EVIA Paradigma: учебник

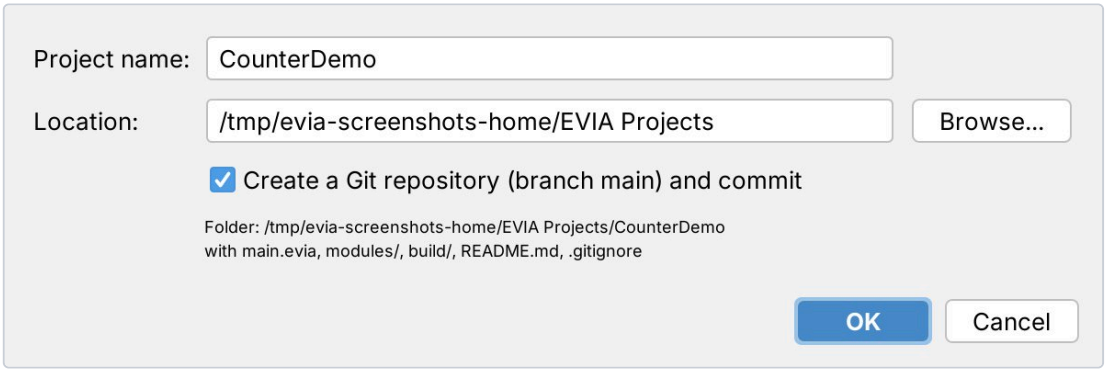
В этом учебнике мы соберём небольшой проект с циклом и классом, запустим его, посмотрим, в какой код на C и C++ он превращается, и поместим его в Git. В конце мы запрограммируем плату Arduino или ESP32. Все снимки экрана сделаны в настоящем приложении. Интерфейс программы на английском, поэтому названия меню и кнопок даны так, как они выглядят на экране.

Начиная с версии 0.4.1 программа поддерживает **тёмную тему** (**View → Theme**) и выглядит как IntelliJ IDEA (шрифты Inter и JetBrains Mono); руководство **Help → Guide** можно читать по-русски и по-армянски. Скриншоты в этом учебнике сделаны в светлой теме.

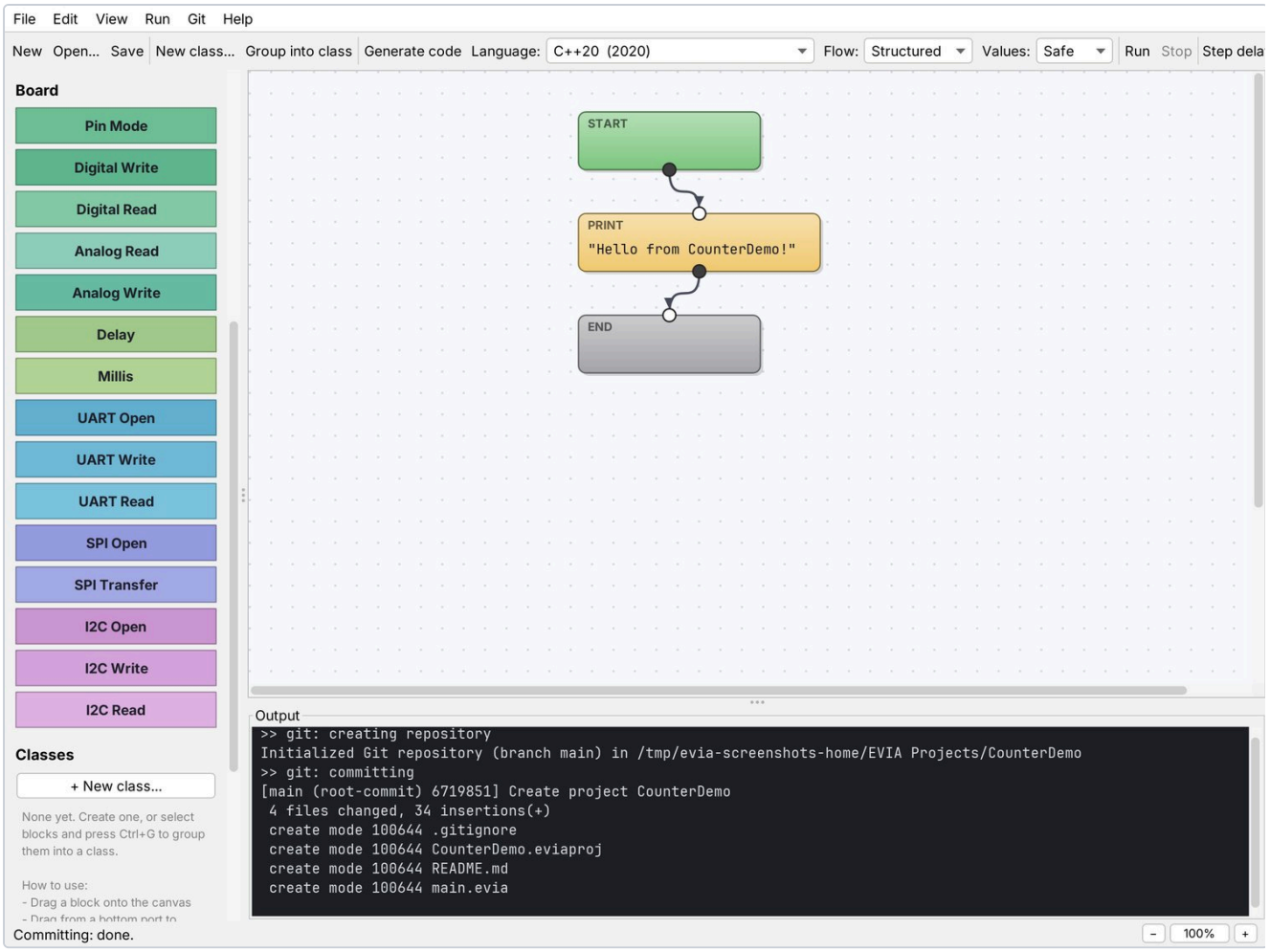
Скачать EVIA Paradigma: последний выпуск (установщики для Windows, macOS и Linux уже содержат Java).

1. Создайте проект

File → New Project... (Ctrl+Shift+N). Введите имя и папку; если git установлен, оставьте отмеченным *Create a Git repository*.



Проект откроется с начальной программой *Start → Print → End*. Панель вывода показывает, что git создал репозиторий и первый коммит.

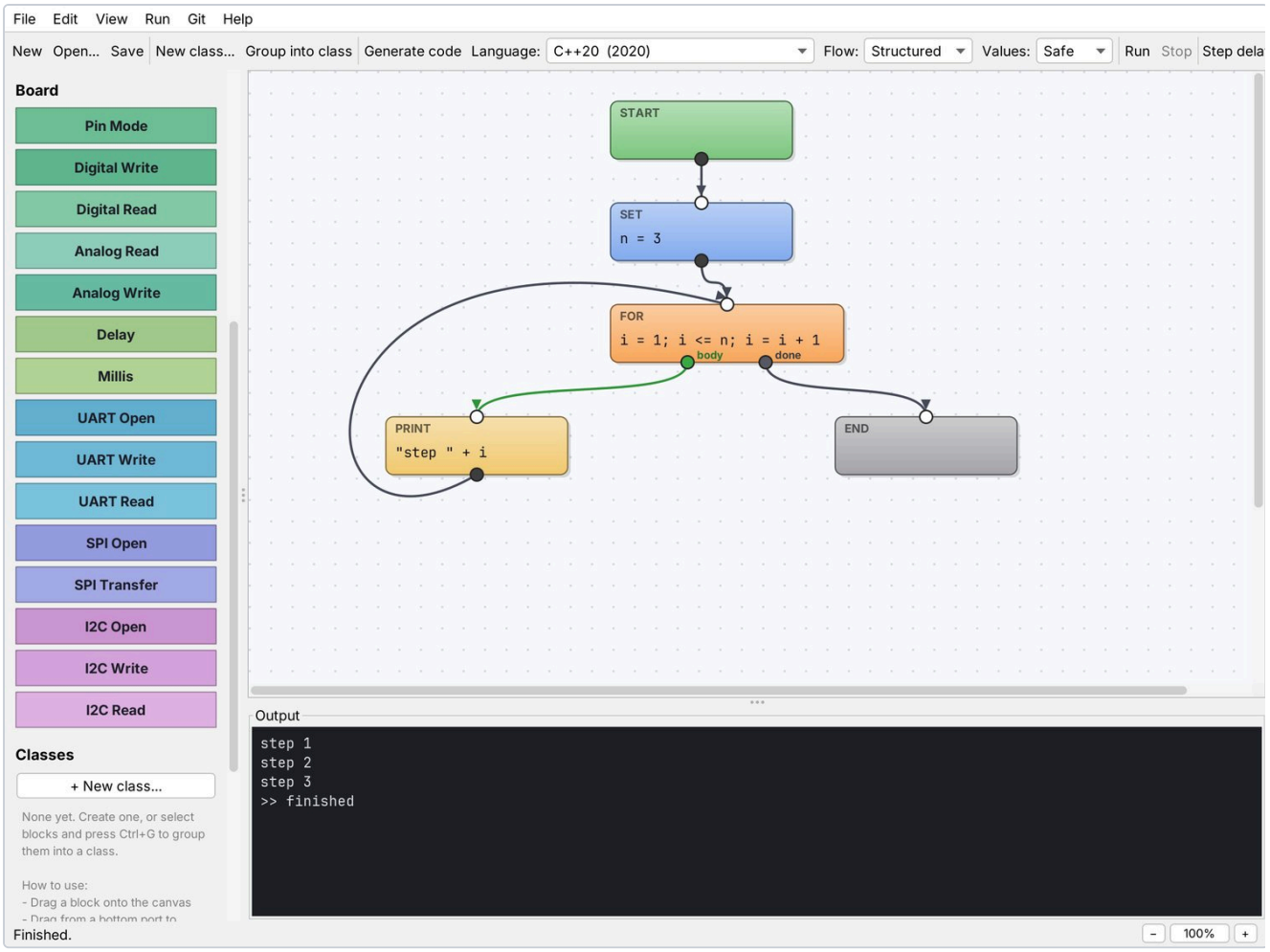


2. Соберите цикл

Перетаскивайте блоки из палитры на холст и соединяйте их: тяните от нижнего порта блока (тёмная точка) к верхнему порту следующего блока (белая точка). Двойной щелчок по блоку открывает его код для редактирования.

- **Set**: `n = 3`
- **For**: `i = 1; i ⩽ n; i = i + 1`. Его порт `body` ведёт к **Print** `"step " + i`; конец тела цикла соедините **обратно с блоком For**, чтобы начался следующий проход. Порт `done` ведёт к **End**.

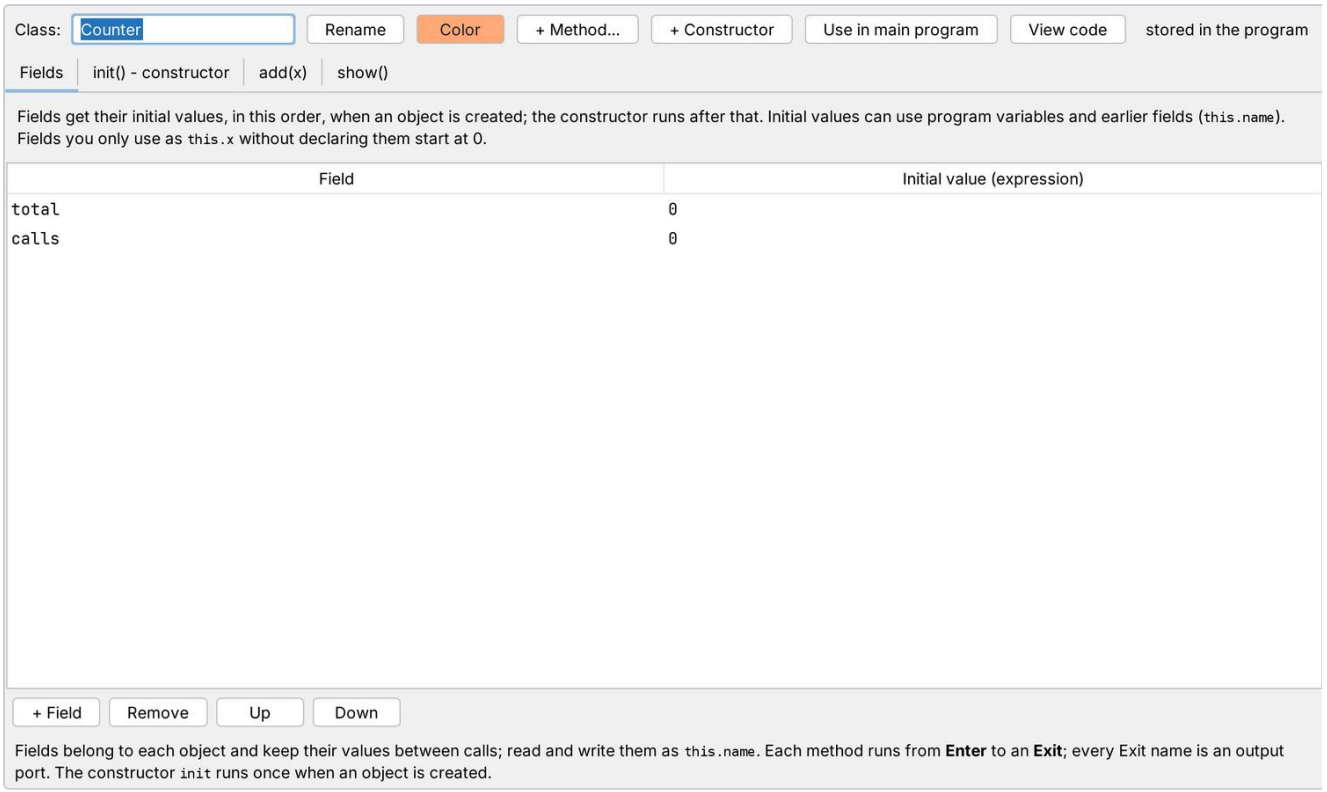
Нажмите **Run** (F5). Задайте *Step delay*, чтобы видеть, как перемещается активный блок. Колесо мыши над схемой меняет масштаб.



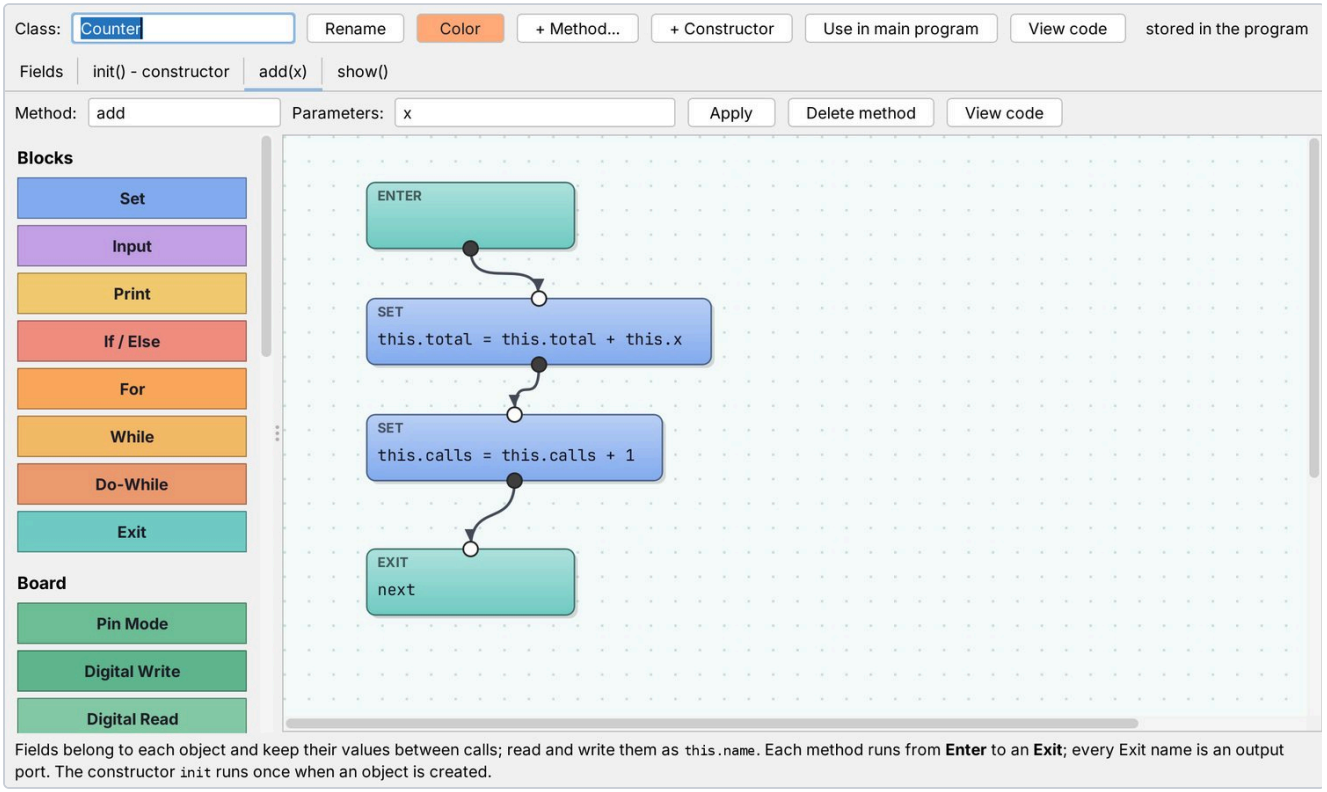
3. Создайте класс

Edit → New Class... (или *+ New class...* в палитре), назовите его `Counter`. В окне класса есть вкладка **Fields** и по одной вкладке на каждый метод.

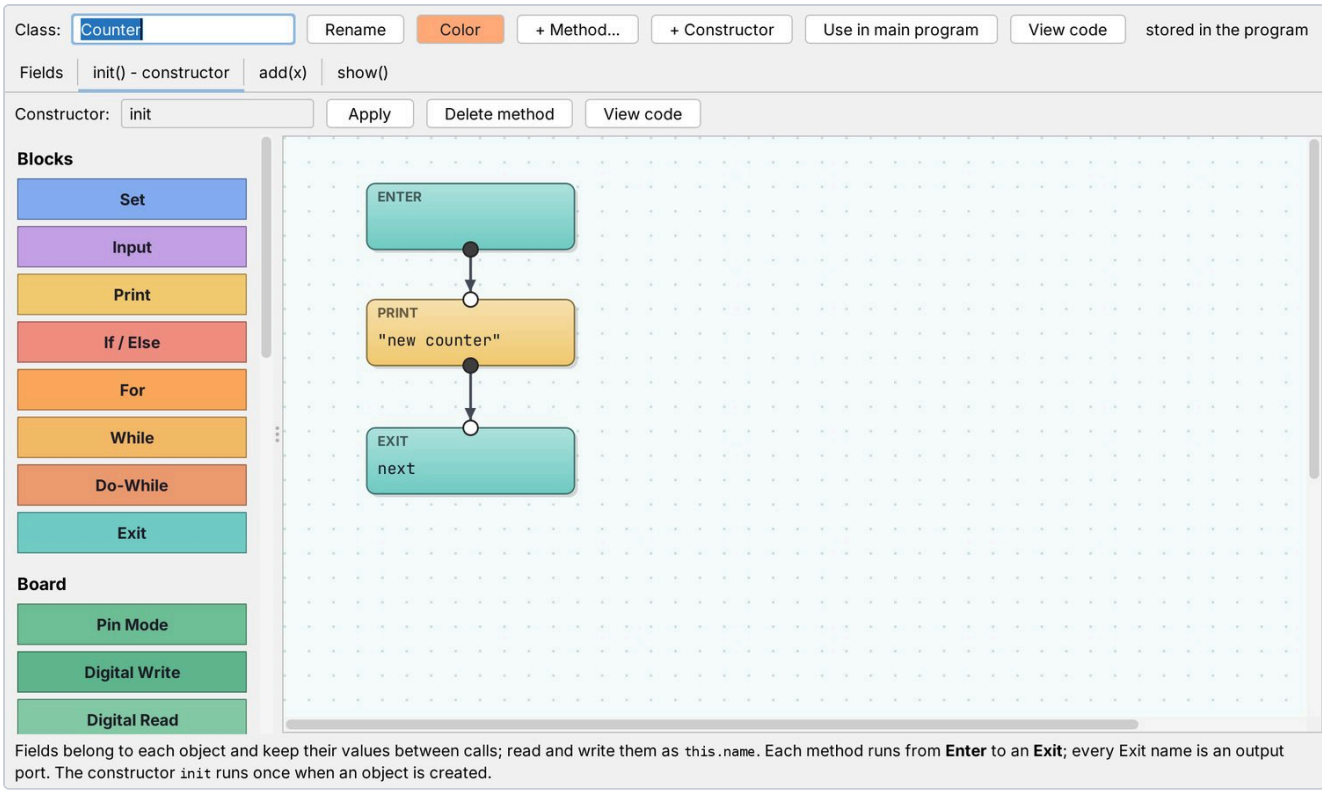
Fields: добавьте `total` и `calls`, оба с начальным значением `0`. У каждого объекта свои копии полей.



Методы: переименуйте первый метод в `add` с параметром `x` (нажмите **Apply**), затем соберите его тело от **Enter** до **Exit**. Внутри метода поля и параметры записываются как `this.имя`:

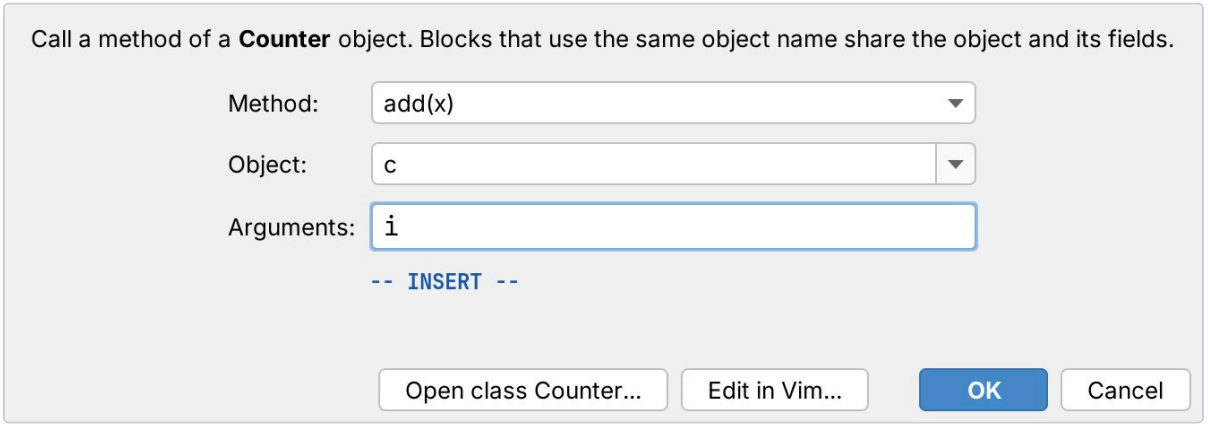


+ Constructor добавляет `init` — он выполняется один раз при создании объекта (после того как поля получили начальные значения). Здесь он печатает `"new counter"`. **+ Method...** добавляет `show`, который печатает `"total = " + this.total + " after " + this.calls + " calls"`.

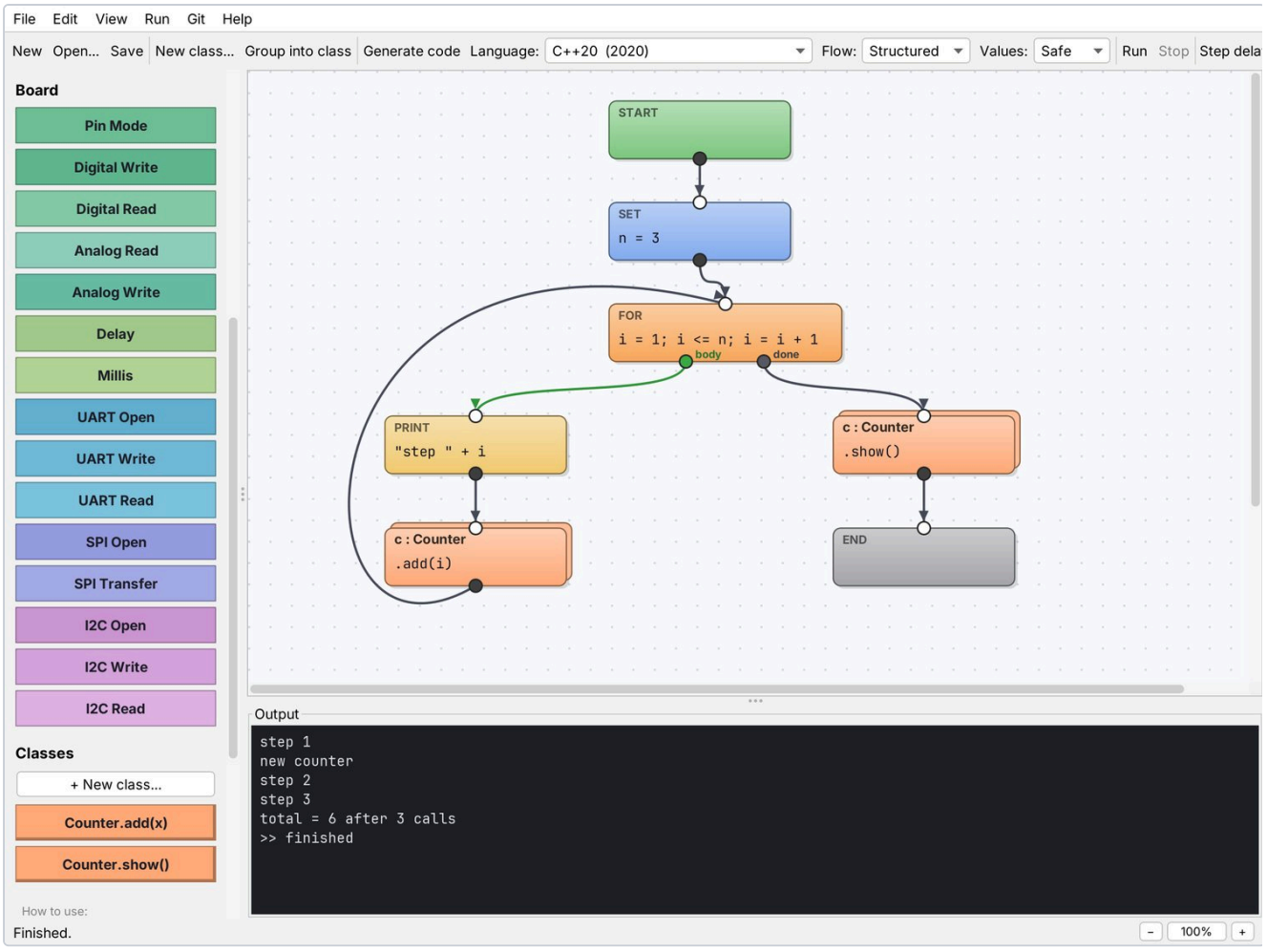


4. Используйте класс

Теперь в палитре есть `Counter.add(x)` и `Counter.show()`. Перетащите `Counter.add(x)` в тело цикла после **Print**, а `Counter.show()` — между `done` и **End**. Двойной щелчок по блоку вызова позволяет выбрать метод, **объект** и аргументы. Дайте обоим блокам имя объекта `c`, чтобы они работали с одним объектом:

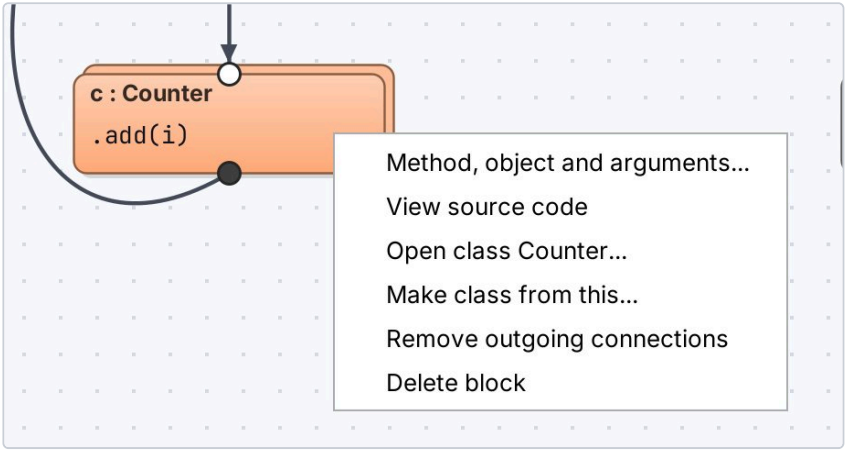


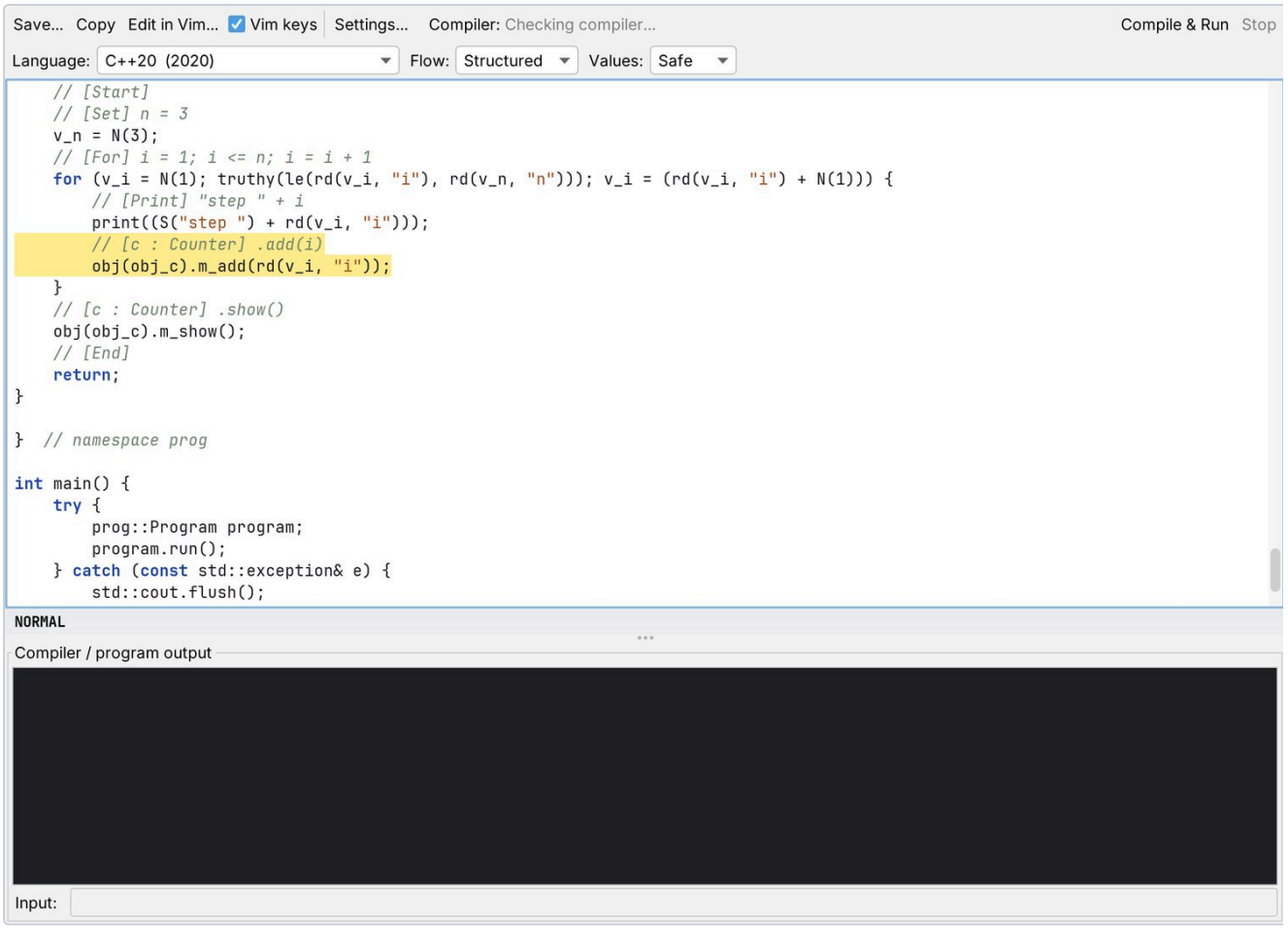
Запустите программу. Объект `c` создаётся при первом использовании (тогда и печатается `new counter`), `add` выполняется три раза, а `show` печатает итог:



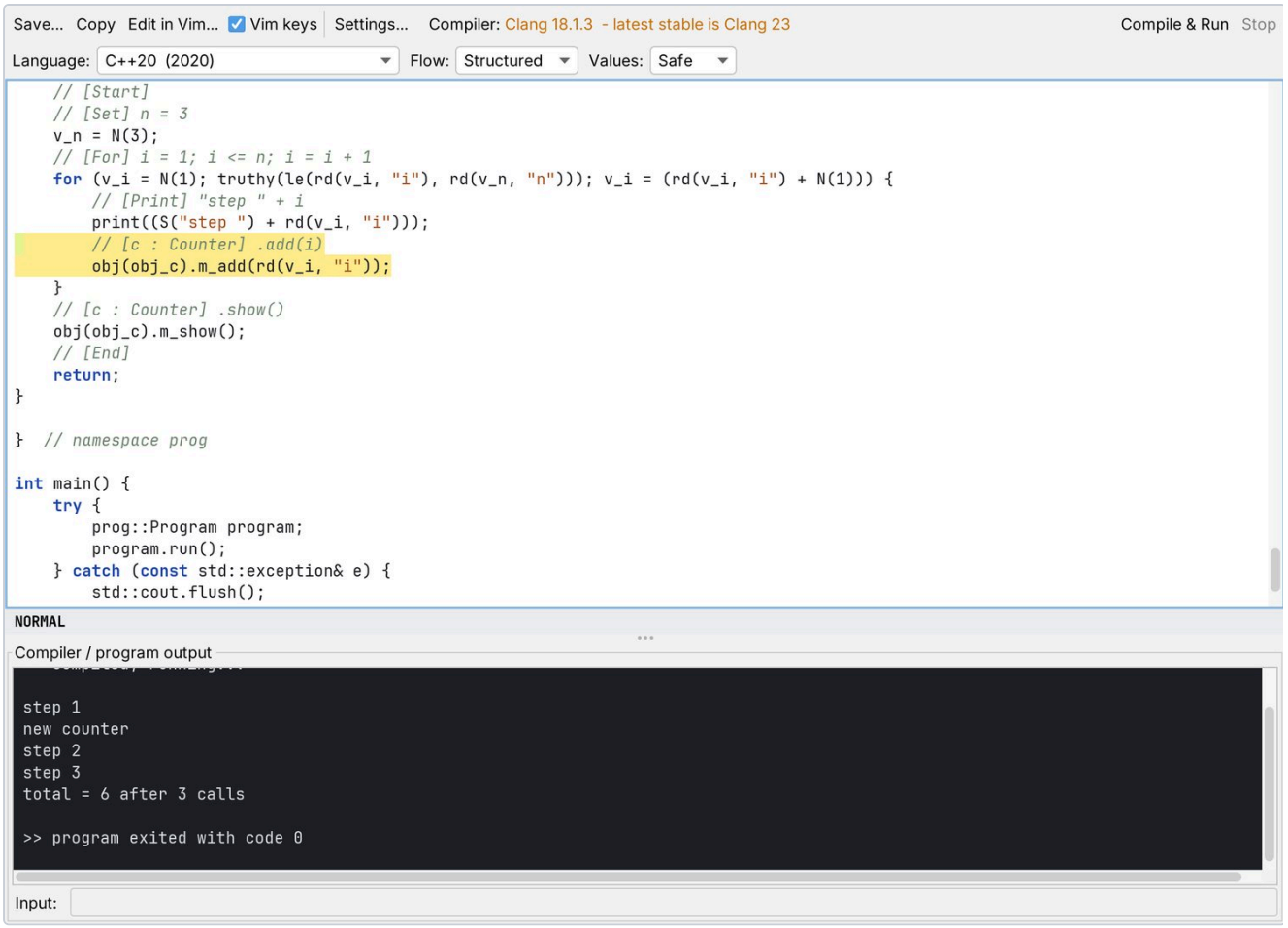
5. Посмотрите код любого блока

Правый щелчок по блоку → **View source code**. Откроется сгенерированный код, в котором подсвечены именно строки этого блока (здесь C++20, язык по умолчанию):



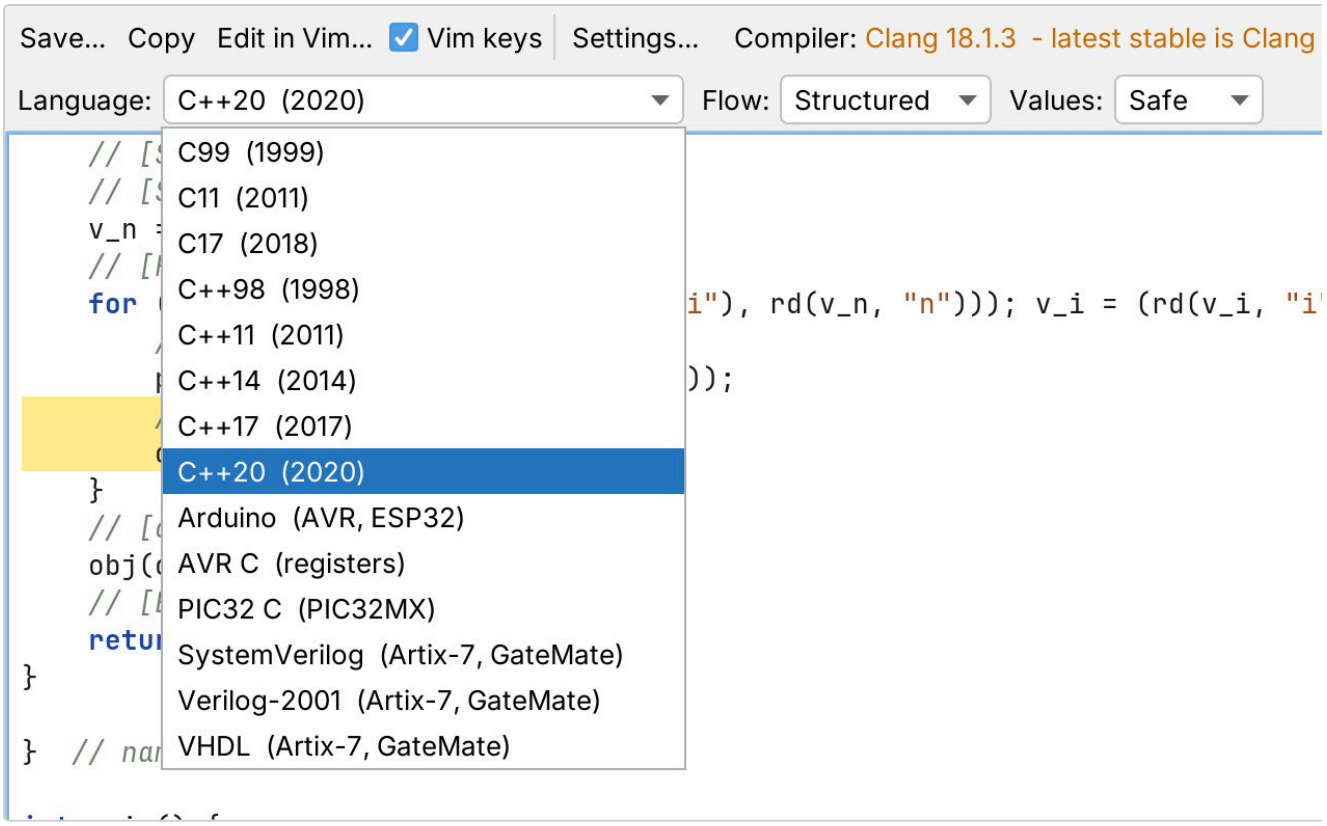


Compile & Run собирает код с помощью Clang или GCC и запускает его. Скомпилированная программа печатает то же, что и интерпретатор. В Windows программа сама устанавливает компилятор: llvm-mingw (Clang) или WinLibs (GCC), Visual Studio не нужна. Выбор компилятора — в **Settings**.

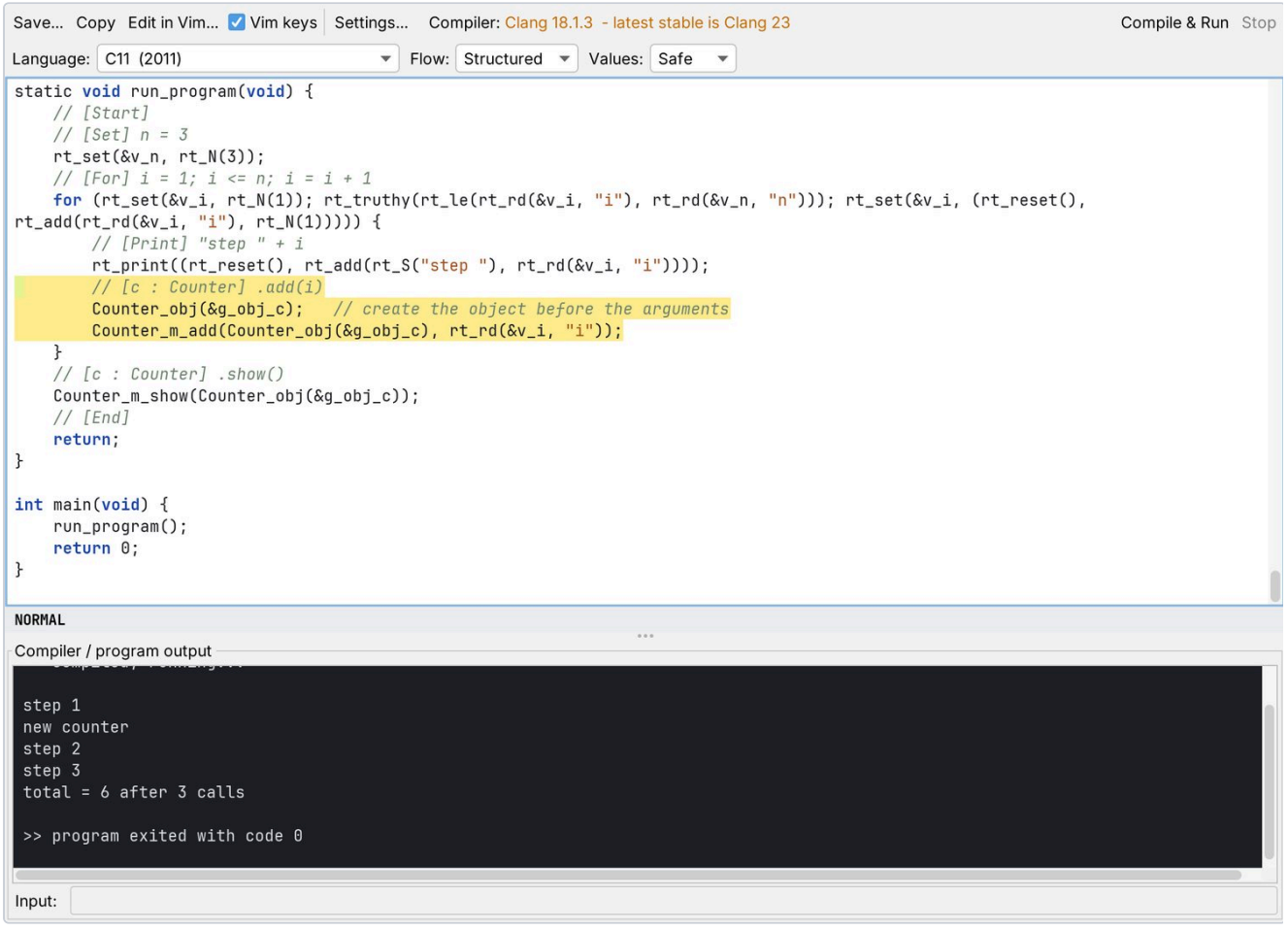


6. Выберите C или другой стандарт C++

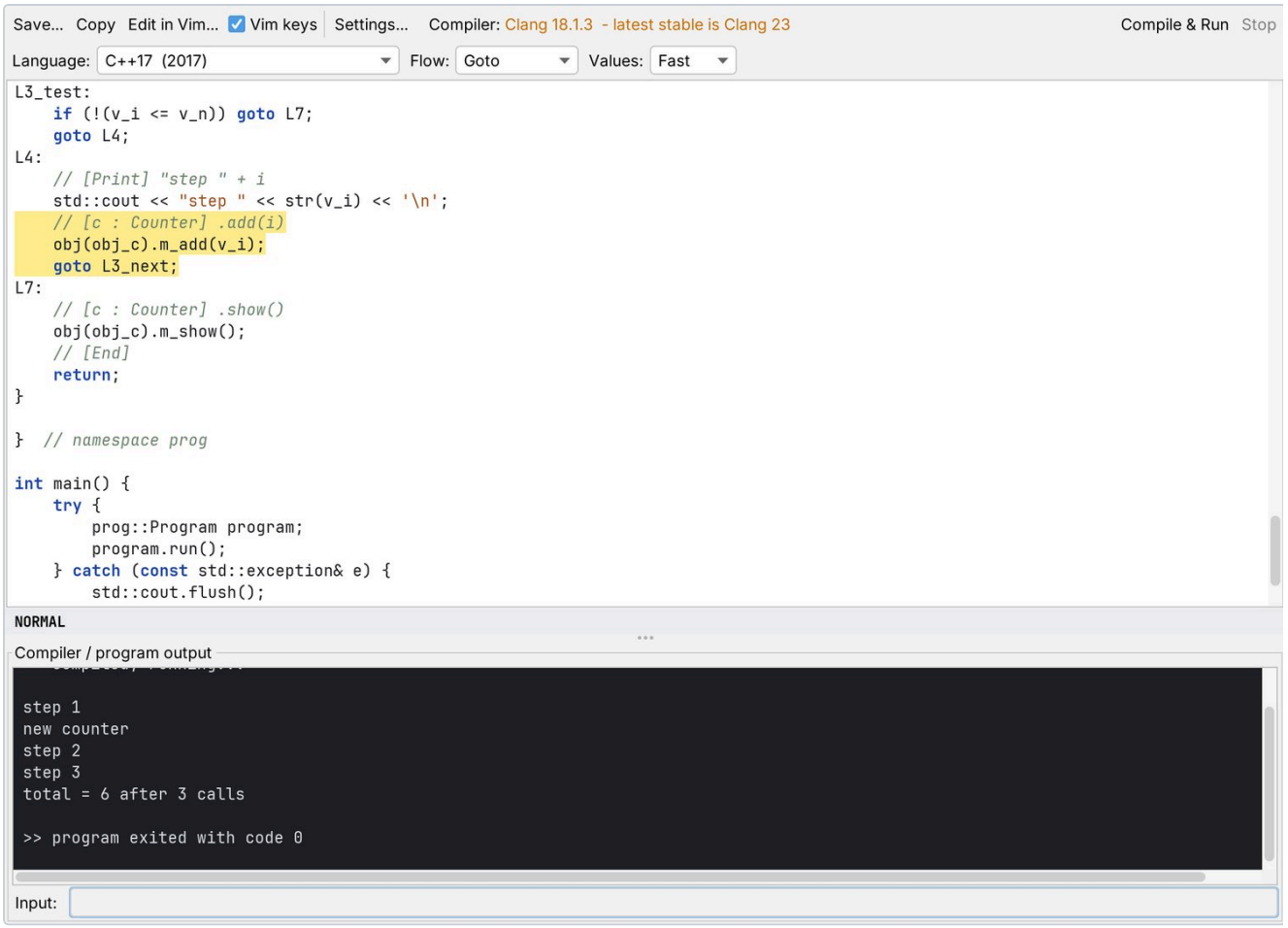
Выберите **Language** в выпадающем списке (в окне кода или на главной панели инструментов) — код сразу генерируется заново, а тот же блок остаётся подсвеченным. **Flow** и **Values** задают стиль кода: **Structured** — if/else, циклы и switch, **Goto** — каждый блок становится меткой, а каждая связь — переходом; **Safe** — числа и текст со всеми проверками интерпретатора, **Fast** — простые `double` без проверок.



Та же программа на **C11**, скомпилированная `clang -std=c11` и запущенная: вывод тот же.

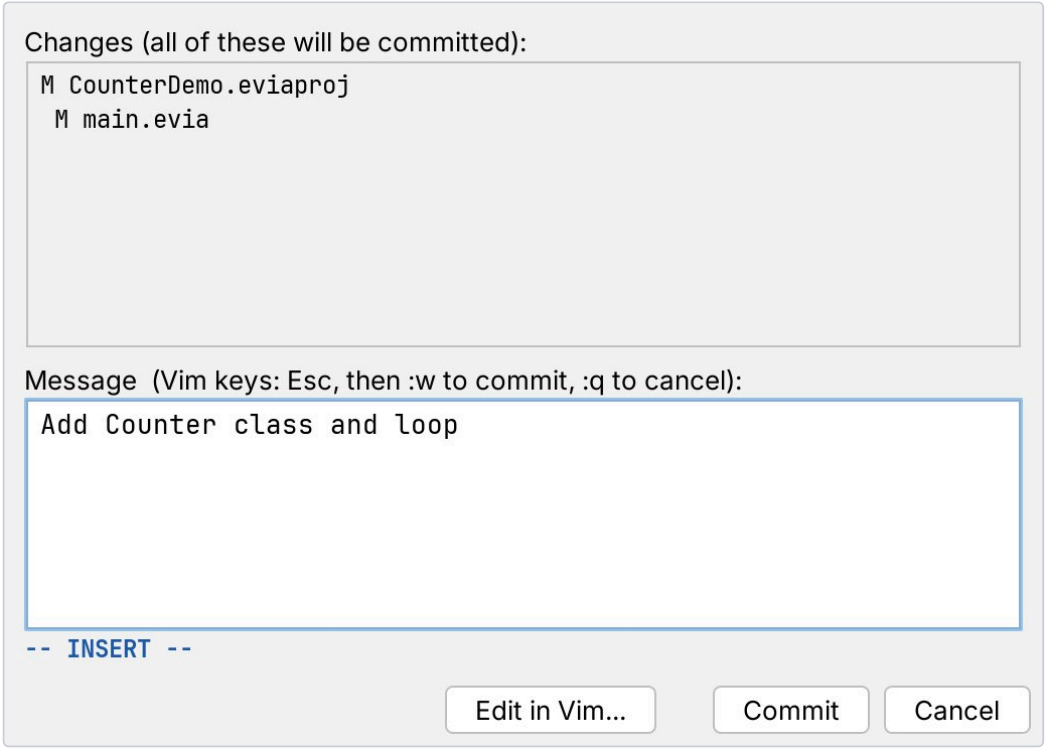


C++17 с потоком **Goto** и значениями **Fast**:

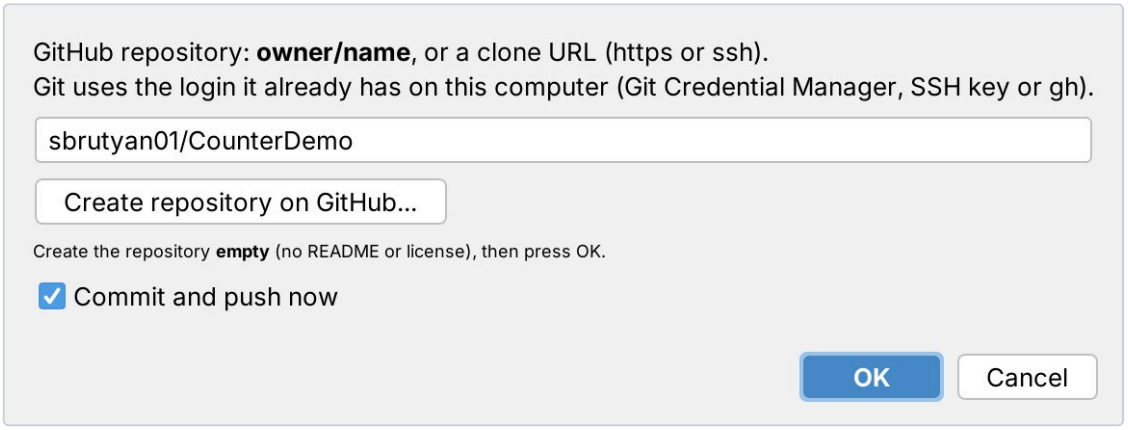


7. Сделайте коммит и опубликуйте на GitHub

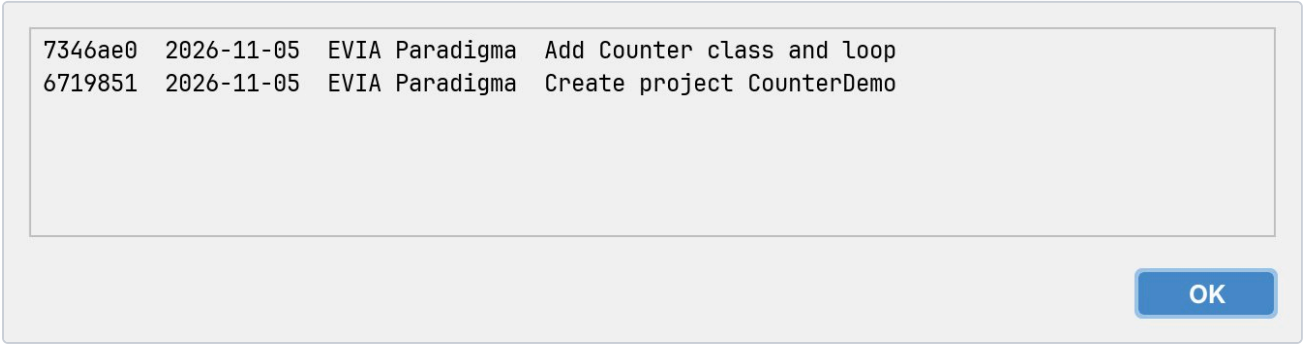
Git → Commit... (Ctrl+K) сохраняет все изменения и делает коммит. В поле сообщения работают клавиши Vim (`Esc`, затем `:w` — коммит).



Git → Connect to GitHub...: введите `владелец/репозиторий`. *Create repository on GitHub...* открывает github.com с уже заполненным именем; создайте **пустой** репозиторий, нажмите **OK** — и проект будет отправлен. Git использует вход, который уже настроен на вашем компьютере (Git Credential Manager, SSH-ключ или `gh auth login`).



Git → History показывает список коммитов:

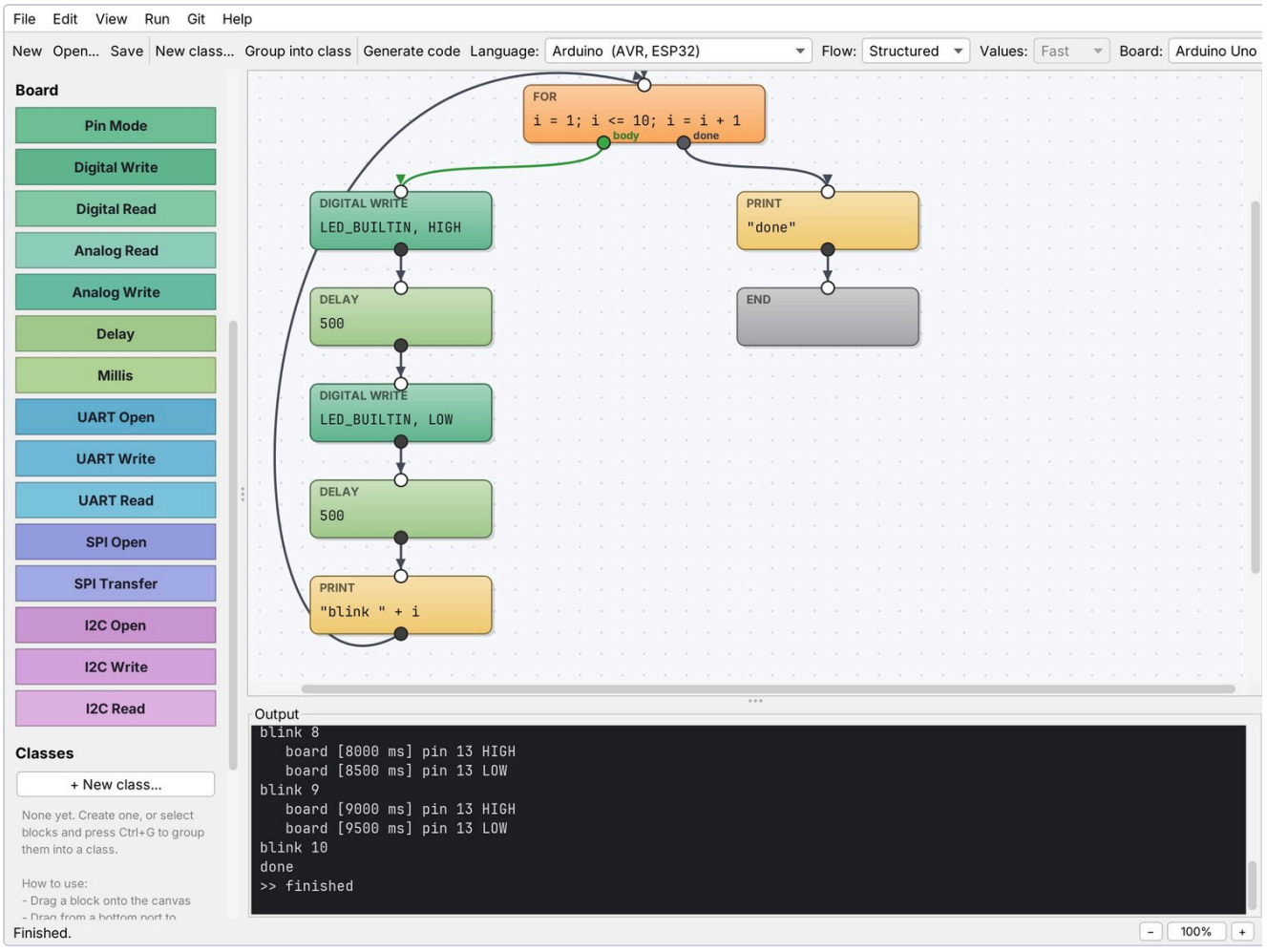


8. Запрограммируйте плату Arduino или ESP32

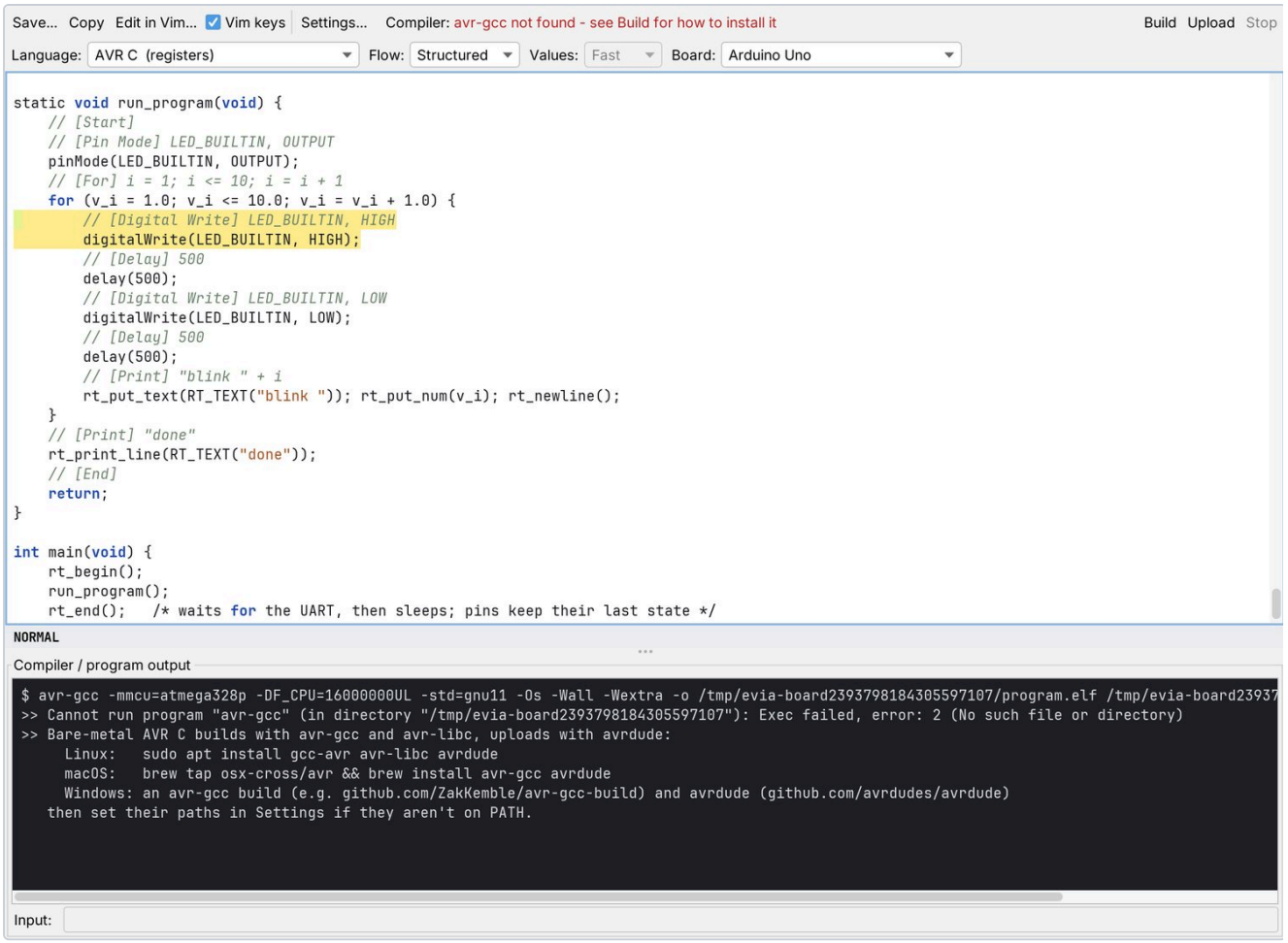
File → Open Example → Arduino and ESP32 Boards → Blink the built-in LED. Блоки **Board** (зелёные, в разделе *Board* палитры) настраивают пины, читают кнопки и датчики и делают паузы:

Блок	Код	Что делает
Pin Mode	`LED_BUILTIN, OUTPUT`	`OUTPUT`, `INPUT` или `INPUT_PULLUP`
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
Analog Write	`5, brightness`	ШИМ, 0..255
Delay	`500`	ждёт столько миллисекунд
Millis	`t`	миллисекунды с начала программы

Run (F5) имитирует Arduino Uno: в выводе видно каждое изменение пинов со временем.



Выберите **Language: Arduino** и плату в списке **Board**, затем **Generate code** (F7): один скетч подходит для любой из 12 плат (Uno, Nano, Mega 2560, Leonardo, Micro, Pro Mini, ESP32, ESP32-S2, -S3, -C3, -C6, -H2). **Build** собирает его через arduino-cli, **Upload** загружает на плату; Print и Input работают через Serial Monitor на скорости 115200 бод. **AVR C** пишет ту же программу без ядра Arduino, прямо на регистрах микроконтроллера, и собирает её avr-gcc.



9. Общайтесь с другими микросхемами: UART, SPI и I2C, и PIC32

Ещё три программы в **File → Open Example → Arduino and ESP32 Boards** используют блоки шин: *UART: send a byte, read it back*, *SPI: transfer a byte* и *I2C: write and read a device register*. Блоки находятся в разделе *Board*:

Блок	Код	Что делает
UART Open	`9600`	запускает последовательный порт с этой скоростью в бодах
UART Write	`65`	отправляет один байт (0..255)
UART Read	`c`	следующий принятый байт или -1, если ничего не пришло
SPI Open	`1000000`	SPI-мастер с этой частотой в Гц (режим 0, старший бит первым)
SPI Transfer	`reply = 66`	отправляет байт и сохраняет байт, пришедший одновременно
I2C Open	`100000`	I2C-мастер с этой частотой в Гц
I2C Write	`80, 0, 42`	адрес (7 бит), регистр, значение
I2C Read	`v = 80, 0`	сохраняет значение регистра или -1, если никто не отвечает

Шину нужно открыть до использования. **Run** имитирует их: байты из UART Write возвращаются через UART Read, вход SPI соединён с его выходом, а по адресу 80 отвечает одно устройство I2C с 256 регистрами (80 — это 0x50; числа десятичные).

Для микросхемы Microchip PIC32MX выберите **Language: PIC32 C** и микросхему в списке **Board** (девять микросхем семейств MX1xx/2xx, MX3xx/4xx и MX5xx/6xx/7xx), затем **Generate code** (F7). Получится bare-metal C для бесплатного компилятора Microchip XC32: тактирование, пины, ШИМ, АЦП, UART, SPI и I2C настраиваются через регистры. **Build** запускает `xc32-gcc` и `xc32-bin2hex`; **Upload** запускает команду из **Settings** (по умолчанию `pic32prog {hex}`). Пины выбранного семейства перечислены в начале созданного файла.

Обратите внимание: код PIC32 проверен по настоящим заголовочным файлам Microchip, но ещё не собирался XC32 и не запускался на микросхеме. Считайте первую сборку на своей плате проверкой.

10. Аппаратура для FPGA

Выберите **Язык: SystemVerilog**, **Verilog-2001** или **VHDL**, затем плату в списке **Плата**: Digilent Nexys A7-100T, Arty A7-35 или A7-100, Basys 3 (Artix-7), либо Olimex GateMateA1-EVB и Cologne Chip GateMate A1 evaluation board. Откройте **Файл → Открыть пример → FPGA Boards → Blink** и нажмите **Создать код** (F7).

Каждый блок становится одним состоянием автомата и занимает один такт. Выводы фиксируются при сборке, поэтому пишите их именами: `LED0`, `SW3`, `BTNC`, `JA1`. В первом комментарии файла лежат ограничения выводов; с `-o` в командной строке они также записываются как `.xdc` (Vivado) или `.ccf` (GateMate).

- **Сборка** и **Загрузка** используют Vivado или открытую цепочку для Artix-7 (выбор в **Настройках**), а для GateMate — yosys, nextpnr-himbaechel, gmpack и openFPGALoader.
- Числа целые 32-битные, поэтому `5 / 2` на FPGA равно 2, а в **Запуске** — 2.5.
- Классы, Input, Analog Read, SPI, I2C и текст в переменных пока не поддерживаются; ошибка называет блок.

Обратите внимание: результат моделировался и синтезировался открытыми средствами, но не проходил через Vivado и не загружался в плату.

11. Отправьте числа дальше по MQTT

Откройте **Файл → Открыть пример → Arduino and ESP32 Boards → MQTT sensor**. Программа публикует температуру брокеру, отправляет себе команду и читает её обратно.

Пять блоков **MQTT** находятся в группе *Board*: *MQTT Connect*, *MQTT Publish*, *MQTT Subscribe*, *MQTT Has Message* и *MQTT Read*. Имя сети, пароль, брокер и темы — фиксированные тексты: они попадают в программу при сборке.

- **Запуск** отвечает из брокера на вашем компьютере, поэтому программу можно попробовать без оборудования.

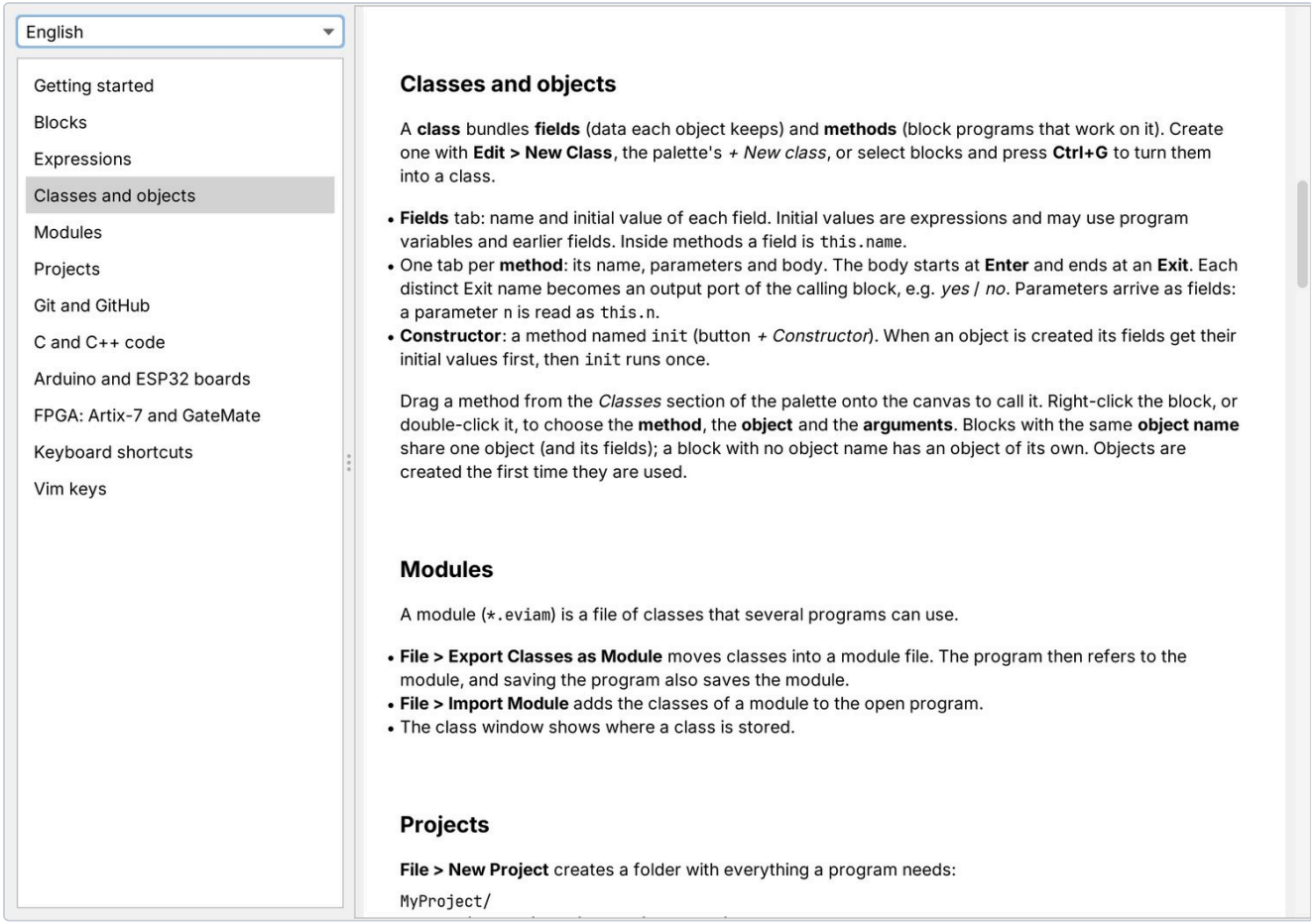
- **PIC32 C** управляет модулем ESP8266 или ESP32 с прошивкой AT через UART2, а Print остаётся на UART1.
- **Arduino** на плате **ESP32** использует собственный WiFi платы.
- Плата AVR отказывается от этих блоков: у неё один последовательный порт, и его занимает Print.

Сообщение — это число. *MQTT Read* возвращает -1, если ничего не пришло, поэтому сначала спросите *MQTT Has Message*, если настоящая -1 возможна.

Обратите внимание: пакеты MQTT проверяются в CI на настоящем брокере, но ни модуля WiFi, ни ESP32, ни микросхемы PIC32 в этой проверке не было. MQTT через порт Ethernet чипов MX695/795 пока не поддерживается.

12. Что дальше

- Делитесь классами между программами: **File → Export Classes as Module...**, затем **File → Import Module...** в другой программе.
- В **File → Open Example** есть готовые программы, в том числе `classes.evia` (поля, конструктор, методы, модуль) и примеры для плат.
- **Help → Tutorial (PDF)** открывает этот учебник на английском, русском, армянском, немецком, упрощённом и традиционном китайском или японском.
- **Help → Guide** (F1) описывает каждый блок, выражения, классы, проекты, Git, сочетания клавиш и клавиши Vim.



Подробнее обо всех возможностях — в README (на английском).