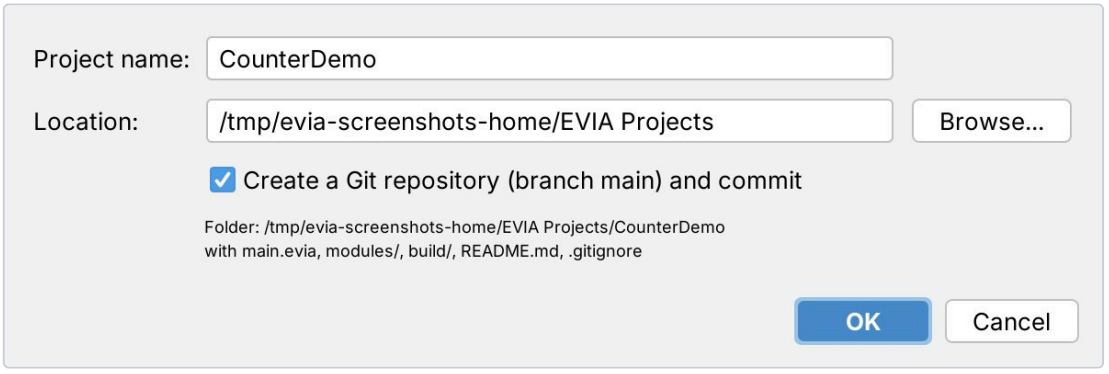
EVIA Paradigma：教學

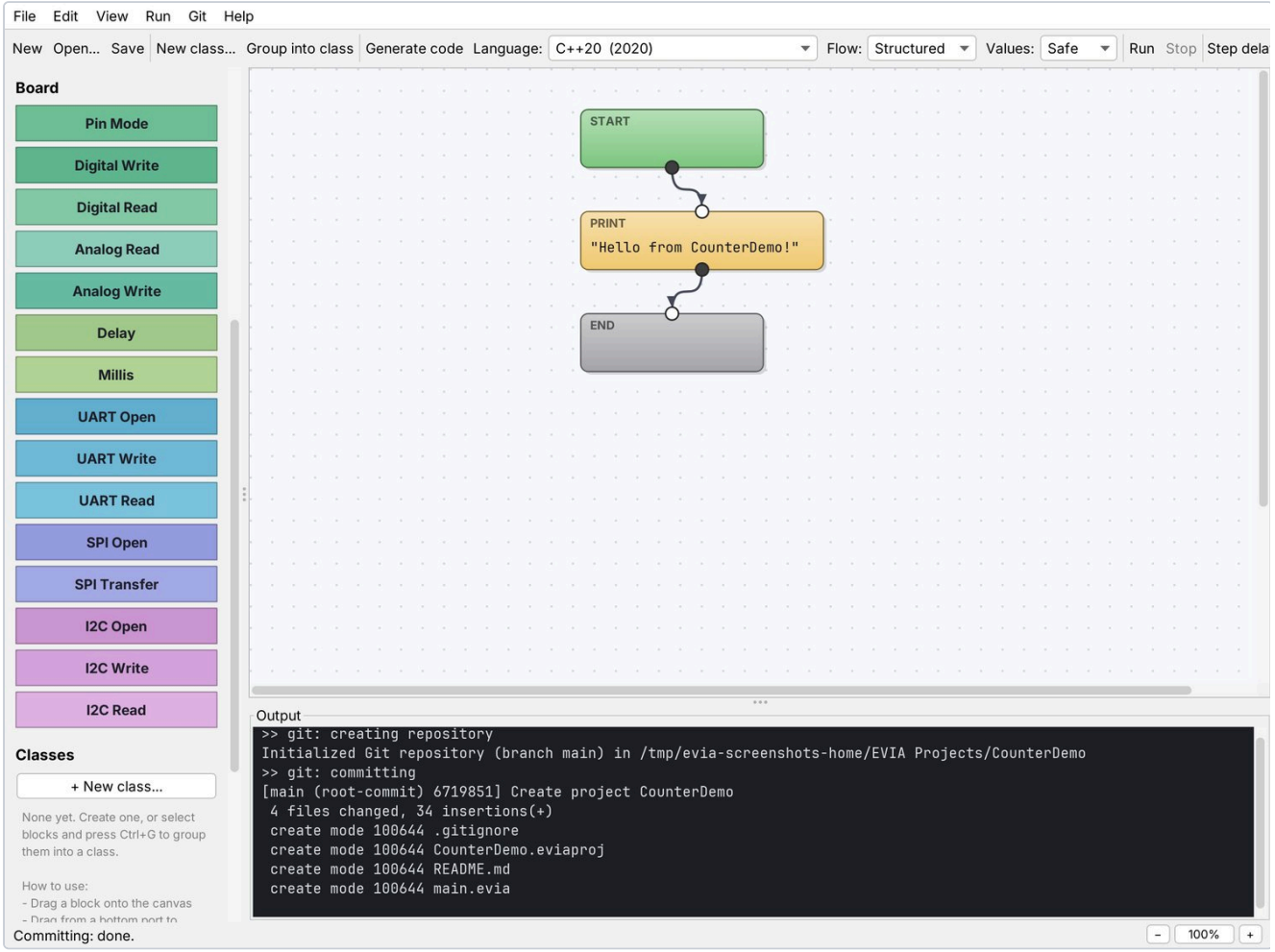
本教學將建立一個包含迴圈與類別的小型專案，執行程式、檢視產生的 C 與 C++ 程式碼，並將專案納入 Git。最後也會編寫 Arduino 或 ESP32 開發板程式。所有螢幕擷圖都來自實際應用程式。介面為英文，因此選單與按鈕名稱保留畫面上的英文寫法。

1. 建立專案

選擇 **File → New Project...**（Ctrl+Shift+N），輸入名稱與位置。若已安裝 git，請保留 *Create a Git repository* 選項。



專案會以初始程式 *Start → Print → End* 開啟。輸出面板會顯示 git 已建立儲存庫與第一筆 commit。

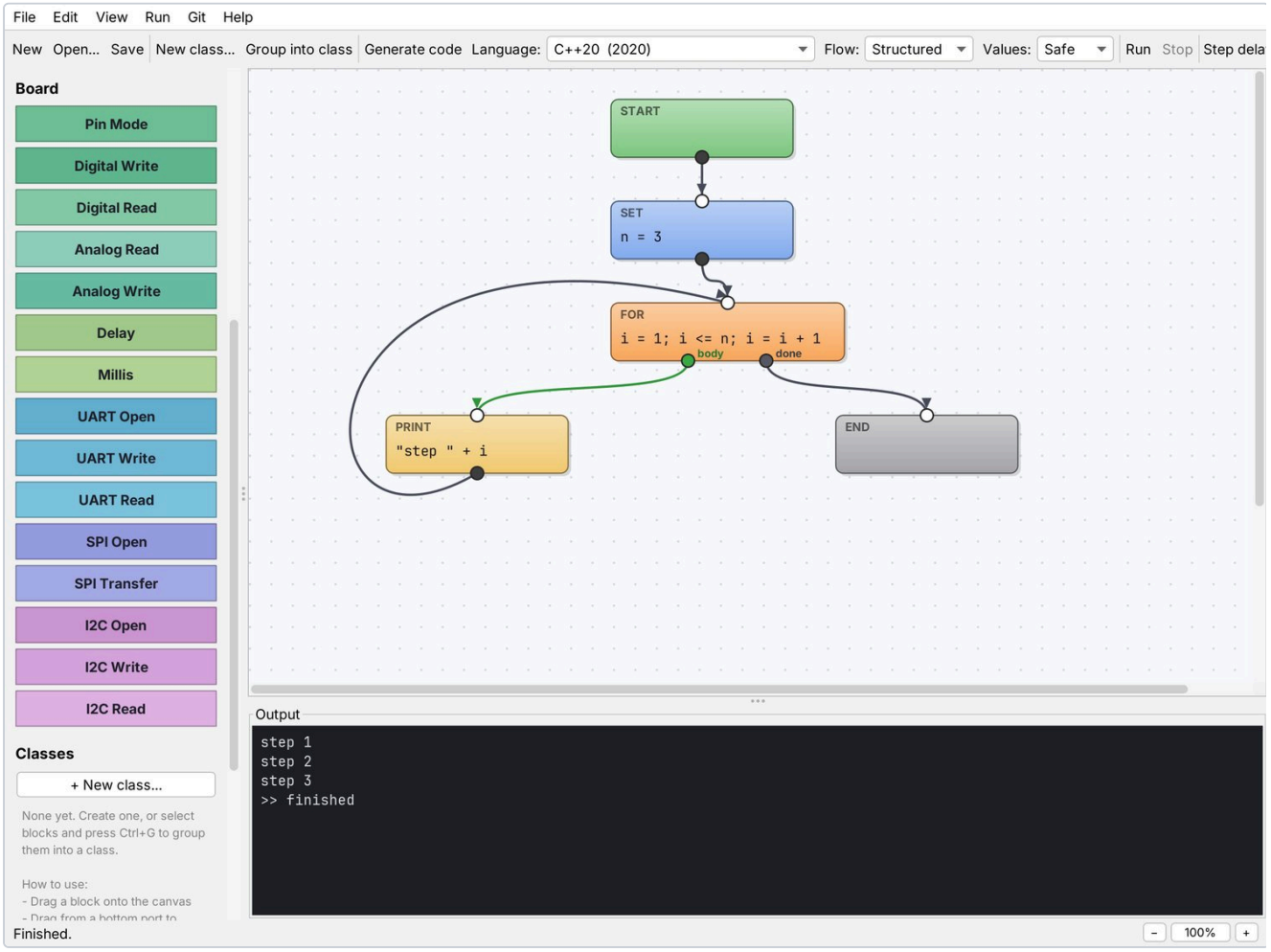


2. 建立迴圈

將積木從面板拖曳到畫布並連接：從積木底部連接埠（深色圓點）拖到下一個積木頂部連接埠（白色圓點）。雙擊積木可編輯其程式碼。

- **Set**：`n = 3`
- **For**：`i = 1; i ⩽ n; i = i + 1`。將 `body` 連接埠接到 **Print** `"step " + i`；把迴圈本體結尾**接回 For 積木**，即可開始下一輪。`done` 連接埠接到 **End**。

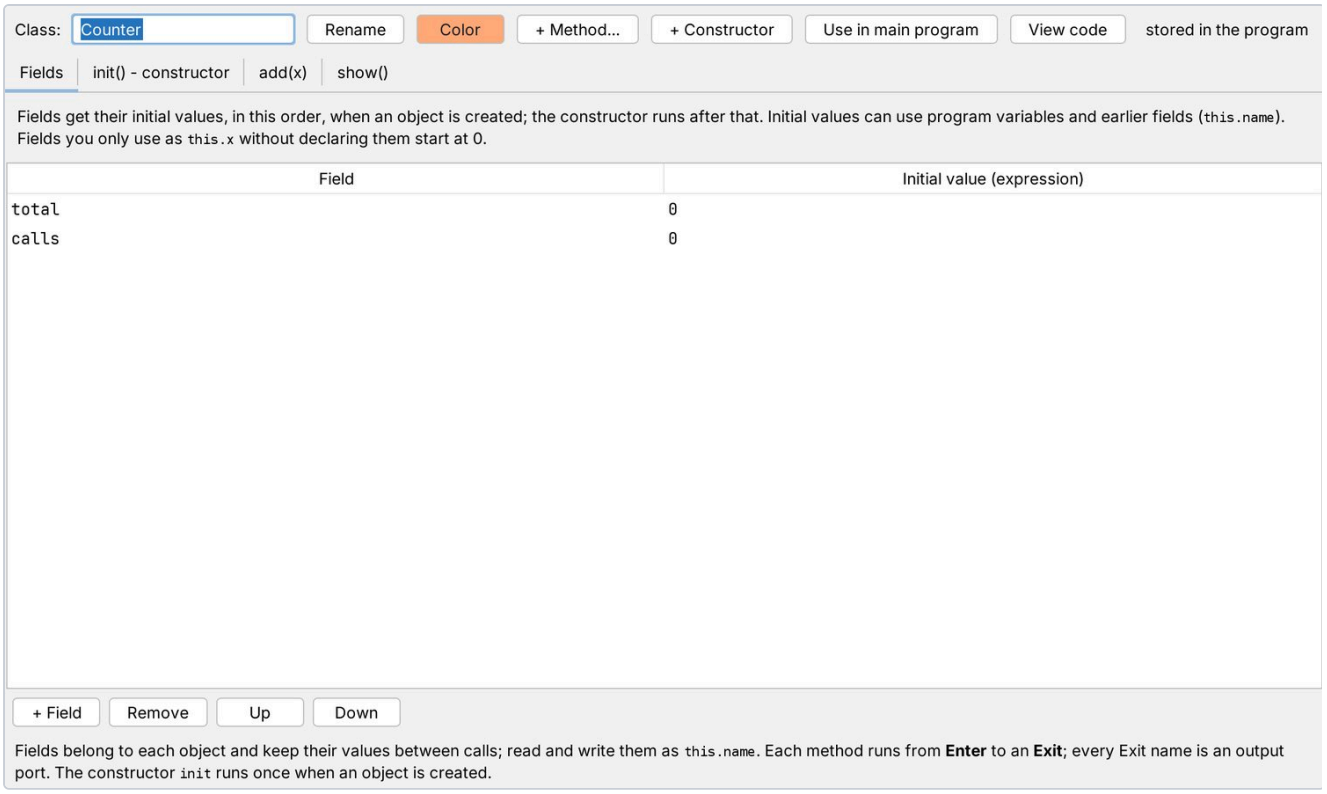
按 **Run**（F5）執行。設定 *Step delay* 可觀察目前執行積木的移動。在圖表上滾動滑鼠滾輪即可縮放。



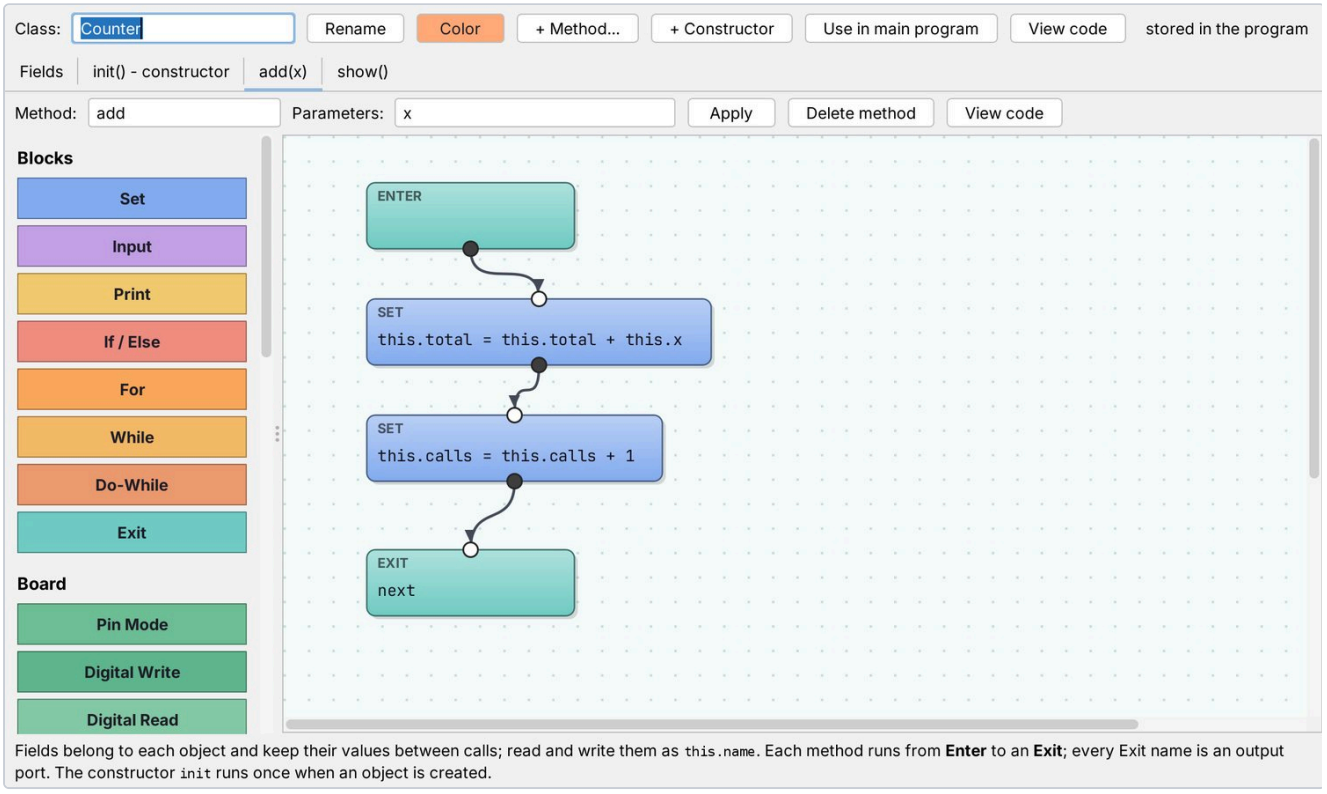
3. 建立類別

選擇 **Edit → New Class...**（或面板中的 *+ New class...*），命名為 `Counter`。類別視窗包含 **Fields** 分頁，以及每個方法各自的分頁。

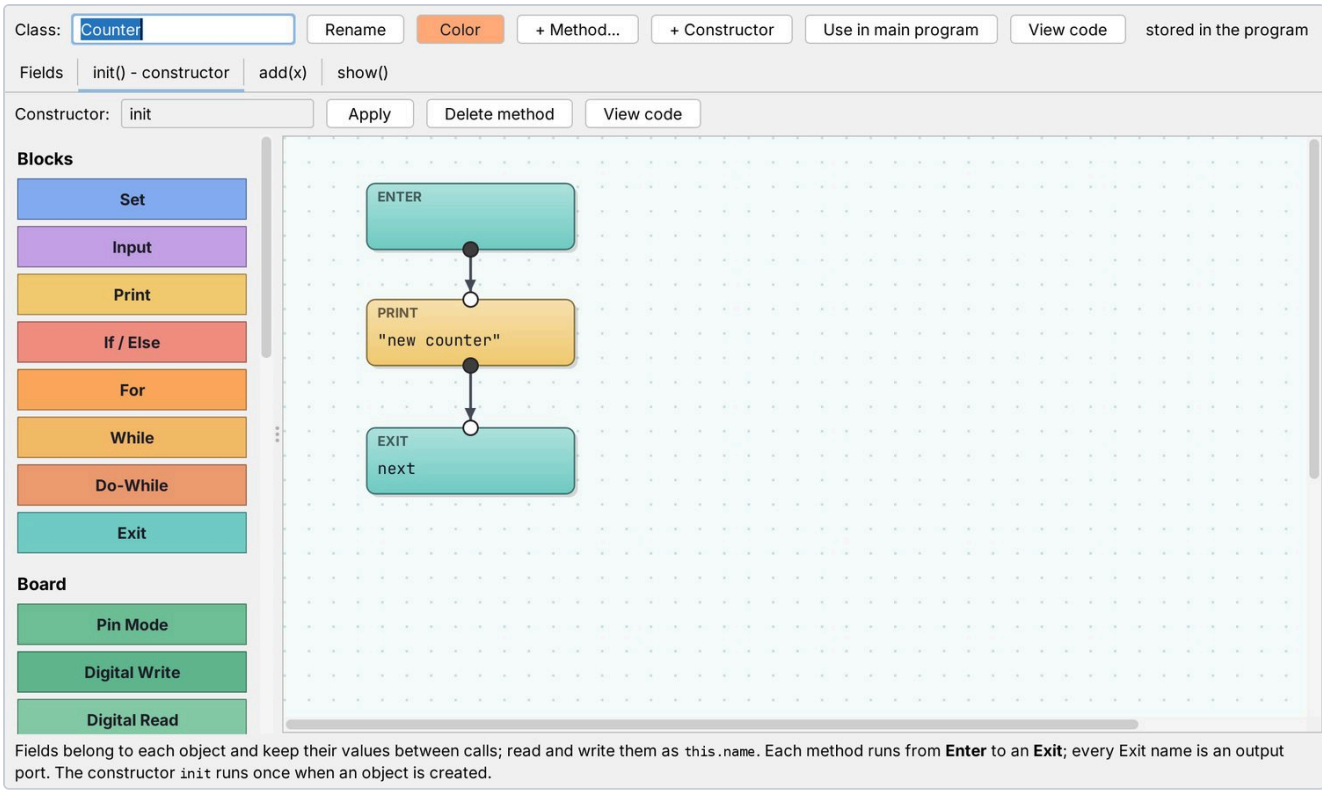
在 **Fields** 中新增 `total` 和 `calls`，初始值都設為 `0`。每個物件都有自己的欄位副本。



將第一個方法重新命名為帶有參數 `x` 的 `add`（按 **Apply**），接著從 **Enter** 到 **Exit** 建立方法本體。在方法內，欄位與參數寫成 `this.name`：

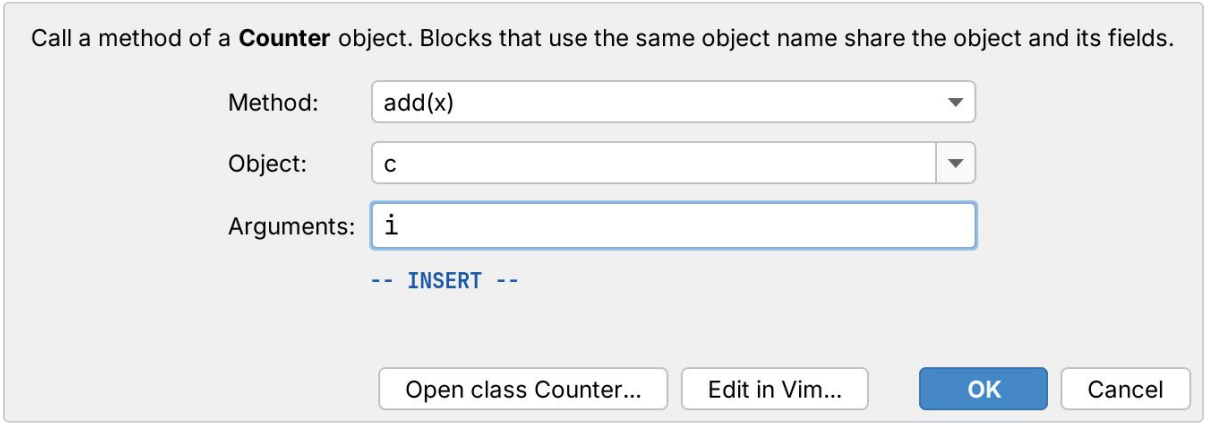


+ Constructor 會新增 `init`，它在建立物件時執行一次（欄位取得初始值之後）。此處讓它輸出 `"new counter"`。**+ Method...** 會新增 `show`，輸出 `"total = " + this.total + " after " + this.calls + " calls"`。

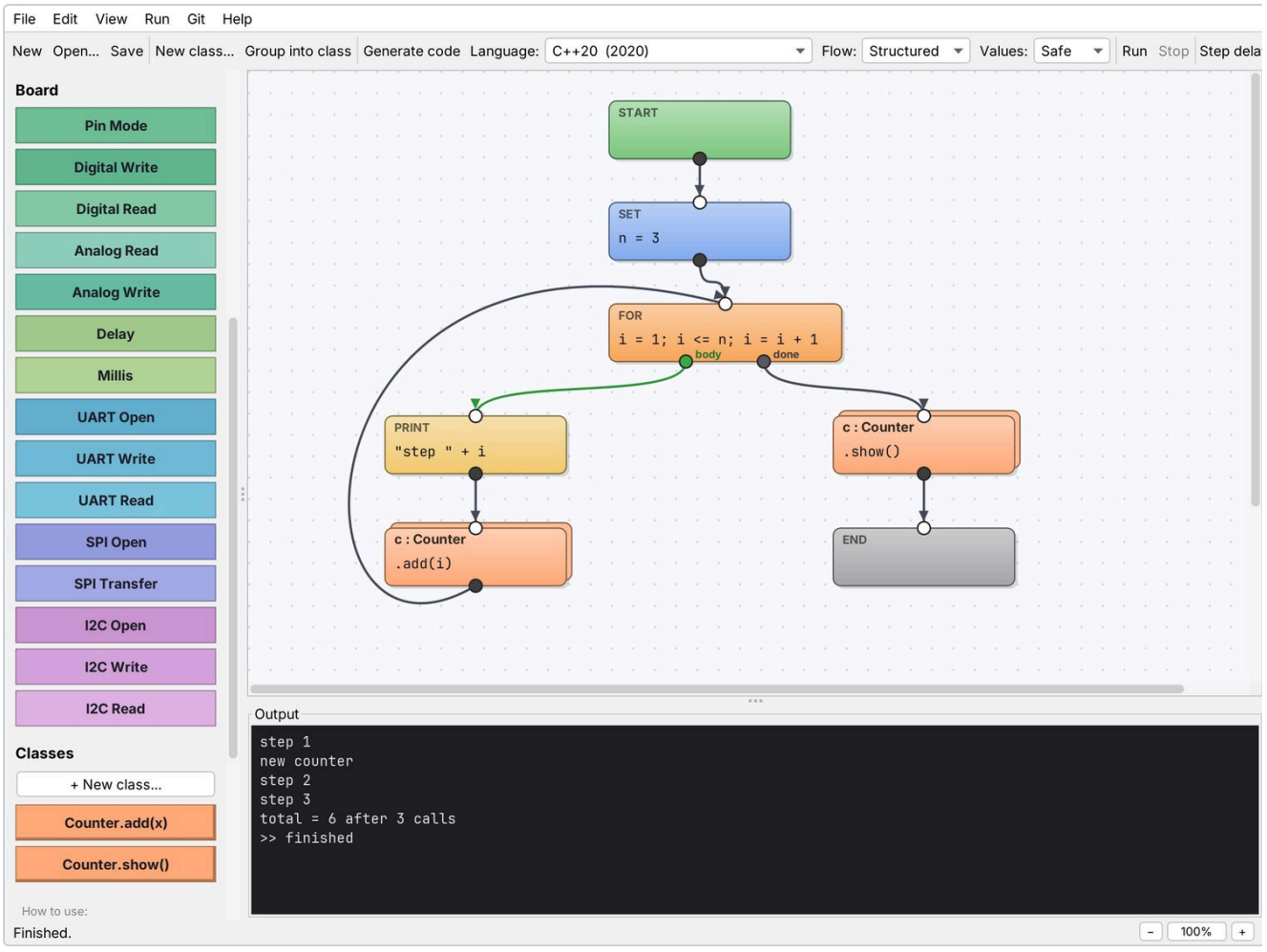


4. 使用類別

面板現在會列出 `Counter.add(x)` 和 `Counter.show()`。將 `Counter.add(x)` 拖到迴圈本體中 **Print** 之後，將 `Counter.show()` 放在 `done` 與 **End** 之間。雙擊呼叫積木可選擇方法、**物件**與引數。兩個積木都使用物件名稱 `c`，讓它們共用同一個物件。

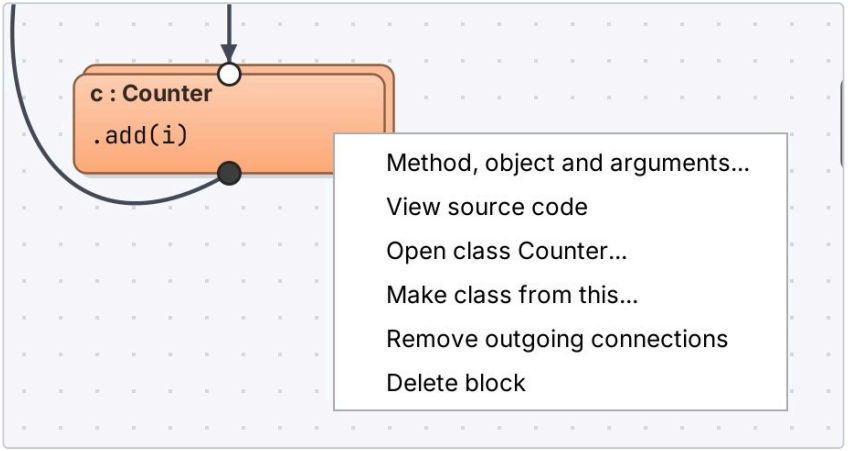


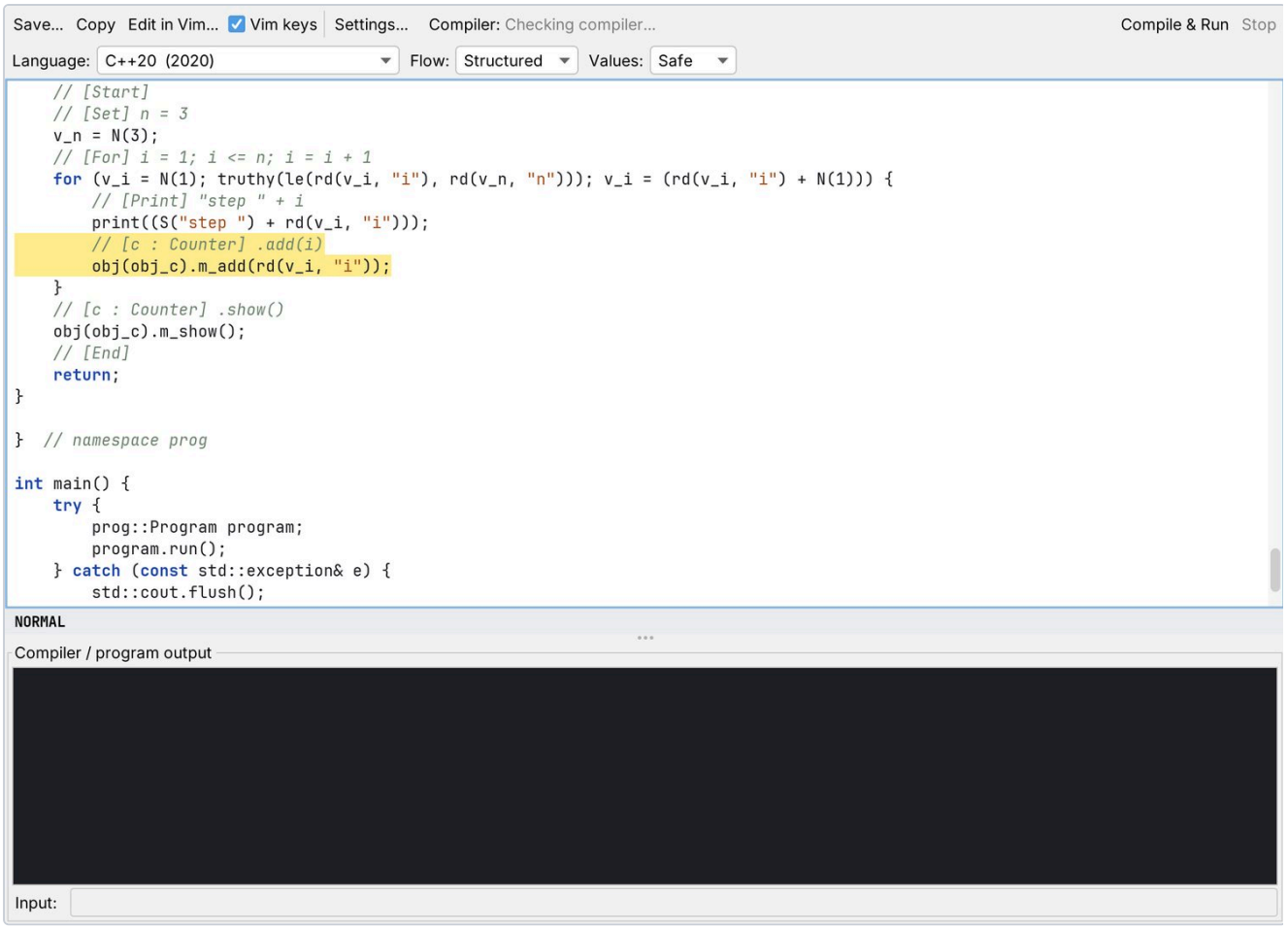
執行程式。第一次使用時會建立 `c`（此時會輸出 `new counter`），`add` 執行三次，`show` 則輸出總計。



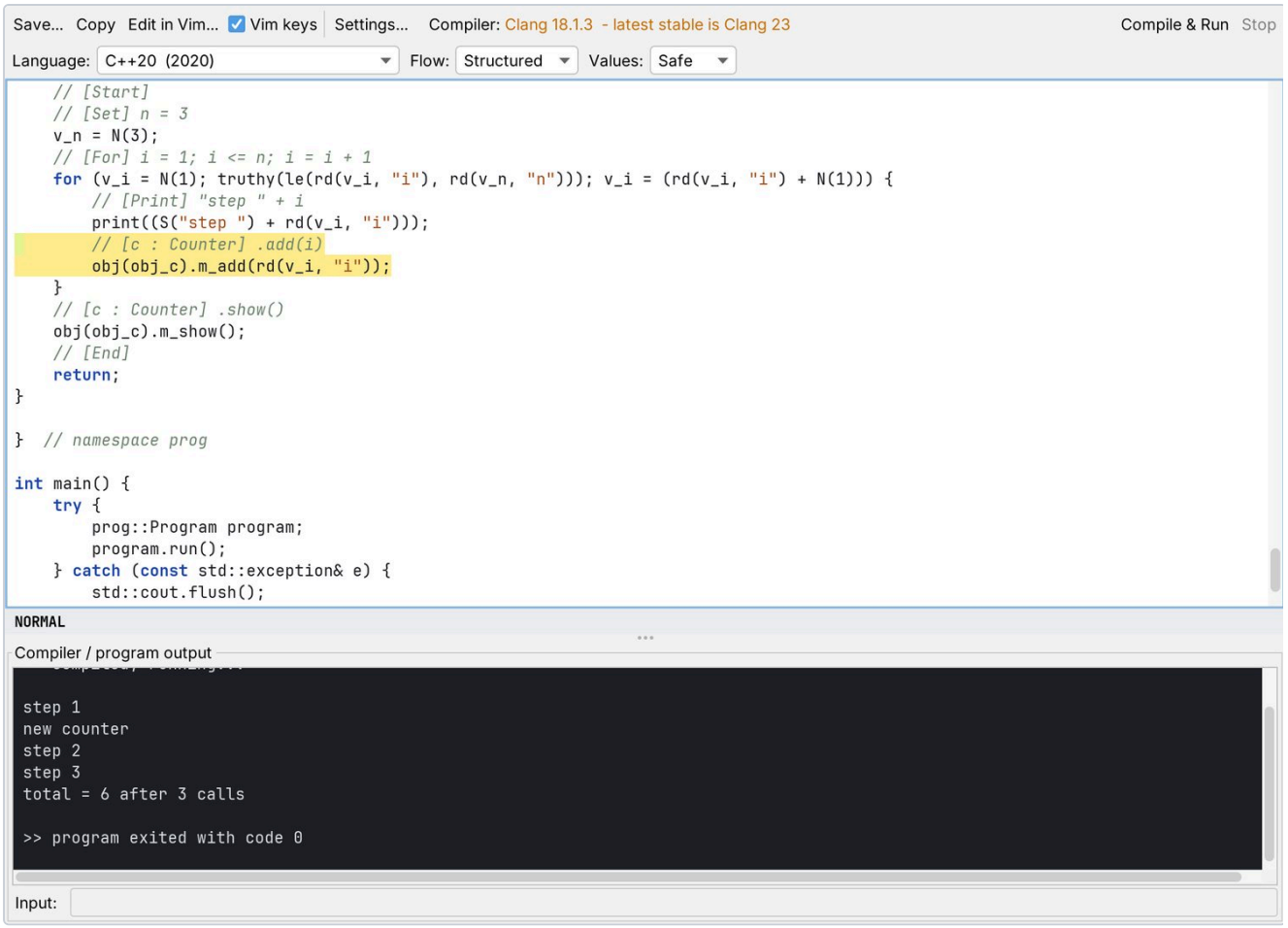
5. 檢視任一積木的程式碼

在積木上按右鍵並選擇 **View source code**。產生的程式碼會開啟，並反白顯示該積木對應的程式碼行（此處預設為 C++20）。



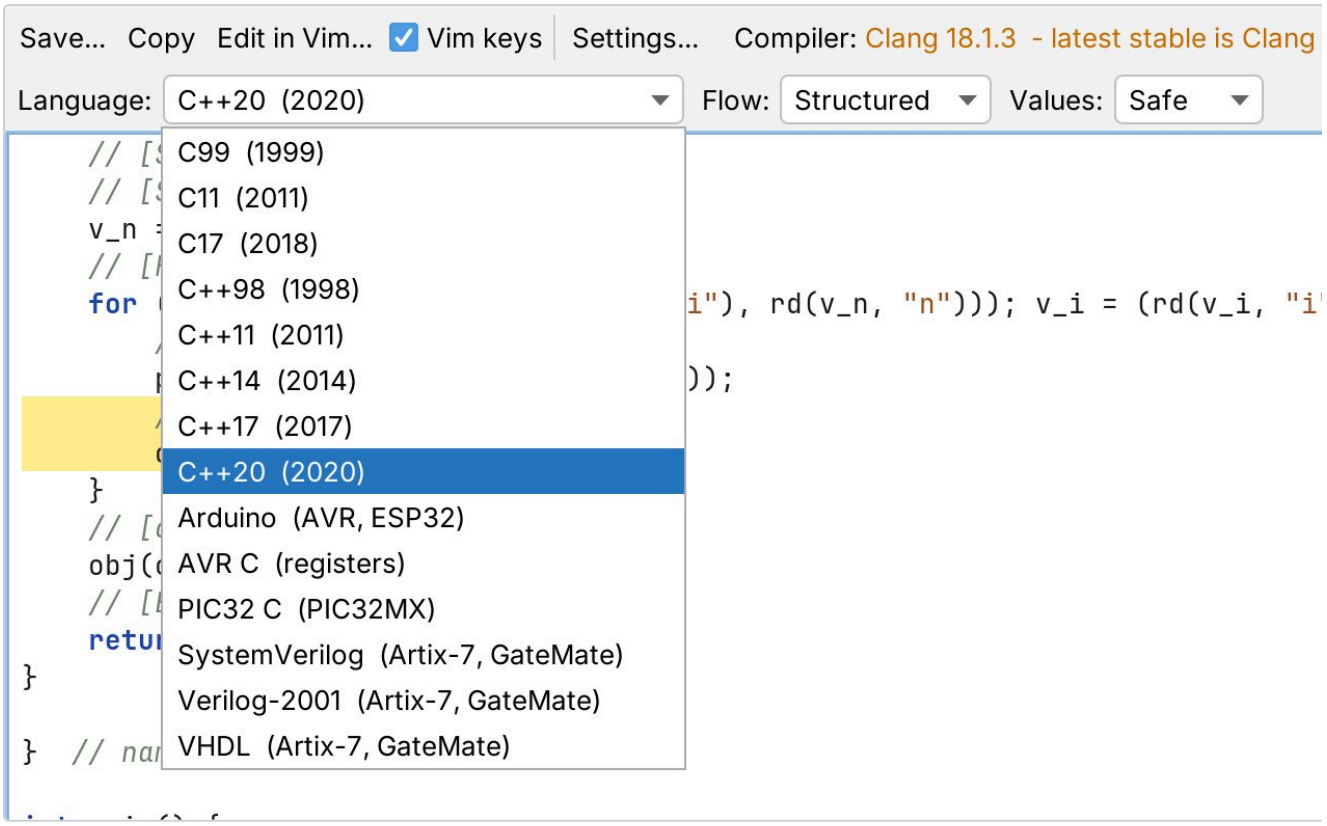


Compile & Run 會以 Clang 或 GCC 編譯並執行。編譯後程式的輸出與直譯器相同。Windows 上應用程式會為你安裝 llvm-mingw（Clang）或 WinLibs（GCC），不需要 Visual Studio。

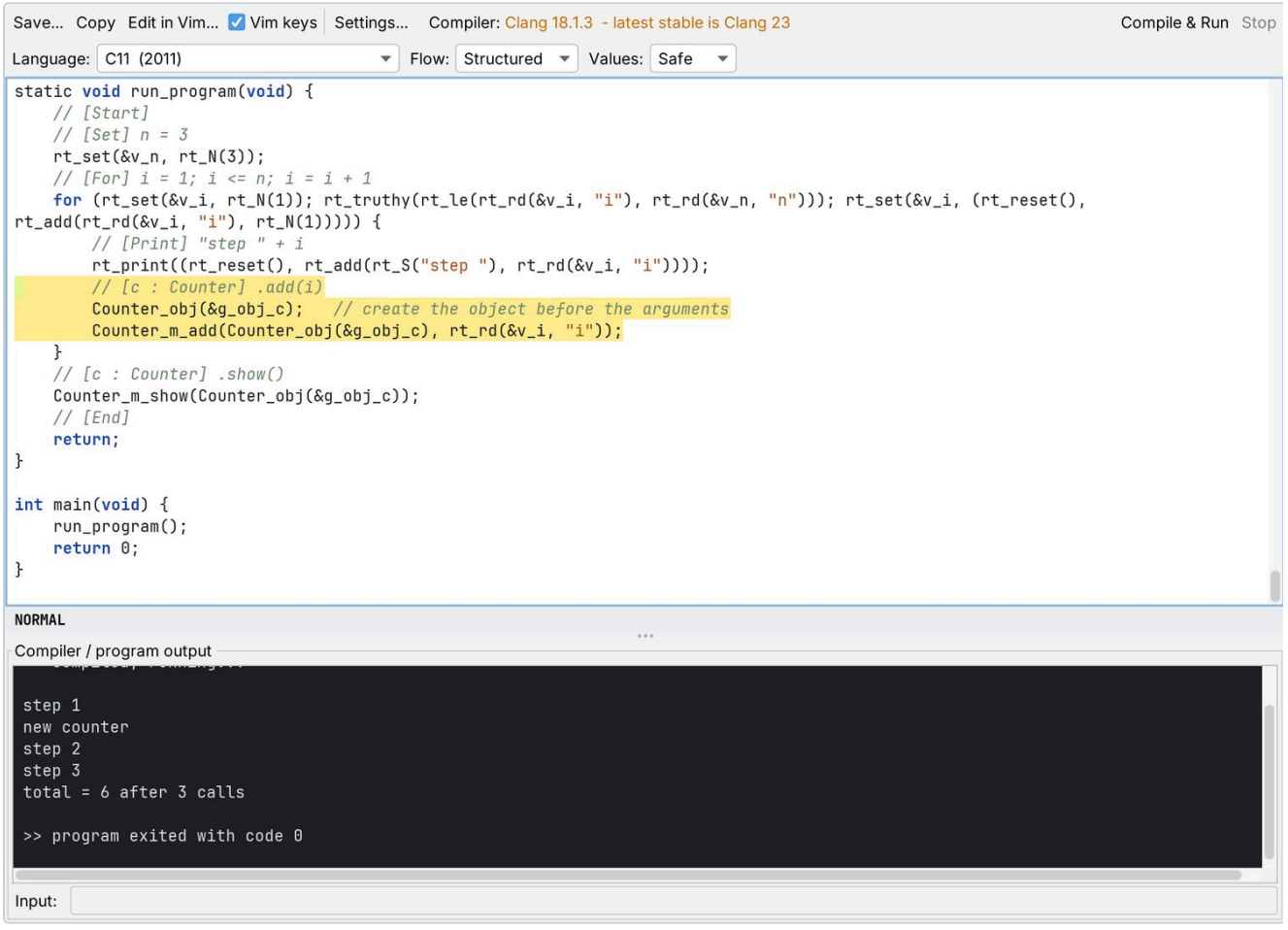


6. 選擇 C 或其他 C++ 標準

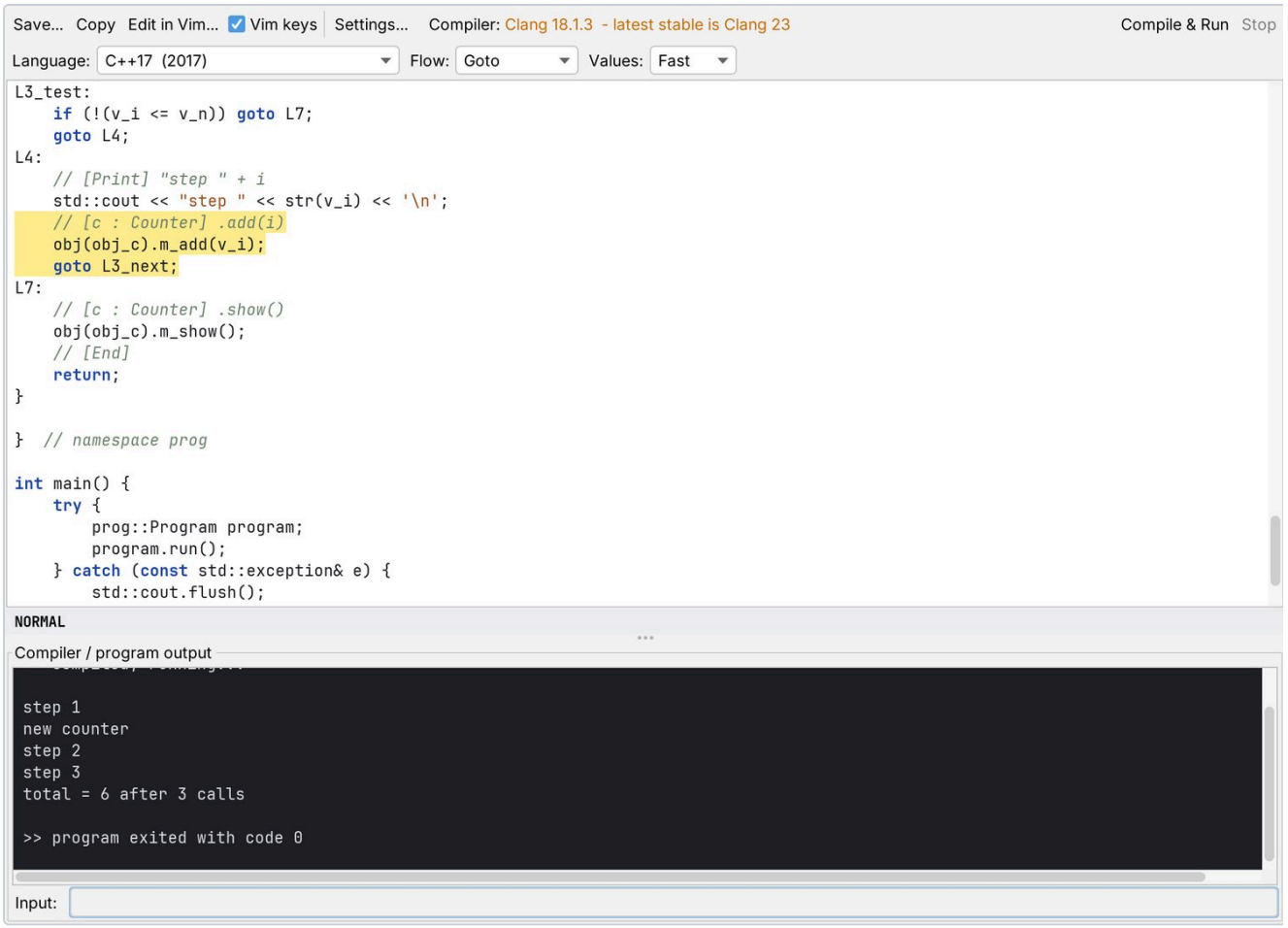
在程式碼視窗或主工具列的下拉選單選擇 **Language**，程式碼會立即重新產生，原本選取的積木仍保持反白。**Flow** 和 **Values** 用來選擇程式風格：**Structured** 使用 if/else、迴圈與 switch；**Goto** 將每個積木轉為標籤、每條連線轉為跳躍。**Safe** 保留文字型別與直譯器檢查；**Fast** 使用沒有執行期檢查的普通 `double`。



同一程式也可產生 **C11**，以 `clang -std=c11` 編譯並執行，輸出相同。

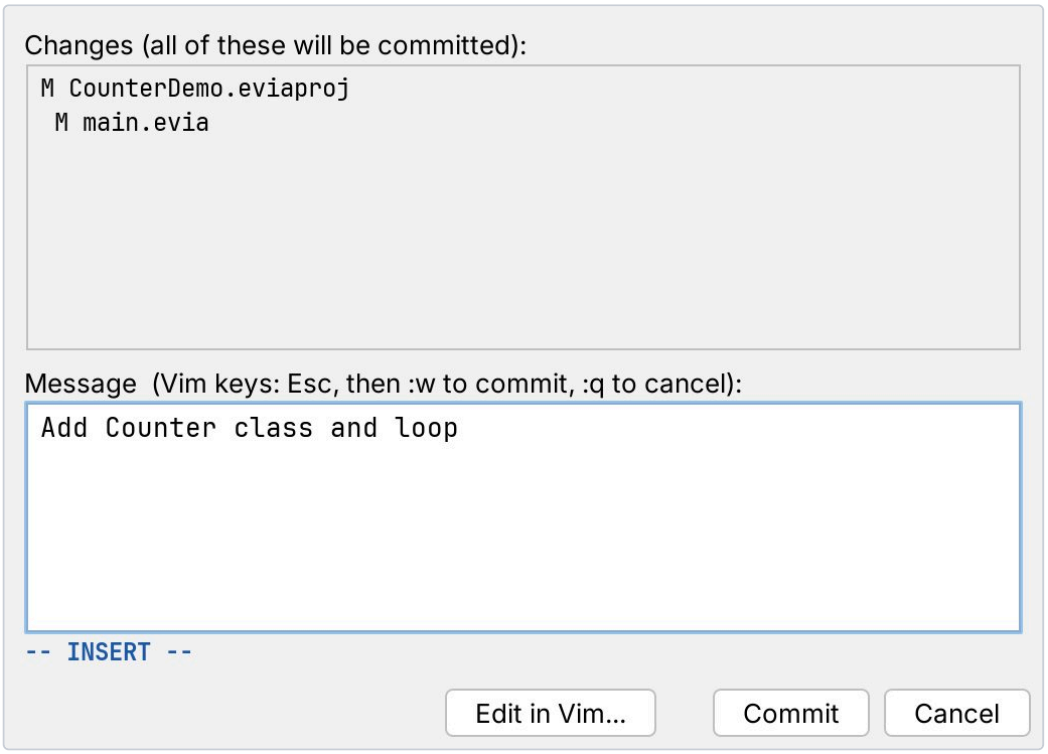


C++17 搭配 **Goto** 流程與 **Fast** 值：

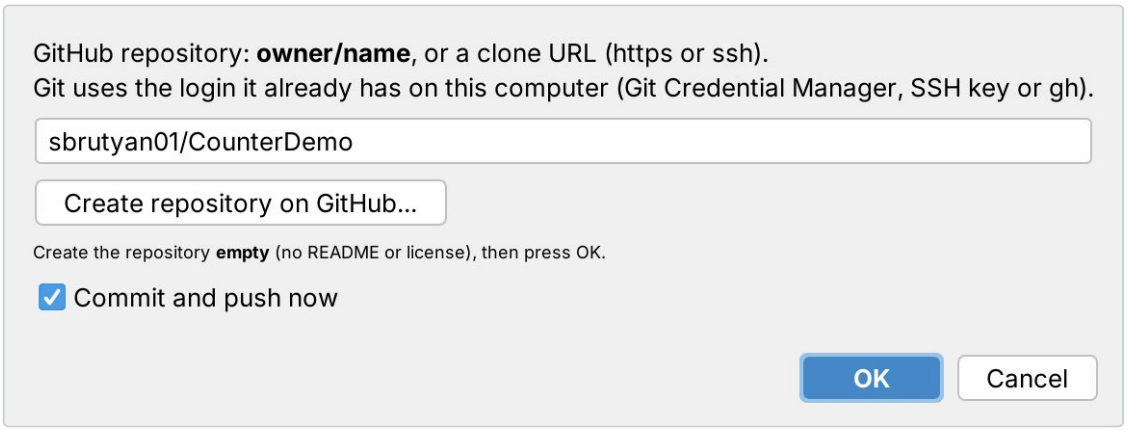


7. 提交變更並發佈到 GitHub

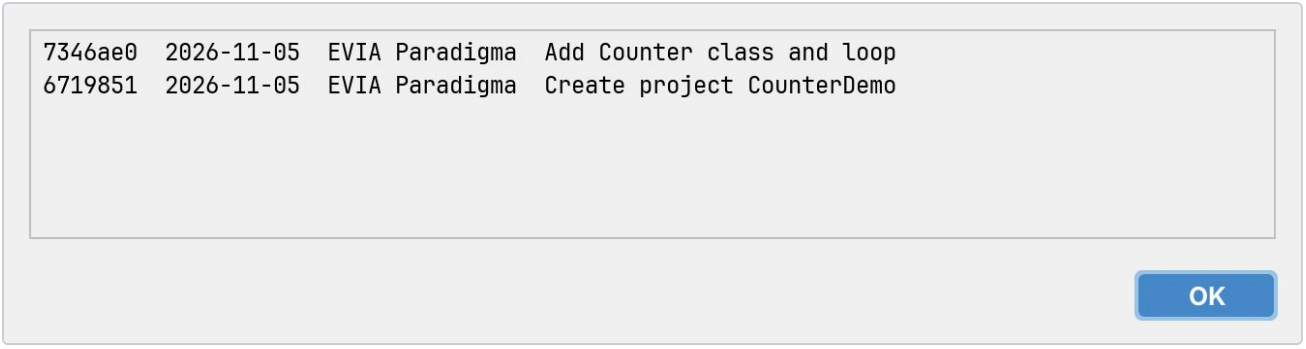
Git → Commit...（Ctrl+K）會儲存並提交所有變更。訊息欄位支援 Vim 按鍵（按 `Esc`，再輸入 `:w` 即可提交）。



選擇 **Git → Connect to GitHub...** 並輸入 `owner/repo`。*Create repository on GitHub...* 會在 github.com 開啟已填入名稱的頁面；建立一個**空白**儲存庫，按 **OK** 即可推送專案。Git 會使用電腦上已設定的認證資訊（Git Credential Manager、SSH 金鑰或 `gh auth login`）。

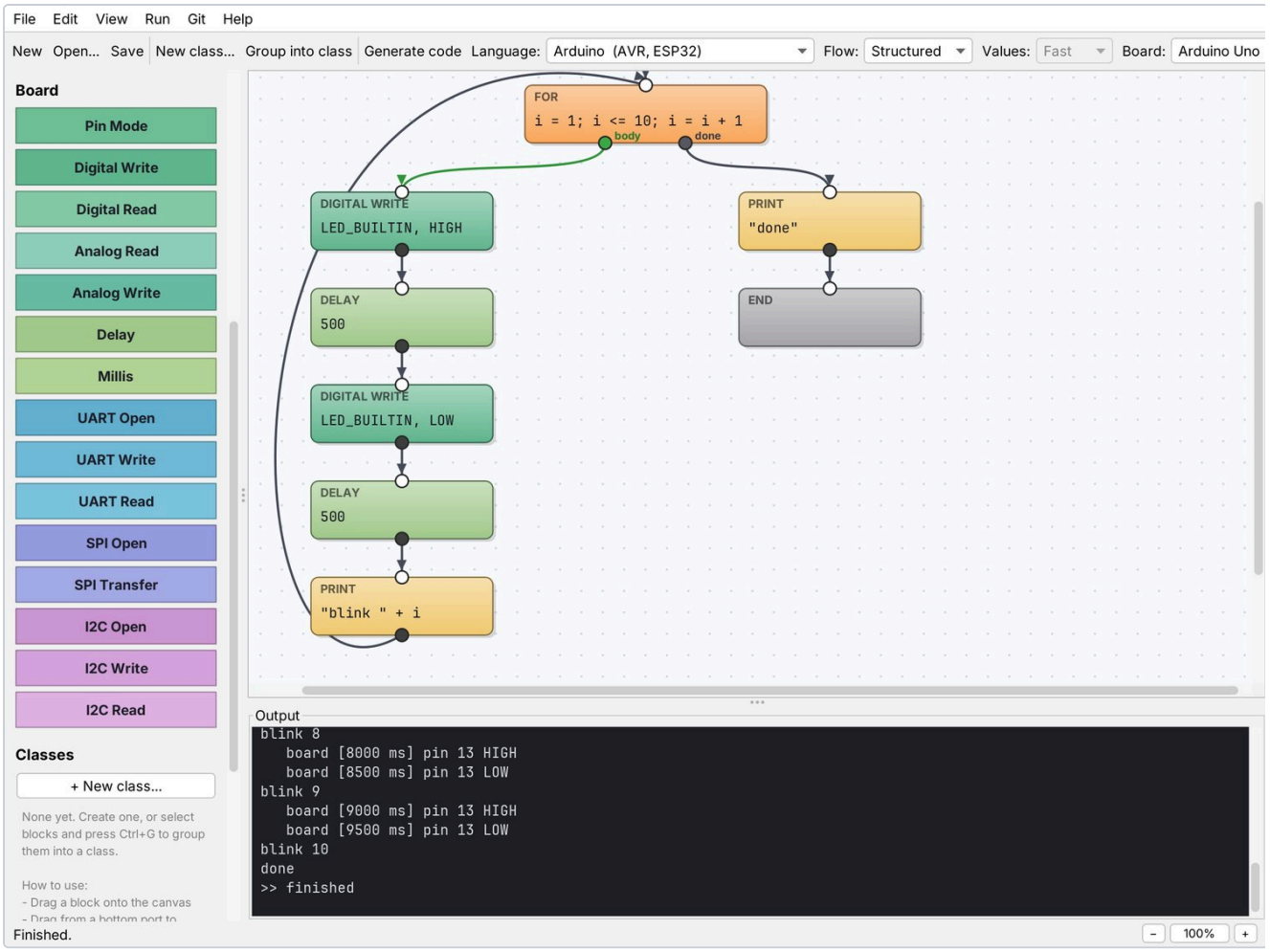


Git → History 會列出 commit：

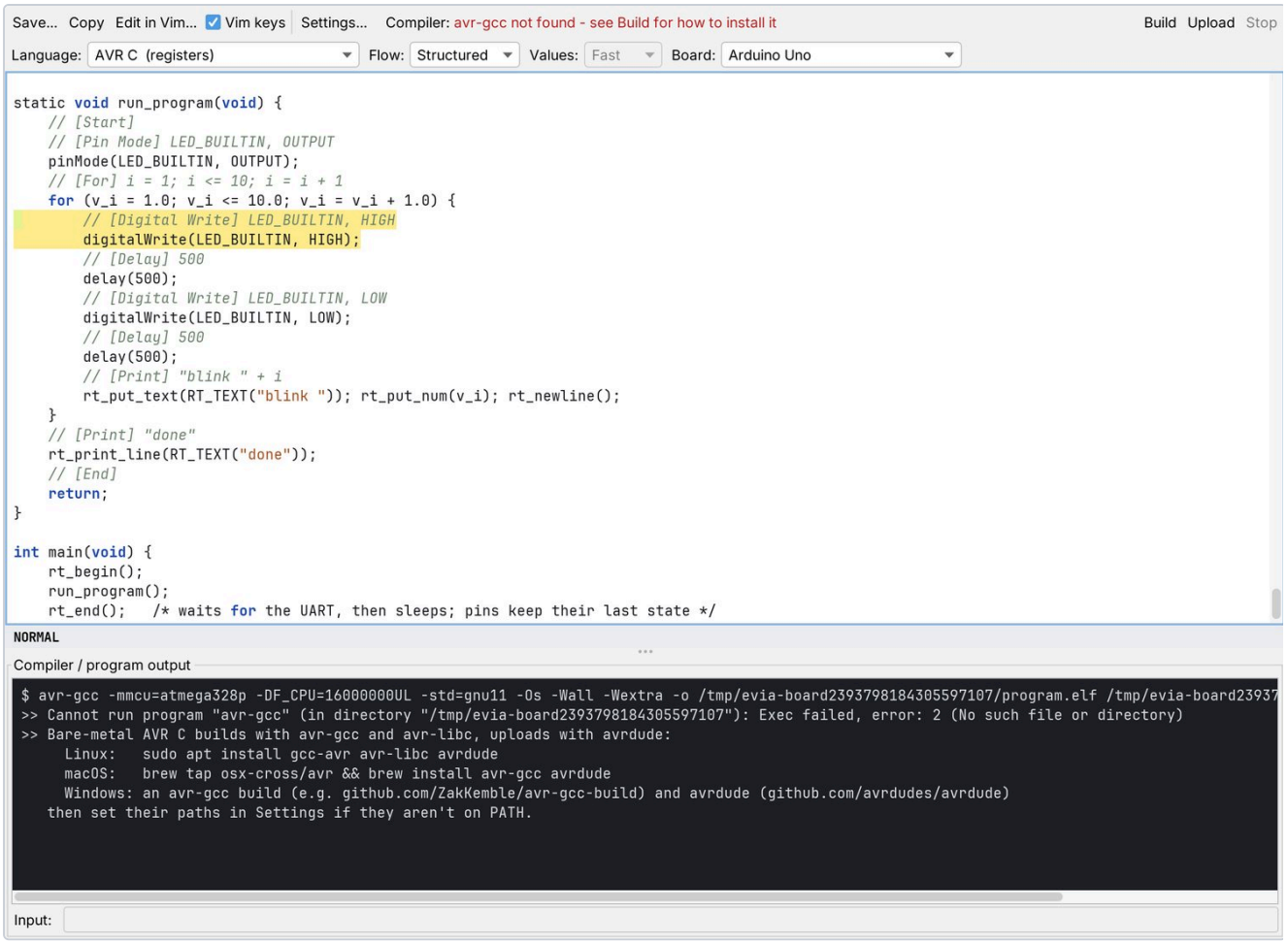


8. 為 Arduino 或 ESP32 開發板編寫程式

選擇 **File → Open Example → Arduino and ESP32 Boards → Blink the built-in LED**。面板 *Board* 群組中的綠色 **Board** 積木可設定接腳、讀取按鈕與感測器，並加入等待。按 **Run**（F5）可模擬 Arduino Uno；輸出會顯示每次接腳變更及其時間。



選擇 **Language: Arduino** 和 **Board**，再按 **Generate code**（F7）：產生的程式適用於 12 種開發板。**Build** 使用 arduino-cli 編譯，**Upload** 將程式上傳到開發板；Print 與 Input 透過 115200 baud 的 Serial Monitor 運作。**AVR C** 不使用 Arduino 核心，而是直接操作晶片暫存器產生相同程式，並以 avr-gcc 編譯。



9. 與其他晶片通訊：UART、SPI、I2C 與 PIC32

File → Open Example → Arduino and ESP32 Boards 中還有三個使用匯流排積木的範例：*UART: send a byte, read it back*、*SPI: transfer a byte*、*I2C: write and read a device register*。使用前先開啟匯流排。**Run** 會模擬它們：UART 寫入的位元組會由 UART 讀回，SPI 輸入連到輸出，I2C 位址 80 有一個含 256 個暫存器的裝置回應（80 即 0x50；數值為十進位）。

積木	程式碼	功能
UART Open	`9600`	以指定 baud rate 啟動序列埠
UART Write	`65`	傳送一個位元組（0..255）
UART Read	`c`	讀取下一個位元組；沒有資料時回傳 -1
SPI Open	`1000000`	以指定頻率（Hz）啟動 SPI 主控端（模式 0，最高位元先傳）
SPI Transfer	`reply = 66`	傳送一個位元組，並儲存同時收到的位元組
I2C Open	`100000`	以指定頻率（Hz）啟動 I2C 主控端
I2C Write	`80, 0, 42`	位址（7 位元）、暫存器、值
I2C Read	`v = 80, 0`	儲存暫存器值；沒有裝置回應時為 -1

若使用 Microchip PIC32MX 晶片，請選擇 **Language: PIC32 C**，再從 **Board** 清單選擇晶片（MX1xx/2xx、MX3xx/4xx、MX5xx/6xx/7xx 系列共九款），然後按 **Generate code**（F7）。產生的裸機 C 可使用 Microchip 免費的 XC32 編譯器：時脈、接腳、PWM、ADC、UART、SPI 與 I2C 都透過暫存器

設定。**Build** 會執行 `xc32-gcc` 和 `xc32-bin2hex`；**Upload** 會執行 **Settings** 中設定的命令（預設為 `pic32prog {hex}`）。所選晶片系列的接腳列於產生檔案開頭。

注意： PIC32 程式碼已使用 Microchip 實際的裝置標頭檔檢查，但尚未以 XC32 編譯，也未在晶片上執行。請將首次在開發板上的建置視為測試。

10. 為 FPGA 建立硬體

選擇 **Language: SystemVerilog**、**Verilog-2001** 或 **VHDL**，再從 **Board** 清單選擇 Digilent Nexys A7-100T、Arty A7-35 或 A7-100、Basys 3（Artix-7），或 Olimex GateMateA1-EVB、Cologne Chip GateMate A1 evaluation board。開啟 **File → Open Example → FPGA Boards → Blink**，再按 **Generate code**（F7）。

每個積木會成為有限狀態機的一個狀態，並佔用一個時脈週期。設計建置時接腳已固定，因此請使用名稱：`LED0`、`SW3`、`BTNC`、`JA1`。接腳限制寫在檔案第一個註解中；在命令列使用 `-o` 時，也會另存為 `.xdc`（Vivado）或 `.ccf`（GateMate）。

- **Build** 與 **Upload** 在 Artix-7 上使用 Vivado 或開放原始碼工具流程（於 **Settings** 選擇）；GateMate 使用 yosys、nextpnr-himbaechel、gmpack 與 openFPGALoader。
- 數值為 32 位元整數，因此 FPGA 上 `5 / 2` 得到 2，而 **Run** 得到 2.5。
- 目前尚不支援類別、Input、Analog Read、SPI、I2C 與變數中的文字；錯誤訊息會指出該積木。

注意： 輸出已使用開放原始碼工具模擬與綜合，但尚未經過 Vivado，也未載入實際開發板。

11. 用 MQTT 把數字送出去

開啟 **File → Open Example → Arduino and ESP32 Boards → MQTT sensor**。這個程式向代理伺服器發布溫度，再給自己 送一條命令並讀回來。

MQTT 的五個積木在 *Board* 群組裡：*MQTT Connect*、*MQTT Publish*、*MQTT Subscribe*、*MQTT Has Message* 與 *MQTT Read*。網路名稱、密碼、代理伺服器與主題都是固定文字，因為它們在建置時寫進程式裡。

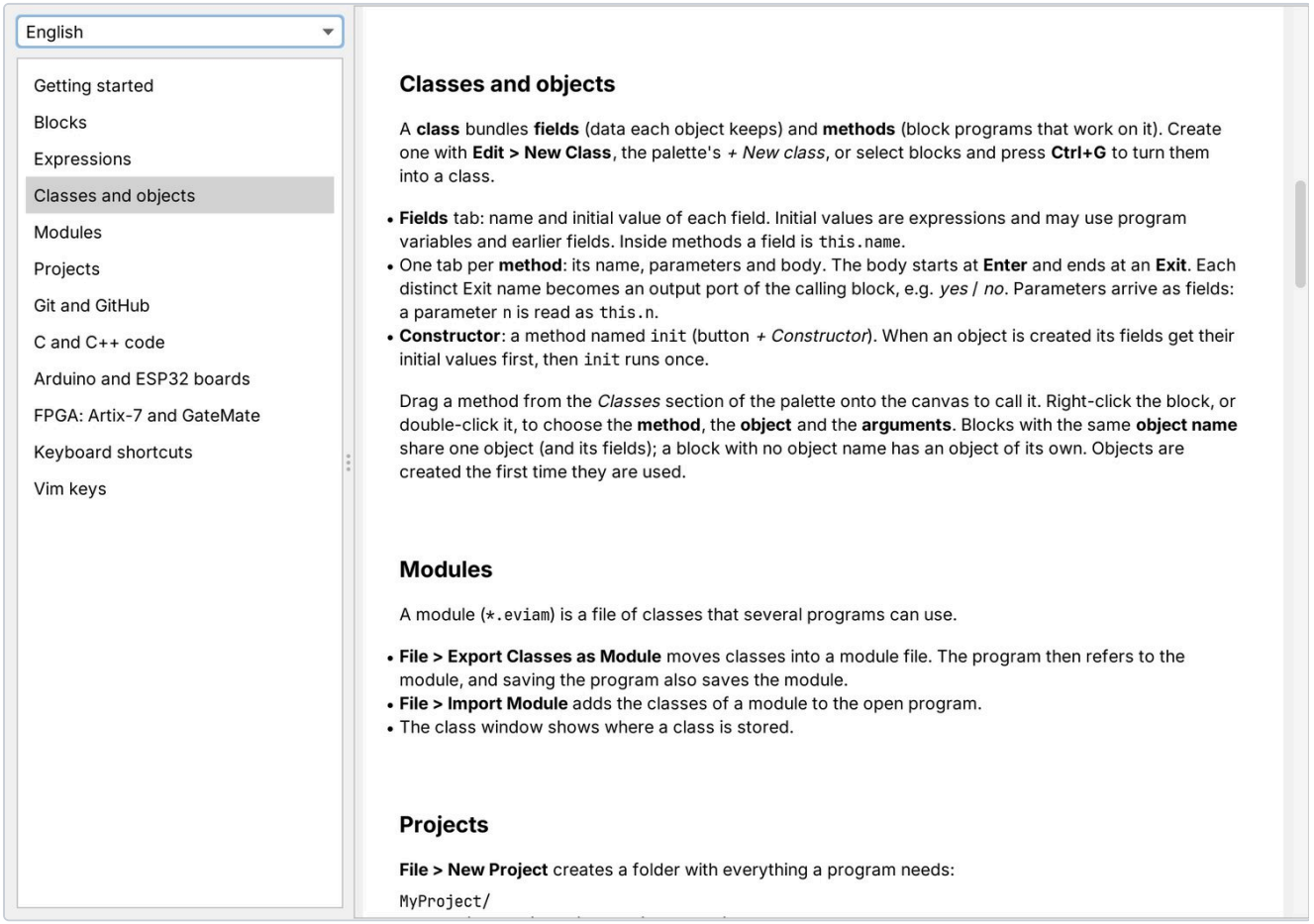
- **Run** 由你電腦上的代理伺服器作答，所以沒有硬體也能試。
- **PIC32 C** 透過 UART2 驅動帶 AT 韌體的 ESP8266 或 ESP32 模組，Print 仍用 UART1。
- **ESP32** 板上的 **Arduino** 使用板子自己的 WiFi。
- AVR 板會拒絕這些積木：它只有一個序列埠，而 Print 正在用。

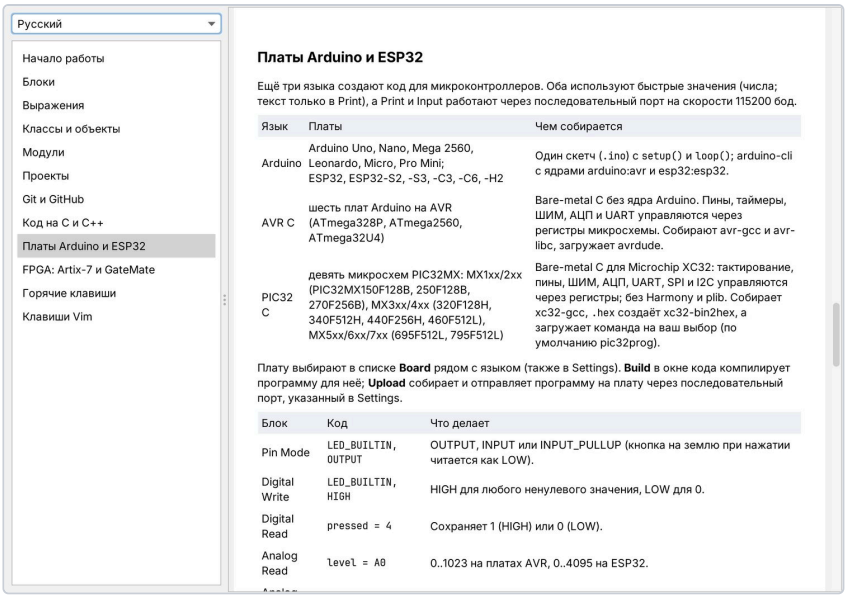
訊息是一個數字。沒有訊息時 *MQTT Read* 回傳 -1，所以如果真的可能收到 -1，先用 *MQTT Has Message* 判斷。

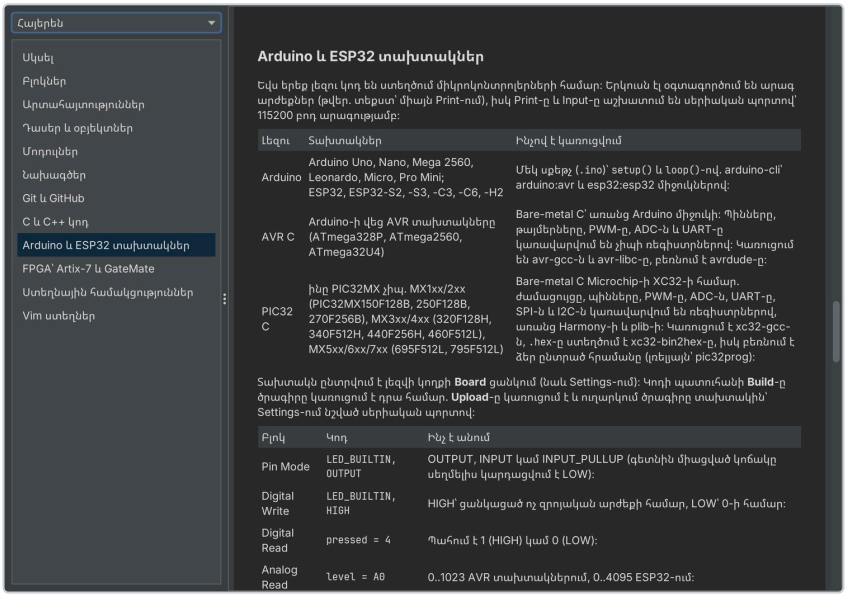
請注意： MQTT 封包在 CI 中對著真實的代理伺服器檢查過，但沒有 WiFi 模組、ESP32 或 PIC32 晶片參與。 MX695/795 晶片的乙太網路埠 MQTT 還不支援。

12. 繼續探索

- 在不同程式間共用類別：選擇 **File → Export Classes as Module...**，再於另一個程式選擇 **File → Import Module...**。
- **File → Open Example** 提供現成程式，例如 `classes.evia`（欄位、建構函式、方法與模組）。
- **Help → Tutorial (PDF)** 可開啟英文、俄文、亞美尼亞文、德文、簡體中文、繁體中文或日文教學。
- **Help → Guide**（F1）以英文、俄文與亞美尼亞文說明所有積木、運算式、類別、專案、Git、快捷鍵和 Vim 按鍵。使用 **View → Theme** 切換淺色與深色主題。




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: