EVIA Paradigma. ուսուցում

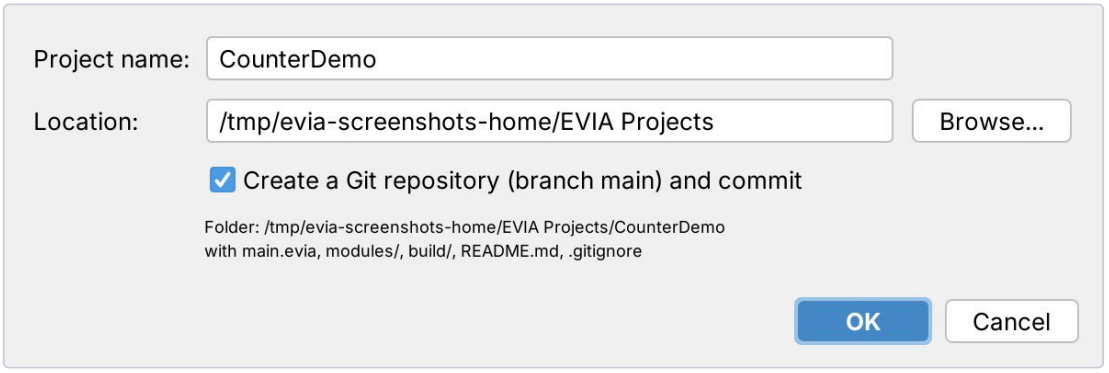
Այս ուսուցման ընթացքում մենք կստեղծենք ցիկլով և դասով փոքր նախագիծ, կգործարկենք այն, կտեսնենք, թե ինչ C և C++ կոդի է այն վերածվում, և կպահենք այն Git-ում: Վերջում կծրագրավորենք Arduino կամ ESP32 տախտակ: Բոլոր սքրինշոթներն արված են իրական հավելվածում: Ծրագրի ինտերֆեյսն անգլերեն է, ուստի ընտրացանկերի և կոճակների անունները բերված են այնպես, ինչպես երևում են էկրանին:

0.4.1 տարբերակից ծրագիրն ունի **մուգ թեմա** (**View → Theme**) և նման է IntelliJ IDEA-ին (Inter և JetBrains Mono տառատեսակներ). **Help → Guide** ուղեցույցը կարելի է կարդալ նաև ռուսերեն և հայերեն: Այս ուսուցման սքրինշոթներն արված են բաց թեմայով:

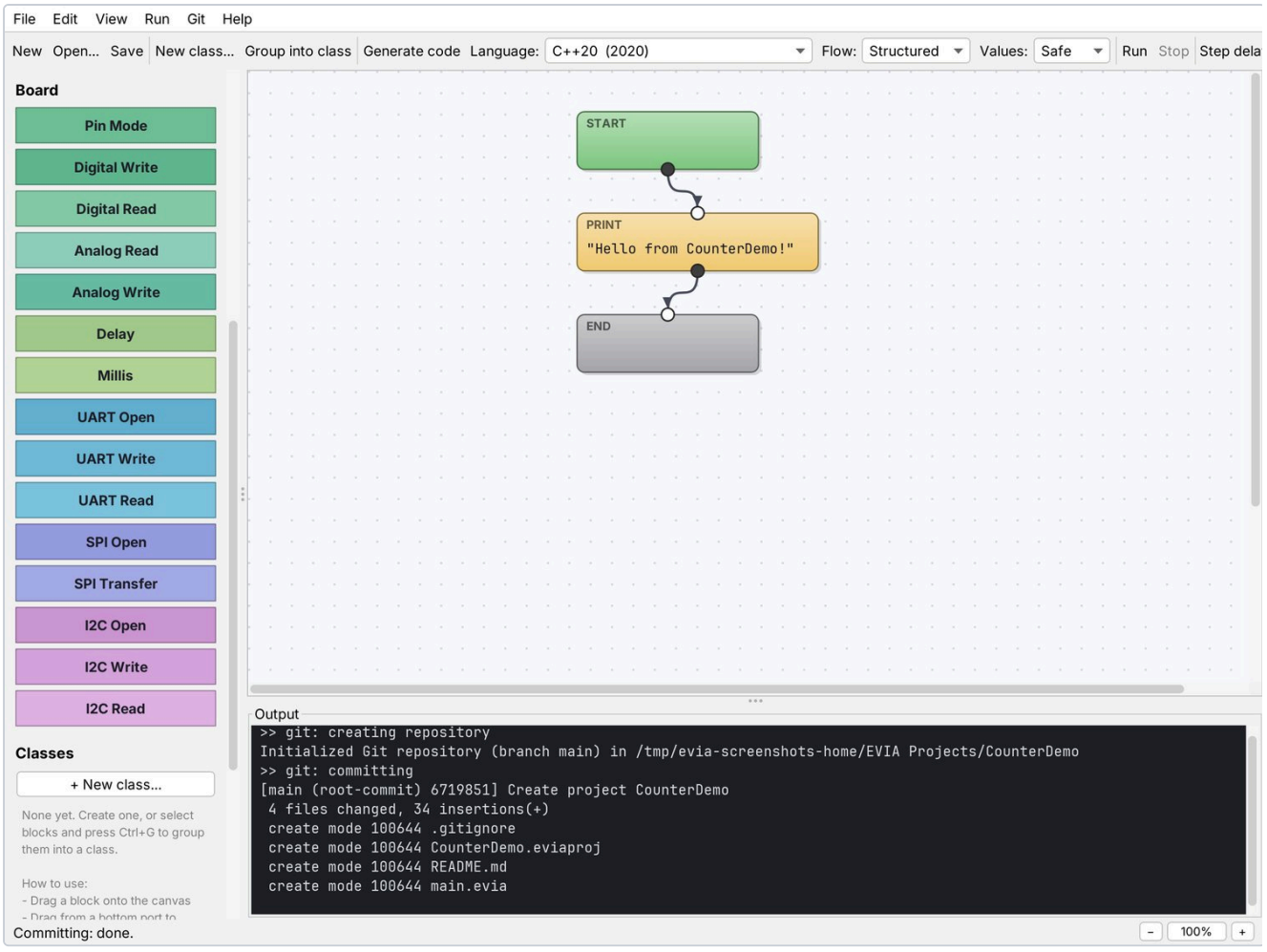
Ներբեռնել EVIA Paradigma-ն՝ վերջին թողարկումը (Windows-ի, macOS-ի և Linux-ի տեղադրիչներն արդեն պարունակում են Java):

1. Ստեղծեք նախագիծ

File → New Project... (Ctrl+Shift+N): Մուտքագրեք անունը և թղթապանակը. եթե git-ը տեղադրված է, թողեք *Create a Git repository*-ն նշված:



Նախագիծը բացվում է սկզբնական ծրագրով՝ *Start → Print → End*: Ելքի վահանակը ցույց է տալիս, որ git-ը ստեղծել է պահոցը և առաջին commit-ը:

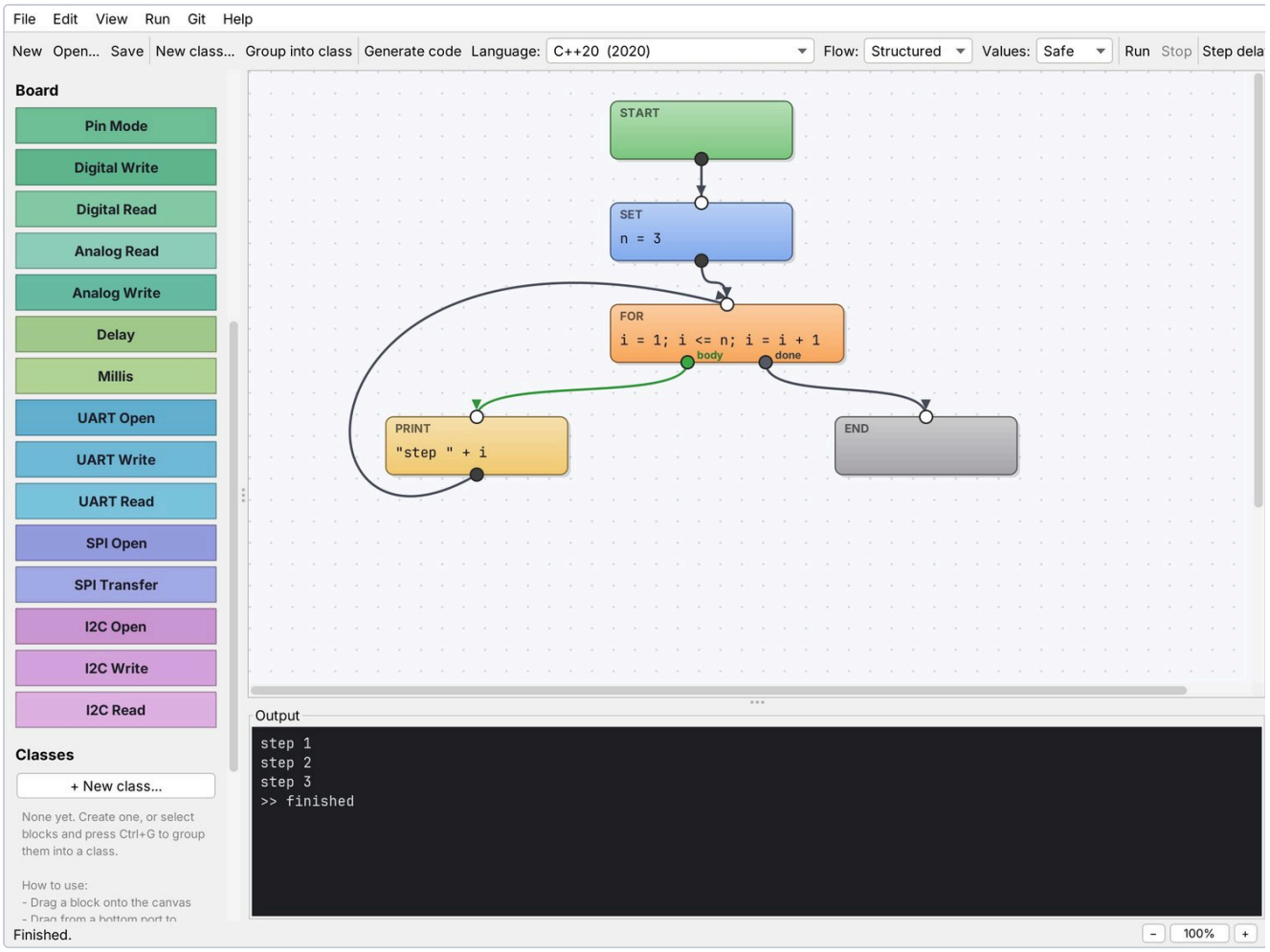


2. Կառուցեք ցիկլ

Բլոկները գունապնակից քաշեք կտավի վրա և միացրեք դրանք. քաշեք բլոկի ներքևի պորտից (մուգ կետ) դեպի հաջորդ բլոկի վերևի պորտը (սպիտակ կետ): Բլոկի վրա կրկնակի սեղմումը բացում է նրա կոդը խմբագրելու համար:

- **Set**. `n = 3`
- **For**. `i = 1; i ⩽ n; i = i + 1`: Նրա `body` պորտը տանում է դեպի **Print** `"step " + i`. ցիկլի մարմնի վերջը միացրեք **հետ՝ For բլոկին**, որպեսզի սկսվի հաջորդ անցումը: `done` պորտը տանում է դեպի **End**:

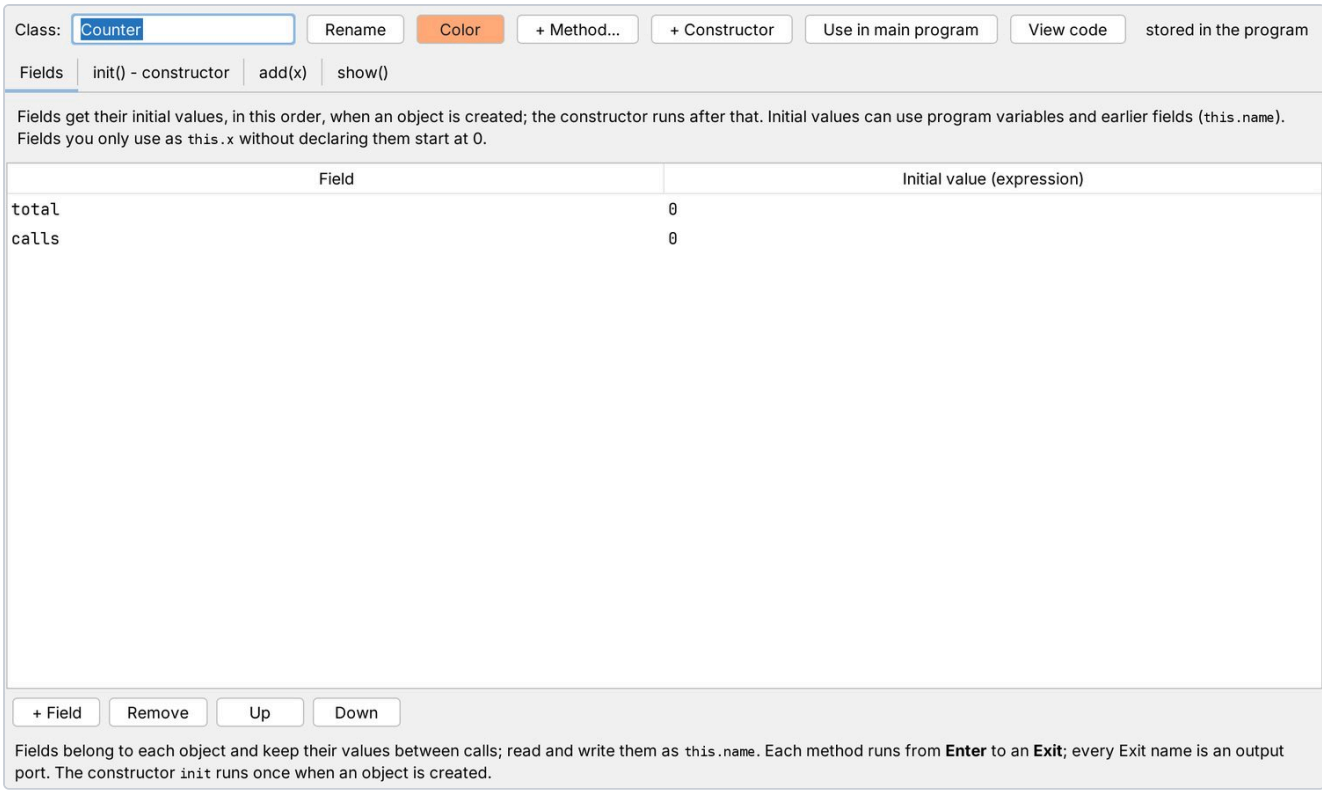
Սեղմեք **Run** (F5): Նշեք *Step delay*-ը, որպեսզի տեսնեք, թե ինչպես է շարժվում ակտիվ բլոկը: Մկնիկի անիվը սխեմայի վրա փոխում է մասշտաբը:



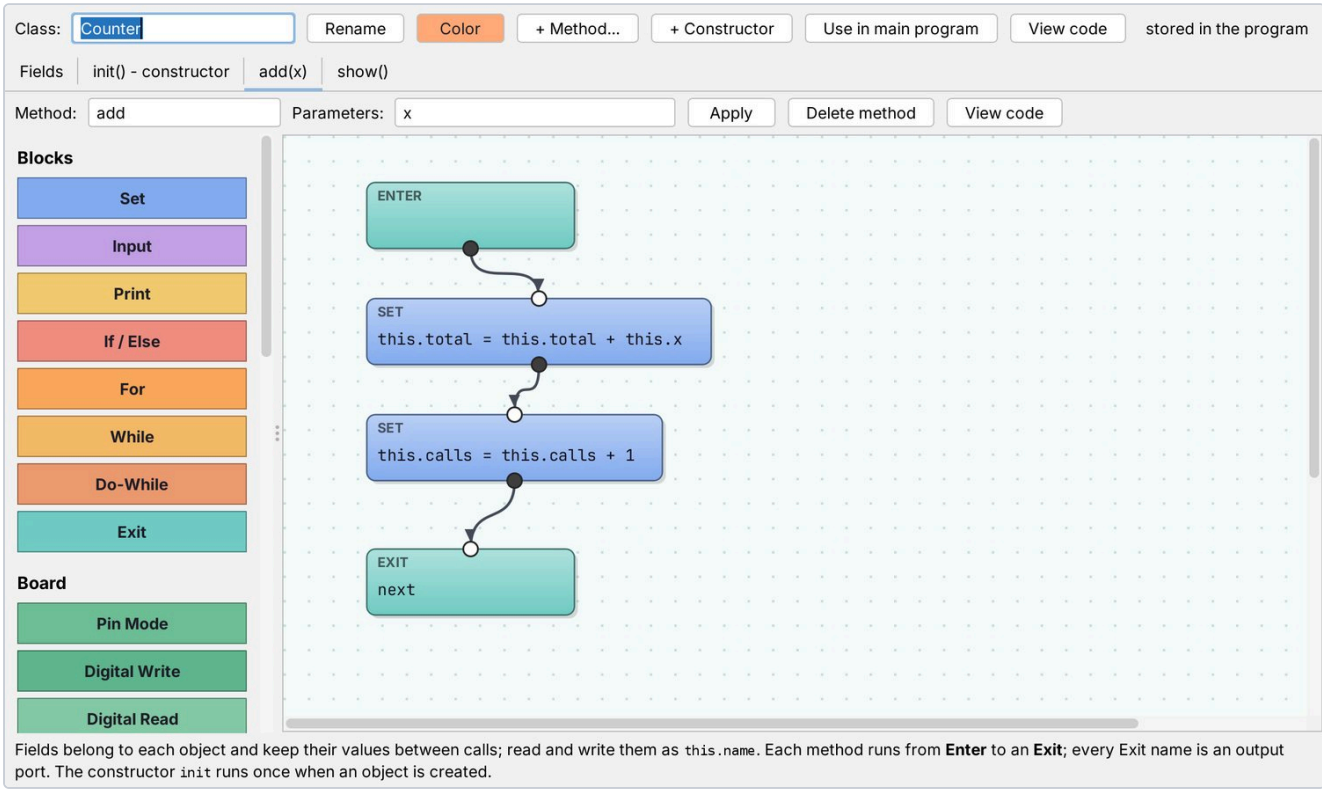
3. Ստեղծեք դաս

Edit → New Class... (կամ *+ New class...* գունապնակում), անվանեք այն `Counter`: Դասի պատուհանում կա **Fields** ներդիր և յուրաքանչյուր մեթոդի համար մեկական ներդիր:

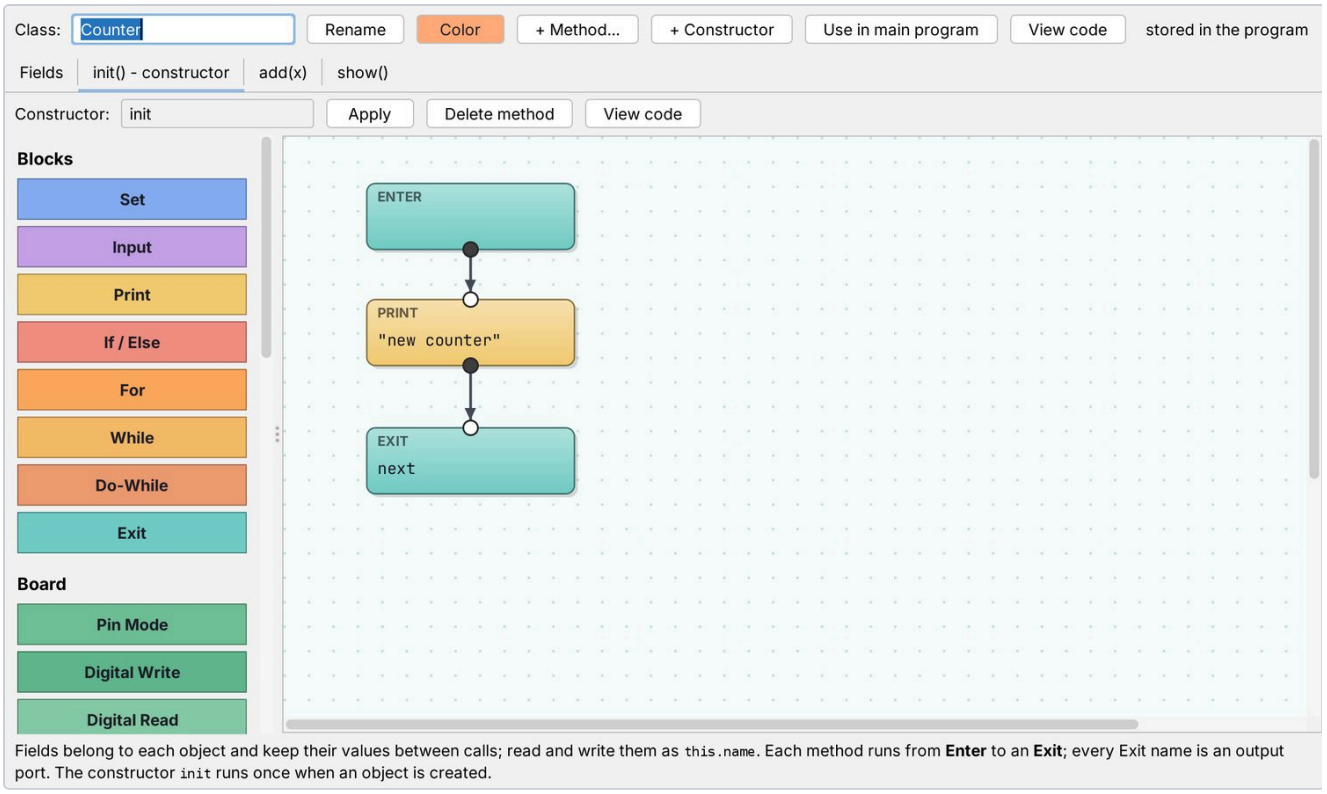
Fields. ավելացրեք `total` և `calls` դաշտերը, երկուսն էլ `0` սկզբնական արժեքով: Յուրաքանչյուր օբյեկտ ունի դաշտերի իր պատճենները:



Մեթոդներ. առաջին մեթոդը վերանվանեք `add`՝ `x` պարամետրով (սեղմեք **Apply**), ապա կառուցեք դրա մարմինը **Enter**-ից մինչև **Exit**: Մեթոդի ներսում դաշտերն ու պարամետրերը գրվում են `this.անուն` ձևով:

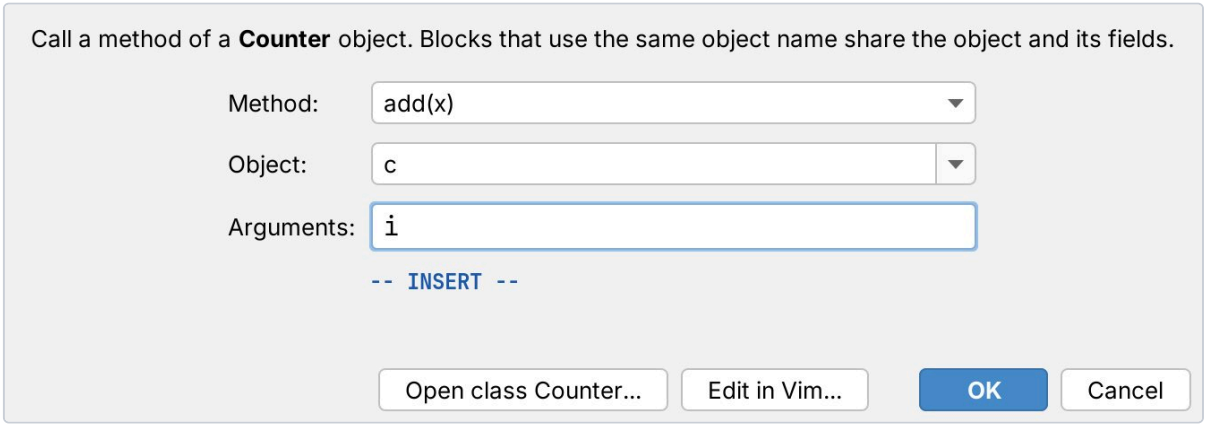


+ Constructor-ը ավելացնում է `init`-ը, որը կատարվում է մեկ անգամ՝ օբյեկտը ստեղծելիս (այն բանից հետո, երբ դաշտերը ստացել են իրենց սկզբնական արժեքները): Այստեղ այն տպում է `"new counter"`: **+ Method...**-ը ավելացնում է `show`-ը, որը տպում է `"total = " + this.total + " after " + this.calls + " calls"`:

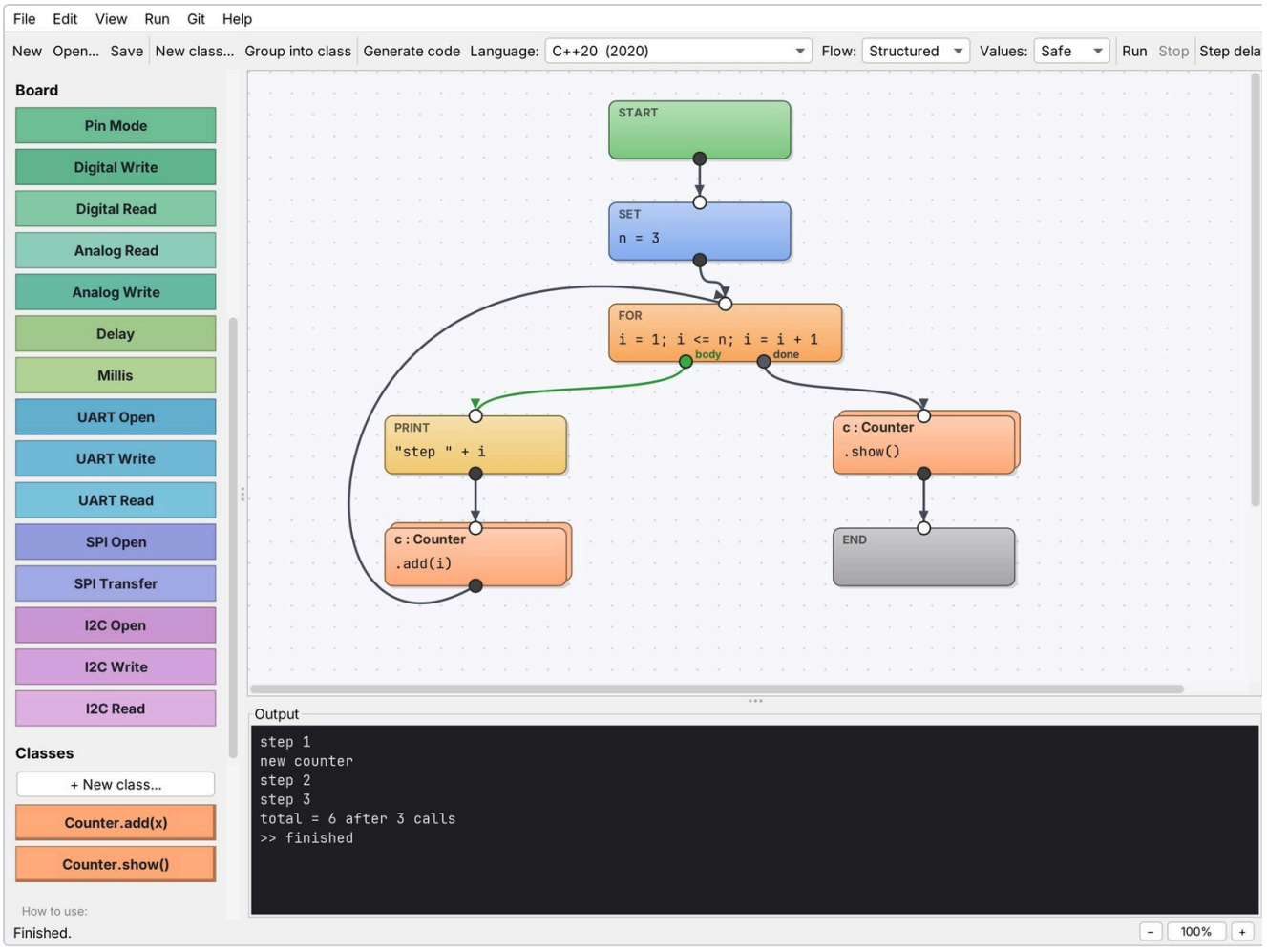


4. Օգտագործեք դասը

Այժմ գունապնակում կան `Counter.add(x)` և `Counter.show()`: Քաշեք `Counter.add(x)`-ը ցիկլի մարմնի մեջ՝ **Print**-ից հետո, իսկ `Counter.show()`-ը՝ `done`-ի և **End**-ի միջև: Կանչի բլոկի վրա կրկնակի սեղմումով կարելի է ընտրել մեթոդը, **օբյեկտը** և արգումենտները: Երկու բլոկներին էլ տվեք `c` օբյեկտի անունը, որպեսզի նրանք աշխատեն նույն օբյեկտի հետ:

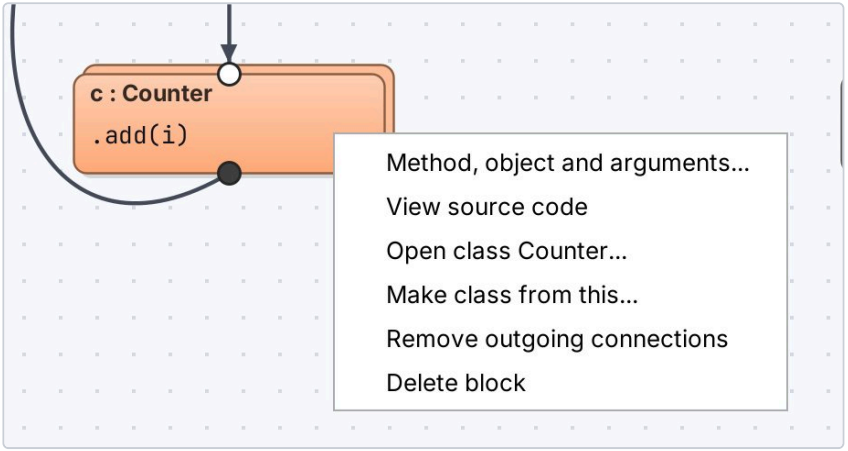


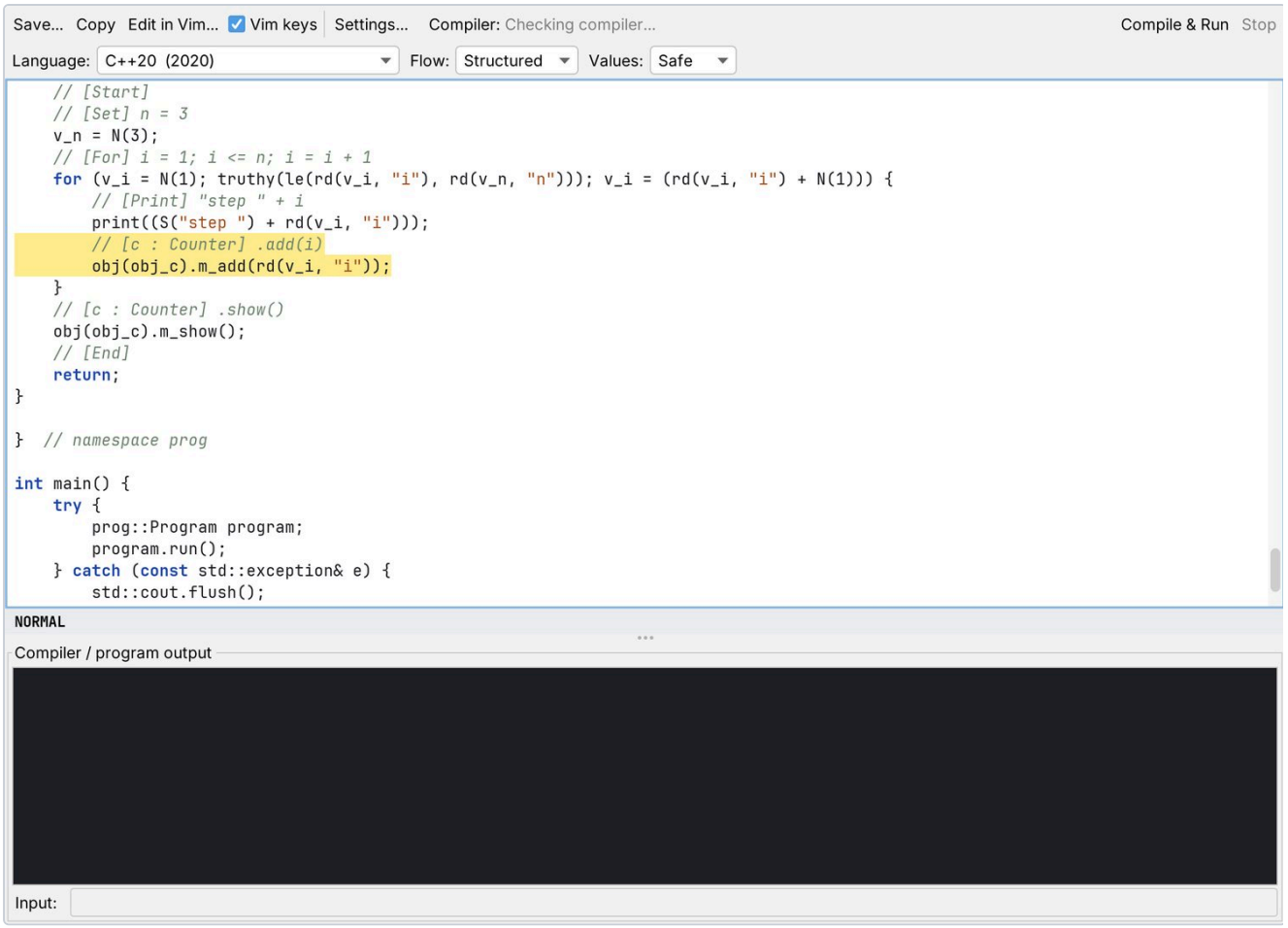
Գործարկեք ծրագիրը: `c`-ն ստեղծվում է առաջին օգտագործման ժամանակ (այդ պահին էլ տպվում է `new counter`), `add`-ը կատարվում է երեք անգամ, իսկ `show`-ը տպում է արդյունքը:



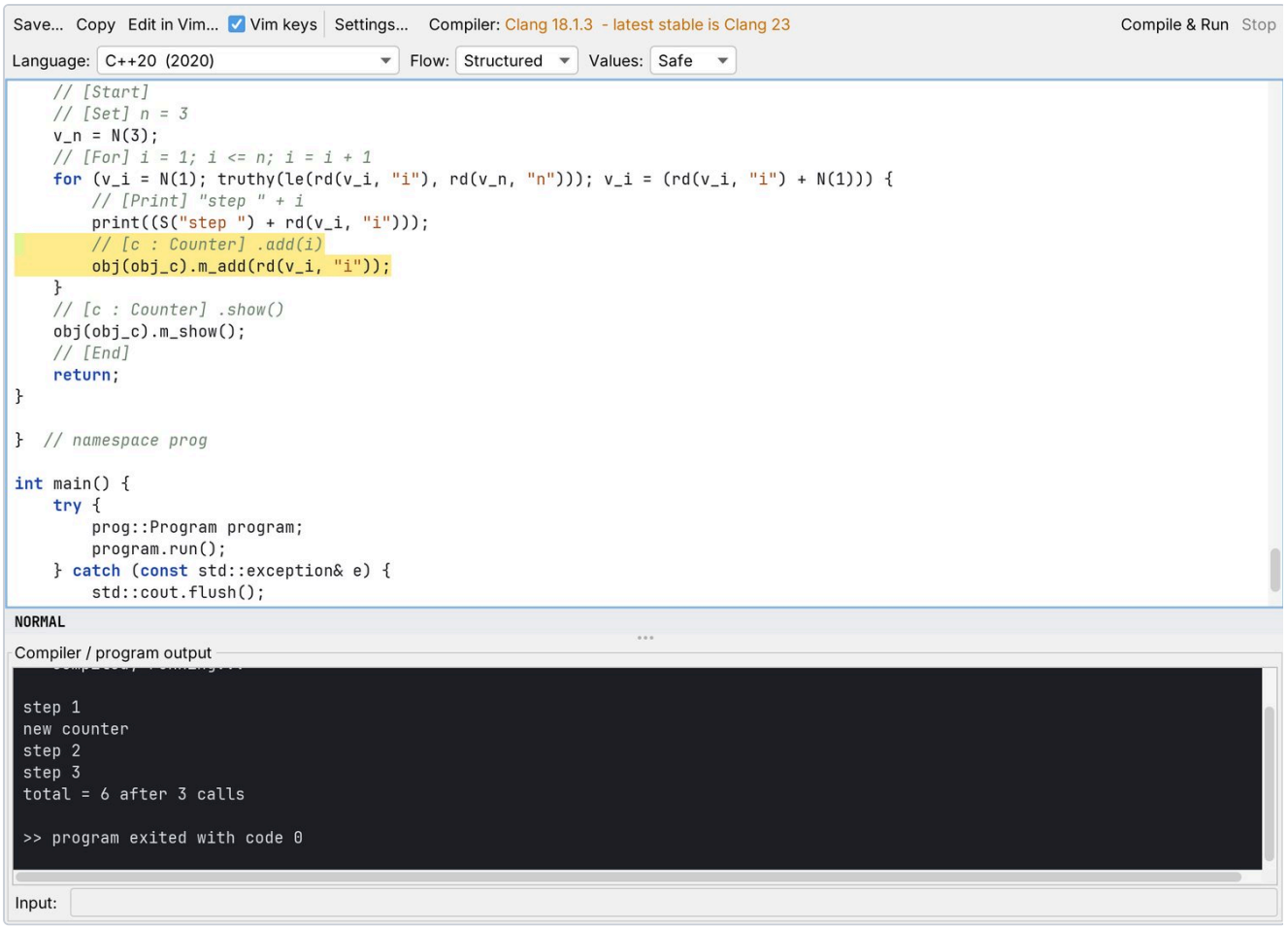
5. Տեսեք ցանկացած բլոկի կոդը

Աջ սեղմում բլոկի վրա → **View source code**: Կբացվի գեներացված կոդը, որտեղ ընդգծված են հենց այդ բլոկի տողերը (այստեղ՝ C++20, լռելյայն լեզուն):



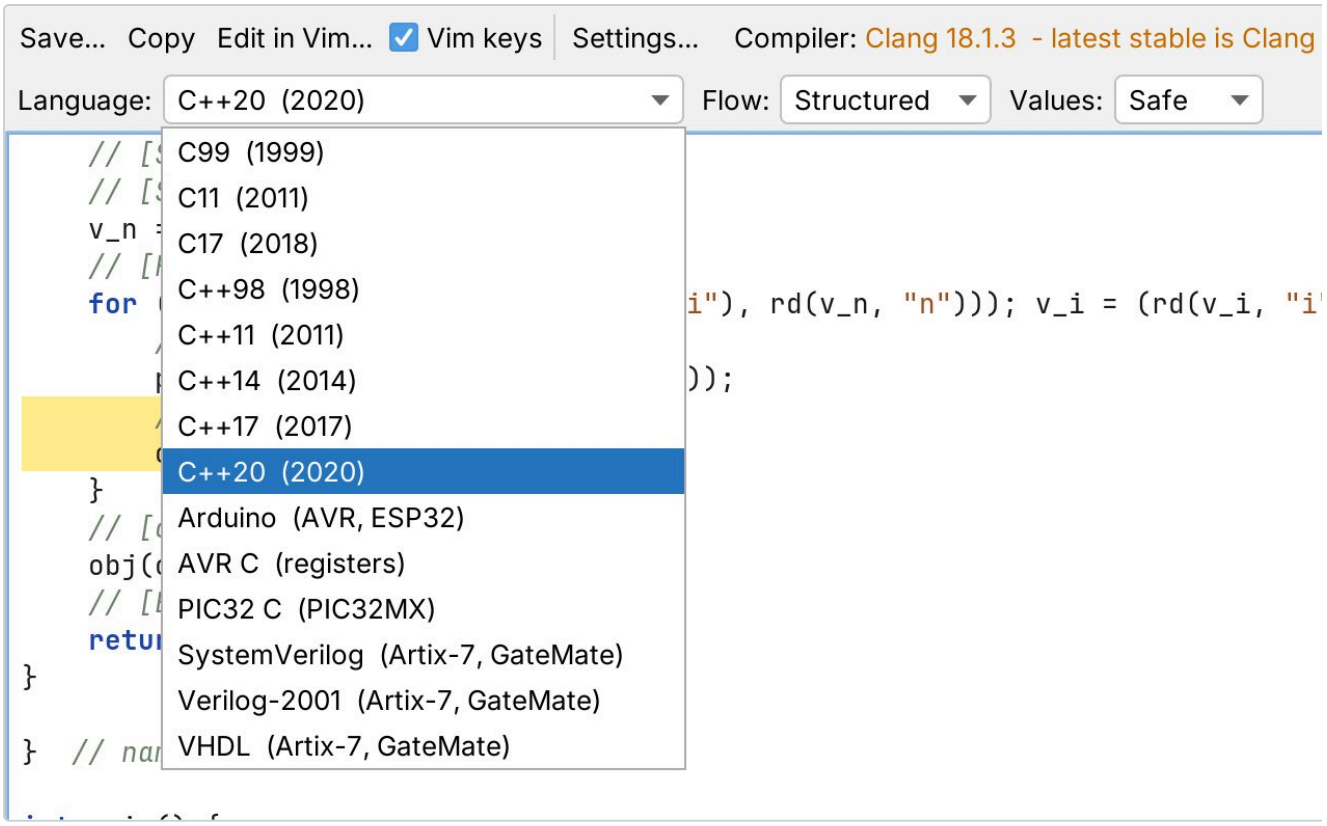


Compile & Run-ը կառուցում է կոդը Clang-ով կամ GCC-ով և գործարկում այն: Կոմպիլյացված ծրագիրը տպում է նույնը, ինչ ինտերպրետատորը: Windows-ում ծրագիրն ինքն է տեղադրում կոմպիլյատորը՝ llvm-mingw (Clang) կամ WinLibs (GCC), Visual Studio պետք չէ: Կոմպիլյատորն ընտրվում է **Settings**-ում:

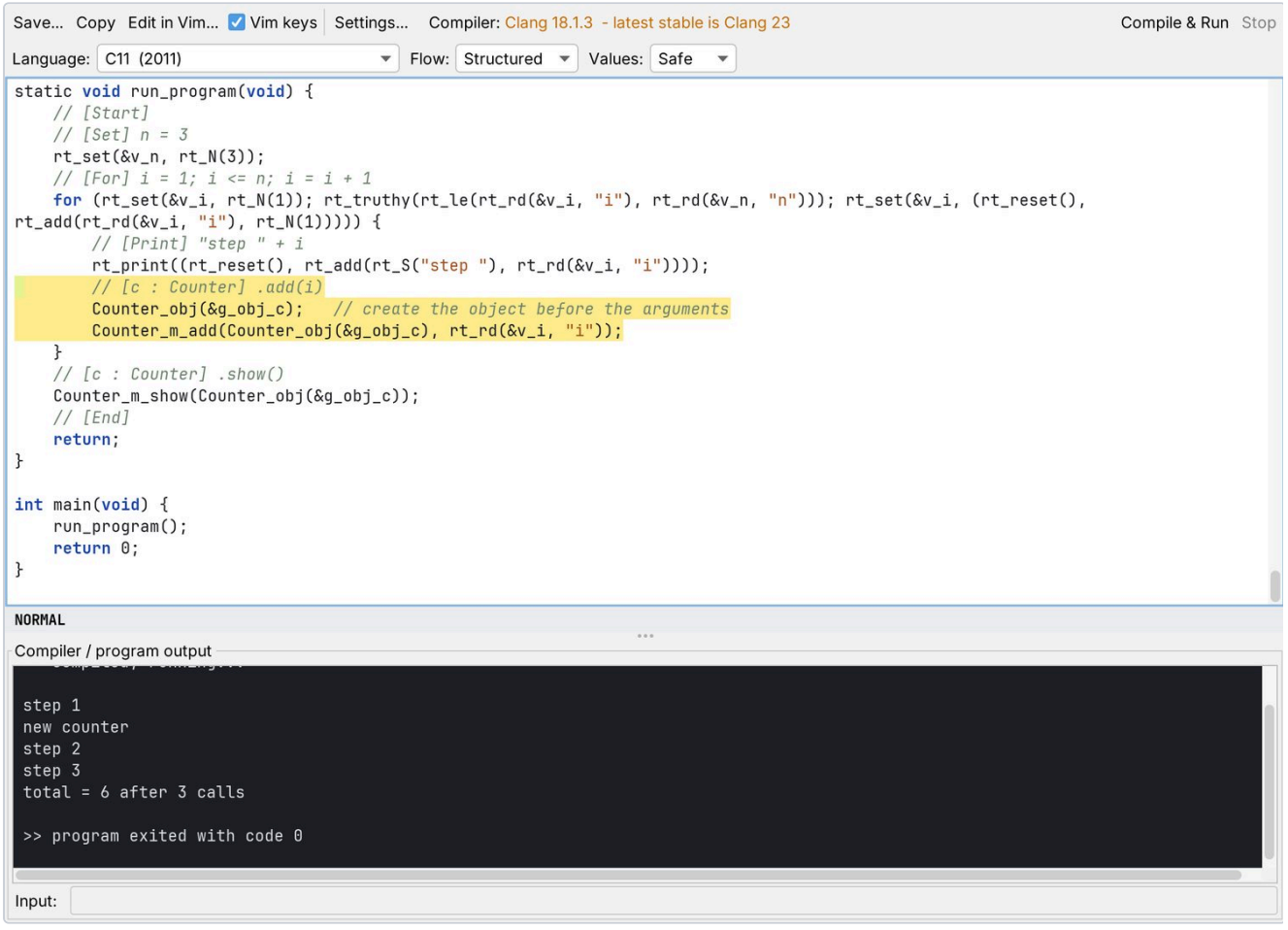


6. Ընտրեք C կամ C++-ի այլ ստանդարտ

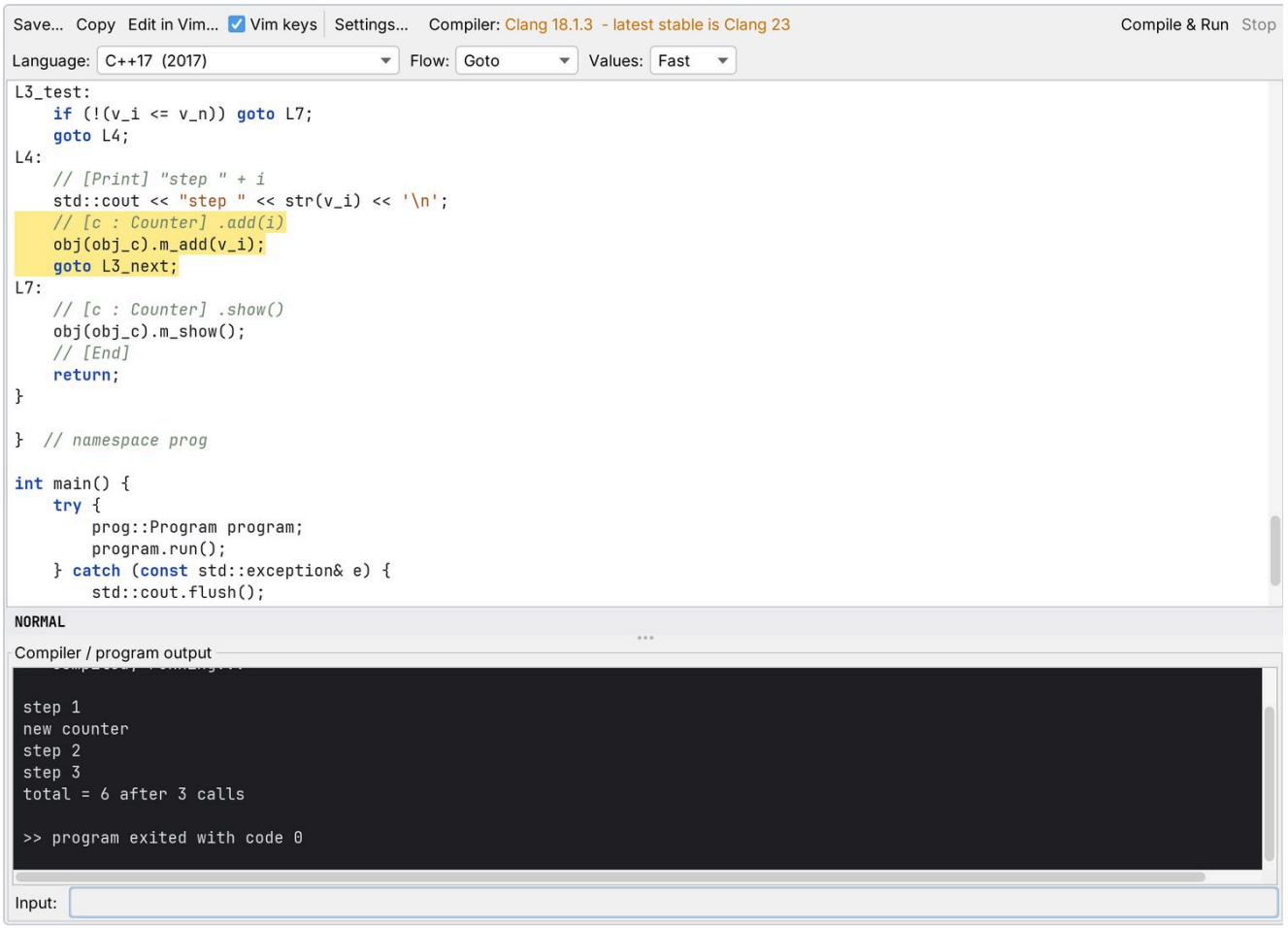
Ընտրեք **Language**-ը բացվող ցանկում (կոդի պատուհանում կամ գլխավոր գործիքագոտում). կոդն անմիջապես նորից գեներացվում է, և նույն բլոկը մնում է ընդգծված: **Flow**-ն և **Values**-ը որոշում են կոդի ոճը. **Structured**՝ if/else, ցիկլեր և switch, **Goto**՝ յուրաքանչյուր բլոկ դառնում է նշիչ, իսկ յուրաքանչյուր կապ՝ անցում. **Safe**՝ թվեր և տեքստ՝ ինտերպրետատորի բոլոր ստուգումներով, **Fast**՝ պարզ `double`-ներ՝ առանց ստուգումների:



Նույն ծրագիրը **C11**-ով՝ կոմպիլյացված `clang -std=c11`-ով և գործարկված. ելքը նույնն է:

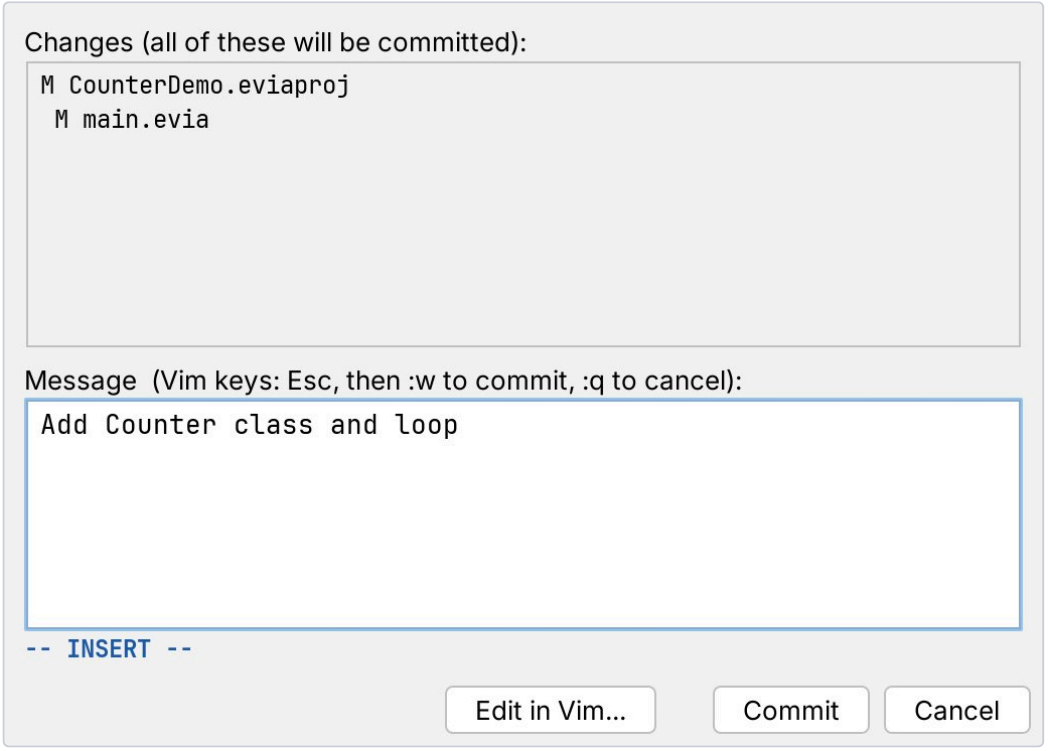


C++17՝ **Goto** հոսքով և **Fast** արժեքներով:

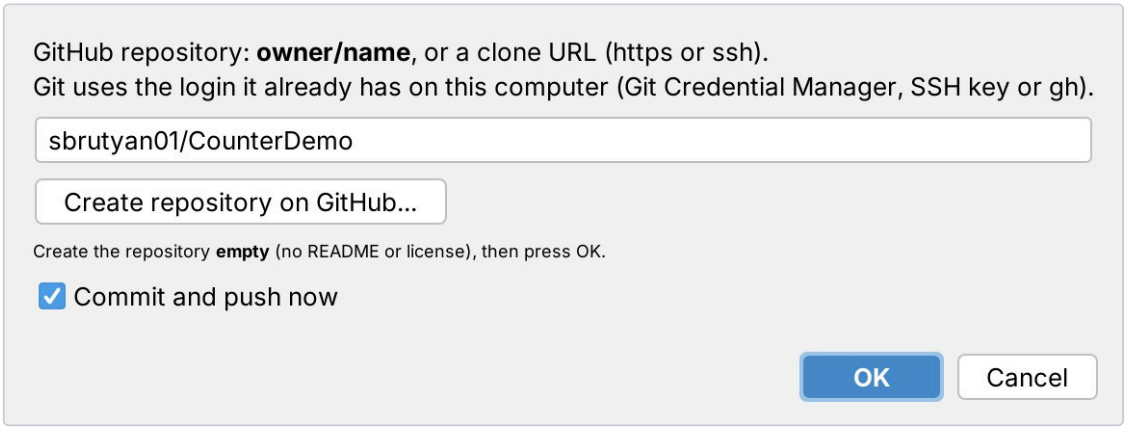


7. Կատարեք commit և հրապարակեք GitHub-ում

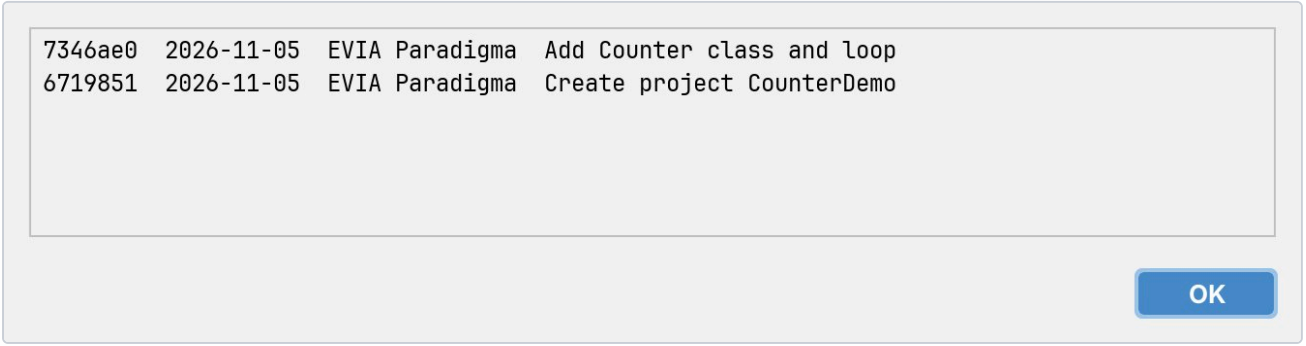
Git → Commit... (Ctrl+K)-ը պահում է բոլոր փոփոխությունները և կատարում commit: Հաղորդագրության դաշտում աշխատում են Vim ստեղները (`Esc`, ապա `:w`՝ commit):



Git → Connect to GitHub... մուտքագրեք `տեր/պահոց`: *Create repository on GitHub...*-ը բացում է github.com-ը՝ արդեն լրացված անունով. ստեղծեք **դատարկ** պահոց, սեղմեք **OK**, և նախագիծը կուղարկվի: Git-ն օգտագործում է այն մուտքը, որն արդեն կարգավորված է ձեր համակարգչում (Git Credential Manager, SSH բանալի կամ `gh auth login`):



Git → History-ն ցույց է տալիս commit-ների ցանկը:

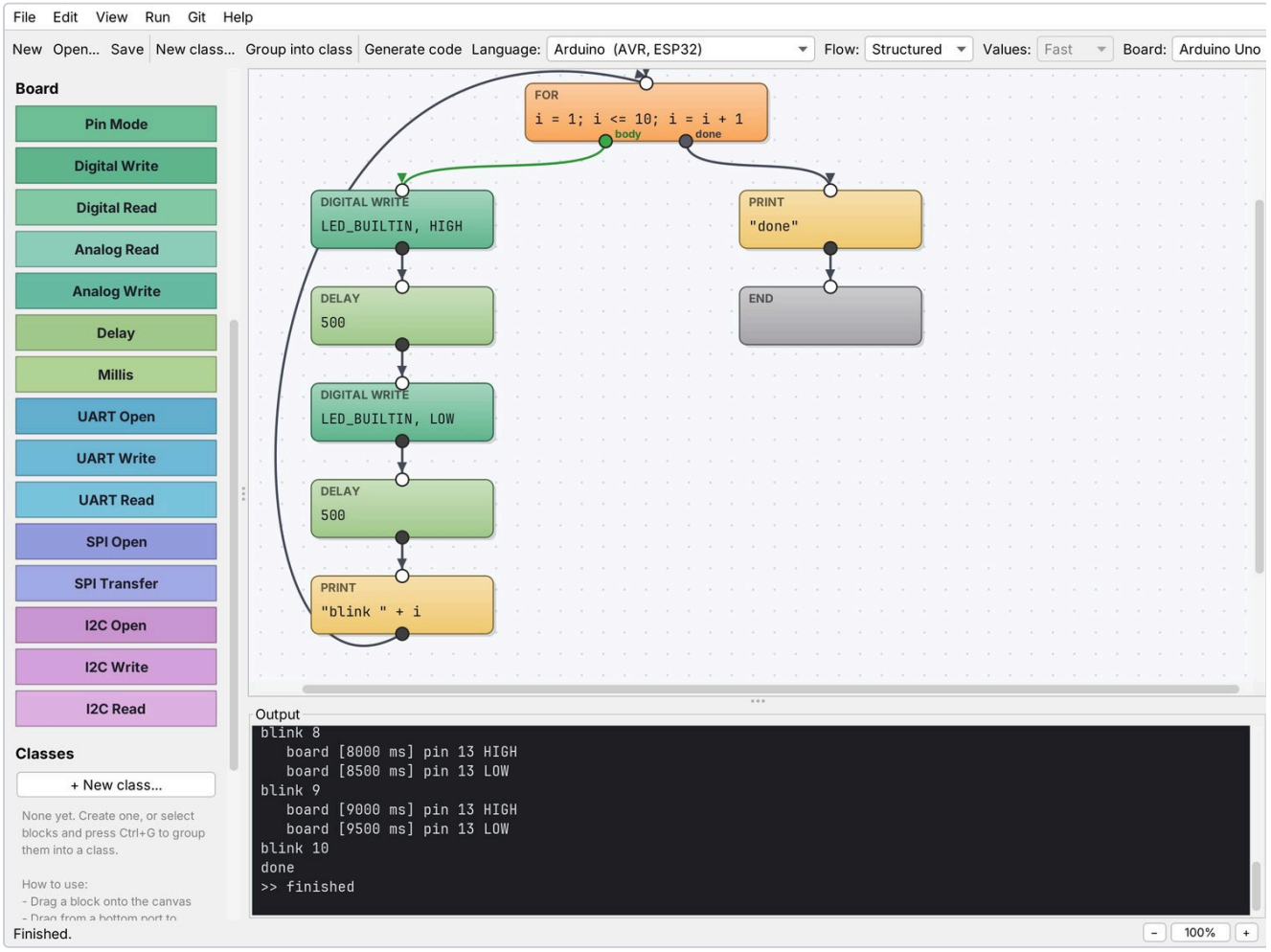


8. Ծրագրավորեք Arduino կամ ESP32 տախտակ

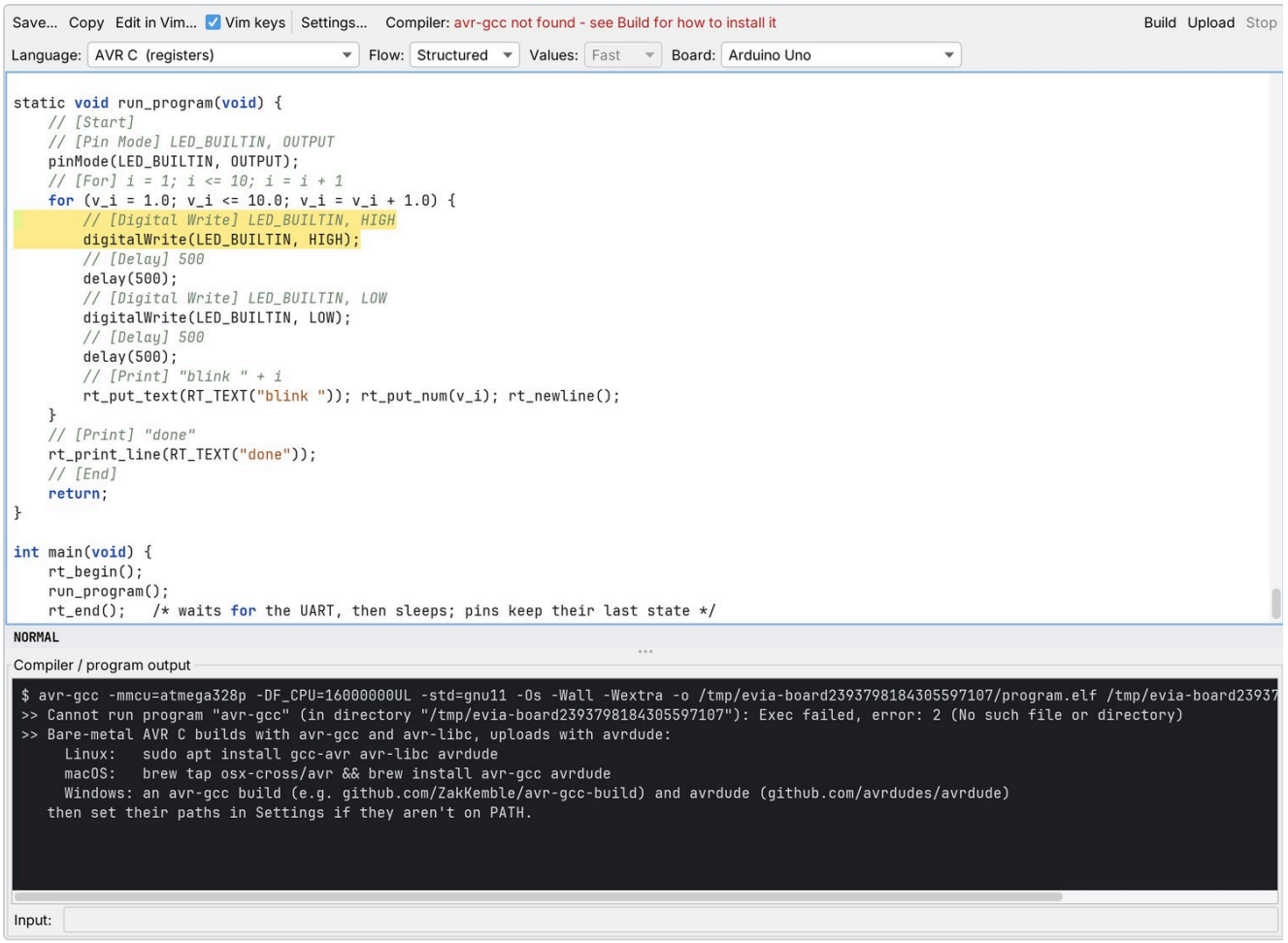
File → Open Example → Arduino and ESP32 Boards → Blink the built-in LED: **Board** բլոկները (կանաչ, գունապնակի *Board* բաժնում) կարգավորում են պինները, կարդում են կոճակներն ու սենսորները և սպասում:

Բլոկ	Կոդ	Ինչ է անում
Pin Mode	`LED_BUILTIN, OUTPUT`	`OUTPUT`, `INPUT` կամ `INPUT_PULLUP`
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
Analog Write	`5, brightness`	PWM, 0..255
Delay	`500`	սպասում է այդքան միլիվայրկյան
Millis	`t`	ծրագրի սկզբից անցած միլիվայրկյանները

Run-ը (F5) նմանակում է Arduino Uno. ելքում երևում է պինների յուրաքանչյուր փոփոխություն՝ իր ժամանակով:



Ընտրեք **Language: Arduino** և տախտակը **Board** ցանկում, ապա **Generate code** (F7). մեկ սքեթչը հարմար է 12 տախտակներից ցանկացածի համար (Uno, Nano, Mega 2560, Leonardo, Micro, Pro Mini, ESP32, ESP32-S2, -S3, -C3, -C6, -H2): **Build**-ը այն կառուցում է arduino-cli-ով, **Upload**-ը բեռնում է տախտակի վրա. Print-ը և Input-ը աշխատում են Serial Monitor-ով՝ 115200 բոդ արագությամբ: **AVR C**-ն գրում է նույն ծրագիրը առանց Arduino միջուկի՝ ուղիղ միկրոկոնտրոլերի ռեգիստրներով, և այն կառուցվում է avr-gcc-ով:



9. Շփվեք այլ չիպերի հետ՝ UART, SPI և I2C, և PIC32

File → Open Example → Arduino and ESP32 Boards-ում ևս երեք ծրագիր օգտագործում են գծերի (bus) բլոկները՝ *UART: send a byte, read it back*, *SPI: transfer a byte* և *I2C: write and read a device register*: Բլոկները *Board* խմբում են.

Բլոկ	Կոդ	Ինչ է անում
UART Open	`9600`	գործարկում է սերիական պորտը այդ բոդ արագությամբ
UART Write	`65`	ուղարկում է մեկ բայթ (0..255)
UART Read	`c`	հաջորդ ստացված բայթը, կամ -1, եթե ոչինչ չի եկել
SPI Open	`1000000`	SPI master՝ այդ հաճախականությամբ (Հց), ռեժիմ 0, ավագ բիտը առաջինը
SPI Transfer	`reply = 66`	ուղարկում է բայթ և պահում է միաժամանակ ստացված բայթը
I2C Open	`100000`	I2C master՝ այդ հաճախականությամբ (Հց)
I2C Write	`80, 0, 42`	հասցե (7 բիթ), ռեգիստր, արժեք
I2C Read	`v = 80, 0`	պահում է ռեգիստրի արժեքը, կամ -1, եթե ոչ ոք չի պատասխանում

Գիծն օգտագործելուց առաջ պետք է բացել: **Run**-ը նմանակում է դրանք. UART Write-ի բայթերը վերադառնում են UART Read-ով, SPI-ի մուտքը միացված է նրա ելքին, իսկ 80 հասցեով պատասխանում է 256 ռեգիստրով մեկ I2C սարք (80-ը 0x50-ն է. թվերը տասնորդական են):

Microchip-ի PIC32MX չիպի համար ընտրեք **Language: PIC32 C** և չիպը **Board** ցանկում (ինը չիպ՝ MX1xx/2xx, MX3xx/4xx և MX5xx/6xx/7xx ընտանիքներից), ապա **Generate code** (F7): Արդյունքը bare-metal C է Microchip-ի անվճար XC32 կոմպիլյատորի համար. ժամացույցը, պինները, PWM-ը, ADC-ն, UART-ը, SPI-ն և I2C-ն կարգավորվում են ռեգիստրներով: **Build**-ը գործարկում է `xc32-gcc` և `xc32-bin2hex`, իսկ **Upload**-ը՝ **Settings**-ում նշված հրամանը (լռելյայն՝ `pic32prog {hex}`): Ընտրված ընտանիքի պինները թվարկված են ստեղծված ֆայլի սկզբում:

Ուշադրություն. PIC32-ի կոդը ստուգվել է Microchip-ի իրական վերնագրային ֆայլերով, բայց դեռ չի կառուցվել XC32-ով և չի աշխատել չիպի վրա: Առաջին կառուցումը ձեր տախտակի վրա համարեք փորձարկում:

10. Սարքակազմ FPGA-ի համար

Ընտրեք **Լեզու՝ SystemVerilog**, **Verilog-2001** կամ **VHDL**, ապա տախտակ **Board** ցանկից՝ Digilent Nexys A7-100T, Arty A7-35 կամ A7-100, Basys 3 (Artix-7), կամ Olimex GateMateA1-EVB և Cologne Chip GateMate A1 evaluation board: Բացեք **File → Open Example → FPGA Boards → Blink** և սեղմեք **Generate code** (F7):

Յուրաքանչյուր բլոկ ավտոմատի մեկ վիճակ է և տևում է մեկ տակտ: Կոնտակտները ֆիքսվում են կառուցելիս, ուստի գրեք դրանք անուններով՝ `LED0`, `SW3`, `BTNC`, `JA1`: Ֆայլի առաջին մեկնաբանությունը պարունակում է կոնտակտների սահմանափակումները. հրամանի տողում `-o`-ի դեպքում դրանք գրվում են նաև որպես `.xdc` (Vivado) կամ `.ccf` (GateMate):

- **Build** և **Upload**՝ Artix-7-ի համար Vivado կամ բաց շղթա (ընտրվում է **Settings**-ում), GateMate-ի համար՝ yosys, nextpnr-himbaechel, gmpack և openFPGALoader:
- Թվերը ամբողջ 32-բիթանոց են, ուստի `5 / 2`-ը FPGA-ում 2 է, իսկ **Run**-ում՝ 2.5:
- Դասերը, Input, Analog Read, SPI, I2C և տեքստը փոփոխականներում դեռ չեն աջակցվում. սխալը նշում է բլոկը:

Ուշադրություն. արդյունքը սիմուլացվել և սինթեզվել է բաց գործիքներով, բայց չի անցել Vivado-ով և չի բեռնվել տախտակի մեջ:

11. Ուղարկեք թվերը MQTT-ով

Բացեք **File → Open Example → Arduino and ESP32 Boards → MQTT sensor**: Ծրագիրը ջերմաստիճան է հրապարակում բրոքերին, ինքն իրեն հրաման է ուղարկում և կարդում այն:

MQTT-ի հինգ բլոկները *Board* խմբում են՝ *MQTT Connect*, *MQTT Publish*, *MQTT Subscribe*, *MQTT Has Message* և *MQTT Read*: Ցանցի անունը, գաղտնաբառը, բրոքերը և թեմաները ֆիքսված տեքստեր են, քանի որ գրվում են ծրագրի մեջ կառուցելիս:

- **Run**-ը պատասխանում է ձեր համակարգչի բրոքերից, այնպես որ ծրագիրը կարելի է փորձել առանց սարքավորման:

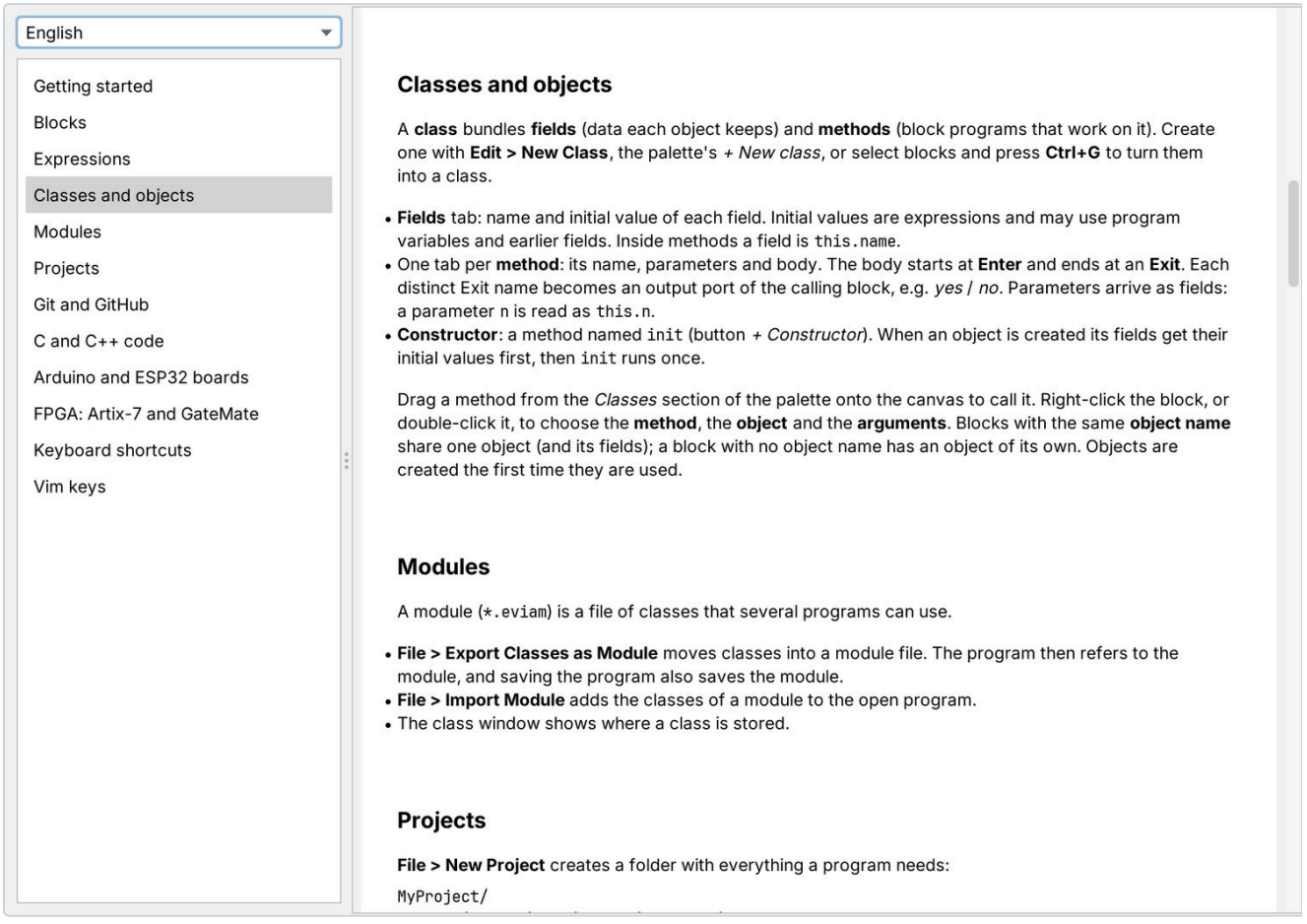
- **PIC32 C**-ն AT որոնվածքով ESP8266 կամ ESP32 մոդուլ է վարում UART2-ով, իսկ Print-ը մնում է UART1-ին:
- **Arduino**-ն **ESP32** տախտակի վրա օգտագործում է տախտակի սեփական WiFi-ը:
- AVR տախտակը մերժում է այս բլոկները՝ այն ունի մեկ սերիական պորտ, և այն զբաղեցնում է Print-ը:

Հաղորդագրությունը թիվ է: *MQTT Read*-ը տալիս է -1, երբ ոչինչ չի եկել, ուստի նախ հարցրեք *MQTT Has Message*-ին, եթե իրական -1 կարող է գալ:

Ուշադրություն. MQTT փաթեթները ստուգվում են CI-ում իրական բրոքերով, բայց ոչ WiFi մոդուլ, ոչ ESP32, ոչ PIC32 չիպ այդ ստուգման մաս չեն կազմել: MX695/795 չիպերի Ethernet պորտով MQTT-ն դեռ չի աջակցվում:

12. Ինչ անել հետո

- Կիսվեք դասերով ծրագրերի միջև. **File → Export Classes as Module…**, ապա **File → Import Module…** մեկ այլ ծրագրում:
- **File → Open Example**-ում կան պատրաստի ծրագրեր, այդ թվում `classes.evia`-ն (դաշտեր, կոնստրուկտոր, մեթոդներ, մոդուլ) և տախտակների օրինակներ:
- **Help → Tutorial (PDF)**-ը բացում է այս ուսուցումը յոթ լեզուներից որևէ մեկով:
- **Help → Guide**-ը (F1) բացատրում է յուրաքանչյուր բլոկ, արտահայտությունները, դասերը, նախագծերը, Git-ը, ստեղնային համակցությունները և Vim ստեղները:



Բոլոր հնարավորությունների մասին մանրամասները՝ README-ում (անգլերեն):