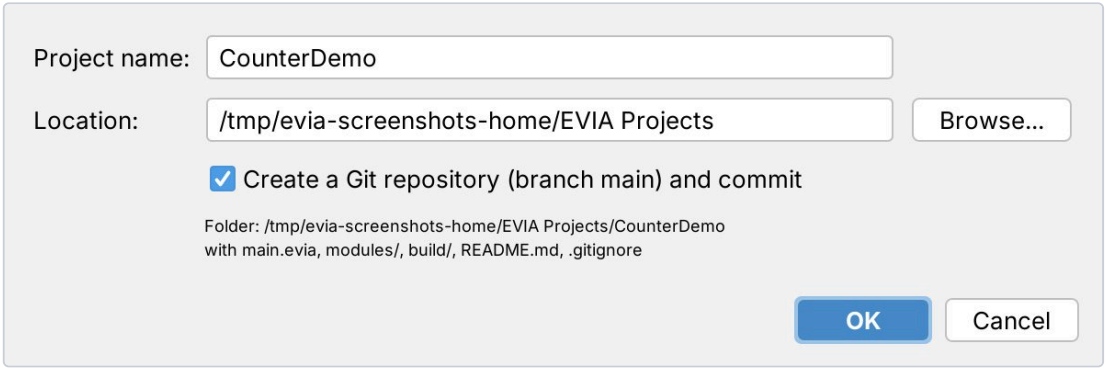
EVIA Paradigma：チュートリアル

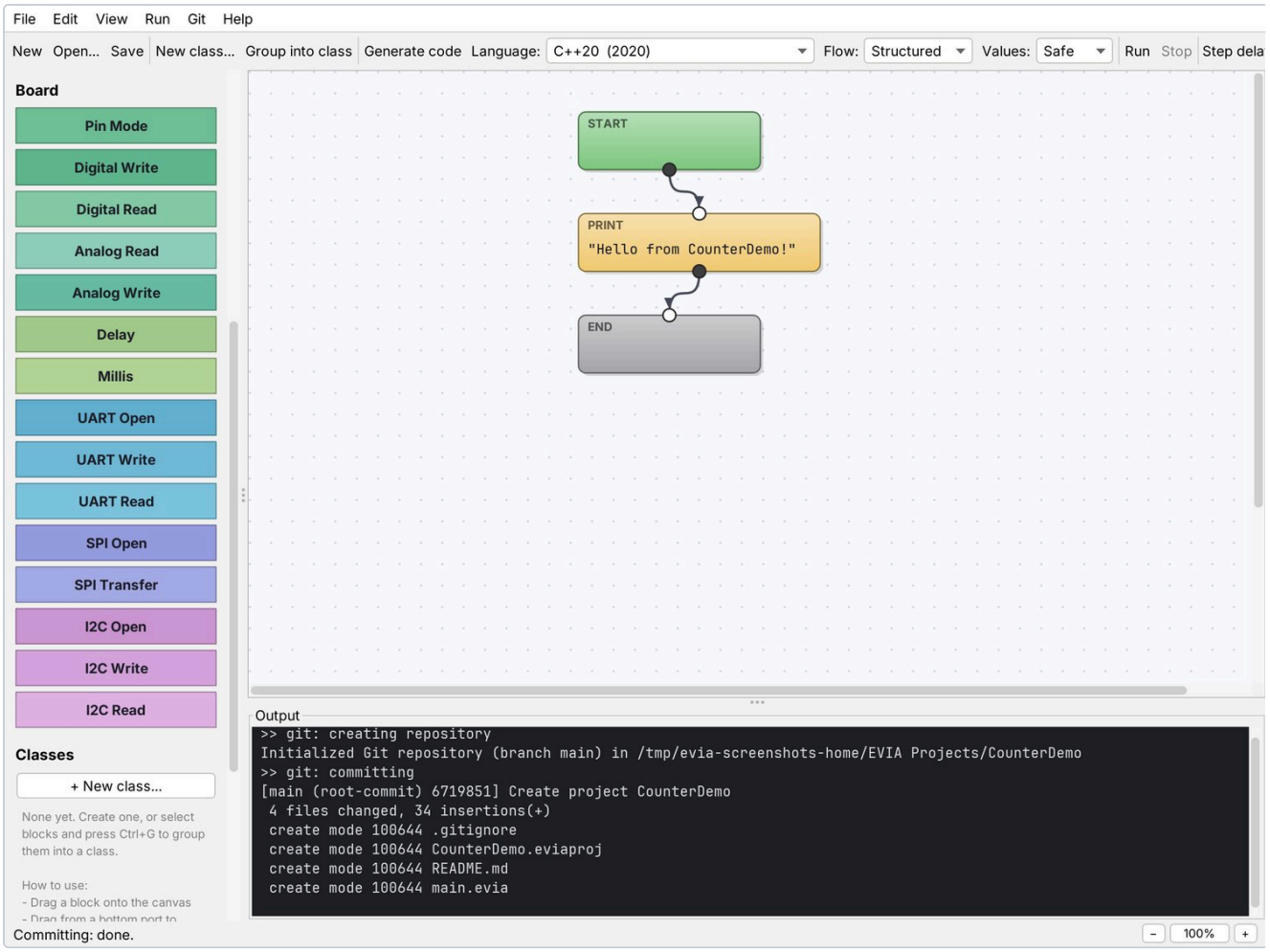
このチュートリアルでは、ループとクラスを使った小さなプロジェクトを作成して実行し、生成される C/C++ コードを確認して、Git で管理します。最後に Arduino または ESP32 ボードのプログラムも作成します。画像はすべて実際のアプリケーションから取得したものです。アプリの UI は英語のため、メニューやボタン名は画面に表示される英語表記のまま記載しています。

1. プロジェクトを作成する

File → New Project...（Ctrl+Shift+N）を選び、名前と保存先を入力します。git がインストール済みなら、*Create a Git repository* をオンのままにします。



プロジェクトは初期プログラム *Start → Print → End* とともに開きます。出力パネルには、git がリポジトリと最初のコミットを作成したことが表示されます。

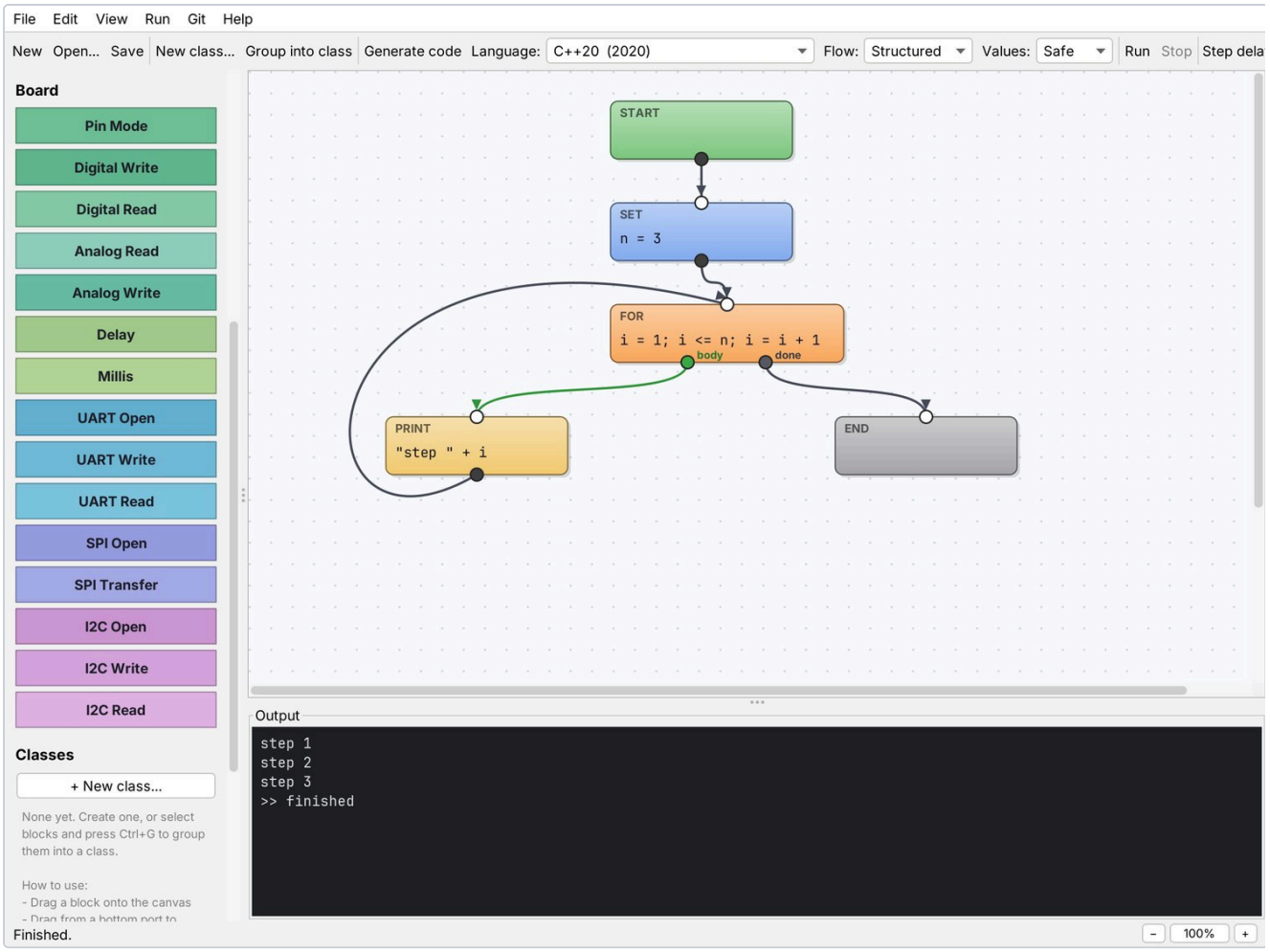


2. ループを作成する

パレットからブロックをキャンバスにドラッグして接続します。ブロック下部のポート（黒い点）から、次のブロック上部のポート（白い点）へドラッグします。ブロックをダブルクリックするとコードを編集できます。

- **Set**：`n = 3`
- **For**：`i = 1; i ⩽ n; i = i + 1`。`body` ポートを **Print** `"step " + i` に接続します。ループ本体の末尾を**For ブロックへ戻して接続**し、次の反復に進めます。`done` ポートは **End** に接続します。

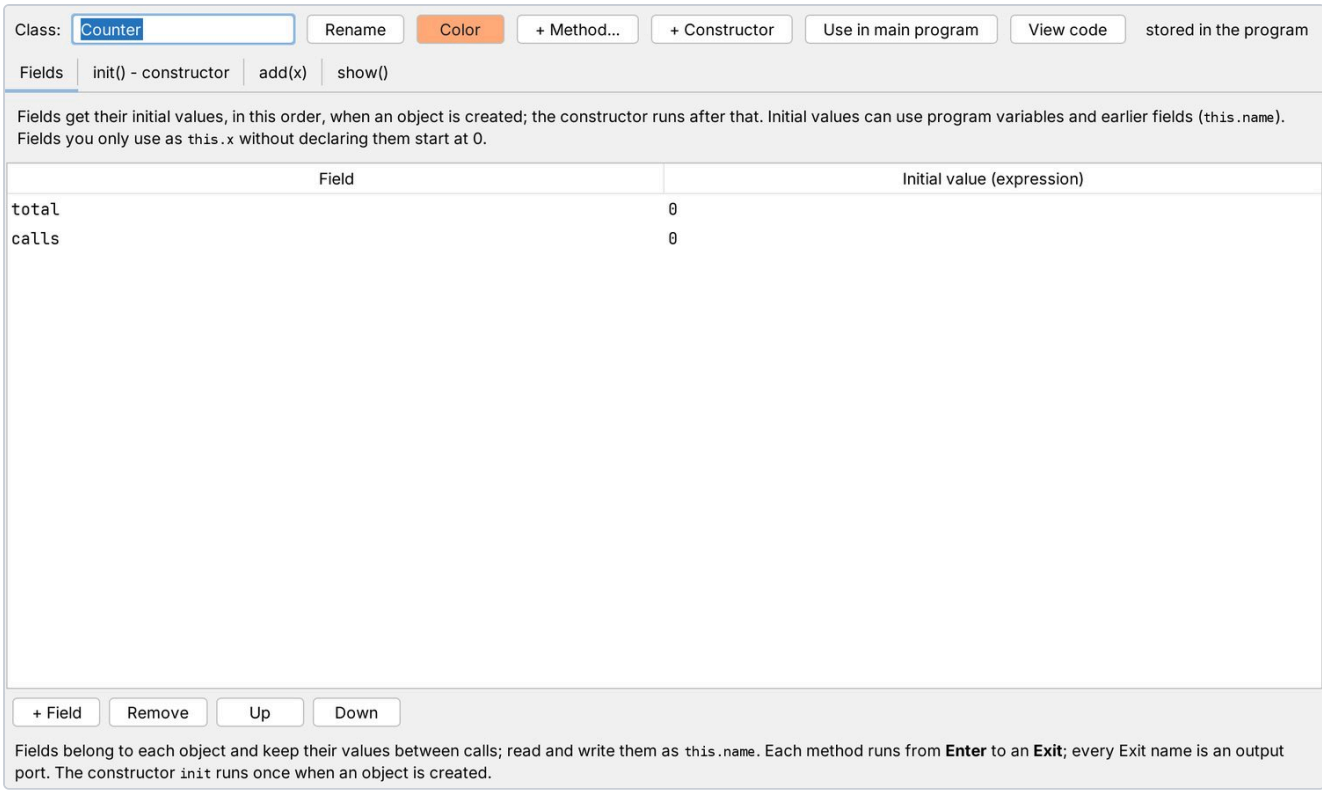
Run（F5）を押します。*Step delay* を設定すると、実行中のブロックが移動する様子を確認できます。図の上でマウスホイールを回すと拡大・縮小できます。



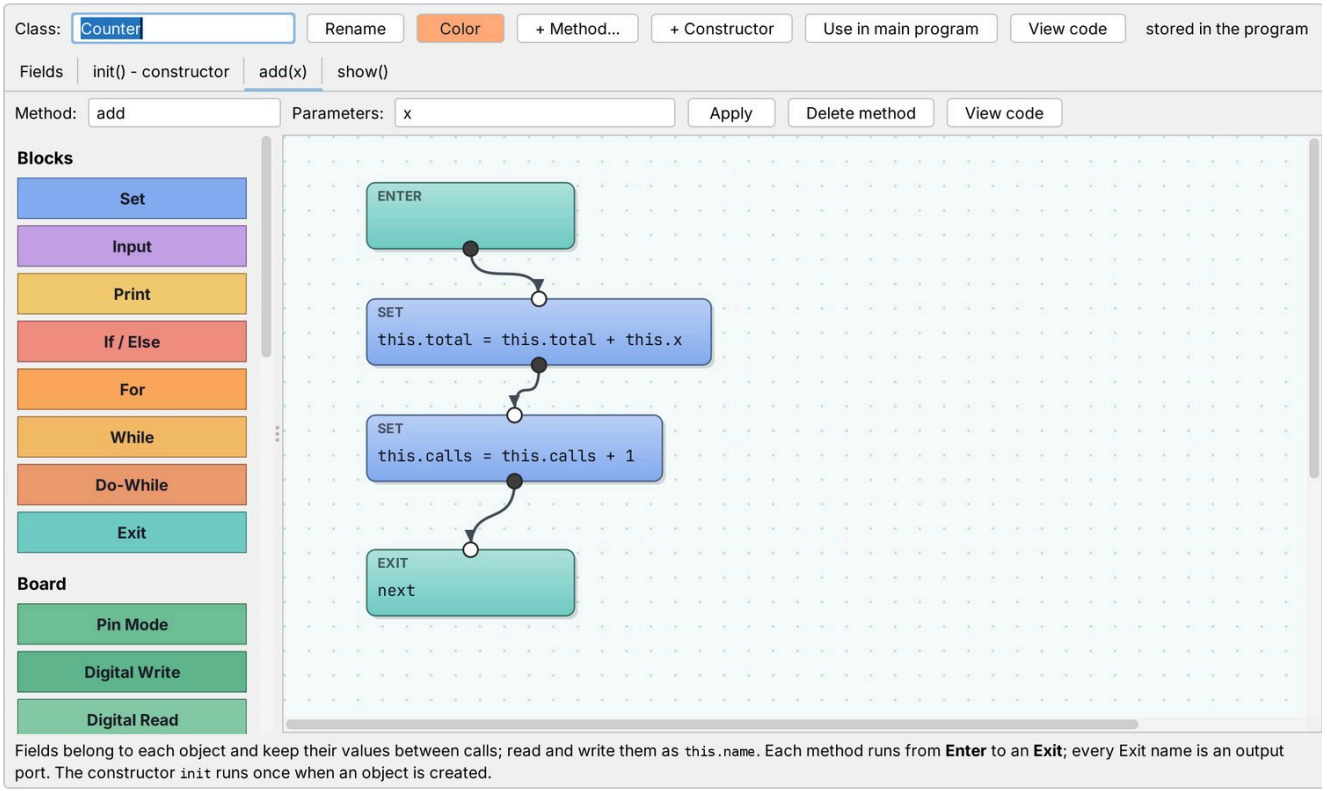
3. クラスを作成する

Edit → New Class...（またはパレットの *+ New class...*）を選び、クラス名を `Counter` にします。クラスウィンドウには **Fields** タブと、メソッドごとのタブがあります。

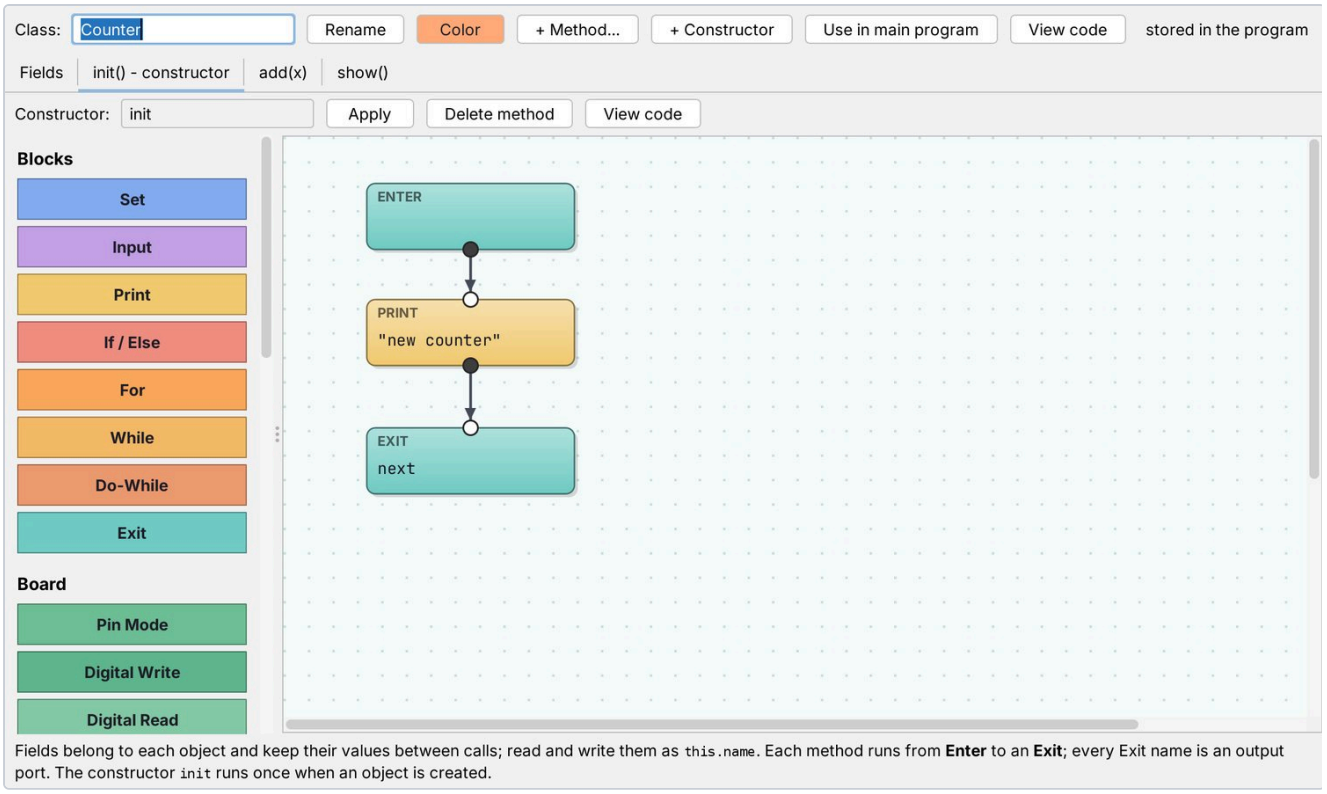
Fields に `total` と `calls` を追加し、どちらも初期値を `0` にします。各オブジェクトは独自のフィールドを持ちます。



最初のメソッド名を、引数 `x` を持つ `add` に変更し（**Apply** を押す）、**Enter** から **Exit** までメソッド本体を作ります。メソッド内では、フィールドと引数を `this.name` の形式で記述します。

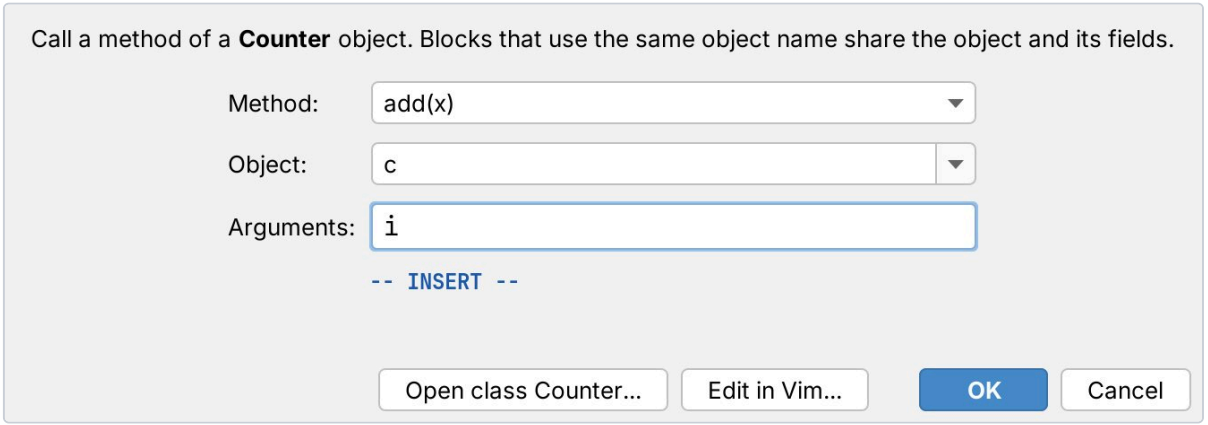


+ Constructor は `init` を追加します。これはオブジェクト作成時に一度だけ実行されます（フィールドの初期化後）。ここでは `"new counter"` を出力します。**+ Method...** で `show` を追加し、`"total = " + this.total + " after " + this.calls + " calls"` を出力します。

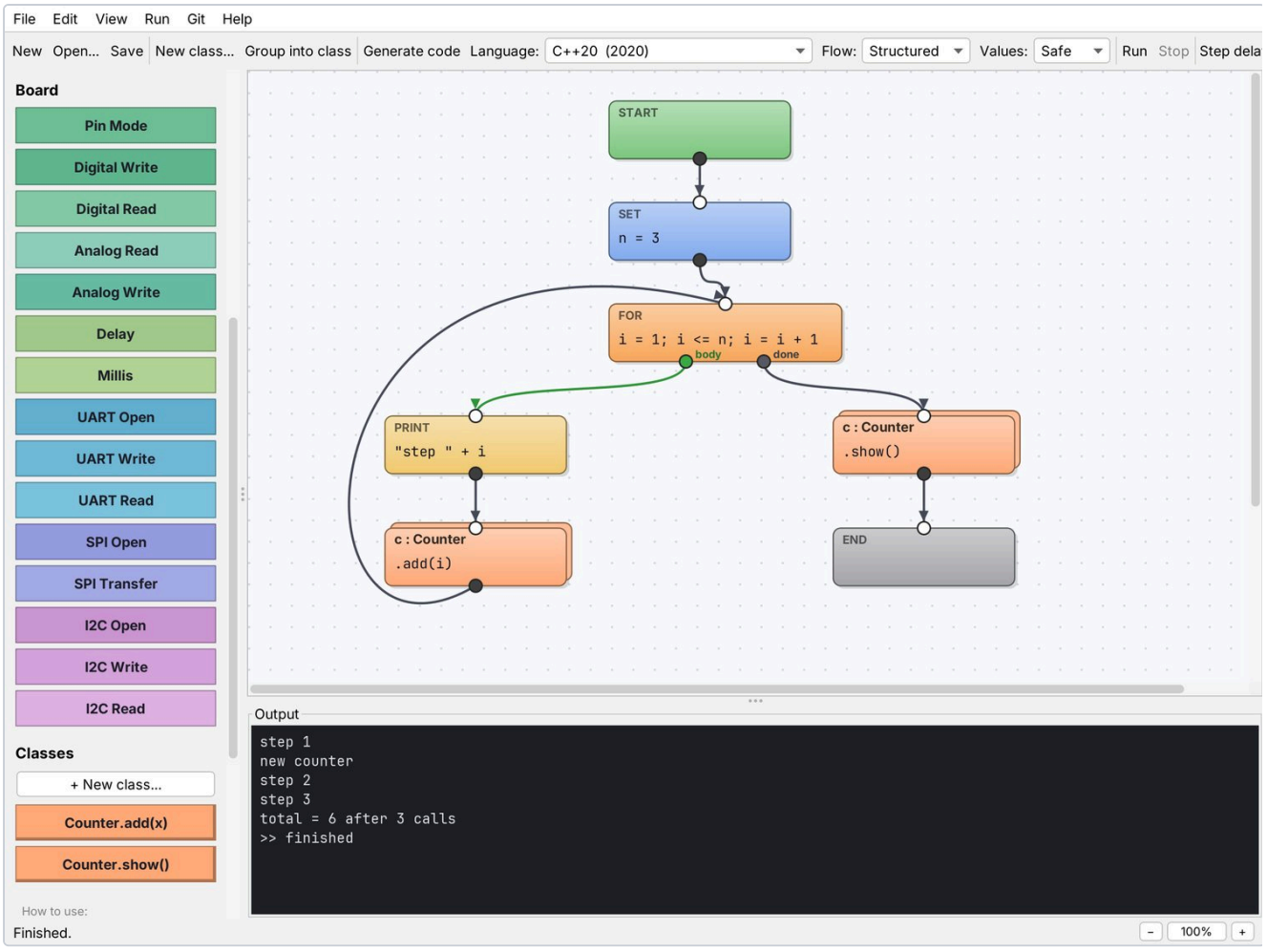


4. クラスを使う

パレットに `Counter.add(x)` と `Counter.show()` が表示されます。`Counter.add(x)` をループ本体の **Print** の後に、`Counter.show()` を `done` と **End** の間にドラッグします。呼び出しブロックをダブルクリックすると、メソッド、**オブジェクト**、引数を選べます。両方のブロックでオブジェクト名を `c` にして、同じオブジェクトを使います。

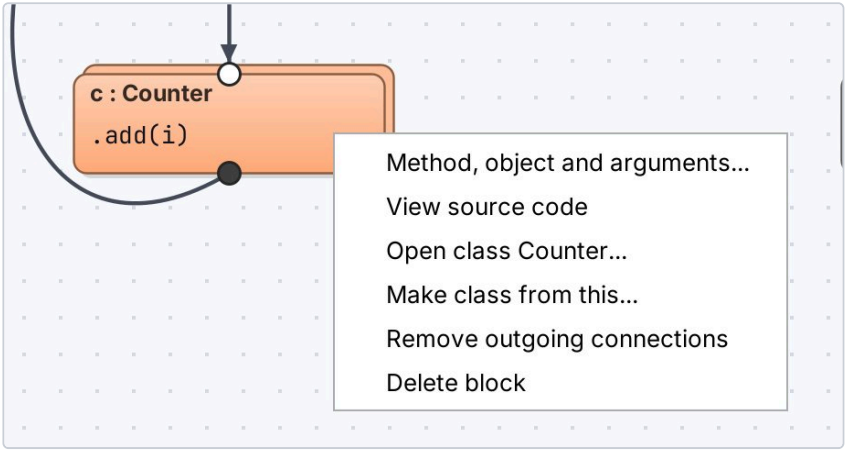


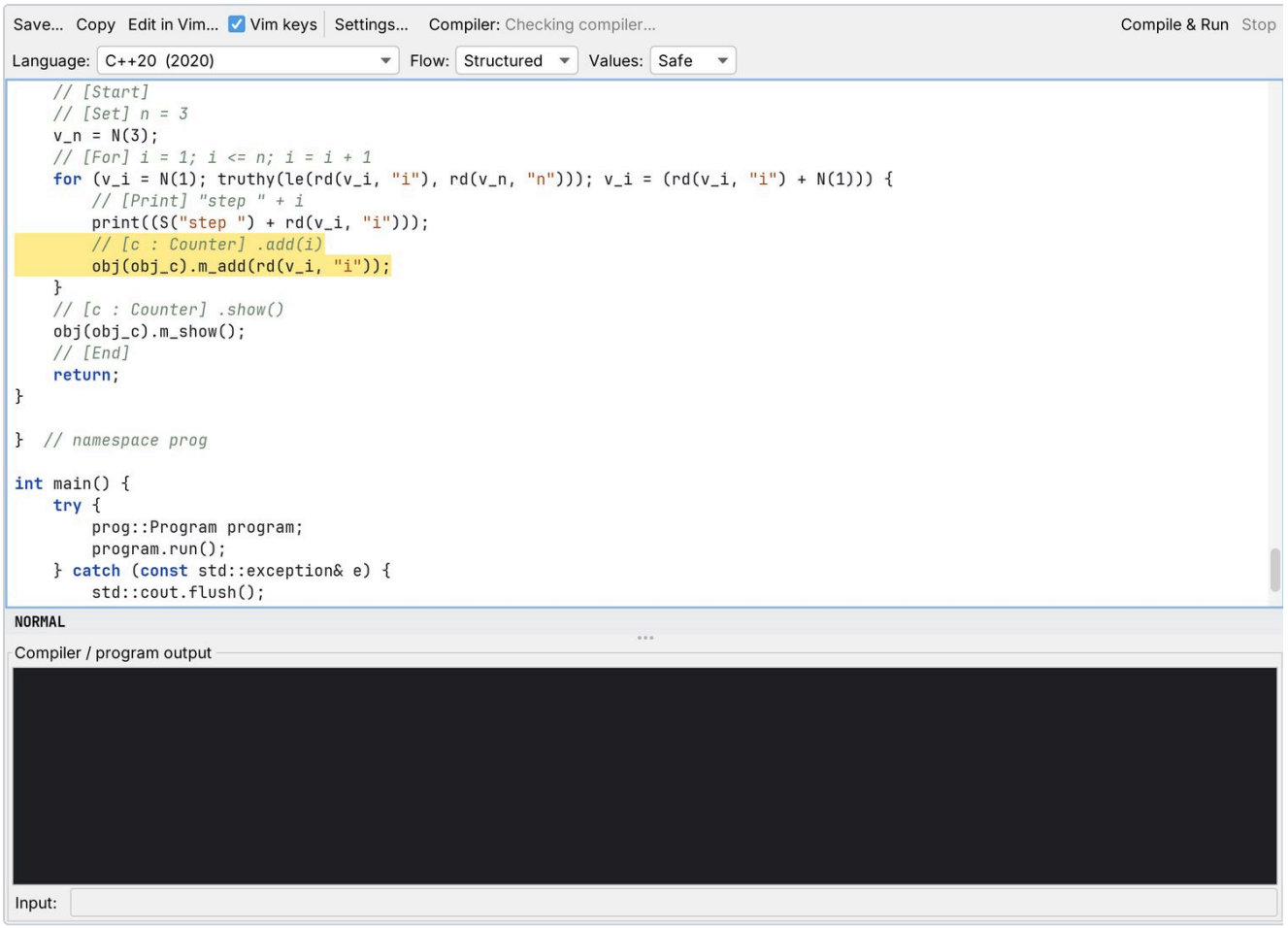
実行すると、最初の使用時に `c` が作成され（そのとき `new counter` が出力されます）、`add` が3回実行され、`show` が合計を出力します。



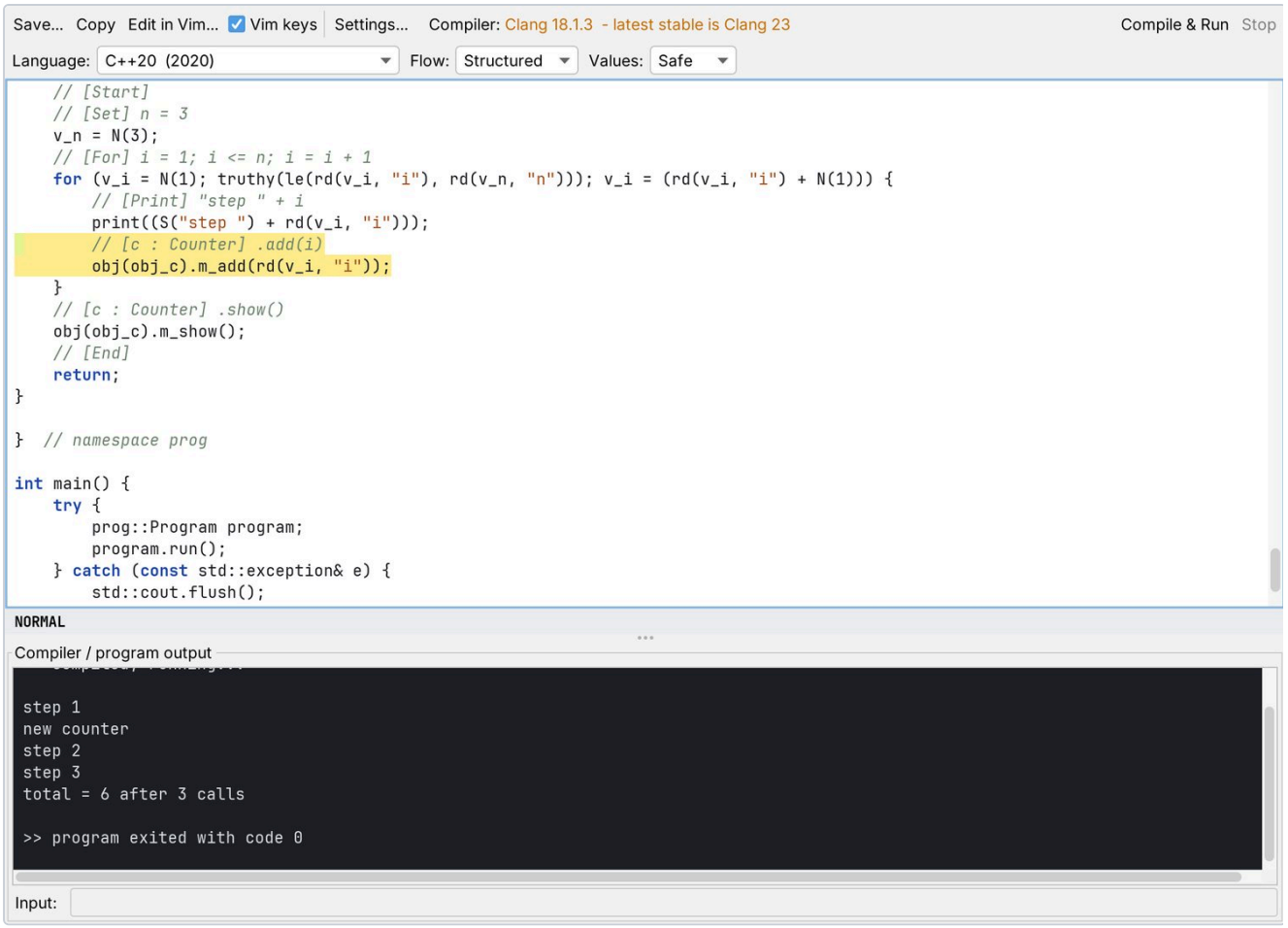
5. ブロックのコードを確認する

ブロックを右クリックして **View source code** を選びます。生成されたコードが開き、そのブロックに対応する行が強調表示されます（ここでは既定の C++20）。



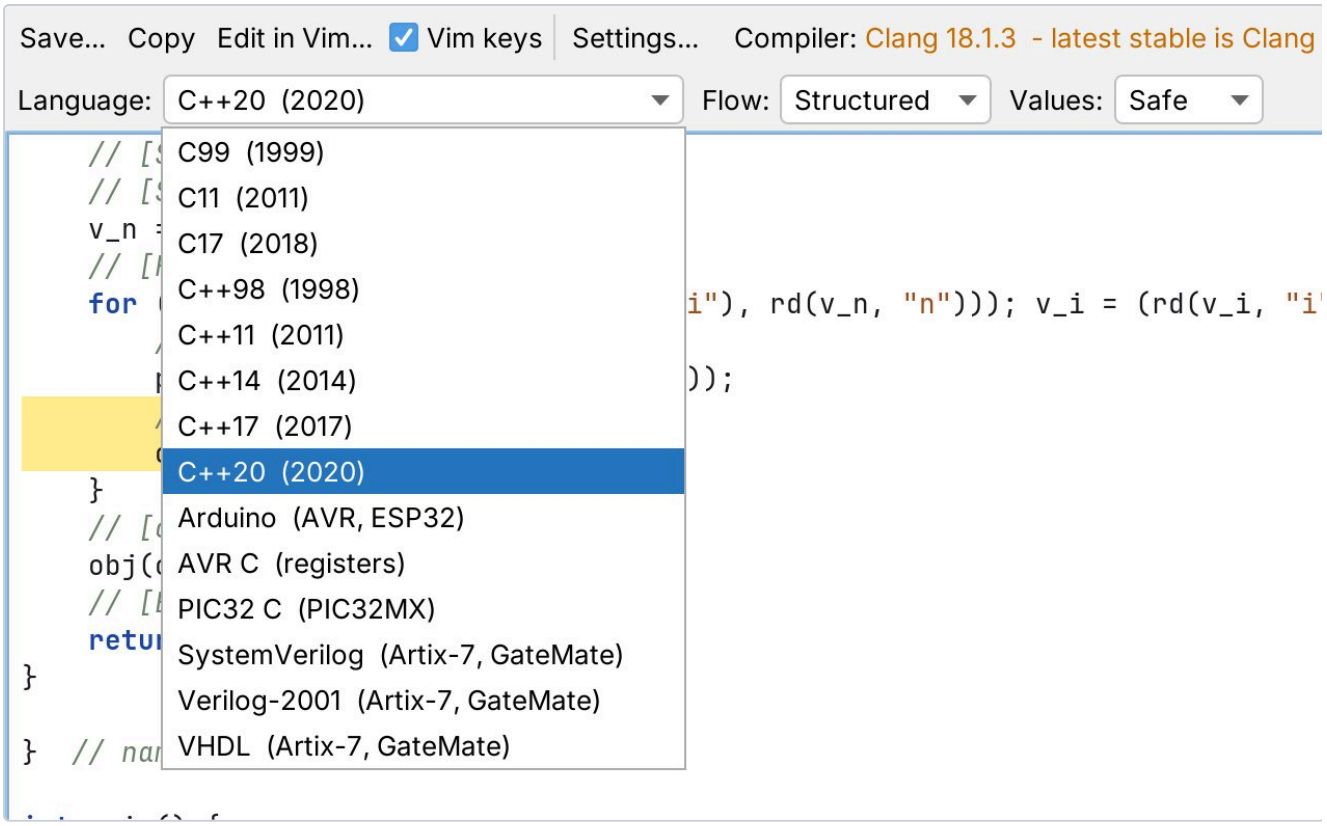


Compile & Run は Clang または GCC でコンパイルして実行します。コンパイル後の出力はインタープリターと同じです。Windows ではアプリが llvm-mingw（Clang）または WinLibs（GCC）をインストールするため、Visual Studio は不要です。

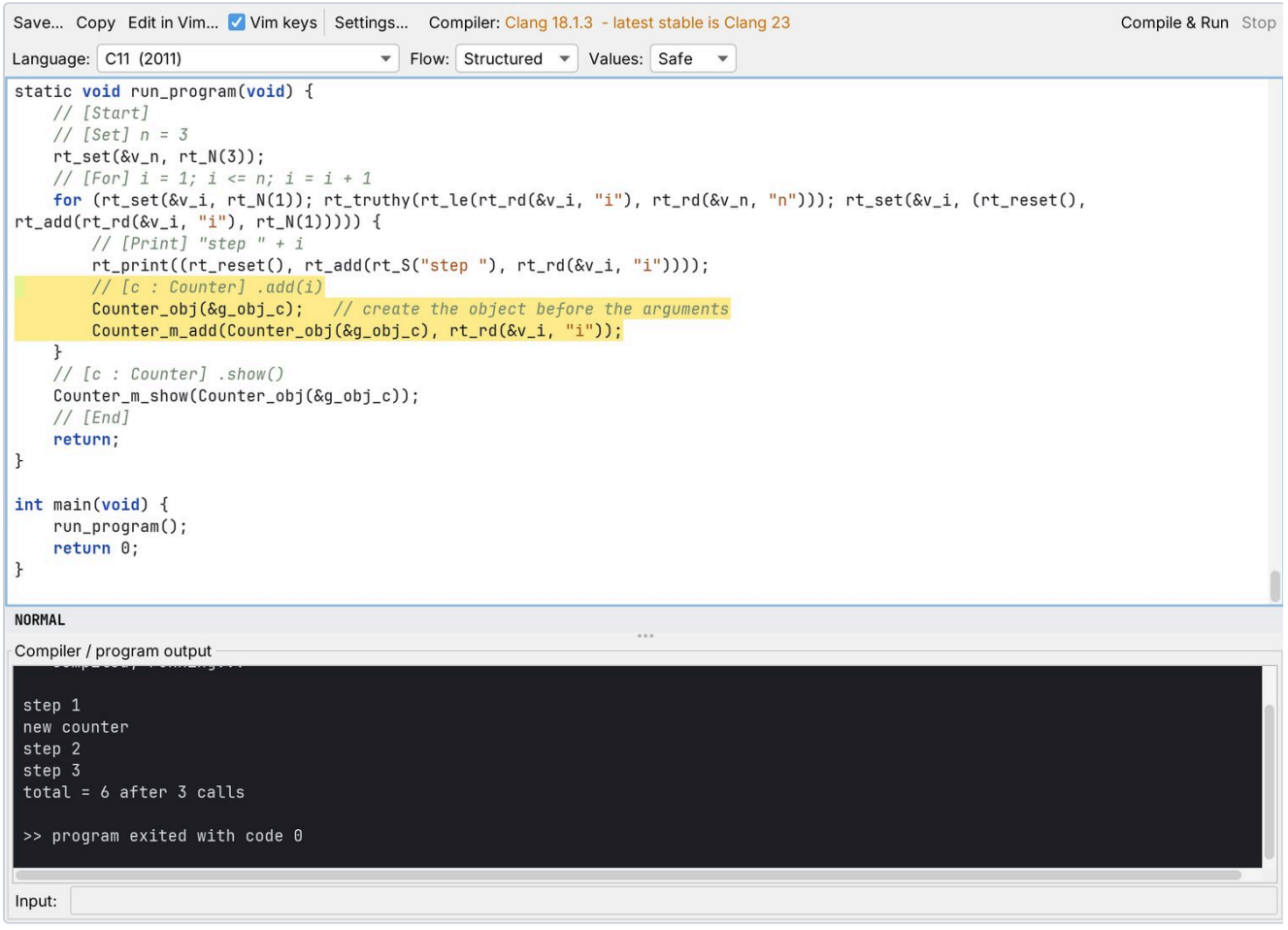


6. C または別の C++ 標準を選ぶ

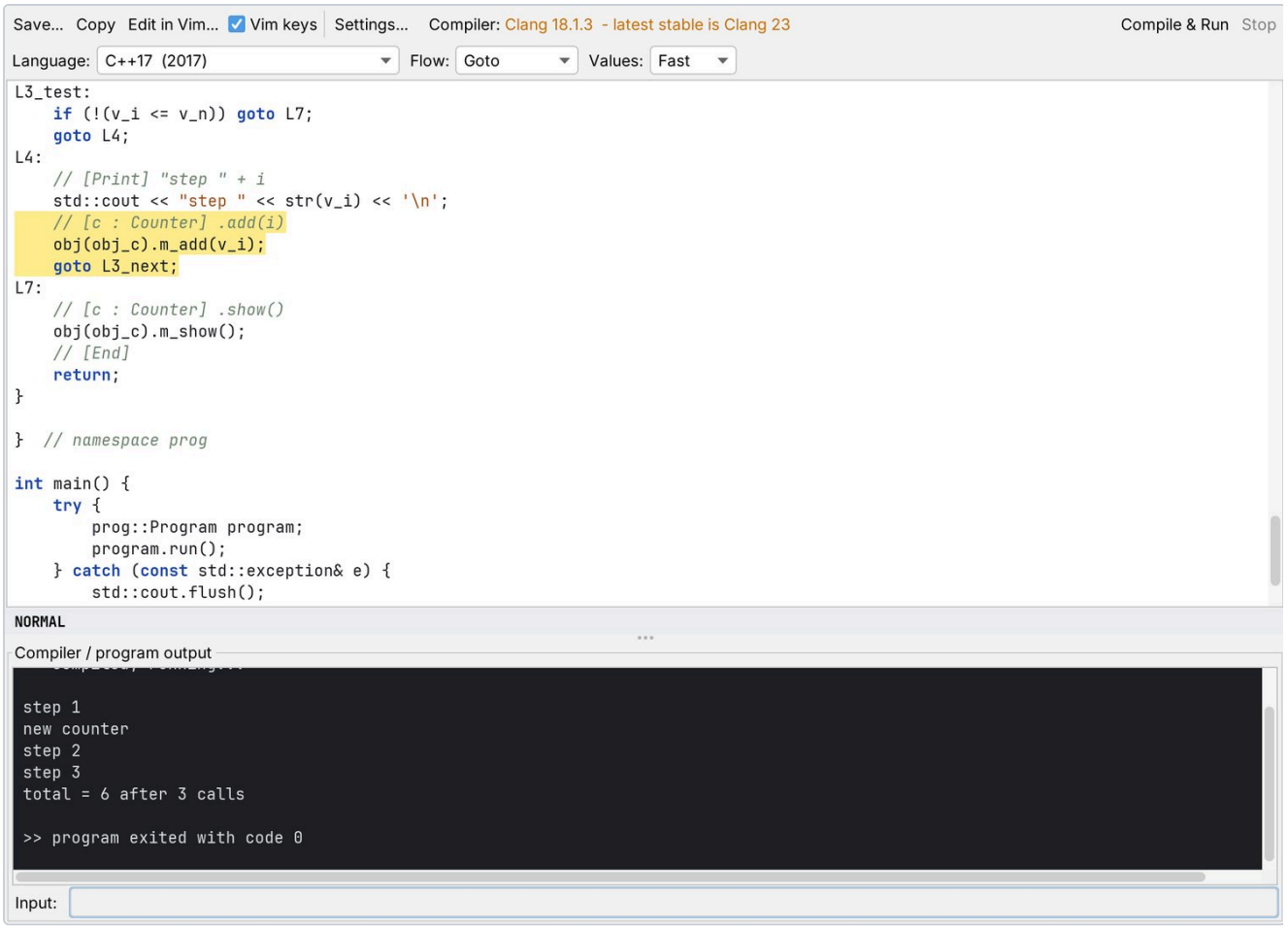
コードウィンドウまたはメインツールバーの **Language** ドロップダウンから言語を選びます。コードはすぐに再生成され、同じブロックが強調表示されたままになります。**Flow** と **Values** でコードの形式を選択します。**Structured** は if/else、ループ、switch を使い、**Goto** は各ブロックをラベル、各接続をジャンプにします。**Safe** はテキスト型とインタープリターのチェックを維持し、**Fast** は実行時チェックのない単純な `double` を使います。



同じプログラムを **C11** で生成し、`clang -std=c11` でコンパイルして実行しても、出力は同じです。

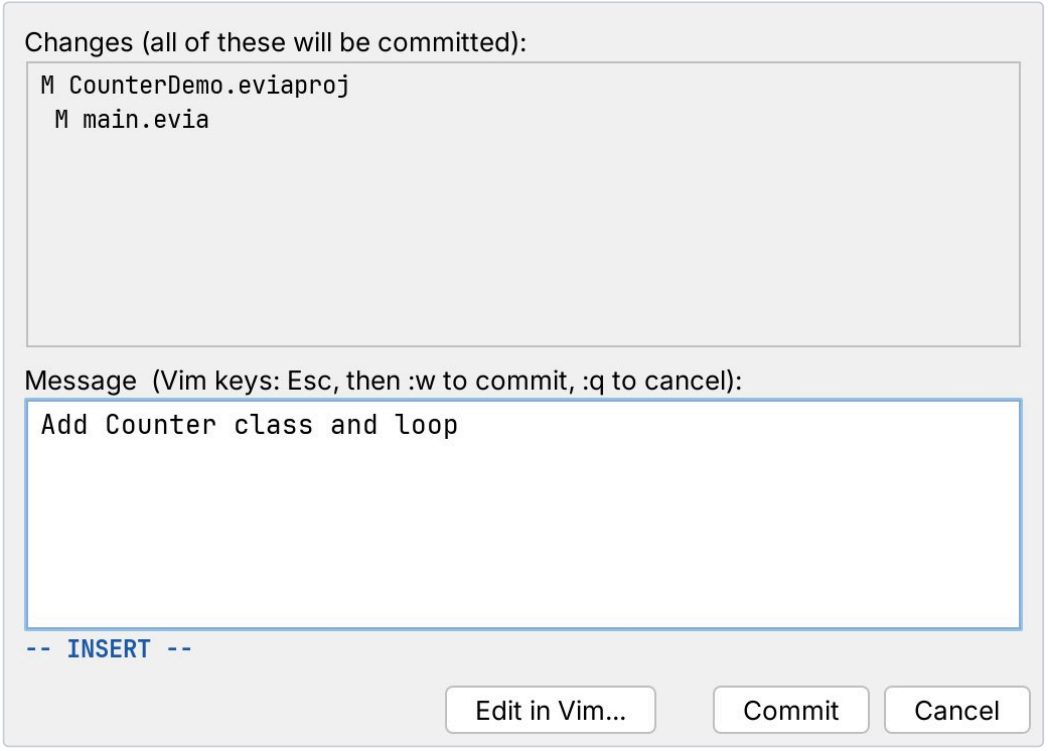


C++17 で **Goto** フローと **Fast** 値を選んだ例：

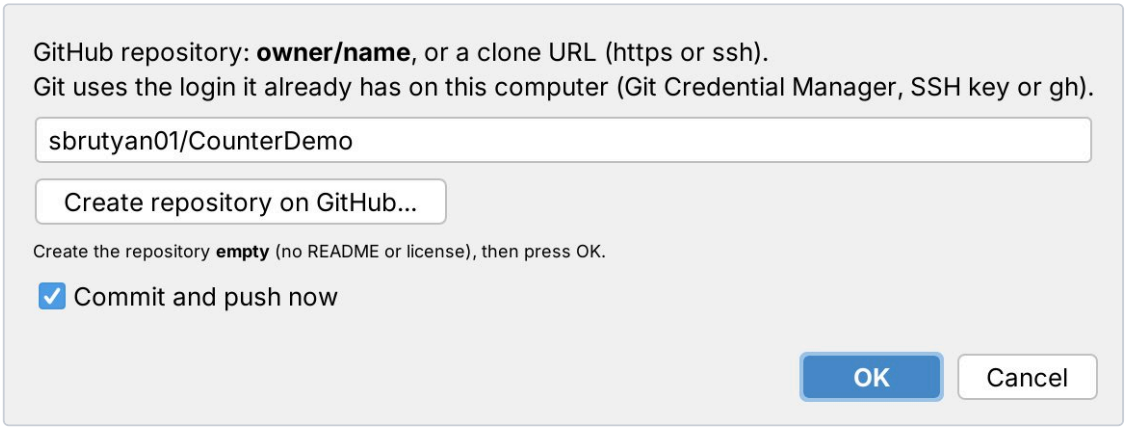


7. GitHub にコミットして公開する

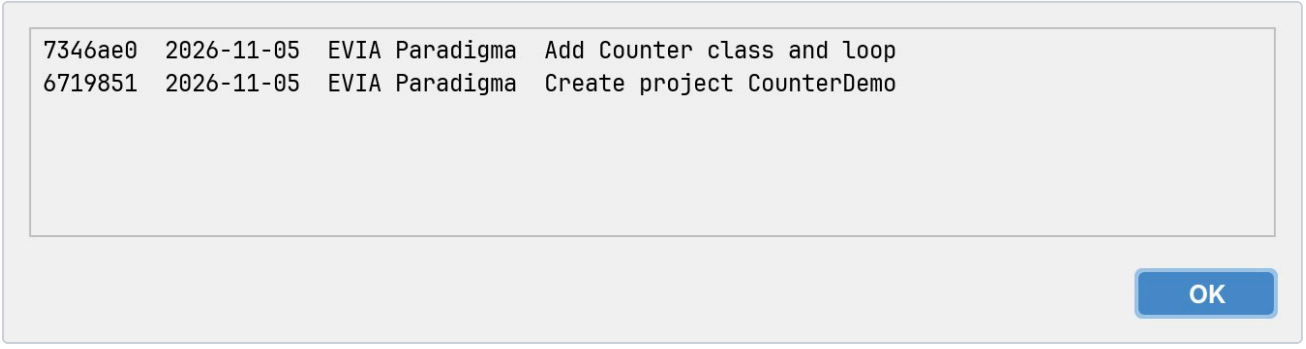
Git → Commit...（Ctrl+K）ですべての変更を保存してコミットします。メッセージ欄では Vim キーが使えます（`Esc` の後に `:w` でコミット）。



Git → Connect to GitHub... を選び、`owner/repo` を入力します。*Create repository on GitHub...* を選ぶと、名前が入力された状態で github.com が開きます。**空の**リポジトリを作成して **OK** を押すと、プロジェクトが push されます。Git はコンピューターで設定済みの認証情報（Git Credential Manager、SSH キー、または `gh auth login`）を使用します。

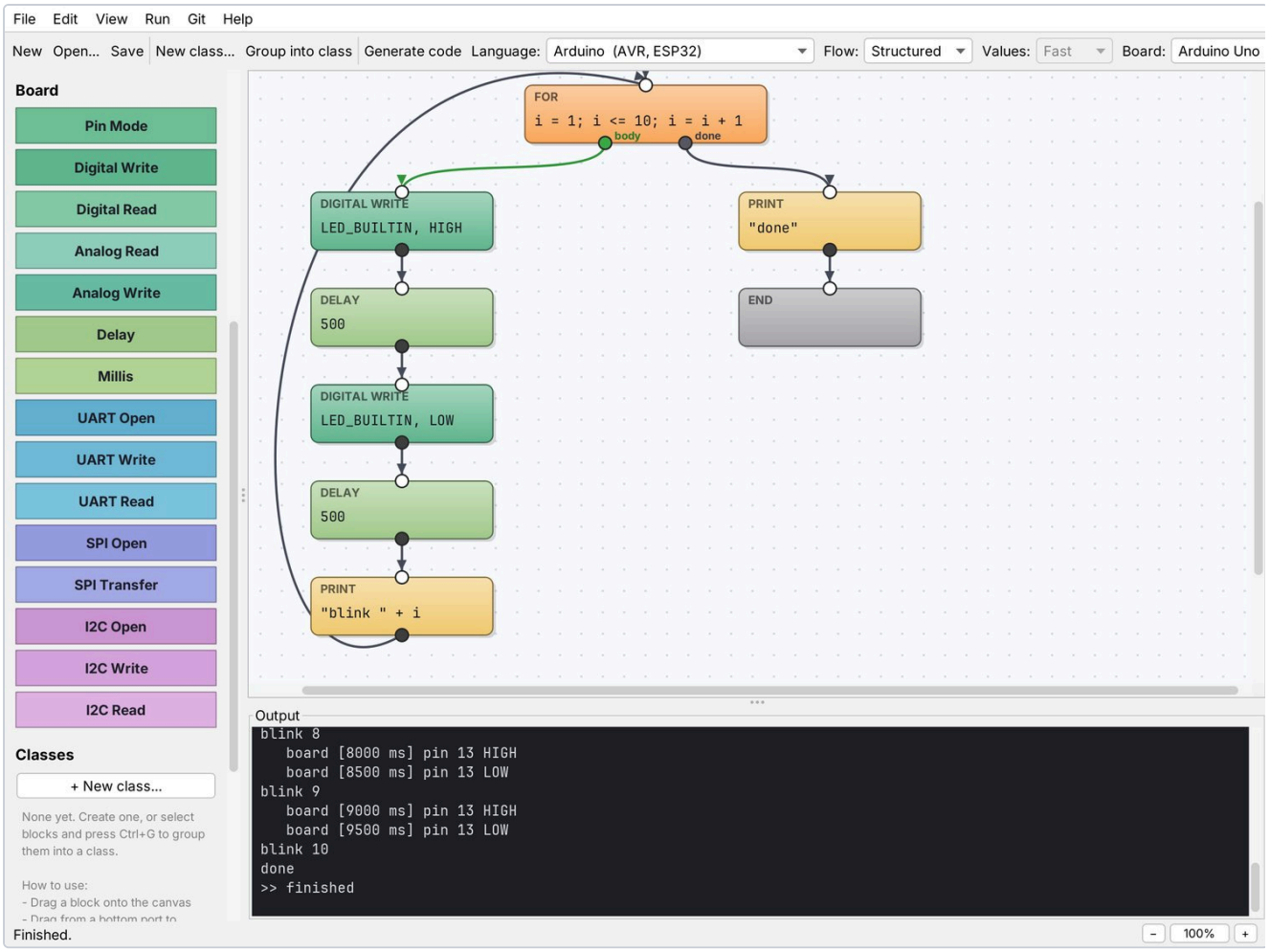


Git → History にはコミット履歴が表示されます。

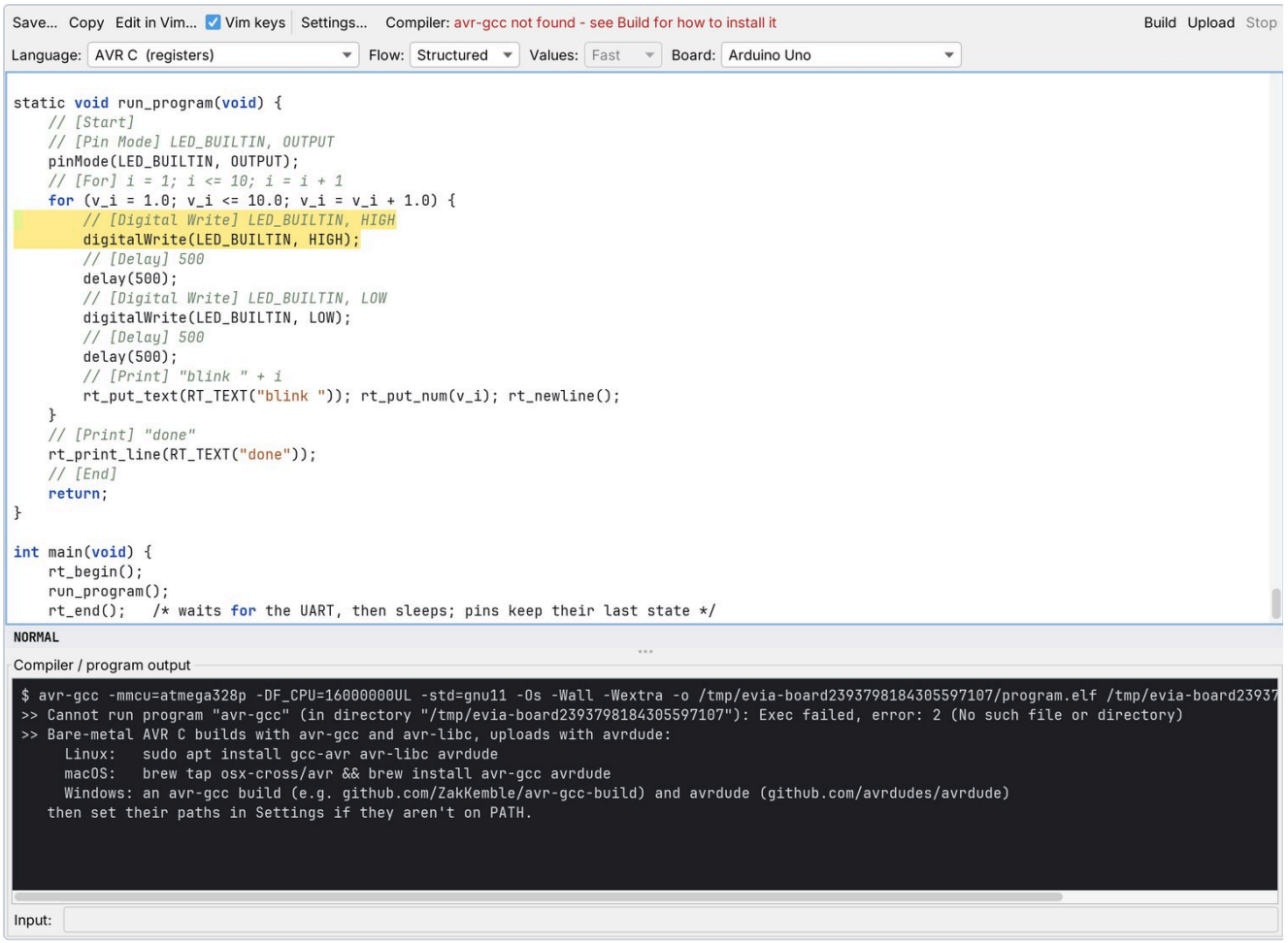


8. Arduino または ESP32 ボードをプログラムする

File → Open Example → Arduino and ESP32 Boards → Blink the built-in LED を開きます。パレットの *Board* グループにある緑色の **Board** ブロックで、ピンの設定、ボタンやセンサーの読み取り、待機を行います。**Run**（F5）は Arduino Uno をシミュレートし、出力に各ピンの変化と時刻を表示します。



Language: Arduino と **Board** を選び、**Generate code**（F7）を押します。生成されるスケッチは12種類のボードに対応します。**Build** は arduino-cli でコンパイルし、**Upload** はボードへ書き込みます。Print と Input は 115200 baud の Serial Monitor を使います。**AVR C** は Arduino コアを使わず、チップのレジスターを直接操作する同じプログラムを生成し、avr-gcc でビルドします。



9. UART、SPI、I2C、PIC32 で他のチップと通信する

File → Open Example → Arduino and ESP32 Boards にある3つの例では、バスブロックを使います。*UART: send a byte, read it back*、*SPI: transfer a byte*、*I2C: write and read a device register* です。使用前にバスを開きます。**Run** では、UART の送信バイトは読み取り側に戻り、SPI は入力と出力が接続され、I2C はアドレス80に256レジスターのデバイスがあるものとして動作します（80は0x50、数値は10進数）。

ブロック	コード	動作
UART Open	`9600`	指定した baud rate でシリアルポートを開始
UART Write	`65`	1バイト（0..255）を送信
UART Read	`c`	次の受信バイト。なければ -1
SPI Open	`1000000`	指定周波数（Hz）の SPI マスターを開始（モード0、MSB先行）
SPI Transfer	`reply = 66`	1バイトを送信し、同時に受信したバイトを保存
I2C Open	`100000`	指定周波数（Hz）の I2C マスターを開始
I2C Write	`80, 0, 42`	アドレス（7ビット）、レジスター、値
I2C Read	`v = 80, 0`	レジスター値を保存。応答がなければ -1

Microchip PIC32MX チップでは、**Language: PIC32 C** と **Board** リストのチップ（MX1xx/2xx、MX3xx/4xx、MX5xx/6xx/7xx 系の9種類）を選び、**Generate code**（F7）を押します。生成されるのは無料の Microchip XC32 コンパイラー用ベアメタル C です。クロック、ピン、PWM、ADC、UART、SPI、I2C をレジスターで設定します。**Build** は `xc32-gcc` と `xc32-bin2hex` を実行し、**Upload** は **Settings** のコマンド（既定は `pic32prog {hex}`）を実行します。選択したチップファミリーのピンは生成ファイルの冒頭に記載されています。

注意： PIC32 コードは Microchip の実際のデバイスヘッダーに対して確認済みですが、XC32 でのビルドや実機での実行は未確認です。ボードでの最初のビルドはテストとして扱ってください。

10. FPGA 用のハードウェアを作る

Language: SystemVerilog、**Verilog-2001**、または **VHDL** を選び、**Board** から Digilent Nexys A7-100T、Arty A7-35 または A7-100、Basys 3（Artix-7）、Olimex GateMateA1-EVB、Cologne Chip GateMate A1 evaluation board を選びます。**File → Open Example → FPGA Boards → Blink** を開いて **Generate code**（F7）を押します。

各ブロックはステートマシンの1状態になり、1クロックサイクルを使います。設計のビルド時にピンは固定されるため、`LED0`、`SW3`、`BTNC`、`JA1` のような名前で指定します。ピン制約はファイル先頭のコメントにあります。コマンドラインで `-o` を指定すると `.xdc`（Vivado）または `.ccf`（GateMate）としても出力されます。

- Artix-7 の **Build** と **Upload** は Vivado またはオープンソースのフローを使用します（**Settings** で選択）。GateMate では yosys、nextpnr-himbaechel、gmpack、openFPGALoader を使います。
- 数値は32ビット整数です。そのため FPGA での `5 / 2` は2ですが、**Run** では2.5です。
- クラス、Input、Analog Read、SPI、I2C、変数内のテキストは未対応です。エラーに該当ブロックが表示されます。

注意： 出力はオープンソースツールでシミュレーションと合成を行っていますが、Vivado での処理や実機への書き込みは行っていません。

11. MQTT で数値を送り出す

File → Open Example → Arduino and ESP32 Boards → MQTT sensor を開きます。温度をブローカーに公開し、自分宛てに コマンドを送って読み戻すプログラムです。

MQTT のブロックは *Board* グループに5つあります。*MQTT Connect*、*MQTT Publish*、*MQTT Subscribe*、 *MQTT Has Message*、*MQTT Read* です。ネットワーク名、パスワード、ブローカー、トピックは固定のテキストです。 ビルド時にプログラムへ書き込まれるからです。

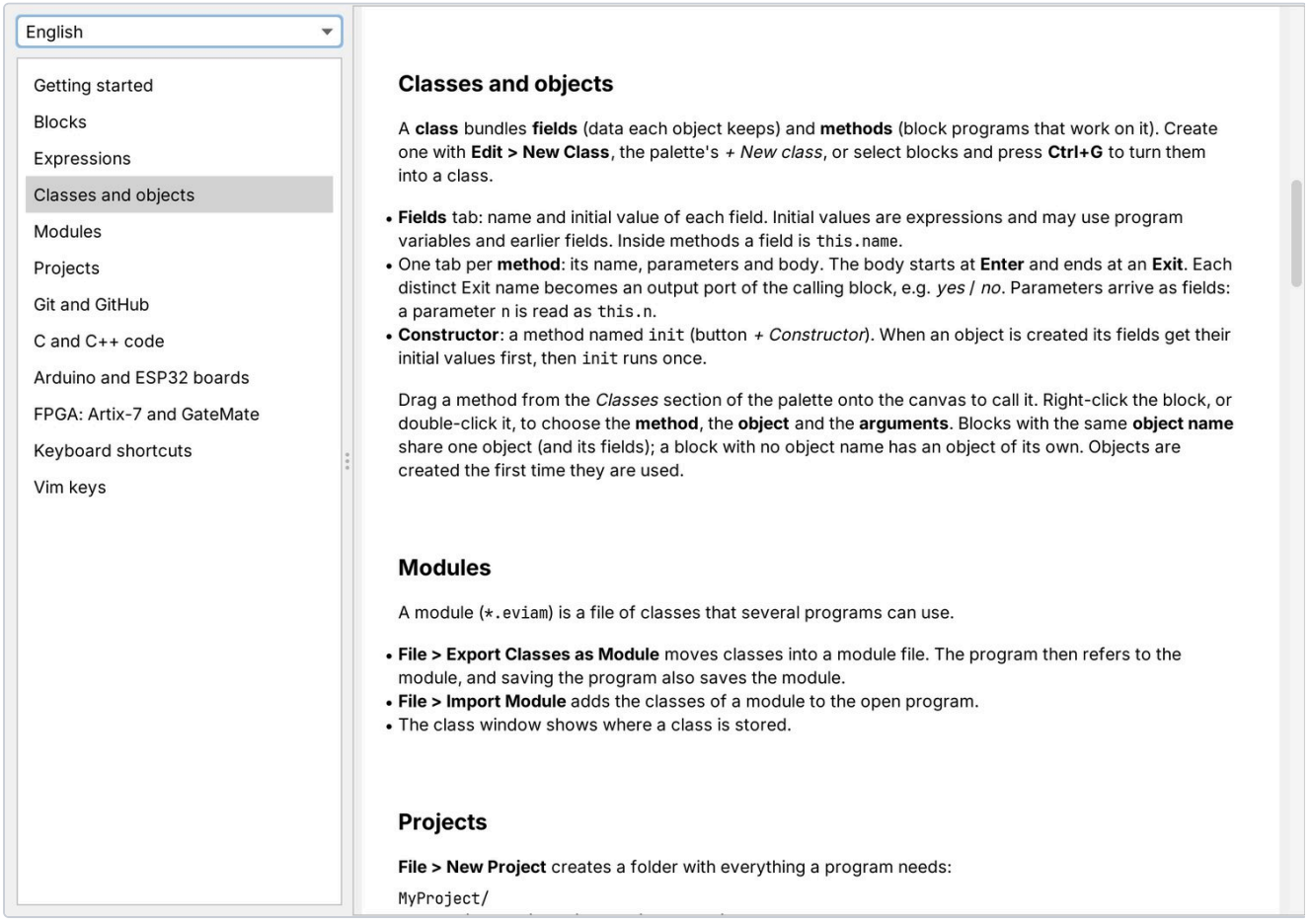
- **Run** は自分のパソコン上のブローカーが応答するので、機器がなくても試せます。
- **PIC32 C** は AT ファームウェアの ESP8266／ESP32 モジュールを UART2 で動かし、Print は UART1 のままです。
- **ESP32** ボードの **Arduino** はボード自身の WiFi を使います。
- AVR ボードはこれらのブロックを拒否します。シリアルポートが1つしかなく、Print が使っているからです。

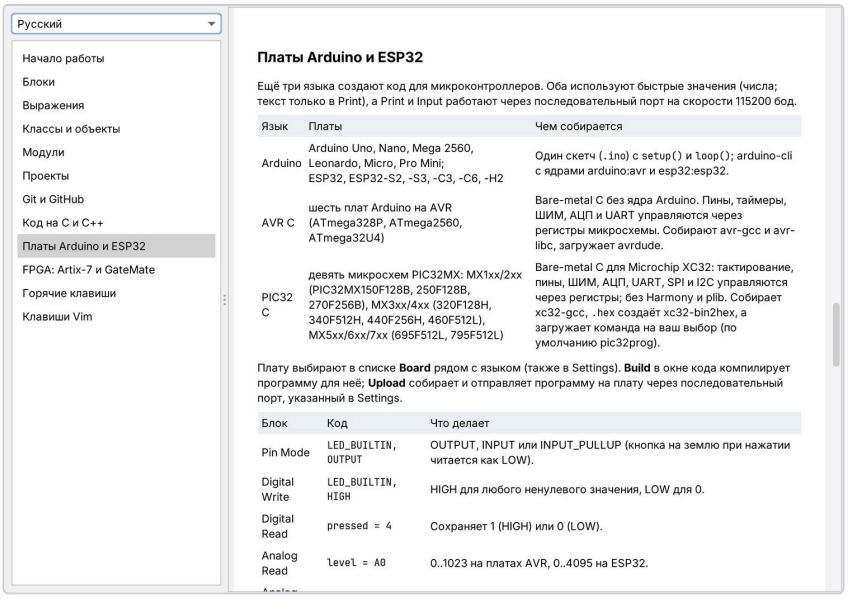
メッセージは数値です。何も届いていないとき *MQTT Read* は -1 を返すので、本物の -1 があり得る場合は先に *MQTT Has Message* で確かめてください。

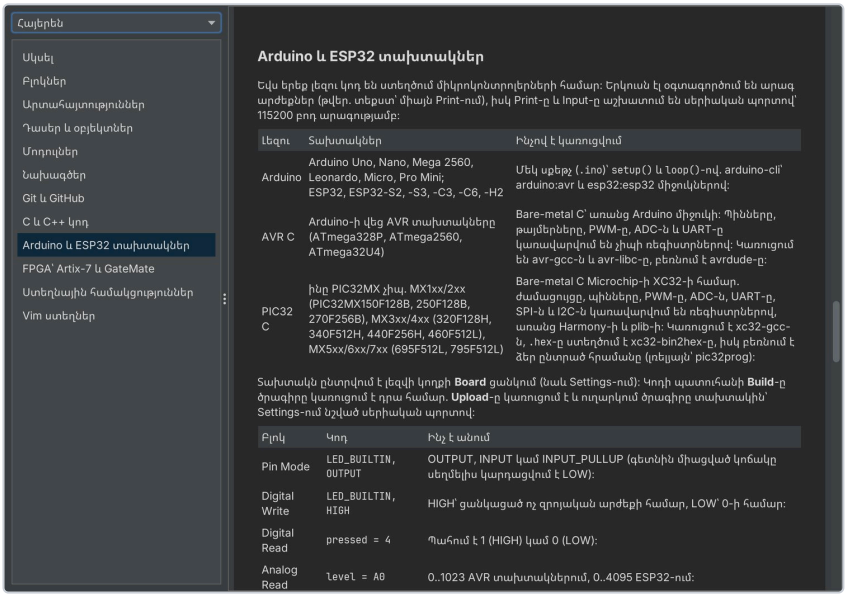
注意: MQTT のパケットは CI で実際のブローカーに対して検査していますが、WiFi モジュール、ESP32、PIC32 チップは 含まれていません。MX695/795 の Ethernet ポート経由の MQTT はまだ対応していません。

12. さらに使いこなす

- クラスを複数のプログラムで共有するには、**File → Export Classes as Module...** を選び、別のプログラムで **File → Import Module...** を選びます。
- **File → Open Example** には、フィールド、コンストラクター、メソッド、モジュールを含む `classes.evia` などのサンプルがあります。
- **Help → Tutorial (PDF)** から英語、ロシア語、アルメニア語、ドイツ語、簡体字中国語、繁体字中国語、日本語のチュートリアルを開けます。
- **Help → Guide**（F1）では、ブロック、式、クラス、プロジェクト、Git、ショートカット、Vim キーを英語、ロシア語、アルメニア語で説明します。**View → Theme** でライト／ダークテーマを切り替えます。




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: