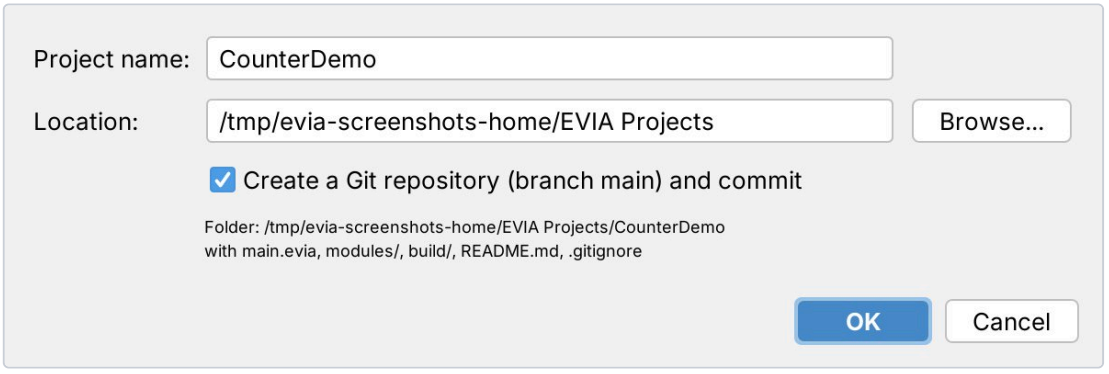
EVIA Paradigma：教程

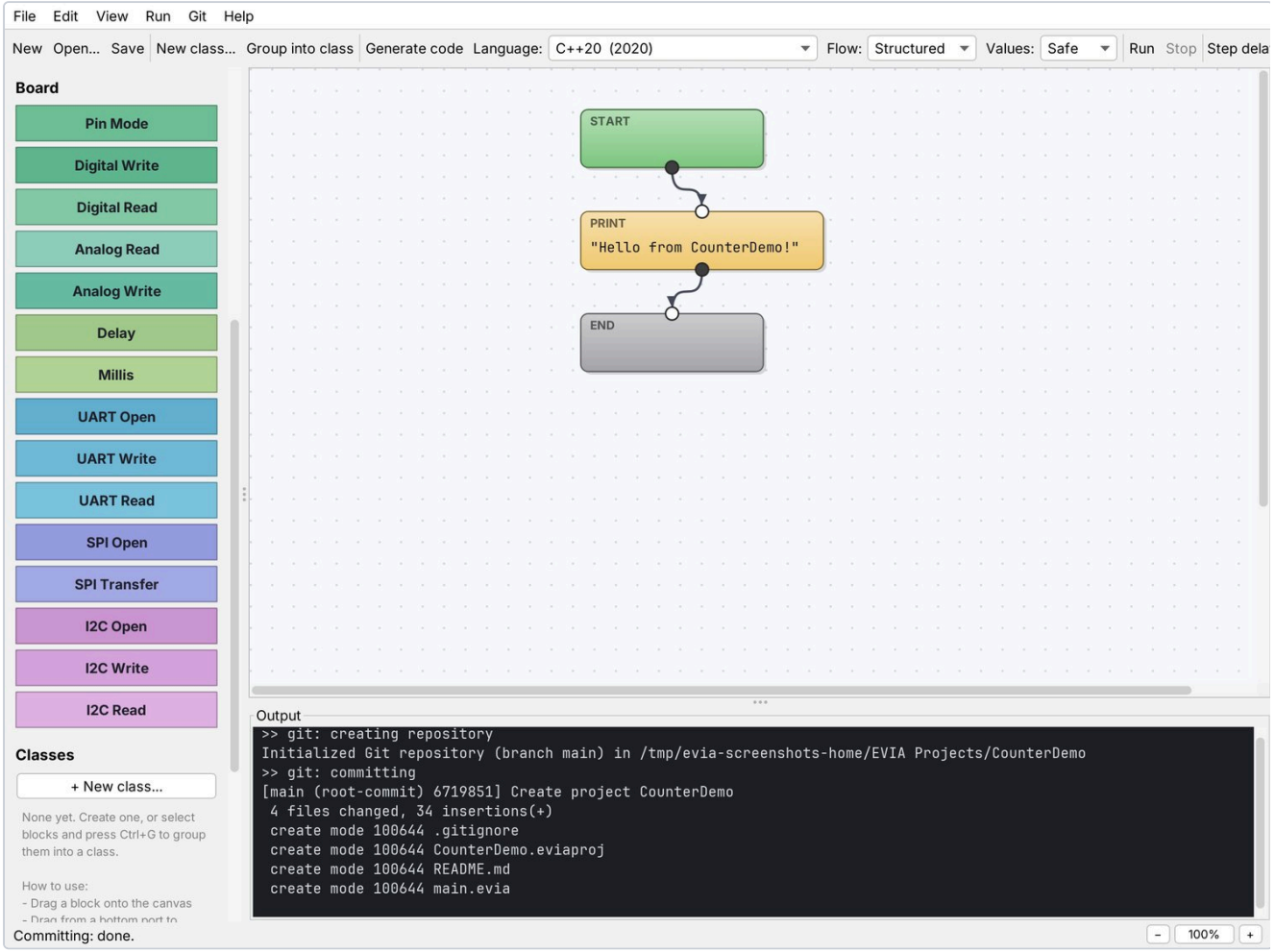
本教程将创建一个包含循环和类的小型项目，运行程序，查看生成的 C 和 C++ 代码，并将项目纳入 Git。最后还会编写 Arduino 或 ESP32 开发板程序。所有截图均来自实际应用。应用界面为英文，因此菜单和按钮名称保留屏幕上的英文写法。

1. 创建项目

选择 **File → New Project...**（Ctrl+Shift+N），输入名称和位置。如果已安装 git，请保留 *Create a Git repository* 选项。



项目会以初始程序 *Start → Print → End* 打开。输出面板会显示 git 已创建仓库和首次提交。

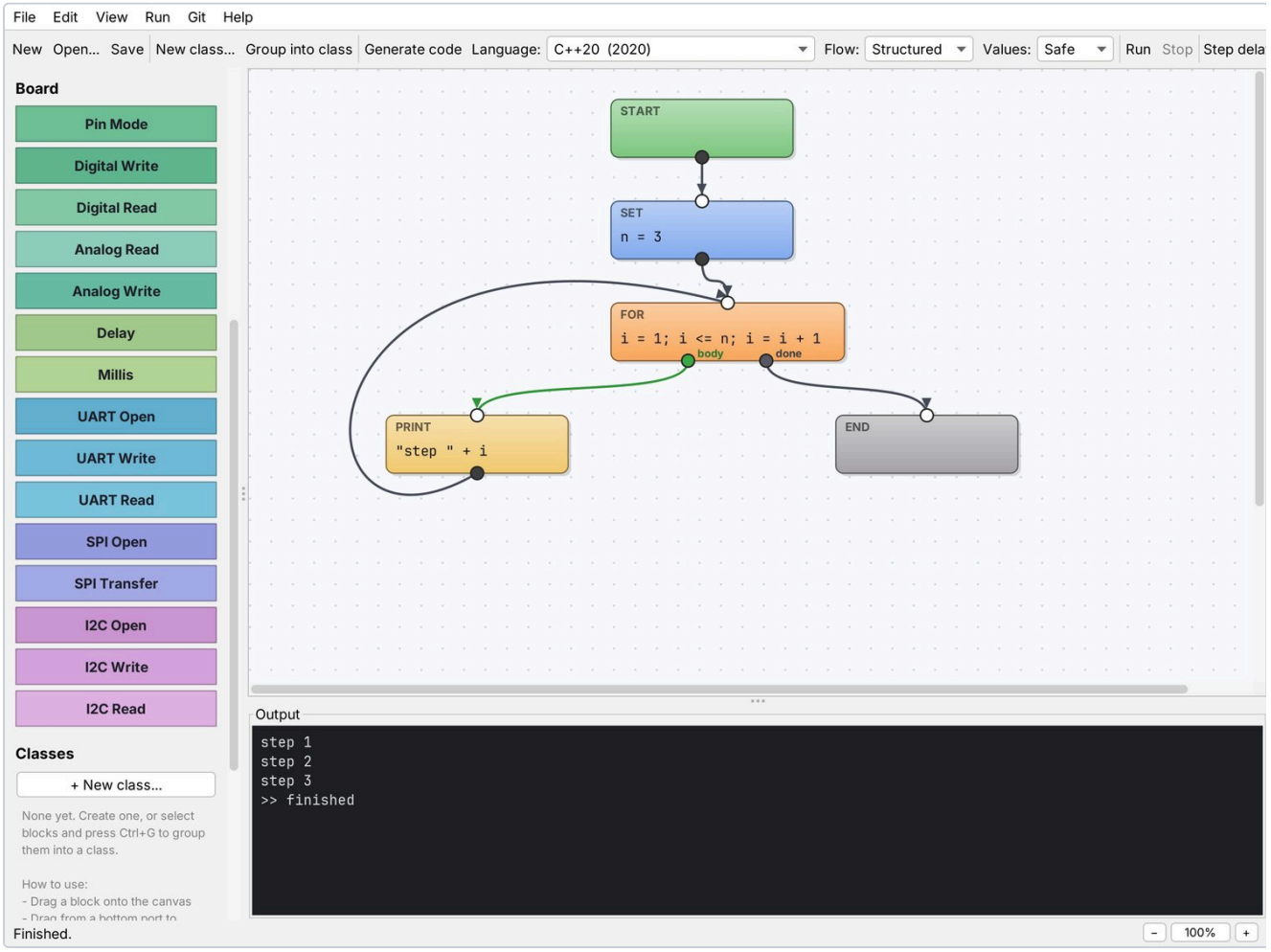


2. 搭建循环

将积木从面板拖到画布并连接：从一个积木底部端口（深色圆点）拖到下一个积木顶部端口（白色圆点）。双击积木可编辑其代码。

- **Set**：`n = 3`
- **For**：`i = 1; i ⩽ n; i = i + 1`。将 `body` 端口连接到 **Print** `"step " + i`；把循环体末尾**连回 For 积木**，以开始下一轮。`done` 端口连接到 **End**。

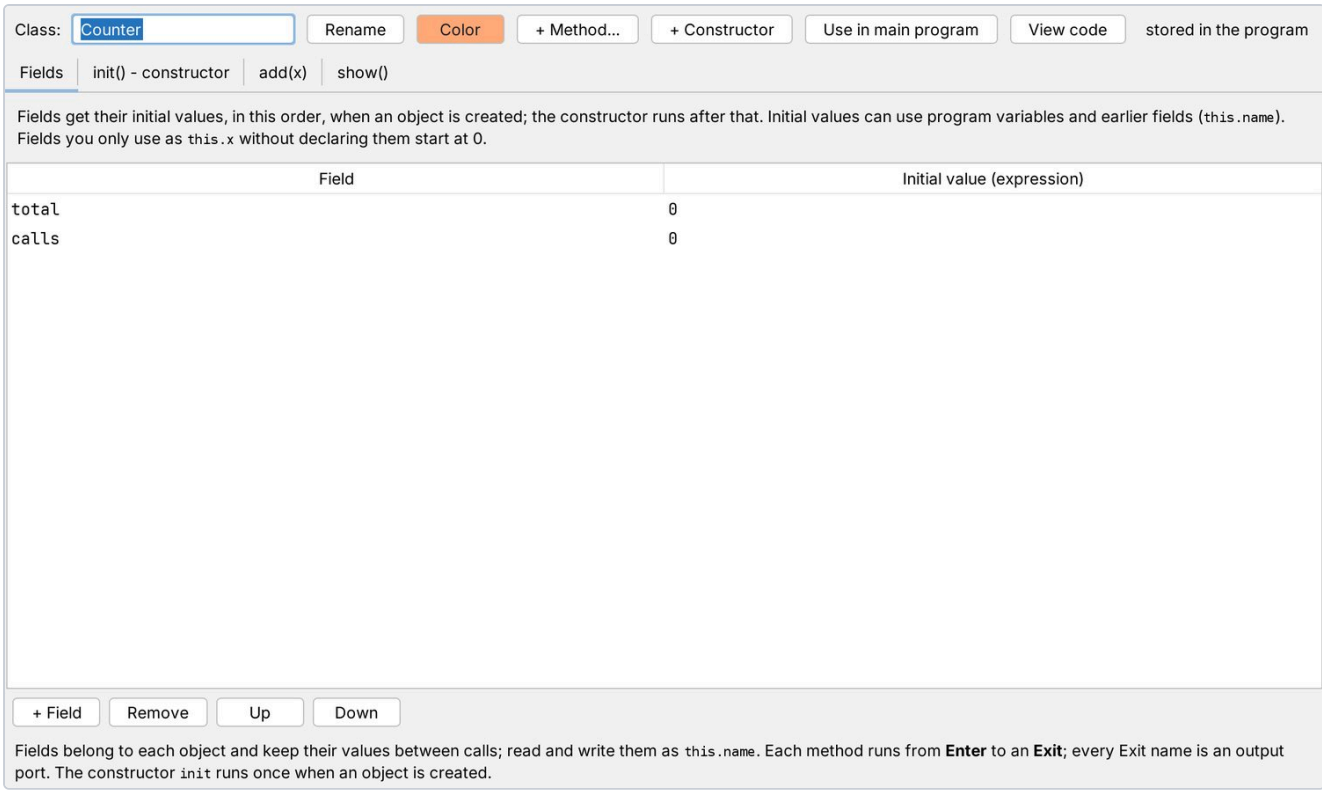
按 **Run**（F5）运行。设置 *Step delay* 可以观察当前执行积木的移动。在图表上滚动鼠标滚轮可缩放。



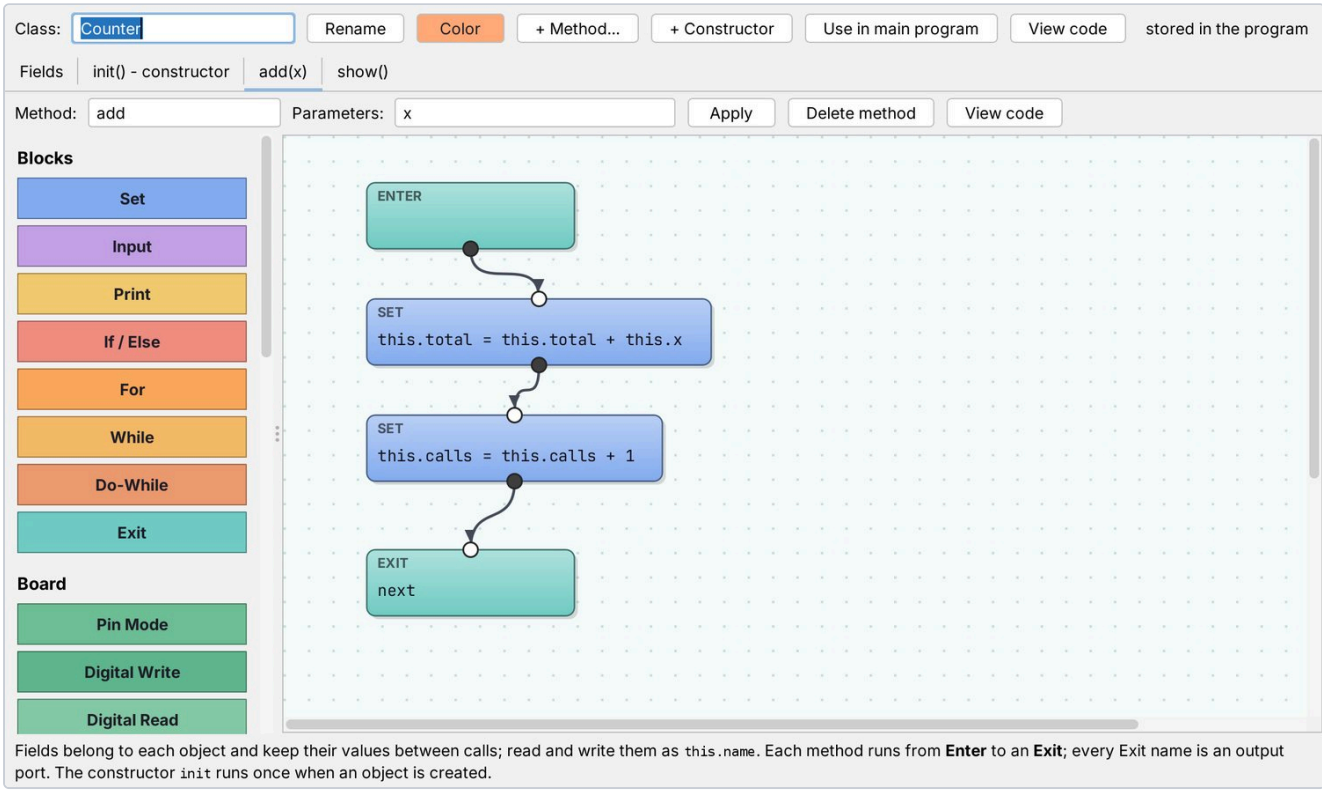
3. 创建类

选择 **Edit → New Class...**（或面板中的 *+ New class...*），命名为 `Counter`。类窗口有一个 **Fields** 标签页，以及每个方法各自的标签页。

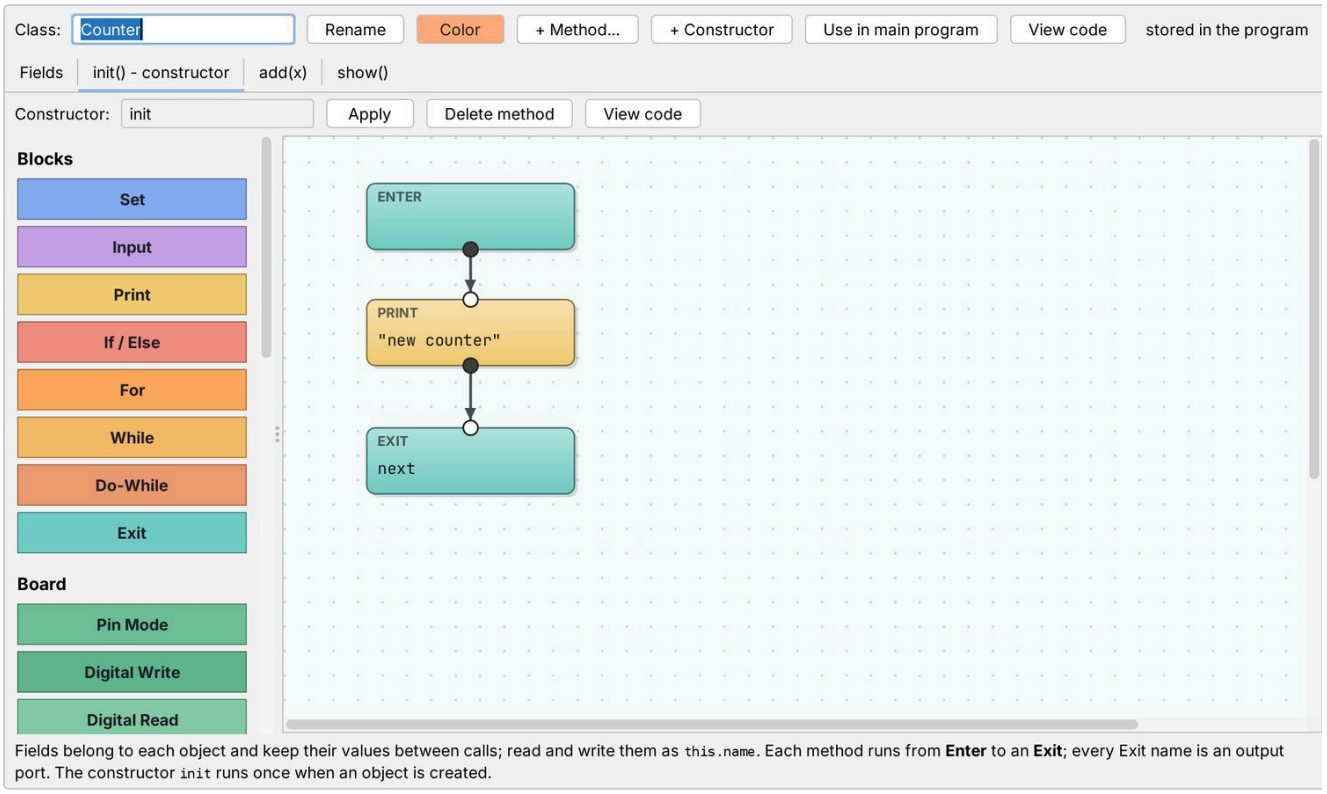
在 **Fields** 中添加 `total` 和 `calls`，初始值都设为 `0`。每个对象都有自己的字段副本。



将第一个方法改名为带参数 `x` 的 `add`（按 **Apply**），然后从 **Enter** 到 **Exit** 搭建方法体。在方法中，字段和参数写作 `this.name`：

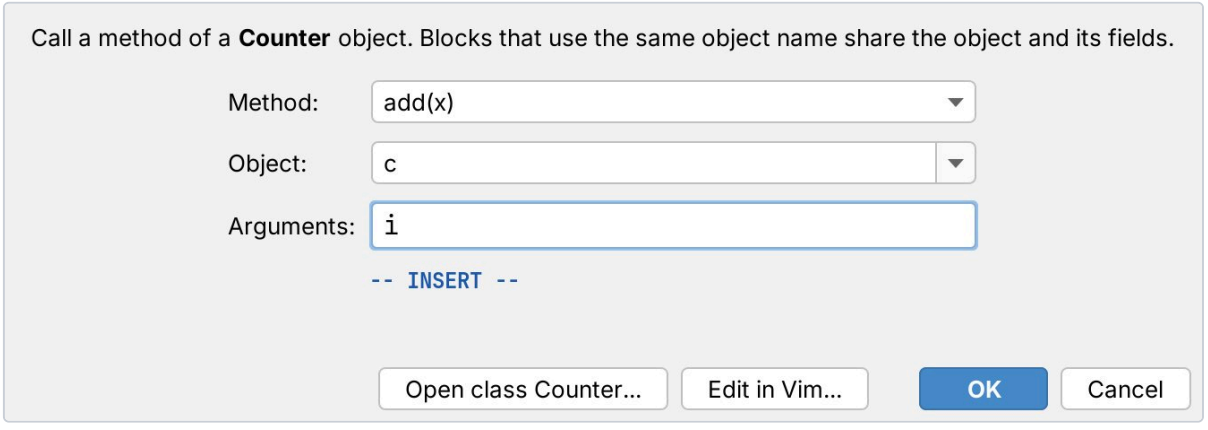


+ Constructor 会添加 `init`，它在创建对象时运行一次（字段初始化之后）。这里让它输出 `"new counter"`。**+ Method...** 会添加 `show`，输出 `"total = " + this.total + " after " + this.calls + " calls"`。

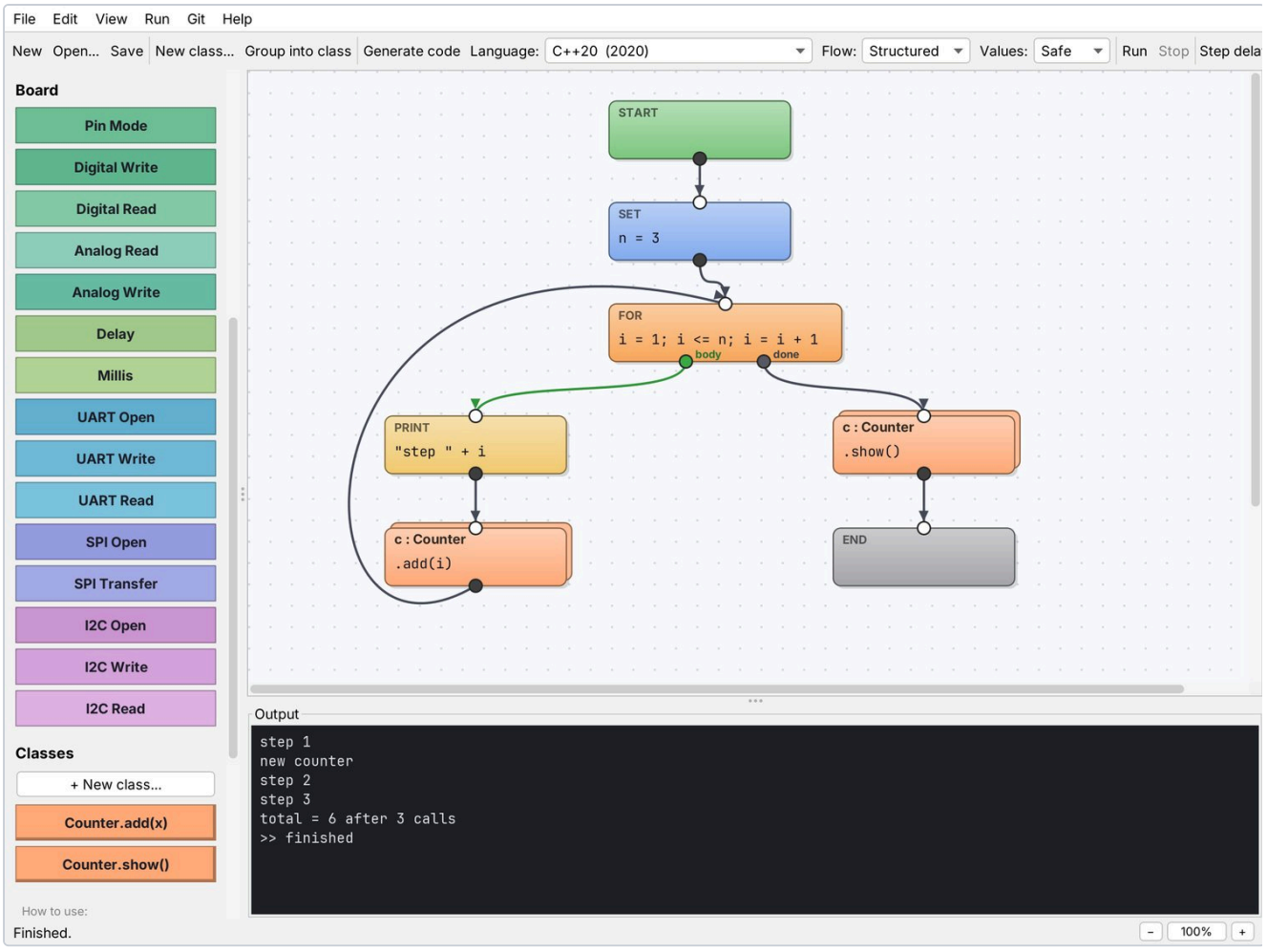


4. 使用类

面板现在会列出 `Counter.add(x)` 和 `Counter.show()`。将 `Counter.add(x)` 拖到循环体中的 **Print** 后面，将 `Counter.show()` 放在 `done` 和 **End** 之间。双击调用积木可选择方法、**对象**和参数。两个积木都使用对象名 `c`，使它们共享同一个对象。

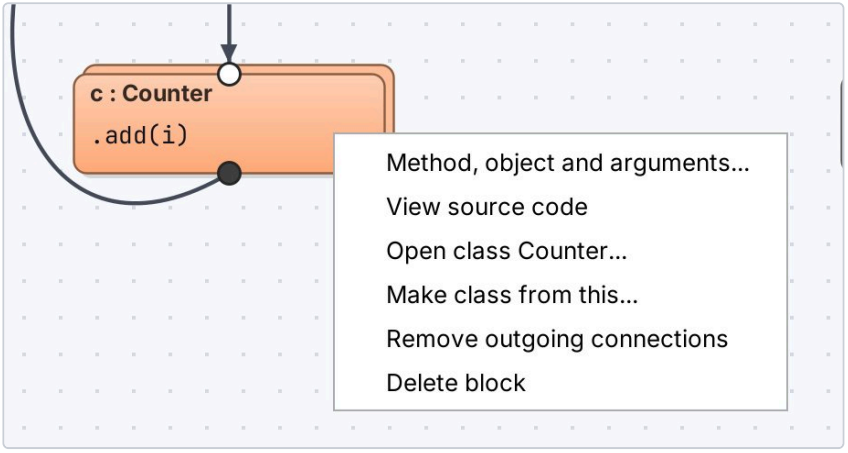


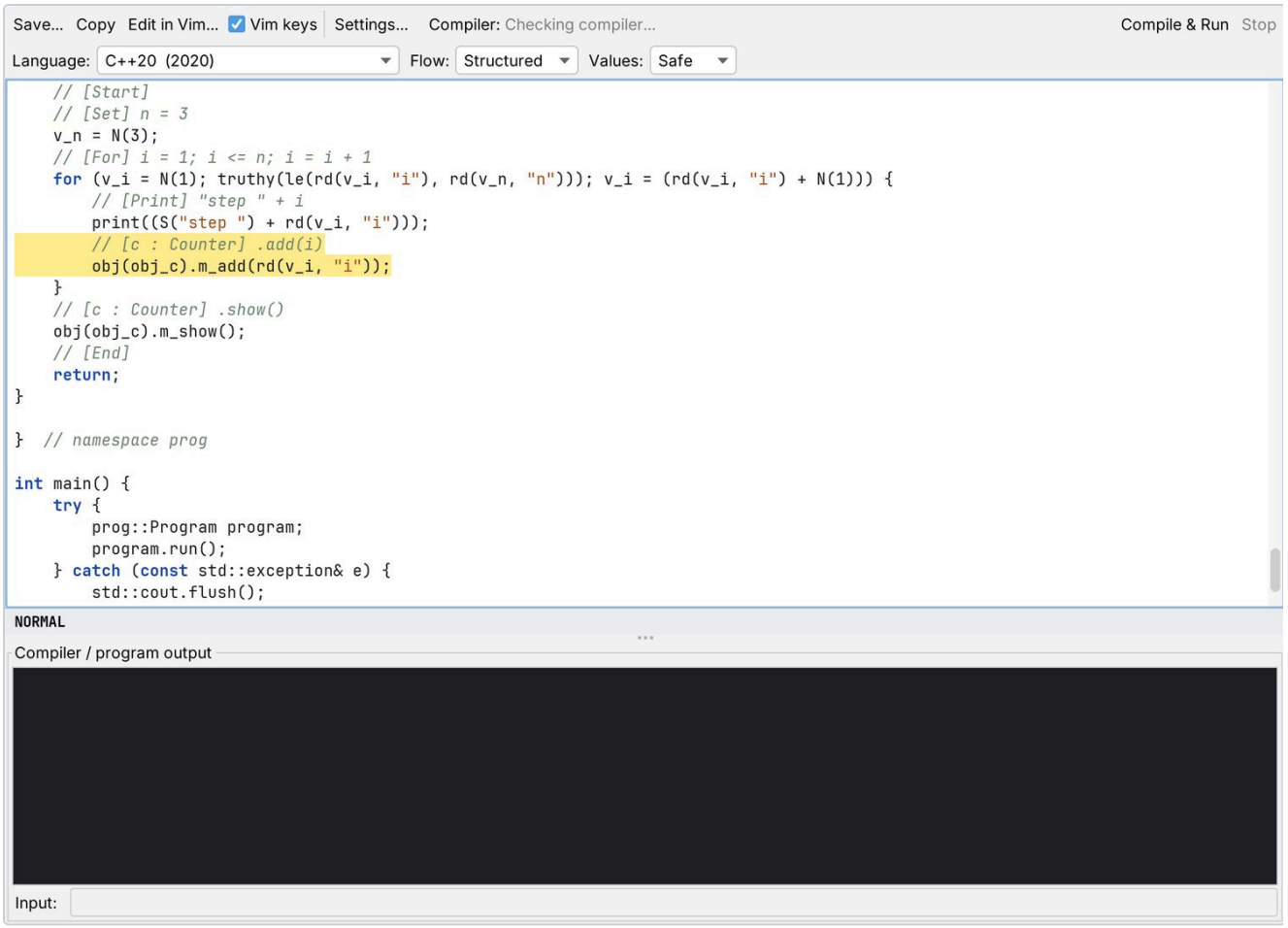
运行程序。首次使用时会创建 `c`（此时输出 `new counter`），`add` 执行三次，`show` 输出合计结果。



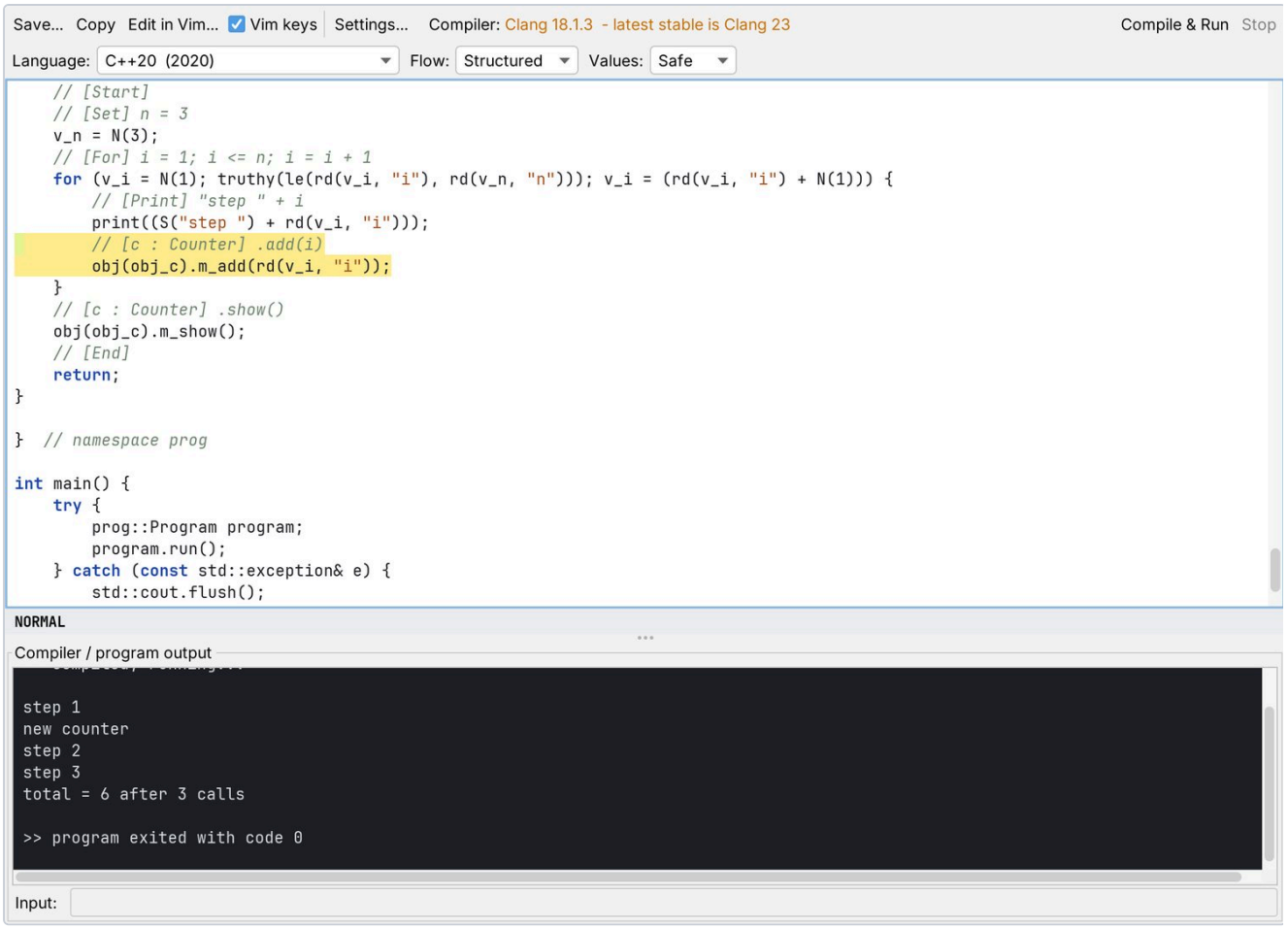
5. 查看任意积木的代码

右键单击积木并选择 **View source code**。生成的代码会打开，并突出显示该积木对应的代码行（此处默认使用 C++20）。



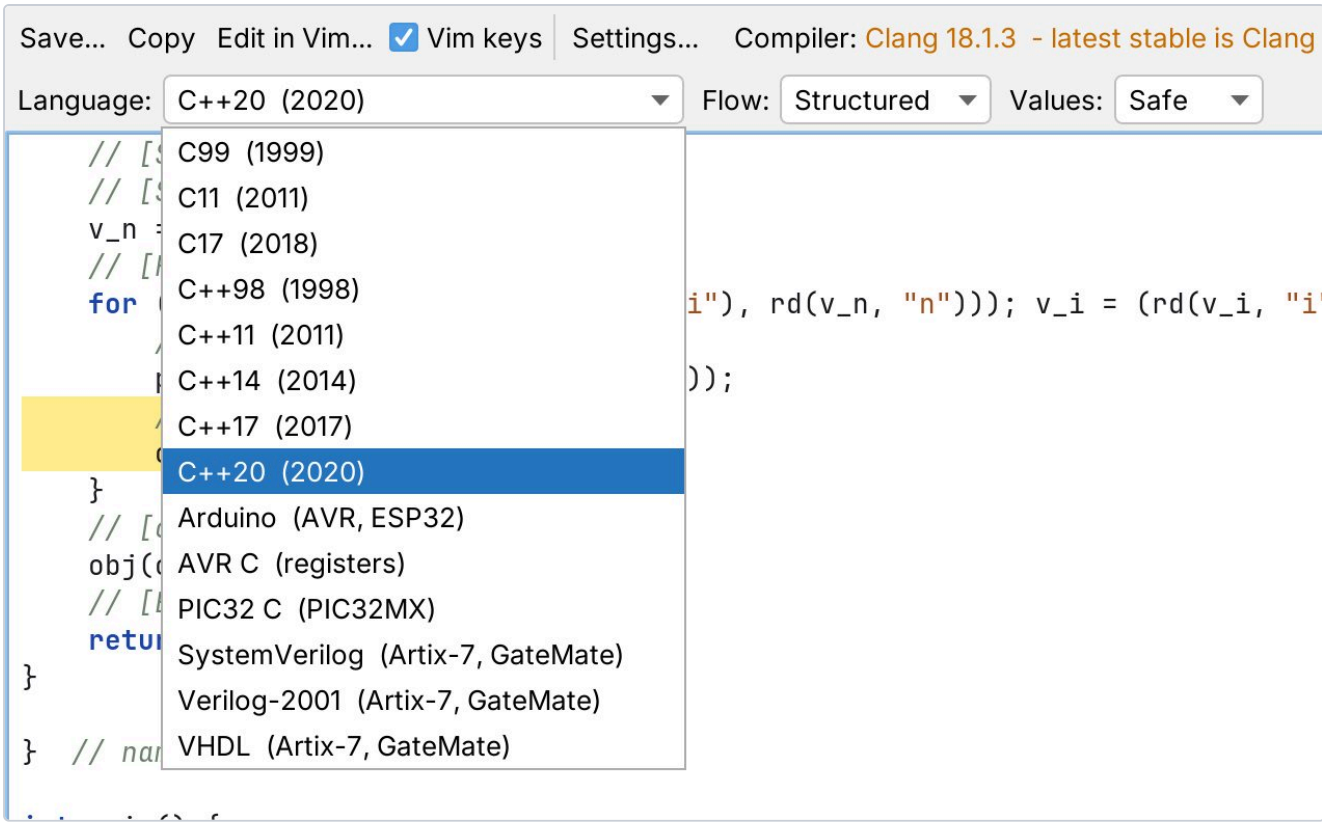


Compile & Run 使用 Clang 或 GCC 编译并运行程序。编译后程序的输出与解释器一致。Windows 上应用会为你安装 llvm-mingw（Clang）或 WinLibs（GCC），无需 Visual Studio。

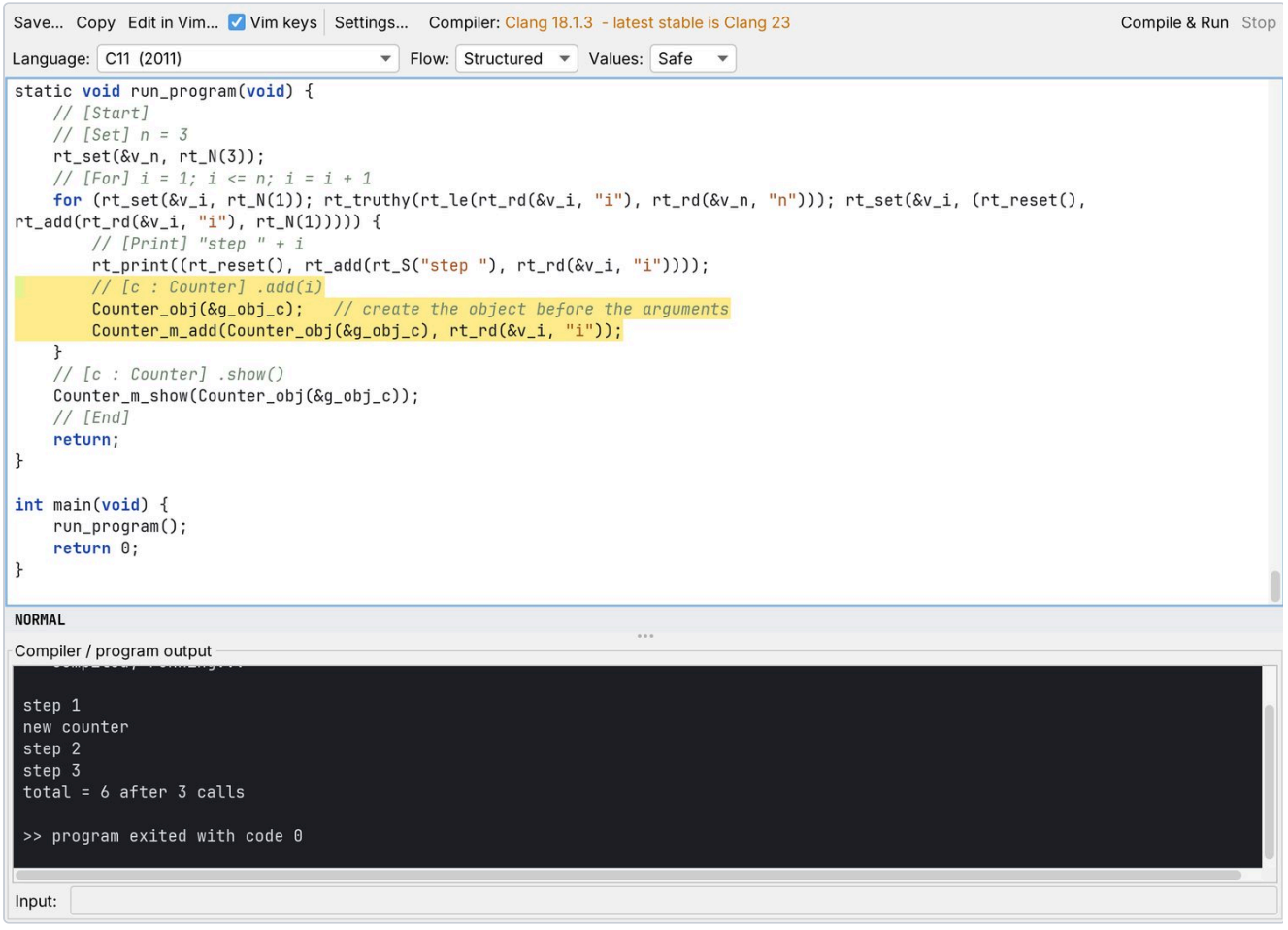


6. 选择 C 或其他 C++ 标准

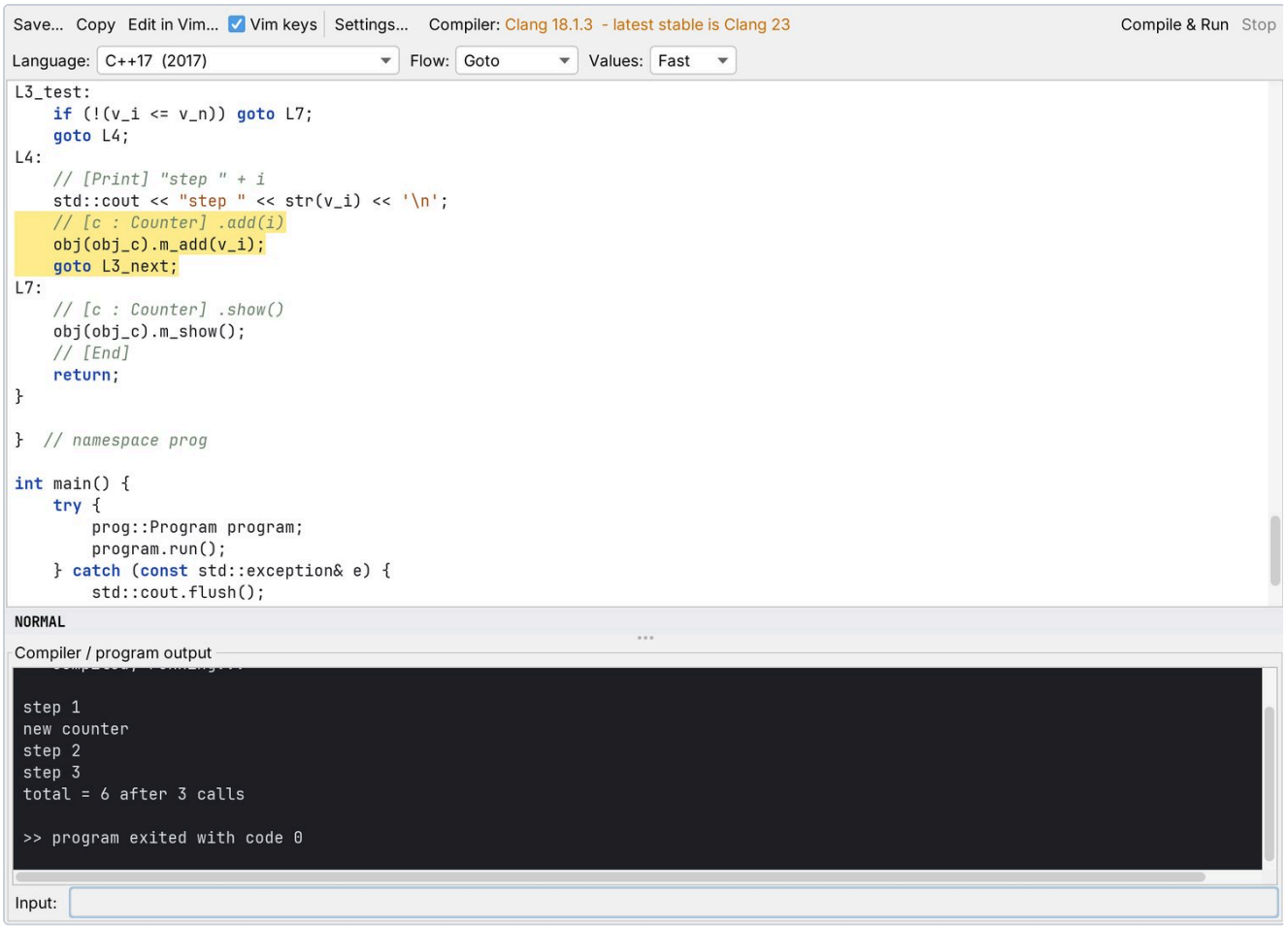
在代码窗口或主工具栏的下拉列表中选择 **Language**，代码会立即重新生成，原先选中的积木仍会高亮。**Flow** 和 **Values** 用于选择代码风格：**Structured** 使用 if/else、循环和 switch；**Goto** 将每个积木变成标签、将每条连线变成跳转。**Safe** 保留文本类型和解释器的检查；**Fast** 使用不做运行时检查的普通 `double`。



同一程序也可生成 **C11**，用 `clang -std=c11` 编译运行，输出相同。

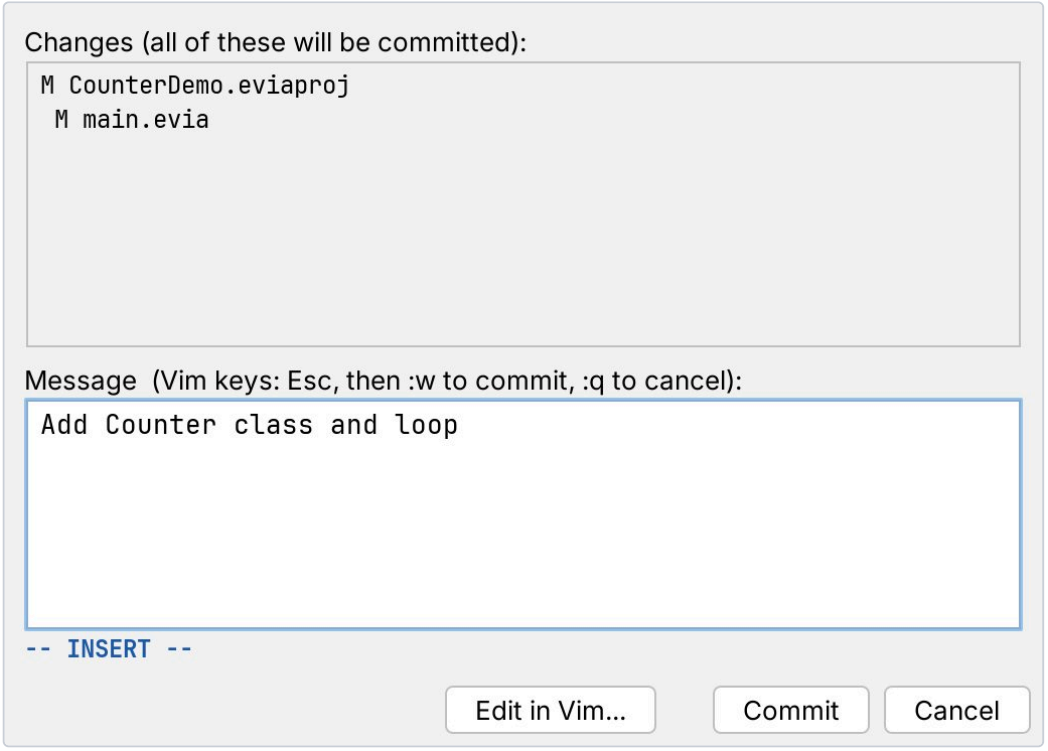


C++17 配合 **Goto** 流程和 **Fast** 值：

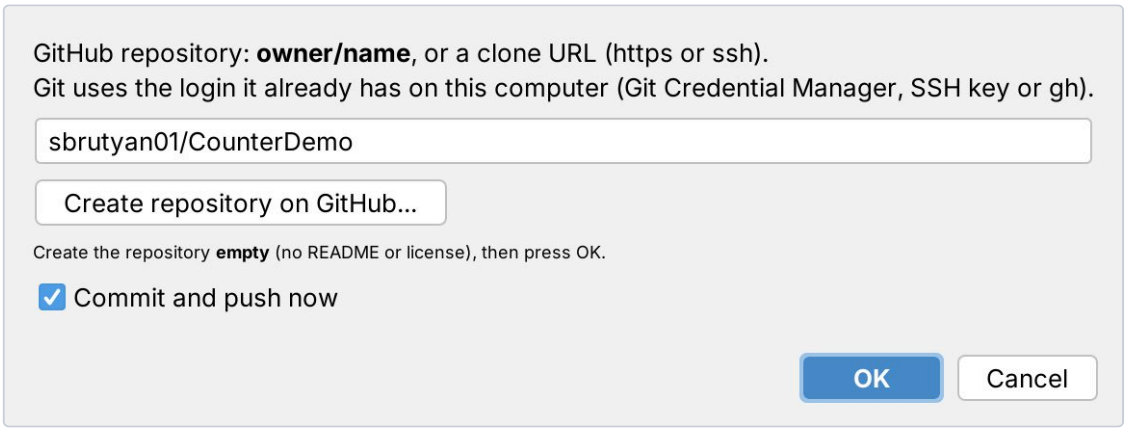


7. 提交更改并发布到 GitHub

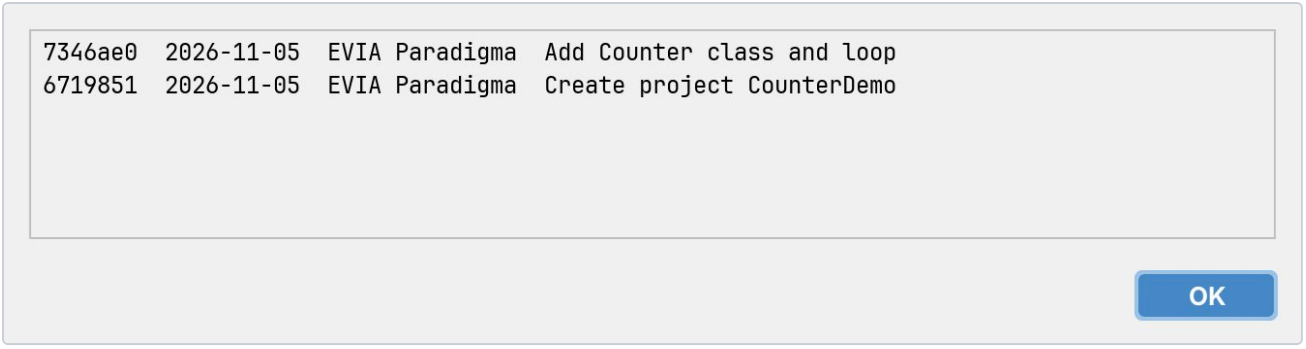
Git → Commit...（Ctrl+K）会保存并提交所有更改。提交消息框支持 Vim 按键（按 `Esc`，再输入 `:w` 提交）。



选择 **Git → Connect to GitHub…** 并输入 `owner/repo`。*Create repository on GitHub…* 会在 github.com 打开已填好名称的页面；创建一个**空**仓库，按 **OK** 即可推送项目。Git 使用电脑上已配置的凭据（Git Credential Manager、SSH 密钥或 `gh auth login`）。

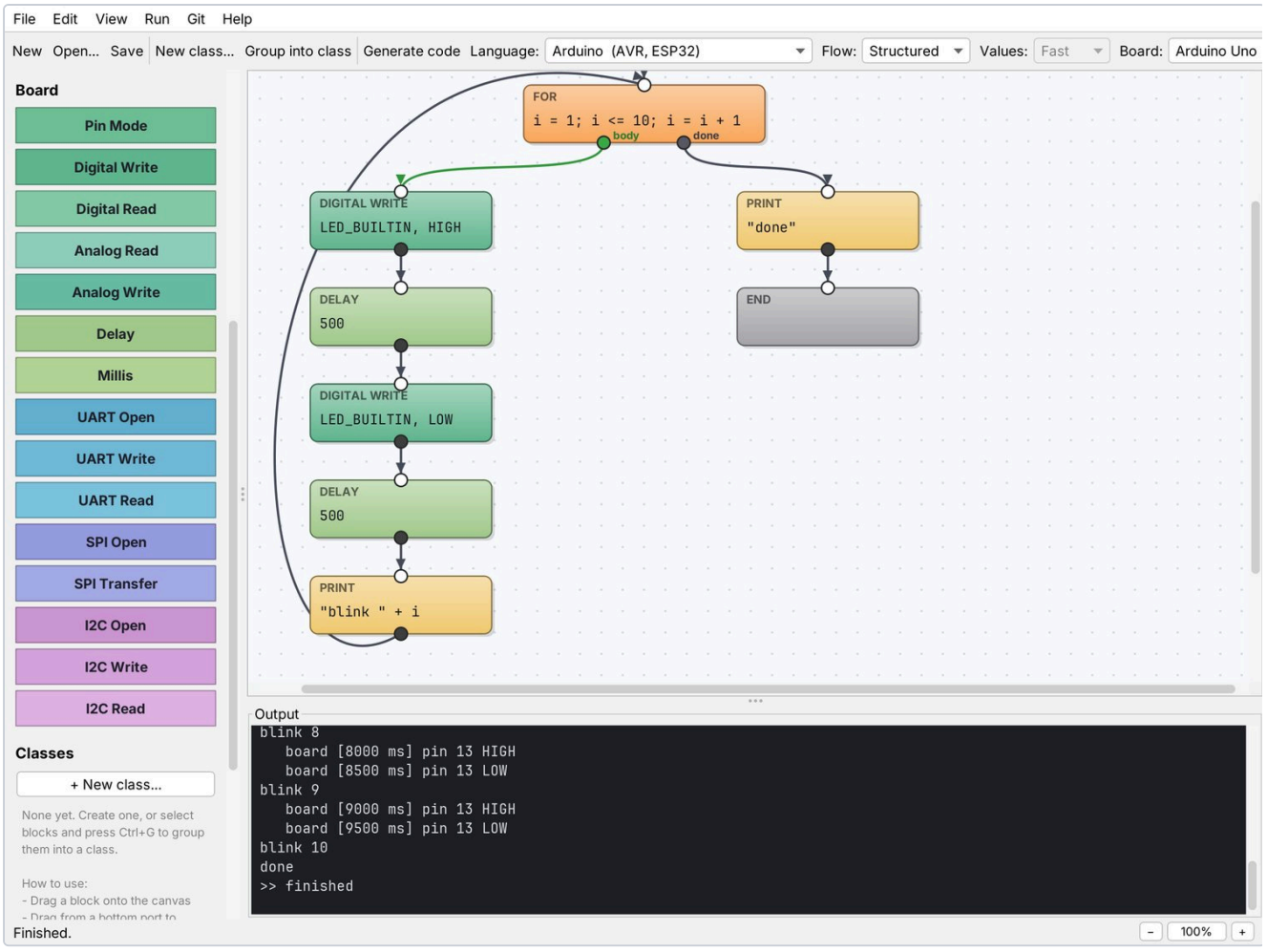


Git → History 会列出提交记录：

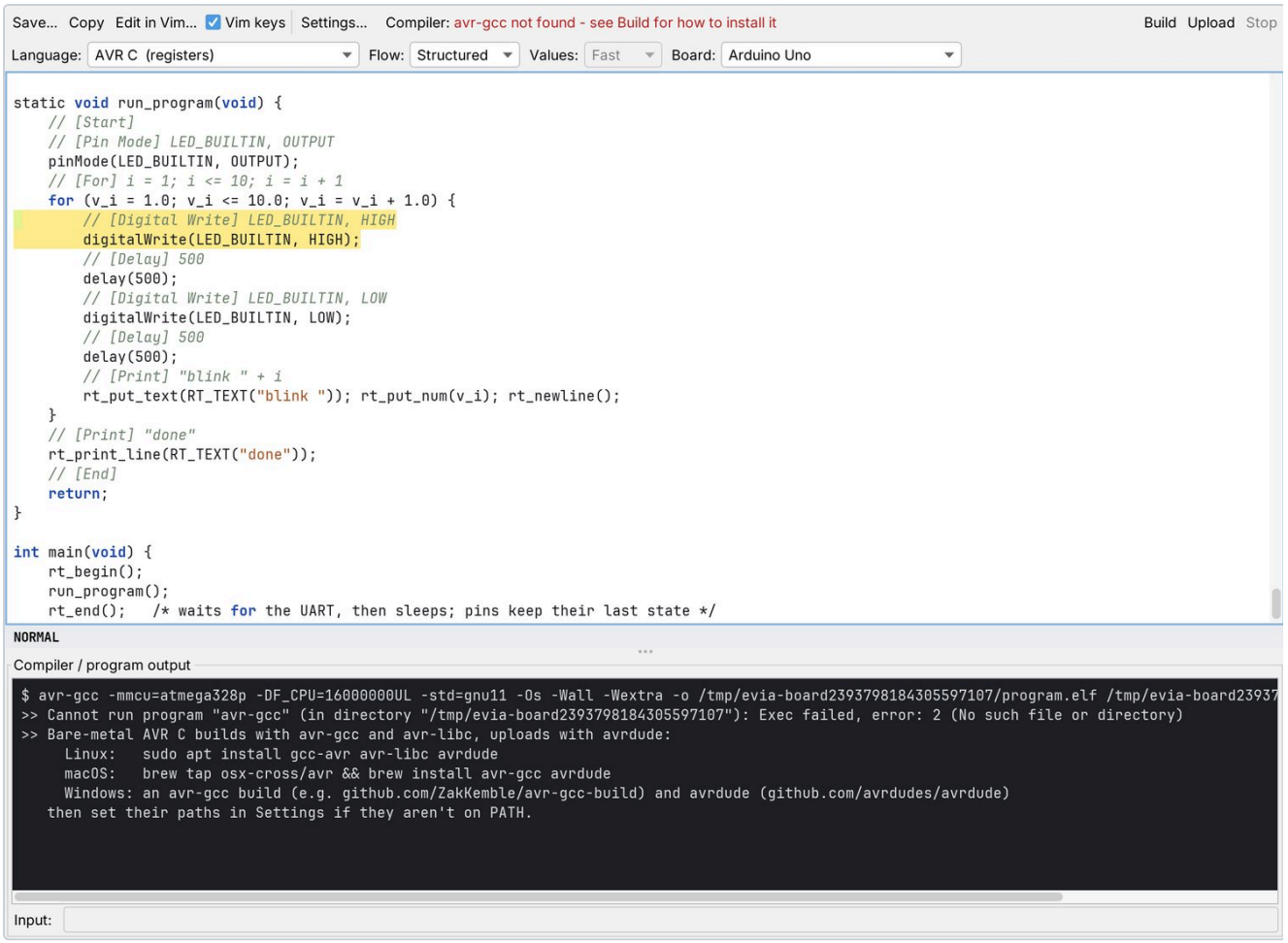


8. 为 Arduino 或 ESP32 开发板编程

选择 **File → Open Example → Arduino and ESP32 Boards → Blink the built-in LED**。面板 *Board* 分组中的绿色 **Board** 积木用于配置引脚、读取按钮和传感器及等待。按 **Run**（F5）可模拟 Arduino Uno；输出会显示每次引脚变化及其时间。



选择 **Language: Arduino** 和一个 **Board**，然后按 **Generate code**（F7）：生成的草图适用于 12 种开发板。**Build** 使用 arduino-cli 编译，**Upload** 将程序上传到开发板；Print 和 Input 通过 115200 波特率的 Serial Monitor 工作。**AVR C** 不使用 Arduino 核心，而是直接操作芯片寄存器来生成同一程序，并用 avr-gcc 编译。



9. 与其他芯片通信：UART、SPI、I2C 和 PIC32

File → Open Example → Arduino and ESP32 Boards 下的另外三个示例使用总线积木：*UART: send a byte, read it back*、*SPI: transfer a byte* 和 *I2C: write and read a device register*。使用前先打开总线。**Run** 会模拟这些总线：UART 写入的数据可由 UART 读取，SPI 输入连接到输出，I2C 地址 80 处有一个包含 256 个寄存器的设备（80 即 0x50；数值为十进制）。

积木	代码	功能
UART Open	`9600`	以指定波特率启动串口
UART Write	`65`	发送一个字节（0..255）
UART Read	`c`	读取下一个字节；若无数据则为 -1
SPI Open	`1000000`	以指定频率（Hz）启动 SPI 主机（模式 0，最高位先传）
SPI Transfer	`reply = 66`	发送一个字节，并保存同时收到的字节
I2C Open	`100000`	以指定频率（Hz）启动 I2C 主机
I2C Write	`80, 0, 42`	地址（7 位）、寄存器、数值
I2C Read	`v = 80, 0`	保存寄存器值；无设备响应时为 -1

为 Microchip PIC32MX 芯片选择 **Language: PIC32 C**，再在 **Board** 列表中选择芯片（MX1xx/2xx、MX3xx/4xx、MX5xx/6xx/7xx 系列共九种），然后按 **Generate code**（F7）。生成的是供 Microchip 免费 XC32 编译器使用的裸机 C 代码：时钟、引脚、PWM、ADC、UART、SPI 和 I2C 均通过寄存器设置。

Build 会运行 `xc32-gcc` 和 `xc32-bin2hex`；**Upload** 会运行 **Settings** 中配置的命令（默认 `pic32prog {hex}`）。所选芯片系列的引脚列在生成文件开头。

注意： PIC32 代码已根据 Microchip 的实际设备头文件检查，但尚未用 XC32 编译，也未在芯片上运行。请将首次在开发板上的构建视为测试。

10. 为 FPGA 构建硬件

选择 **Language: SystemVerilog**、**Verilog-2001** 或 **VHDL**，再从 **Board** 列表选择 Digilent Nexys A7-100T、Arty A7-35 或 A7-100、Basys 3（Artix-7），或 Olimex GateMateA1-EVB、Cologne Chip GateMate A1 evaluation board。打开 **File → Open Example → FPGA Boards → Blink**，然后按 **Generate code**（F7）。

每个积木会成为状态机的一个状态，并占用一个时钟周期。设计构建时引脚已固定，因此请使用名称：`LED0`、`SW3`、`BTNC`、`JA1`。引脚约束写在文件首条注释中；命令行使用 `-o` 时还会另存为 `.xdc`（Vivado）或 `.ccf`（GateMate）。

- **Build** 和 **Upload** 对 Artix-7 使用 Vivado 或开源工具链（在 **Settings** 中选择）；GateMate 使用 yosys、nextpnr-himbaechel、gmpack 和 openFPGALoader。
- 数值是 32 位整数，因此 FPGA 上 `5 / 2` 的结果是 2，而 **Run** 中是 2.5。
- 目前尚不支持类、Input、Analog Read、SPI、I2C 以及变量中的文本；错误消息会指出不支持的积木。

注意： 输出已使用开源工具进行仿真和综合，但尚未通过 Vivado，也未下载到实际开发板。

11. 用 MQTT 把数字发出去

打开 **File → Open Example → Arduino and ESP32 Boards → MQTT sensor**。这个程序向代理服务器发布温度，再给自己 发一条命令并读回来。

MQTT 的五个积木在 *Board* 组里：*MQTT Connect*、*MQTT Publish*、*MQTT Subscribe*、*MQTT Has Message* 和 *MQTT Read*。网络名称、密码、代理服务器和主题都是固定文本，因为它们在构建时写进程序里。

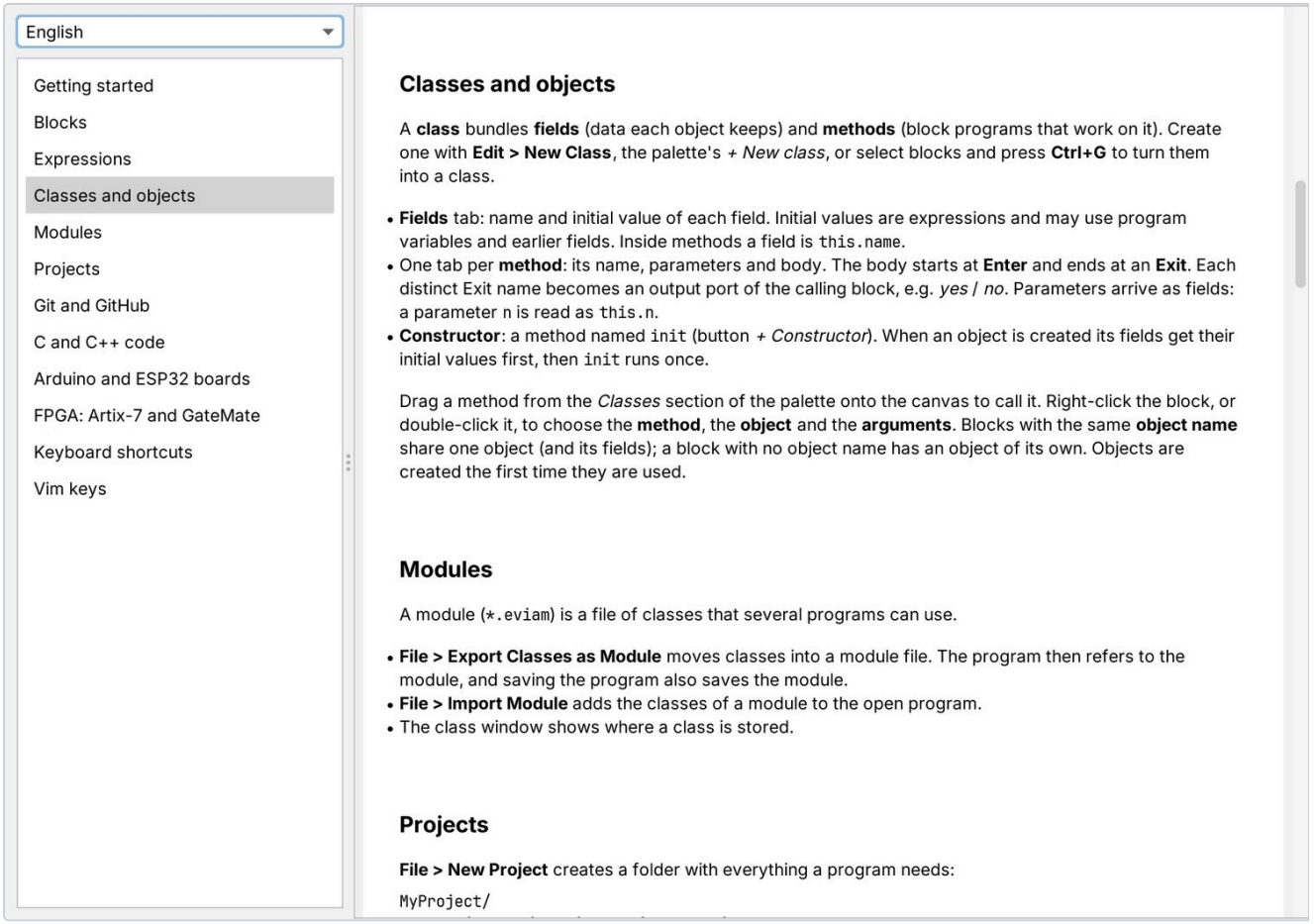
- **Run** 由你电脑上的代理服务器作答，所以没有硬件也能试。
- **PIC32 C** 通过 UART2 驱动带 AT 固件的 ESP8266 或 ESP32 模块，Print 仍用 UART1。
- **ESP32** 板上的 **Arduino** 使用板子自己的 WiFi。
- AVR 板会拒绝这些积木：它只有一个串口，而 Print 正在用。

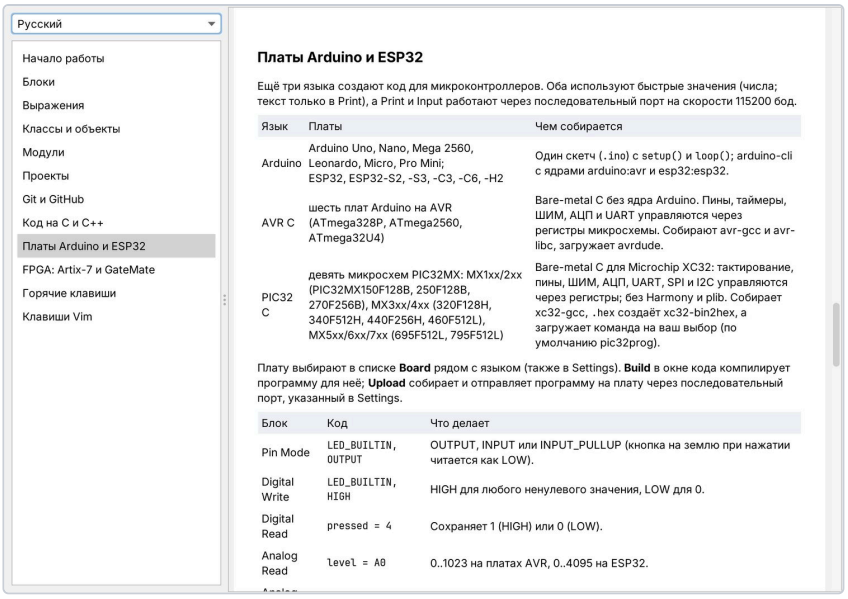
消息是一个数字。没有消息时 *MQTT Read* 返回 -1，所以如果真的可能收到 -1，先用 *MQTT Has Message* 判断。

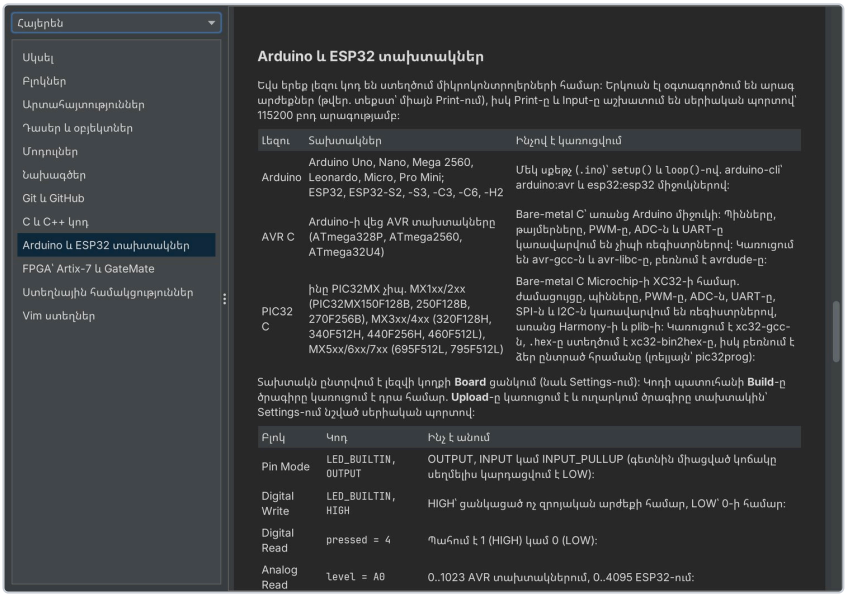
请注意： MQTT 报文在 CI 中对着真实的代理服务器检查过，但没有 WiFi 模块、ESP32 或 PIC32 芯片参与。 MX695/795 芯片的以太网口 MQTT 还不支持。

12. 后续探索

- 在程序间共享类：选择 **File → Export Classes as Module...**，再在另一个程序中选择 **File → Import Module...**。
- **File → Open Example** 提供现成程序，包括 `classes.evia`（字段、构造函数、方法和模块）。
- **Help → Tutorial (PDF)** 可打开英语、俄语、亚美尼亚语、德语、简体中文、繁体中文或日语教程。
- **Help → Guide**（F1）介绍所有积木、表达式、类、项目、Git、快捷键和 Vim 按键，提供英文、俄文和亚美尼亚文版本。通过 **View → Theme** 切换浅色和深色主题。




Русский
Начало работы
Блоки
Выражения
Классы и объекты
Модули
Проекты
Git и GitHub
Код на C и C++
Платы Arduino и ESP32
FPGA: Artix-7 и GateMate
Горячие клавиши
Клавиши Vim
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.


Հայերեն
Սկսել
Բլոկներ
Արտահայտություններ
Դասեր և օբյեկտներ
Մոդուլներ
Նախագծեր
Git և GitHub
C և C++ կոդ
Arduino և ESP32 տախտակներ
FPGA՝ Artix-7 և GateMate
Ստեղնային համակցություններ
Vim ստեղներ
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: