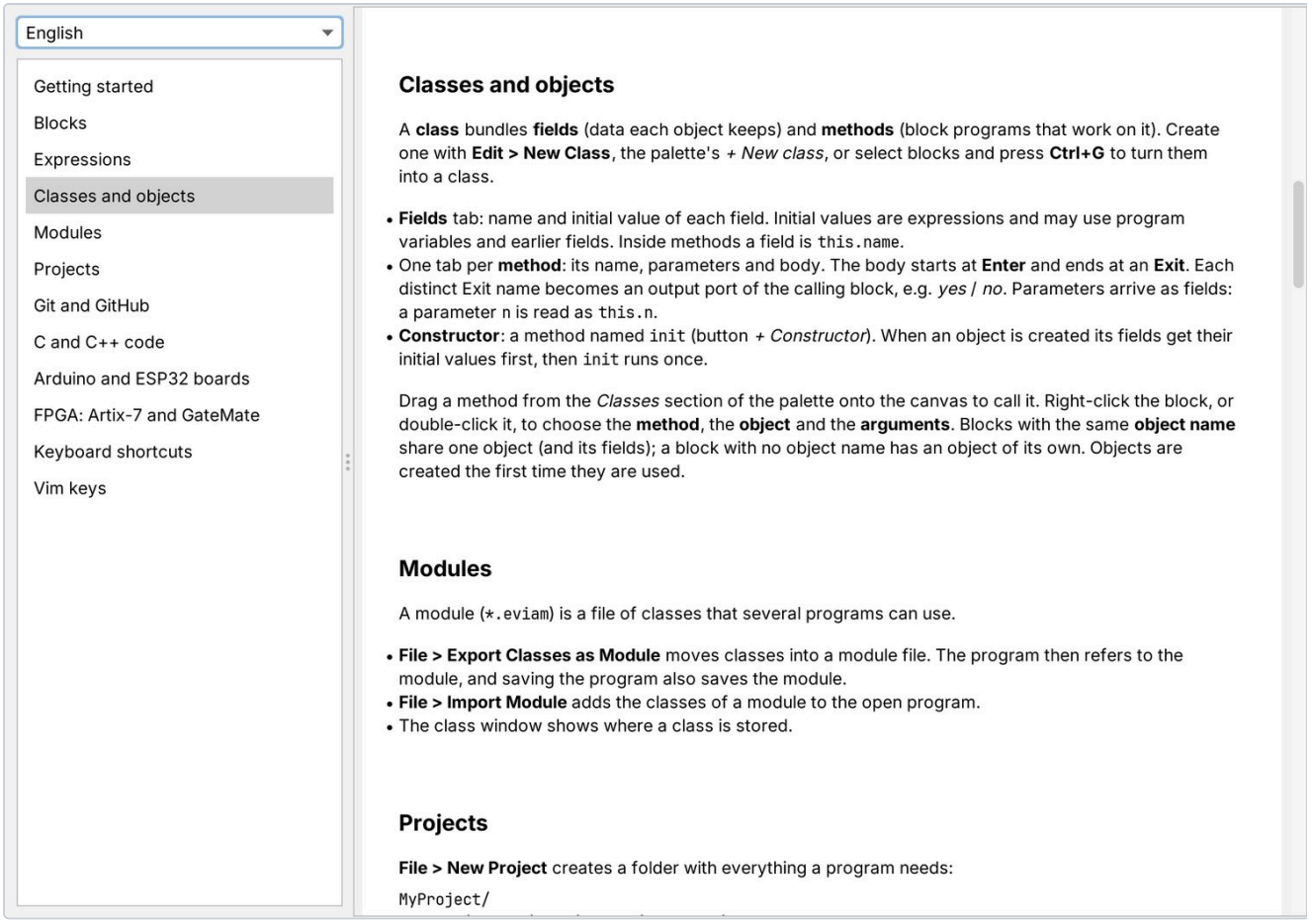
- **Run** answers from a broker on your computer, so the program can be tried without hardware.
- **PIC32 C** drives an ESP8266 or ESP32 module with AT firmware on UART2, while Print keeps UART1.
- **Arduino** on an **ESP32** board uses the board's own WiFi.
- An AVR board refuses the blocks: it has one serial port and Print uses it.

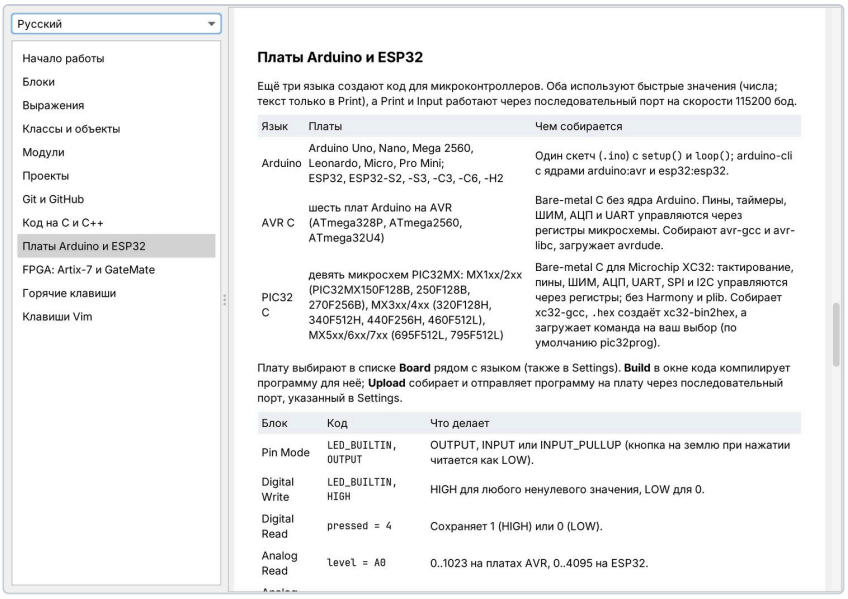
A message is a number. *MQTT Read* gives -1 when nothing has arrived, so ask *MQTT Has Message* first if a real -1 could turn up.

Please note: the MQTT packets are checked against a real broker in CI, but no WiFi module, ESP32 or PIC32 chip was part of that. MQTT over the Ethernet port of the MX695/795 chips is not supported yet.

12. Keep going

- Share classes between programs: **File → Export Classes as Module...**, then **File → Import Module...** in another program.
- **File → Open Example** has ready-made programs, including `classes.evia` (fields, constructor, methods, a module).
- **Help → Tutorial (PDF)** opens this tutorial in English, Russian, Armenian, German, Simplified Chinese, Traditional Chinese or Japanese.
- **Help → Guide** (F1) explains every block, expressions, classes, projects, Git, shortcuts and Vim keys, in English, Russian or Armenian (the list above the topics). **View → Theme** switches between the light and dark look.





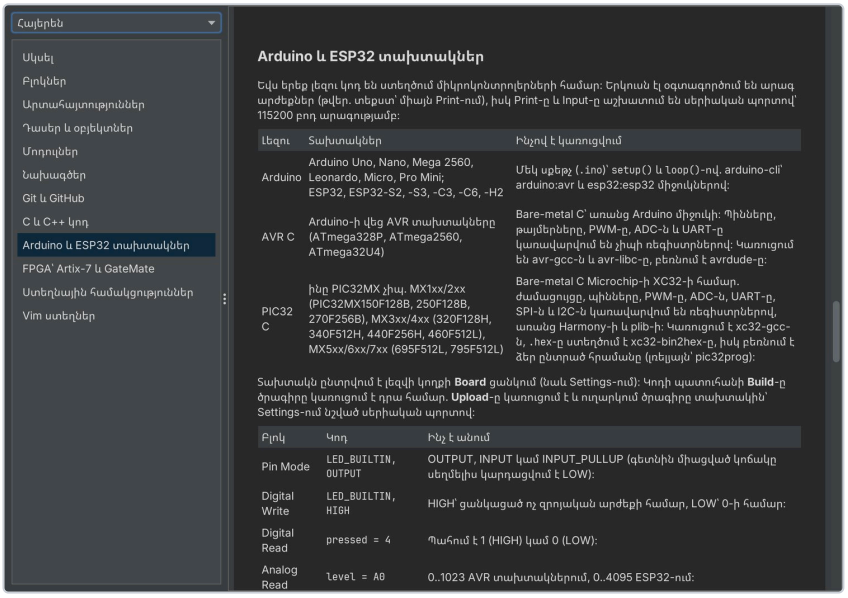
Платы Arduino и ESP32

Ещё три языка создают код для микроконтроллеров. Оба используют быстрые значения (числа; текст только в Print), а Print и Input работают через последовательный порт на скорости 115200 бод.

Язык	Платы	Чем собирается
Arduino	Arduino Uno, Nano, Mega 2560, Leonardo, Micro, Pro Mini; ESP32, ESP32-S2, -S3, -C3, -C6, -H2	Один скетч (`.ino`) с `setup()` и `loop()`; arduino-cli с ядрами arduino:avr и esp32:esp32.
AVR C	шесть плат Arduino на AVR (ATmega328P, ATmega2560, ATmega32U4)	Bare-metal C без ядра Arduino. Пины, таймеры, ШИМ, АЦП и UART управляются через регистры микросхемы. Собирают avr-gcc и avr-libc, загружает avrdude.
PIC32 C	девять микросхем PIC32MX: MX1xx/2xx (PIC32MX150F128B, 250F128B, 270F256B), MX3xx/4xx (320F128H, 340F512H, 440F256H, 460F512L), MX5xx/6xx/7xx (695F512L, 795F512L)	Bare-metal C для Microchip XC32: тактирование, пины, ШИМ, АЦП, UART, SPI и I2C управляются через регистры; без Harmony и plib. Собирает xc32-gcc, `.hex` создаёт xc32-bin2hex, а загружает команда на ваш выбор (по умолчанию pic32prog).

Плату выбирают в списке **Board** рядом с языком (также в Settings). **Build** в окне кода компилирует программу для неё; **Upload** собирает и отправляет программу на плату через последовательный порт, указанный в Settings.

Блок	Код	Что делает
Pin Mode	`LED_BUILTIN, OUTPUT`	OUTPUT, INPUT или INPUT_PULLUP (кнопка на землю при нажатии читается как LOW).
Digital Write	`LED_BUILTIN, HIGH`	HIGH для любого ненулевого значения, LOW для 0.
Digital Read	`pressed = 4`	Сохраняет 1 (HIGH) или 0 (LOW).
Analog Read	`level = A0`	0..1023 на платах AVR, 0..4095 на ESP32.



Arduino և ESP32 տախտակներ

Եվս երեք լեզու կոդ են ստեղծում միկրոկոնտրոլերների համար: Երկուսն էլ օգտագործում են արագ արժեքներ (թվեր. տեքստ՝ միայն Print-ում), իսկ Print-ը և Input-ը աշխատում են սերիական պորտով՝ 115200 բոդ արագությամբ:

Լեզու	Տախտակներ	Ինչով է կառուցվում
Arduino	Arduino Uno, Nano, Mega 2560, Leonardo, Micro, Pro Mini; ESP32, ESP32-S2, -S3, -C3, -C6, -H2	Մեկ սքեթչ (`.ino`)՝ `setup()` և `loop()`-ով. arduino-cli՝ arduino:avr և esp32:esp32 միջուկներով:
AVR C	Arduino-ի վեց AVR տախտակները (ATmega328P, ATmega2560, ATmega32U4)	Bare-metal C՝ առանց Arduino միջուկի: Պինները, թայմերները, PWM-ը, ADC-ն և UART-ը կառավարվում են չիպի ռեգիստրներով: Կառուցում են avr-gcc-ն և avr-libc-ը, բեռնում է avrdude-ը:
PIC32 C	ինը PIC32MX չիպ. MX1xx/2xx (PIC32MX150F128B, 250F128B, 270F256B), MX3xx/4xx (320F128H, 340F512H, 440F256H, 460F512L), MX5xx/6xx/7xx (695F512L, 795F512L)	Bare-metal C Microchip-ի XC32-ի համար. ժամացույցը, պինները, PWM-ը, ADC-ն, UART-ը, SPI-ն և I2C-ն կառավարվում են ռեգիստրներով, առանց Harmony-ի և plib-ի: Կառուցում է xc32-gcc-ն, `.hex`-ը ստեղծում է xc32-bin2hex-ը, իսկ բեռնում է ձեր ընտրած հրամանը (լռելյայն՝ pic32prog):

Տախտակն ընտրվում է լեզվի կողքի **Board** ցանկում (նաև Settings-ում): Կոդի պատուհանի **Build**-ը ծրագիրը կառուցում է դրա համար. **Upload**-ը կառուցում է և ուղարկում ծրագիրը տախտակին՝ Settings-ում նշված սերիական պորտով:

Բլոկ	Կոդ	Ինչ է անում
Pin Mode	`LED_BUILTIN, OUTPUT`	OUTPUT, INPUT կամ INPUT_PULLUP (գետնին միացված կոճակը սեղմելիս կարդացվում է LOW):
Digital Write	`LED_BUILTIN, HIGH`	HIGH՝ ցանկացած ոչ զրոյական արժեքի համար, LOW՝ 0-ի համար:
Digital Read	`pressed = 4`	Պահում է 1 (HIGH) կամ 0 (LOW):
Analog Read	`level = A0`	0..1023 AVR տախտակներում, 0..4095 ESP32-ում: