<https://arduino.ru/Reference>

<https://arduino.ru/Reference>

Официальный сайт компании Arduino по адресу[arduino.cc](http://out.arduino.ru/?redirect=http%3A%2F%2Farduino.cc%2F&baseU=https%3A%2F%2Farduino.ru%2FReference)

[https://arduino.ru/sites/default/files/framework_logo.png](https://arduino.ru/)

[**Arduino.ru**](https://arduino.ru/)

Начало формы

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | |  |

Конец формы

* [**Что такое Ардуино?**](https://arduino.ru/About)

* [**Среда разработки**](https://arduino.ru/Arduino_environment)

* [**Программирование**](https://arduino.ru/Reference)

* [**Контроллеры**](https://arduino.ru/Hardware)

* [**Форум**](https://arduino.ru/forum)

* [**Новый форум**](https://forum.arduino.ru/)

* [**Купить в России**](https://arduino.ru/buy)

**Регистрация новых пользователей и создание новых тем теперь только на новом форуме**[**https://forum.arduino.ru**](https://forum.arduino.ru/)

Программирование Ардуино

Язык программирования устройств Ардуино основан на C/C++. Он прост в освоении, и на данный момент Arduino — это, пожалуй, самый удобный способ программирования устройств на микроконтроллерах.

Базовые и полезные знания, необходимые для успешного программирования под платформу Arduino:

|  |  |
| --- | --- |
| * [Начало работы с Arduino в Windows](https://arduino.ru/Guide/Windows) * [Работа с Arduino Mini](https://arduino.ru/Guide/ArduinoMini) * [Цифровые выводы](https://arduino.ru/Tutorial/DigitalPins) * [Аналоговые входы](https://arduino.ru/Tutorial/AnalogInputPins) * [Широтно-импульсная модуляция](https://arduino.ru/Tutorial/PWM) * [Память в Arduino](https://arduino.ru/Tutorial/Memory) * [Использование аппаратных прерываний в Arduino](https://arduino.ru/Tutorial/Arduino_Interrupts) * [Перепрошивка контроллера Atmega8U2 для Arduino Uno и Mega2560](https://arduino.ru/forum/obshchii/arduino-ne-opredelyaetsya-kompyuterom-i-kak-eto-ispravit#comment-663) | * [Переменные](https://arduino.ru/Tutorial/Variables) * [Функции](https://arduino.ru/Reference/FunctionDeclaration) * [Создание библиотек для Arduino](https://arduino.ru/Hacking/LibraryTutorial) * [Использование сдвигового регистра 74HC595 для увеличения количества выходов](https://arduino.ru/Tutorial/registr_74HC595) * [Прямое управления выходами через регистры микроконтроллера Atmega](https://arduino.ru/Tutorial/Upravlenie_portami_cherez_registry) |

**Справочник языка Ардуино**

Язык Arduino можно разделить на три раздела:

|  |  |  |
| --- | --- | --- |
| Операторы   * [setup](https://arduino.ru/Reference/Setup)() * [loop](https://arduino.ru/Reference/Loop)()   **Управляющие операторы**   * [if](https://arduino.ru/Reference/If) * [if...else](https://arduino.ru/Reference/Else) * [for](https://arduino.ru/Reference/For) * [switch case](https://arduino.ru/Reference/SwitchCase) * [while](https://arduino.ru/Reference/While) * [do... while](https://arduino.ru/Reference/DoWhile) * [break](https://arduino.ru/Reference/Break) * [continue](https://arduino.ru/Reference/Continue) * [return](https://arduino.ru/Reference/Return) * [goto](https://arduino.ru/Reference/Goto)   **Синтаксис**   * [; (semicolon)](https://arduino.ru/Reference/SemiColon) * [{} (curly braces)](https://arduino.ru/Reference/Braces) * [// (single line comment)](https://arduino.ru/Reference/Comments) * [/\* \*/ (multi-line comment)](https://arduino.ru/Reference/Comments)   **Арифметические операторы**   * [= (assignment)](https://arduino.ru/Reference/Assignment) * [+ (addition)](https://arduino.ru/Reference/Arithmetic) * [- (subtraction)](https://arduino.ru/Reference/Arithmetic) * [\* (multiplication)](https://arduino.ru/Reference/Arithmetic) * [/ (division)](https://arduino.ru/Reference/Arithmetic) * [% (modulo)](https://arduino.ru/Reference/Modulo)   **Операторы сравнения**   * [==](https://arduino.ru/Reference/If) (equal to) * [!=](https://arduino.ru/Reference/If) (not equal to) * [<](https://arduino.ru/Reference/If) (less than) * [>](https://arduino.ru/Reference/If) (greater than) * [<=](https://arduino.ru/Reference/If) (less than or equal to) * [>=](https://arduino.ru/Reference/If) (greater than or equal to)   **Логические операторы**   * [&& (И)](https://arduino.ru/Reference/Boolean) * [|| (ИЛИ)](https://arduino.ru/Reference/Boolean) * [! (Отрицание)](https://arduino.ru/Reference/Boolean)   **Унарные операторы**   * [++](https://arduino.ru/Reference/Increment) (increment) * [--](https://arduino.ru/Reference/Increment) (decrement) * [+=](https://arduino.ru/Reference/IncrementCompound) (compound addition) * [-=](https://arduino.ru/Reference/IncrementCompound) (compound subtraction) * [\*=](https://arduino.ru/Reference/IncrementCompound) (compound multiplication) * [/=](https://arduino.ru/Reference/IncrementCompound) (compound division) | Данные  **Константы**   * [HIGH](https://arduino.ru/Reference/Constants) | [LOW](https://arduino.ru/Reference/Constants) * [INPUT](https://arduino.ru/Reference/Constants) | [OUTPUT](https://arduino.ru/Reference/Constants) * [true](https://arduino.ru/Reference/Constants) | [false](https://arduino.ru/Reference/Constants) * [Целочисленные константы](https://arduino.ru/Reference/IntegerConstants) * [Константы с плавающей запятой](https://arduino.ru/Reference/Fpconstants)   **Типы данных**   * [boolean](https://arduino.ru/Reference/BooleanVariables) * [char](https://arduino.ru/Reference/Char) * [byte](https://arduino.ru/Reference/Byte) * [int](https://arduino.ru/Reference/Int) * [unsigned int](https://arduino.ru/Reference/UnsignedInt) * [word](https://arduino.ru/Reference/Word) * [long](https://arduino.ru/Reference/Long) * [unsigned long](https://arduino.ru/Reference/UnsignedLong) * [float](https://arduino.ru/Reference/Float) * [double](https://arduino.ru/Reference/Double) * [string](https://arduino.ru/Reference/String) - массив символов * [String](https://arduino.ru/Reference/StringObject) - объект класса * [массив (array)](https://arduino.ru/Reference/Array) * [void](https://arduino.ru/Reference/Void)   **Преобразование типов данных**   * [char()](https://arduino.ru/Reference/CharCast) * [byte()](https://arduino.ru/Reference/ByteCast) * [int()](https://arduino.ru/Reference/IntCast) * [long()](https://arduino.ru/Reference/LongCast) * [float()](https://arduino.ru/Reference/FloatCast)   **Область видимости переменных и квалификаторы**   * [Область видимости](https://arduino.ru/Reference/Scope) * [static](https://arduino.ru/Reference/Static) * [volatile](https://arduino.ru/Reference/Volatile) * [const](https://arduino.ru/Reference/Const) | Функции  **Цифровой ввод/вывод**   * [pinMode](https://arduino.ru/Reference/PinMode)() * [digitalWrite](https://arduino.ru/Reference/DigitalWrite)() * [digitalRead](https://arduino.ru/Reference/DigitalRead)()   **Аналоговый ввод/вывод**   * [analogRead](https://arduino.ru/Reference/AnalogRead)() * [analogReference()](https://arduino.ru/Reference/AnalogReference) * [analogWrite](https://arduino.ru/Reference/AnalogWrite)()   **Дополнительные фунции ввода/вывода**   * [tone](https://arduino.ru/Reference/Tone)() * [noTone](https://arduino.ru/Reference/NoTone)() * [shiftOut](https://arduino.ru/Reference/ShiftOut)() * [pulseIn](https://arduino.ru/Reference/PulseIn)()   **Работа со временем**   * [millis](https://arduino.ru/Reference/Millis)() * [micros](https://arduino.ru/Reference/Micros)() * [delay](https://arduino.ru/Reference/Delay)() * [delayMicroseconds](https://arduino.ru/Reference/DelayMicroseconds)()   **Математические функции**   * [min](https://arduino.ru/Reference/Min)() * [max](https://arduino.ru/Reference/Max)() * [abs](https://arduino.ru/Reference/Abs)() * [constrain](https://arduino.ru/Reference/Constrain)() * [map](https://arduino.ru/Reference/Map)() * [pow](https://arduino.ru/Reference/Pow)() * [sq](https://arduino.ru/Reference/Sq)() * [sqrt](https://arduino.ru/Reference/Sqrt)()   **Тригонометрические функции**   * [sin](https://arduino.ru/Reference/Sin)() * [cos](https://arduino.ru/Reference/Cos)() * [tan](https://arduino.ru/Reference/Tan)()   **Генераторы случайных значений**   * [randomSeed](https://arduino.ru/Reference/RandomSeed)() * [random](https://arduino.ru/Reference/Random)()   **Внешние прерывания**   * [attachInterrupt()](https://arduino.ru/Reference/AttachInterrupt) * [detachInterrupt()](https://arduino.ru/Reference/DetachInterrupt)   **Функции передачи данных**   * [Serial](https://arduino.ru/Reference/Serial) |

**Библиотеки Arduino**

[Servo](https://arduino.ru/Reference/Library/Servo) — библиотека управления сервоприводами.  
[EEPROM](https://arduino.ru/Reference/Library/EEPROM) — чтение и запись энергонезависимой памяти микроконтроллера.  
[SPI](https://arduino.ru/Reference/Library/SPI) — библиотека, реализующая передачу данных через интерфейс SPI.  
[Stepper](https://arduino.ru/Reference/Library/Stepper) — библиотека управления шаговыми двигателями.

Функция Setup

**setup()**

Функция setup() вызывается, когда стартует скетч. Используется для инициализации переменных, определения режимов работы выводов, запуска используемых библиотек и т.д. Функция setup запускает только один раз, после каждой подачи питания или сброса платы Arduino.

*Пример*

|  |  |
| --- | --- |
| int buttonPin = 3; | |
|  |

|  |  |
| --- | --- |
| void setup() | |
| { |

|  |
| --- |
| Serial.begin(9600); |
| pinMode(buttonPin, INPUT); | |

|  |  |
| --- | --- |
| } | |
|  |

|  |
| --- |
|  |
| void loop() | |

|  |
| --- |
| { |
| // ... | |

|  |
| --- |
| } |

Функция Loop

**loop()**

После вызова функции setup(), которая инициализирует и устанавливает первоначальные значения, функция loop() делает точь-в-точь то, что означает её название, и крутится в цикле, позволяя вашей программе совершать вычисления и реагировать на них. Используйте её для активного управления платой Arduino.

*Пример*

|  |  |
| --- | --- |
| int buttonPin = 3; | |
|  |

|  |  |
| --- | --- |
| // setup инициализирует последовательный порт и кнопку | |
| void setup() |

|  |
| --- |
| { |
| beginSerial(9600); | |

|  |  |
| --- | --- |
| pinMode(buttonPin, INPUT); | |
| } |

|  |
| --- |
|  |
| // в цикле проверяется состояние кнопки, | |

|  |  |
| --- | --- |
| // и на последовательный порт будет отправлено сообщение, если она нажата | |
| void loop() |

|  |
| --- |
| { |
| if (digitalRead(buttonPin) == HIGH) | |

|  |  |
| --- | --- |
| serialWrite('H'); | |
| else |

|  |  |
| --- | --- |
| serialWrite('L'); | |
|  |

|  |  |
| --- | --- |
| delay(1000); | |
| }  **Управляющие операторы**   * [if](https://arduino.ru/Reference/If) * [if...else](https://arduino.ru/Reference/Else) * [for](https://arduino.ru/Reference/For) * [switch case](https://arduino.ru/Reference/SwitchCase) * [while](https://arduino.ru/Reference/While) * [do... while](https://arduino.ru/Reference/DoWhile) * [break](https://arduino.ru/Reference/Break) * [continue](https://arduino.ru/Reference/Continue) * [return](https://arduino.ru/Reference/Return) * [goto](https://arduino.ru/Reference/Goto)   Оператор If  **if (условие) и ==, !=, <, > (операторы сравнения)**  **if**, используется в сочетании с операторами сравнения, проверяет, достигнута ли истинность условия, например, превышает ли входное значение заданное число. Формат оператора **if** следующий:   |  |  | | --- | --- | | if (someVariable > 50) | | | { |  |  |  | | --- | --- | | // выполнять действия | | | } |   Программа проверяет, значение *someVariable* больше чем 50 или нет. Если да, то выполняются определенные действия. Говоря иначе, если выражение в круглых скобках истинно, выполняются операторы внутри фигурных скобок. Если нет, программа пропускает этот код.  Скобки после оператора **if** могут быть опущены. Если так сделано, только следующая строка (обозначенная точкой с запятой) становится оператором, выполняемым в операторе **if**.   |  |  | | --- | --- | | if (x > 120) digitalWrite(LEDpin, HIGH); | | |  |  |  | | --- | | if (x > 120) | | digitalWrite(LEDpin, HIGH); | |  |  | | --- | |  | | if (x > 120){ digitalWrite(LEDpin, HIGH); } | |  |  | | --- | |  | | if (x > 120){ | |  |  | | --- | | digitalWrite(LEDpin1, HIGH); | | digitalWrite(LEDpin2, HIGH); |  |  | | --- | | }                                 // все правильно |   Выражения, которые вычисляются внутри круглых скобок, могут состоять из одного или нескольких операторов.  ***Операторы сравнения***  x == y (x равно y)   x != y (x не равно y)   x < y (x меньше чем y)   x > y (x больше чем y)   x <= y (x меньше чем или равно y)   x >= y (x больше чем или равно y)  **Внимание!**  Следите, чтобы случайно не использовать знак простого равенства (например, if (x = 10)). Знак простого равенства – это оператор присваивания, и устанавливает значение х равное 10 (заносит значение 10 в переменную х). Вместо этого используйте знак двойного равенства (например, if (x == 10)), который является оператором сравнения и проверяет, х равен 10 или нет. Последнее из двух выражений будет истинно, только если х равен 10, но предыдущее выражение всегда верно.  Это связано с тем, что С вычисляет выражение if (x=10) следующим образом: значение 10 присваивается х (помним, что простой знак равенства – это **оператор присваивания**), таким образом, х теперь равен 10. Затем условный **if** вычисляет 10, которое уже равно ИСТИНА, так как любое число, неравное 0, равно ИСТИНА. Поэтому if (x=10) будет всегда иметь логическое значение ИСТИНА, которое не является желательным результатом, когда используется оператор **if**. Вдобавок, переменной х будет присвоено значение 10, что также не является желаемым действием.  **If** также может быть частью разветвленной управляющей конструкции с использованием [**if...else**](https://arduino.ru/Reference/Else)  Оператор If..else  Конструкция**if..else** предоставляет больший контроль над процессом выполнения кода, чем базовый оператор **if**, позволяя осуществлять несколько проверок, объединенных вместе. Например, аналоговый вход может быть проверен и выполнено одно действие, если на входе меньше 500, или другой действие, если на входе 500 или больше. Код при этом может выглядеть так:   |  |  | | --- | --- | | if (pinFiveInput < 500) | | | { |  |  |  | | --- | --- | | // действие A | | | } |  |  |  | | --- | --- | | else | | | { |  |  |  | | --- | --- | | // действие B | | | } |   Другой способ создания переходов со взаимоисключающими проверками использует оператор switch case.  Else позволяет делать отличную от указанной в **if** проверку, чтобы можно было осуществлять сразу несколько взаимоисключающих проверок. Каждая проверка позволяет переходить к следующему за ней оператору не раньше, чем получит логический результат ИСТИНА. Когда проверка с результатом ИСТИНА найдена, запускается вложенная в нее блок операторов, и затем программа игнорирует все следующие строки в конструкции **if..else**. Если ни одна из проверок не получила результат ИСТИНА, по умолчанию выполняется блок операторов в **else**, если последний присутствует, и устанавливается действие по умолчанию.  Отметим, что конструкция**else if**может быть использована с или без заключительного **else** и наоборот. Допускается неограниченное число таких переходов**else if**.   |  |  | | --- | --- | | if (pinFiveInput < 500) | | | { |  |  |  | | --- | --- | | // выполнять действие A | | | } |  |  |  | | --- | --- | | else if (pinFiveInput >= 1000) | | | { |  |  |  | | --- | --- | | // выполнять действие B | | | } |  |  |  | | --- | --- | | else | | | { |  |  |  | | --- | --- | | // выполнять действие  C | | | } |   Другой способ создания переходов со взаимоисключающими проверками использует оператор [switch case](https://arduino.ru/Reference/SwitchCase).  Оператор Switch  Подобно конструкции **if, switch...case** управляет процессом выполнения программы, позволяя программисту задавать альтернативный код, который будет выполняться при разных условиях. В частности, оператор **switch** сравнивает значение переменной со значением, определенном в операторах case. Когда найден оператор **case**, значение которого равно значению переменной,   выполняется программный код в этом операторе.  Ключевое слово **break** является командой выхода из оператора **case** и обычно используется в конце каждого **case**. Без оператора **break** оператор**switch**будет продолжать вычислять  следующие выражения, пока не достигнет **break**или конец оператора**switch.**  **Пример**   |  |  | | --- | --- | | switch (var) { | | | case 1: |  |  |  | | --- | --- | | //выполняется, когда var равно 1 | | | break; |  |  | | --- | | case 2: | | //выполняется когда  var равно 2 | |  |  | | --- | | break; | | default: |  |  |  | | --- | --- | | // выполняется, если не выбрана ни одна альтернатива | | | // default необязателен |  |  | | --- | | } |   **Синтаксис:**   |  |  | | --- | --- | | switch (var) { | | | case label: |  |  |  | | --- | --- | | // код для выполнения | | | break; |  |  | | --- | | case label: | | // код для выполнения | |  |  |  | | --- | --- | | break; | | | default: |  |  |  | | --- | --- | | // код для выполнения | | | } |   **Параметры:**   * *var:* переменная, которая вычисляется для сравнения с вариантами в case * *label*: значение, с которым сравнивается значение переменной   **Смотреть также:**  [if...else](https://arduino.ru/Reference/Else)  Оператор For  Конструкция **for** используется для повторения блока операторов, заключенных в фигурные скобки. Счетчик приращений обычно используется для приращения и завершения цикла. Оператор **for** подходит для любых повторяющихся действий и часто используется в сочетании с массивами коллекций данных/выводов.  Заголовок цикла **for** состоит из трех частей:  **for** (**initialization**;**condition**;**increment**) {операторы выполняющиеся в цикле}  Инициализация (Initialization) выполняется самой первой и один раз. Каждый раз в цикле проверяется условие (condition), если оно верно, выполняется блок операторов и приращение (increment), затем условие проверяется вновь. Когда логическое значение условия становится ложным, цикл завершается.  **Пример**   |  |  | | --- | --- | | // Затемнение светодиода с использованием ШИМ-вывода | | |  |  |  |  | | --- | --- | | int PWMpin = 10; // Светодиод последовательно с резистором 470 ом на 10 выводов | | |  |  |  |  | | --- | --- | | void setup() | | | { |  |  |  | | --- | --- | | // настройка не нужна | | | } |  |  | | --- | |  | |  |  |  |  | | --- | --- | | void loop() | | | { |  |  |  | | --- | --- | | for (int i=0; i <= 255; i++){ | | | analogWrite(PWMpin, i); |  |  |  | | --- | --- | | delay(10); | | | } |  |  | | --- | | } |   **Советы по применению**  Цикл **for** в Си гораздо более гибкий, чем циклы **for** в других языках программирования, например, в Бейсике. Любой из трех или все три элемента заголовка  могут быть опущены, хотя точки с запятой обязательны. Также операторы для инициализации, условия и приращения цикла могут быть любым допустимым в Си операторами с независимыми переменными, и использовать любой тип данных Си, включая данные с плавающей точкой (floats). Эти необычные для цикла **for**  типы операторов позволяют обеспечить программное решение некоторых нестандартных проблем.  Например, использование умножения в операторе счетчика цикла позволяет создавать логарифмическую прогрессию:   |  |  | | --- | --- | | for(int x = 2; x < 100; x = x \* 1.5){ | | | println(x); |  |  | | --- | | } |   Генерируется: 2,3,4,6,9,13,19,28,42,63,94  Другой пример, плавное уменьшение или увеличение уровня сигнала на светодиод с помощью одного цикла **for:**  [код](https://arduino.ru/Reference/For#viewSource)  [расечатать](https://arduino.ru/Reference/For#printSource)[?](https://arduino.ru/Reference/For#about)   |  |  | | --- | --- | | void loop() | | | { |  |  | | --- | | int x = 1; | | for (int i = 0; i > -1; i = i + x){ | |  |  | | --- | | analogWrite(PWMpin, i); | | if (i == 255) x = -1;             // переключение управления на максимуме | |  |  |  | | --- | --- | | delay(10); | | | } |  |  | | --- | | } |   **Смотреть также**   * [while](https://arduino.ru/Reference/While)   Оператор Switch  Подобно конструкции **if, switch...case** управляет процессом выполнения программы, позволяя программисту задавать альтернативный код, который будет выполняться при разных условиях. В частности, оператор **switch** сравнивает значение переменной со значением, определенном в операторах case. Когда найден оператор **case**, значение которого равно значению переменной,   выполняется программный код в этом операторе.  Ключевое слово **break** является командой выхода из оператора **case** и обычно используется в конце каждого **case**. Без оператора **break** оператор**switch**будет продолжать вычислять  следующие выражения, пока не достигнет **break**или конец оператора**switch.**  **Пример**   |  |  | | --- | --- | | switch (var) { | | | case 1: |  |  |  | | --- | --- | | //выполняется, когда var равно 1 | | | break; |  |  | | --- | | case 2: | | //выполняется когда  var равно 2 | |  |  | | --- | | break; | | default: |  |  |  | | --- | --- | | // выполняется, если не выбрана ни одна альтернатива | | | // default необязателен |  |  | | --- | | } |   **Синтаксис:**   |  |  | | --- | --- | | switch (var) { | | | case label: |  |  |  | | --- | --- | | // код для выполнения | | | break; |  |  | | --- | | case label: | | // код для выполнения | |  |  |  | | --- | --- | | break; | | | default: |  |  |  | | --- | --- | | // код для выполнения | | | } |   **Параметры:**   * *var:* переменная, которая вычисляется для сравнения с вариантами в case * *label*: значение, с которым сравнивается значение переменной   **Смотреть также:**  [if...else](https://arduino.ru/Reference/Else) |

Циклы while

**While** будет вычислять в цикле непрерывно и бесконечно до тех пор, пока выражение в круглых скобках, () не станет равно логическому ЛОЖНО. Что-то должно изменять значение проверяемой переменной, иначе выход из цикла **while** никогда не будет достигнут. Это изменение может происходить как в программном коде, например, при увеличении переменной, так и во внешних условиях, например, при тестировании датчика.

**Синтаксис**

|  |  |
| --- | --- |
| while(выражение){ | |
| // оператор(ы) |

|  |
| --- |
| } |

**Параметры**

выражение - (булевский) C-оператор, который возвращает значение истина или ложь

**Пример**

[код](https://arduino.ru/Reference/While#viewSource)

[расечатать](https://arduino.ru/Reference/While#printSource)[?](https://arduino.ru/Reference/While#about)

|  |  |
| --- | --- |
| var = 0; | |
|  |

|  |
| --- |
| while(var < 200){ |
| // выполнить что-то, повторив 200 раз | |

|  |  |
| --- | --- |
| var++; | |
| } |

**См. также**

* [do ... while](https://arduino.ru/Reference/DoWhile)
* [break](https://arduino.ru/Reference/Break)
* [continue](https://arduino.ru/Reference/Continue)
* [return](https://arduino.ru/Reference/Return)

[Назад к Справочнику](https://arduino.ru/Reference)

do ... while

Цикл **do** работает так же, как и цикл **while**, за исключением того, что условие проверяется в конце цикла, таким образом, цикл **do** будет *всегда* выполняться хотя бы раз.

|  |  |
| --- | --- |
| do | |
| { |

|  |  |
| --- | --- |
| // последовательность операторов | |
| } while (проверка условия); |

**Пример**

[код](https://arduino.ru/Reference/DoWhile#viewSource)

[расечатать](https://arduino.ru/Reference/DoWhile#printSource)[?](https://arduino.ru/Reference/DoWhile#about)

|  |  |
| --- | --- |
| do | |
| { |

|  |  |
| --- | --- |
| delay(50);          // подождать, пока датчики стабилизируются | |
| x = readSensors(); // проверить датчики |

|  |
| --- |
| } while (x < 100); |

**См. также**

* [while](https://arduino.ru/Reference/While)
* [break](https://arduino.ru/Reference/Break)
* [continue](https://arduino.ru/Reference/Continue)
* [return](https://arduino.ru/Reference/Return)

[Назад к Справочнику](https://arduino.ru/Reference)

break

**Break** используется для принудительного выхода из циклов **do, for** или **while**, не дожидаясь завершения цикла по условию. Он также используется для выхода из оператора **switch**

**Пример**

[код](https://arduino.ru/Reference/Break#viewSource)

[расечатать](https://arduino.ru/Reference/Break#printSource)[?](https://arduino.ru/Reference/Break#about)

|  |  |
| --- | --- |
| for (x = 0; x < 255; x ++) | |
| { |

|  |
| --- |
| digitalWrite(PWMpin, x); |
| sens = analogRead(sensorPin); | |

|  |  |
| --- | --- |
| if (sens > threshold){      // выходим из цикла если есть сигнал с датчика | |
| x = 0; |

|  |  |
| --- | --- |
| break; | |
| } |

|  |  |
| --- | --- |
| delay(50); | |
| } |

**См. также**

* [while](https://arduino.ru/Reference/While)
* [do ... while](https://arduino.ru/Reference/DoWhile)
* [continue](https://arduino.ru/Reference/Continue)
* [return](https://arduino.ru/Reference/Return)

[Назад к Справочнику](https://arduino.ru/Reference)

continue

Оператор **continue** пропускает оставшиеся операторы в текущем шаге цикла. Вместо них выполняется проверка условного выражения цикла, которая происходит при каждой следующей итерации.

**Пример**

[код](https://arduino.ru/Reference/Continue#viewSource)

[расечатать](https://arduino.ru/Reference/Continue#printSource)[?](https://arduino.ru/Reference/Continue#about)

|  |  |
| --- | --- |
| for (x = 0; x < 255; x ++) | |
| { |

|  |  |
| --- | --- |
| if (x > 40 && x < 120){      // если истина то прыгаем сразу на следующую итерацию цикла | |
| continue; |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| digitalWrite(PWMpin, x); | |
| delay(50); |

|  |
| --- |
|  |
| } | |

**См. также**

* [while](https://arduino.ru/Reference/While)
* [do ... while](https://arduino.ru/Reference/DoWhile)
* [break](https://arduino.ru/Reference/Break)
* [return](https://arduino.ru/Reference/Return)

[Назад к Справочнику](https://arduino.ru/Reference)

return

Прекращает вычисления в функции и возвращает значение из прерванной функции в вызывающую, если это нужно.

**Синтаксис**

*return;*

*return значение;* // обе формы допустимы

**Параметры**

*Значение*: переменная или константа любого типа

**Примеры:**

Функция сравнивает значение на датчике входа с пороговым

[код](https://arduino.ru/Reference/Return#viewSource)

[расечатать](https://arduino.ru/Reference/Return#printSource)[?](https://arduino.ru/Reference/Return#about)

|  |
| --- |
| int checkSensor(){ |
| if (analogRead(0) > 400) { | |

|  |  |
| --- | --- |
| return 1; | |
| else{ |

|  |  |
| --- | --- |
| return 0; | |
| } |

|  |
| --- |
| } |

С помощью ключевого слова return удобно тестировать блоки кода без «закомментирования» больших кусков с возможным ошибочным кодом.

|  |  |
| --- | --- |
| void loop(){ | |
|  |

|  |  |
| --- | --- |
| // здесь блестящая идея тестирования кода | |
|  |

|  |  |
| --- | --- |
| return; | |
|  |

|  |  |
| --- | --- |
| // оставшаяся часть неправильно функционирующего варианта здесь | |
| // этот код никогда не будет выполняться |

|  |
| --- |
|  |
| } | |

**См. также**

* [while](https://arduino.ru/Reference/While)
* [do ... while](https://arduino.ru/Reference/DoWhile)
* [break](https://arduino.ru/Reference/Break)
* [continue](https://arduino.ru/Reference/Continue)

[Назад к Справочнику](https://arduino.ru/Reference)

Оператор goto

Условное «перемещение» выполнения программы к определенной метке-указателю в самой программе, при этом пропускается весь код до самой метки, а исполняется - после нее.

**Синтаксис:**

|  |  |
| --- | --- |
| label: | |
| // |

|  |  |
| --- | --- |
| // какой-либо код | |
| // |

|  |
| --- |
| goto label; // переходим к метке label |

**Замечание по использованию**

Использование **goto** не рекомендуется в С программировании, многие авторы книг не советуют его применять вообще, так как это не является необходимым(с их точки зрения). Причины их негодования заключаются в том, что программист при частом использовании в коде, команды **goto** - может запустить программу в бесконечный цикл, который потом трудно будет найти – отладка программы значительно усложниться. С другой стороны, если взгянуть на ассемблерный код, то там часто используется подобный переход по метке.

При разумном применении, команда может значительно упростить код программы и сохранить время программиста. Например, в случае необходимости выхода из глубоких циклов [**for**](https://arduino.ru/Reference/For), [**while**](https://arduino.ru/Reference/While), проверок [**if**](https://arduino.ru/Reference/If) и прочих многократно вложенных контрукций.

**Пример**

|  |
| --- |
| for(byte r = 0; r < 255; r++){ |
| for(byte g = 255; g > -1; g--){ | |

|  |
| --- |
| for(byte b = 0; b < 255; b++){ |
| if (analogRead(0) > 250){ goto bailout;} | |

|  |  |
| --- | --- |
| // еще код | |
| } |

|  |  |
| --- | --- |
| } | |
| } |

|  |
| --- |
| bailout: |

**Смотрите также**

* [if](https://arduino.ru/Reference/If)
* [switch case](https://arduino.ru/Reference/SwitchCase)
* [do.. while](https://arduino.ru/Reference/DoWhile)

[Справочник языка](https://arduino.ru/Reference)

**Синтаксис**

* [; (semicolon)](https://arduino.ru/Reference/SemiColon)
* [{} (curly braces)](https://arduino.ru/Reference/Braces)
* [// (single line comment)](https://arduino.ru/Reference/Comments)
* [/\* \*/ (multi-line comment)](https://arduino.ru/Reference/Comments)

; (точка с запятой)

Синтаксис **; (точка с запятой)** используется для обозначения конца оператора.

***Пример***

int a = 13;

***Подсказка***

Забытая в конце строки точка с запятой приводит к ошибке компиляции. Текст ошибки может быть либо видимым и ссылаться на пропущенную точку с запятой, либо нет. Если встречается непонятная или похожая на нелогичную ошибка компиляции, одним из первых действий должна быть проверка пропущенных точек с запятой, в коде, непосредственно предшествующем строке, в которой компилятор выдал предупреждение.

{} (фигурные скобки)

Фигурные скобки **{}** (также называются просто «скобки») – важный элемент языка программирования С. Они используются в нескольких различных конструкциях, приведенных ниже, и это может иногда сбивать с толку начинающих.

Открывающая скобка “**{**” должна всегда сопровождаться закрывающей скобкой “**}**”. Это условие, известное как парность (симметричность) фигурных скобок. Arduino IDE (интегрированная среда разработчика) включает подходящий инструмент для проверки парности скобок. Достаточно выделить скобку, или даже поставить курсор сразу же за скобкой, как будет подсвечена её логическая пара.

Сейчас эта возможность работает с ошибкой, так как IDE часто ищет (некорректно) скобку в тексте, который «закомментирован».

Начинающие программисты или программисты, перешедшие на Си с Бейсика, часто считают использование фигурных скобок сбивающим с толку или пугающим. В конце концов, одни и те же фигурные скобки заменяют оператор RETURN в подпрограммах (функциях), оператор ENDIF в условных циклах и оператор NEXT в циклах FOR.

Поскольку использование фигурных скобок столь многогранно, хорошей практикой программирования будет печатать закрывающую фигурную скобку сразу после того, как напечатана открывающая скобка, когда вставляется конструкция, для которой нужно использовать фигурные скобки. Затем возвращаем курсор в позицию между фигурными скобками и начинаем вводить операторы. Ваши скобки всегда будут парными и не лишат вас душевного равновесия.

Непарные скобки могут часто приводить к скрытым, непонятным ошибкам компиляции, которые сложно отследить в большой программе. Из-за их разного использования, скобки также невероятно важны в синтаксической правильности программы и перемещение скобки на одну или две строки часто приводят к значительному воздействию на логику программы.

**Основные способы использования фигурных скобок**

***Функции***

|  |  |  |
| --- | --- | --- |
| 1 | void НазваниеФункции(тип данных аргумента){ | |
| 2 |  |

|  |  |  |
| --- | --- | --- |
| 3 | оператор(ы) | |
| 4 |  |

|  |  |
| --- | --- |
| 5 | } |

***Циклы***

|  |  |  |
| --- | --- | --- |
| 1 | while (логическое выражение) | |
| 2 | { |

|  |  |
| --- | --- |
| 3 |  |
| 4 | оператор(ы) | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | } | |
| 1 | do | |

|  |  |  |
| --- | --- | --- |
| 2 | { | |
| 3 |  |

|  |  |  |
| --- | --- | --- |
| 4 | оператор(ы) | |
| 5 |  |

|  |  |
| --- | --- |
| 6 | } while (логическое выражение); |

|  |  |  |
| --- | --- | --- |
| 1 | for (инициализация; условие окончания цикла; приращения цикла) | |
| 2 | { |

|  |  |
| --- | --- |
| 3 |  |
| 4 | оператор(ы) | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | } | |

***Условные операторы***

|  |  |  |
| --- | --- | --- |
| 01 | if (логическое выражение) | |
| 02 | { |

|  |  |
| --- | --- |
| 03 |  |
| 04 | оператор(ы) | |

|  |  |
| --- | --- |
| 05 |  |
| 06 | } | |

|  |  |  |
| --- | --- | --- |
| 07 | else if (логическое выражение) | |
| 08 | { |

|  |  |
| --- | --- |
| 09 |  |
| 10 | оператор(ы) | |

|  |  |
| --- | --- |
| 11 |  |
| 12 | } | |

|  |  |  |
| --- | --- | --- |
| 13 | else | |
| 14 | { |

|  |  |
| --- | --- |
| 15 |  |
| 16 | оператор(ы) | |

|  |  |
| --- | --- |
| 17 |  |
| 18 | } | |

Комментарии

Комментарии – это строки в программе, которые используются для информирования вас самих или других о том, как работает программа. Они игнорируются компилятором и не экспортируются в процессор, таким образом, они не занимают место в памяти микроконтроллера Atmega.

Комментарии предназначены только для того, чтобы помочь вам понять (или вспомнить), как работает ваша программа или  объяснить это другим. Есть два способа пометить строку как комментарий:

**Пример**

|  |  |
| --- | --- |
| x = 5; // Это комментарий в одной строке. Все после двойного слэша – комментарий | |
| // до конца строки |

|  |
| --- |
|  |
|  |

|  |  |
| --- | --- |
| /\* это многострочный комментарий – используйте его для закоментирования целых кусков кода | |
|  |

|  |  |
| --- | --- |
| if (gwb == 0){   // комментарий в строке допустим внутри многострочного комментария | |
| // но не другой многострочный комментарий |

|  |
| --- |
|  |
| } | |

|  |
| --- |
|  |
| // не забывайте «закрывать» комментарии – они должны быть парными! | |

|  |
| --- |
| \*/ |

**Подсказка**

Во время экспериментов с кодом, «закомментирование» частей программы – подходящий способ удаления строк, в которых могут быть ошибки. Так строки в коде остаются, но превращаются в комментарии, и компилятор просто игнорирует их. Это может быть особенно полезно при локализации проблемы, или когда не получается скомпилировать программу, а сообщение об ошибке при компиляции скрыто или бесполезно.

Комментарии

Комментарии – это строки в программе, которые используются для информирования вас самих или других о том, как работает программа. Они игнорируются компилятором и не экспортируются в процессор, таким образом, они не занимают место в памяти микроконтроллера Atmega.

Комментарии предназначены только для того, чтобы помочь вам понять (или вспомнить), как работает ваша программа или  объяснить это другим. Есть два способа пометить строку как комментарий:

**Пример**

|  |  |
| --- | --- |
| x = 5; // Это комментарий в одной строке. Все после двойного слэша – комментарий | |
| // до конца строки |

|  |
| --- |
|  |
|  |

|  |  |
| --- | --- |
| /\* это многострочный комментарий – используйте его для закоментирования целых кусков кода | |
|  |

|  |  |
| --- | --- |
| if (gwb == 0){   // комментарий в строке допустим внутри многострочного комментария | |
| // но не другой многострочный комментарий |

|  |
| --- |
|  |
| } | |

|  |
| --- |
|  |
| // не забывайте «закрывать» комментарии – они должны быть парными! | |

|  |
| --- |
| \*/ |

**Подсказка**

Во время экспериментов с кодом, «закомментирование» частей программы – подходящий способ удаления строк, в которых могут быть ошибки. Так строки в коде остаются, но превращаются в комментарии, и компилятор просто игнорирует их. Это может быть особенно полезно при локализации проблемы, или когда не получается скомпилировать программу, а сообщение об ошибке при компиляции скрыто или бесполезно.

Арифметические операторы

= оператор присваивания

Присваивает переменной слева от оператора значение переменной или выражения, находящееся справа.

**Пример**

|  |
| --- |
| int sensVal;                 // объявление переменной типа integer |
| sensVal = analogRead(0);       // присваивание переменной sensVal, значения, считанное с аналогового входа 0 | |

**Важно**

Переменная слева от оператора присваивания (=) должна быть способна сохранить присваиваемое значение. Если оно выходит за диапазон допустимых значений, то сохраненное значение будет не верно.

Необходимо различать оператор присваивания (=) и [оператор сравнения](https://arduino.ru/Reference/If) (== двойной знак равенства), который осуществляет проверку на равенство.

**Смотрите также**

* [Операторы сравнения, if](https://arduino.ru/Reference/If)
* [int](https://arduino.ru/Reference/Int)
* [long](https://arduino.ru/Reference/Long)
* [char](https://arduino.ru/Reference/Char)

[Справочник языка](https://arduino.ru/Reference)

Оператор If

**if (условие) и ==, !=, <, > (операторы сравнения)**

**if**, используется в сочетании с операторами сравнения, проверяет, достигнута ли истинность условия, например, превышает ли входное значение заданное число. Формат оператора **if** следующий:

|  |  |
| --- | --- |
| if (someVariable > 50) | |
| { |

|  |  |
| --- | --- |
| // выполнять действия | |
| } |

Программа проверяет, значение *someVariable* больше чем 50 или нет. Если да, то выполняются определенные действия. Говоря иначе, если выражение в круглых скобках истинно, выполняются операторы внутри фигурных скобок. Если нет, программа пропускает этот код.

Скобки после оператора **if** могут быть опущены. Если так сделано, только следующая строка (обозначенная точкой с запятой) становится оператором, выполняемым в операторе **if**.

|  |  |
| --- | --- |
| if (x > 120) digitalWrite(LEDpin, HIGH); | |
|  |

|  |
| --- |
| if (x > 120) |
| digitalWrite(LEDpin, HIGH); | |

|  |
| --- |
|  |
| if (x > 120){ digitalWrite(LEDpin, HIGH); } | |

|  |
| --- |
|  |
| if (x > 120){ | |

|  |
| --- |
| digitalWrite(LEDpin1, HIGH); |
| digitalWrite(LEDpin2, HIGH); |

|  |
| --- |
| }                                 // все правильно |

Выражения, которые вычисляются внутри круглых скобок, могут состоять из одного или нескольких операторов.

***Операторы сравнения***

x == y (x равно y)

 x != y (x не равно y)

 x < y (x меньше чем y)

 x > y (x больше чем y)

 x <= y (x меньше чем или равно y)

 x >= y (x больше чем или равно y)

**Внимание!**

Следите, чтобы случайно не использовать знак простого равенства (например, if (x = 10)). Знак простого равенства – это оператор присваивания, и устанавливает значение х равное 10 (заносит значение 10 в переменную х). Вместо этого используйте знак двойного равенства (например, if (x == 10)), который является оператором сравнения и проверяет, х равен 10 или нет. Последнее из двух выражений будет истинно, только если х равен 10, но предыдущее выражение всегда верно.

Это связано с тем, что С вычисляет выражение if (x=10) следующим образом: значение 10 присваивается х (помним, что простой знак равенства – это **оператор присваивания**), таким образом, х теперь равен 10. Затем условный **if** вычисляет 10, которое уже равно ИСТИНА, так как любое число, неравное 0, равно ИСТИНА. Поэтому if (x=10) будет всегда иметь логическое значение ИСТИНА, которое не является желательным результатом, когда используется оператор **if**. Вдобавок, переменной х будет присвоено значение 10, что также не является желаемым действием.

**If** также может быть частью разветвленной управляющей конструкции с использованием [**if...else**](https://arduino.ru/Reference/Else)

**Сложение, вычитание, умножение и деление**

Операторы **+**, **-**, **\*** и **/** соответственно, возвращают результат выполнения арифметических действий над двумя операндами. Возвращаемый результат будет зависеть от типа данных операндов, например, 9 / 4 возвратит 2, т.к. операнды 9 и 4 имеют [тип **int**](https://arduino.ru/Reference/Int). Также следует следить за тем, чтобы результат не вышел за диапазон допустимых значений для используемого типа данных. Так, например, сложение 1 с переменной типа **int** и значением 32 767 возвратит -32 768. Если операнды имеют разные типы, то тип с более "широким" диапазоном будет использован для вычислений.

Если один из операндов имеет тип[float](https://arduino.ru/Reference/Float)или [double](https://arduino.ru/Reference/Double), то арифметика "с плавающей запятой" будет использована для вычислений.

**Пример**

|  |
| --- |
| y = y + 3; |
| x = x - 7; |

|  |
| --- |
| i = j \* 6; |
| r = r / 5; |

**Синтаксис**

[код](https://arduino.ru/Reference/Arithmetic#viewSource)

[расечатать](https://arduino.ru/Reference/Arithmetic#printSource)[?](https://arduino.ru/Reference/Arithmetic#about)

|  |
| --- |
| result = value1 + value2; |
| result = value1 - value2; |

|  |
| --- |
| result = value1 \* value2; |
| result = value1 / value2; |

**Параметры**

* value1: любая переменная или константа
* value2: любая переменная или константа

**Советы по использованию**

* Помните, что [целочисленные константы](https://arduino.ru/Reference/IntegerConstants) воспринимаются компилятором как тип **int**, следите за вхождением результата в диапазон допустимых значений
* Вычисления с "плавающей запятой" выполняются дольше чем целочисленные.

[Справочник языка](https://arduino.ru/Reference)

% оператор

Возвращает остаток от деления одного [целого (**int)**](https://arduino.ru/Reference/Int) операнда на другой.

**Синтаксис**

|  |
| --- |
| result = dividend % divisor |

**Параметры**

* dividend: делимое
* divisor: делитель

**Возвращаемое значение**

Остаток от деления.

**Пример**

[код](https://arduino.ru/Reference/Modulo#viewSource)

[расечатать](https://arduino.ru/Reference/Modulo#printSource)[?](https://arduino.ru/Reference/Modulo#about)

|  |
| --- |
| x = 7 % 5;   // x имеет значение 2 |
| x = 9 % 5;   // x имеет значение 4 |

|  |
| --- |
| x = 5 % 5;   // x имеет значение 0 |
| x = 4 % 5;   // x имеет значение 4 |

**Советы по использованию %**

Нельзя применить к типу [float](https://arduino.ru/Reference/Float).

[Справочник языка](https://arduino.ru/Reference)

**Операторы сравнения**

* [==](https://arduino.ru/Reference/If) (equal to)
* [!=](https://arduino.ru/Reference/If) (not equal to)
* [<](https://arduino.ru/Reference/If) (less than)
* [>](https://arduino.ru/Reference/If) (greater than)
* [<=](https://arduino.ru/Reference/If) (less than or equal to)
* [>=](https://arduino.ru/Reference/If) (greater than or equal to)

Оператор If

**if (условие) и ==, !=, <, > (операторы сравнения)**

**if**, используется в сочетании с операторами сравнения, проверяет, достигнута ли истинность условия, например, превышает ли входное значение заданное число. Формат оператора **if** следующий:

|  |  |
| --- | --- |
| if (someVariable > 50) | |
| { |

|  |  |
| --- | --- |
| // выполнять действия | |
| } |

Программа проверяет, значение *someVariable* больше чем 50 или нет. Если да, то выполняются определенные действия. Говоря иначе, если выражение в круглых скобках истинно, выполняются операторы внутри фигурных скобок. Если нет, программа пропускает этот код.

Скобки после оператора **if** могут быть опущены. Если так сделано, только следующая строка (обозначенная точкой с запятой) становится оператором, выполняемым в операторе **if**.

|  |  |
| --- | --- |
| if (x > 120) digitalWrite(LEDpin, HIGH); | |
|  |

|  |
| --- |
| if (x > 120) |
| digitalWrite(LEDpin, HIGH); | |

|  |
| --- |
|  |
| if (x > 120){ digitalWrite(LEDpin, HIGH); } | |

|  |
| --- |
|  |
| if (x > 120){ | |

|  |
| --- |
| digitalWrite(LEDpin1, HIGH); |
| digitalWrite(LEDpin2, HIGH); |

|  |
| --- |
| }                                 // все правильно |

Выражения, которые вычисляются внутри круглых скобок, могут состоять из одного или нескольких операторов.

***Операторы сравнения***

x == y (x равно y)

 x != y (x не равно y)

 x < y (x меньше чем y)

 x > y (x больше чем y)

 x <= y (x меньше чем или равно y)

 x >= y (x больше чем или равно y)

**Внимание!**

Следите, чтобы случайно не использовать знак простого равенства (например, if (x = 10)). Знак простого равенства – это оператор присваивания, и устанавливает значение х равное 10 (заносит значение 10 в переменную х). Вместо этого используйте знак двойного равенства (например, if (x == 10)), который является оператором сравнения и проверяет, х равен 10 или нет. Последнее из двух выражений будет истинно, только если х равен 10, но предыдущее выражение всегда верно.

Это связано с тем, что С вычисляет выражение if (x=10) следующим образом: значение 10 присваивается х (помним, что простой знак равенства – это **оператор присваивания**), таким образом, х теперь равен 10. Затем условный **if** вычисляет 10, которое уже равно ИСТИНА, так как любое число, неравное 0, равно ИСТИНА. Поэтому if (x=10) будет всегда иметь логическое значение ИСТИНА, которое не является желательным результатом, когда используется оператор **if**. Вдобавок, переменной х будет присвоено значение 10, что также не является желаемым действием.

**If** также может быть частью разветвленной управляющей конструкции с использованием [**if...else**](https://arduino.ru/Reference/Else)

**Логические операторы**

* [&& (И)](https://arduino.ru/Reference/Boolean)
* [|| (ИЛИ)](https://arduino.ru/Reference/Boolean)
* [! (Отрицание)](https://arduino.ru/Reference/Boolean)

Логические операторы

Логические операторы чаще всего используются в проверке условия оператора [if](https://arduino.ru/Reference/If). Базовые сведения о  логических операциях, смотрите в [Википедии](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F).

**&& (логическое И)**

Истина, если оба операнда истина (true).

|  |  |
| --- | --- |
| if (digitalRead(2) == HIGH  && digitalRead(3) == HIGH) { // считывает состояние двух портов | |
| // ... |

|  |
| --- |
| } |

Истина если оба порта вход/выхода [HIGH](https://arduino.ru/Reference/Constants)

**|| (логическое ИЛИ)**

Истина, если хотя бы один операнд истина, например:

24 24

|  |  |
| --- | --- |
| if (x > 0 || y > 0) { | |
| // ... |

|  |
| --- |
| } |

будет верно (истина) если x или y больше 0.

**! (логическое отрицание)**

True, если операнд false, и наоборот, например:

|  |  |
| --- | --- |
| if (!x) { | |
| // ... |

|  |
| --- |
| } |

условие верно, если x - false (x равно 0).

*Важно* различать логический оператор "И" - **&&** и битовый оператор "И" - **&**. Тоже самое относится к логическому оператору "ИЛИ" - **||** и битовому оператору "ИЛИ" - **|**.

**Пример**

|  |
| --- |
| if (a >= 10 && a <= 20){}   // условие верно, если a больше или равно 10, но меньше или равно 20 |

**Смотрите также**

* [if](https://arduino.ru/Reference/If)

[Справочник языка](https://arduino.ru/Reference)

Логические операторы

Логические операторы чаще всего используются в проверке условия оператора [if](https://arduino.ru/Reference/If). Базовые сведения о  логических операциях, смотрите в [Википедии](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F).

**&& (логическое И)**

Истина, если оба операнда истина (true).

|  |  |
| --- | --- |
| if (digitalRead(2) == HIGH  && digitalRead(3) == HIGH) { // считывает состояние двух портов | |
| // ... |

|  |
| --- |
| } |

Истина если оба порта вход/выхода [HIGH](https://arduino.ru/Reference/Constants) 25

**|| (логическое ИЛИ)**

Истина, если хотя бы один операнд истина, например:

|  |  |
| --- | --- |
| if (x > 0 || y > 0) { | |
| // ... |

|  |
| --- |
| } |

будет верно (истина) если x или y больше 0.

**! (логическое отрицание)**

True, если операнд false, и наоборот, например:

|  |  |
| --- | --- |
| if (!x) { | |
| // ... |

|  |
| --- |
| } |

условие верно, если x - false (x равно 0).

*Важно* различать логический оператор "И" - **&&** и битовый оператор "И" - **&**. Тоже самое относится к логическому оператору "ИЛИ" - **||** и битовому оператору "ИЛИ" - **|**.

**Пример**

|  |
| --- |
| if (a >= 10 && a <= 20){}   // условие верно, если a больше или равно 10, но меньше или равно 20 |

**Смотрите также**

* [if](https://arduino.ru/Reference/If)

[Справочник языка](https://arduino.ru/Reference)

**Унарные операторы**

* [++](https://arduino.ru/Reference/Increment) (increment)
* [--](https://arduino.ru/Reference/Increment) (decrement)
* [+=](https://arduino.ru/Reference/IncrementCompound) (compound addition)
* [-=](https://arduino.ru/Reference/IncrementCompound) (compound subtraction)
* [\*=](https://arduino.ru/Reference/IncrementCompound) (compound multiplication)
* [/=](https://arduino.ru/Reference/IncrementCompound) (compound division)

++ (увеличение значения) / -- (уменьшение значения) 26

Унарные (имеющие один операнд) операторы **++**, **--** увеличивают, уменьшают значение переменной соответственно.

**Синтаксис**

[код](https://arduino.ru/Reference/Increment#viewSource)

[расечатать](https://arduino.ru/Reference/Increment#printSource)[?](https://arduino.ru/Reference/Increment#about)

|  |  |
| --- | --- |
| x++;  // увеличивает значение x на единицу и возвращает старое значение x | |
| ++x;  // увеличивает значение x на единицу и возвращает новое значение x |

|  |
| --- |
|  |
| x-- ;   // уменьшает значение x на единицу и возвращает старое значение x | |

|  |
| --- |
| --x ;   // уменьшает значение x на единицу и возвращает новое значение x |

**Параметры**

* x: переменная типа [**int**](https://arduino.ru/Reference/Int) или[**long**](https://arduino.ru/Reference/Long) (может быть[беззнаковой](https://arduino.ru/Reference/UnsignedInt))

**Возвращаемое значение**

Изначальное или новое, увеличенное или уменьшенное на единицу, значение переменной.

**Пример**

|  |
| --- |
| x = 2; |
| y = ++x;      // x теперь равно 3, y равно 3 | |

|  |
| --- |
| y = x--;      // x равно 2, y равно 3 |

**Смотрите также**

* [**+=**](https://arduino.ru/Reference/IncrementCompound)
* [-=](https://arduino.ru/Reference/IncrementCompound)

[Справочник языка](https://arduino.ru/Reference)

+= , -= , \*= , /=

Короткий способ записи арифметических действий над переменной и одним операндом.

**Синтаксис**

[код](https://arduino.ru/Reference/IncrementCompound#viewSource)

[расечатать](https://arduino.ru/Reference/IncrementCompound#printSource)[?](https://arduino.ru/Reference/IncrementCompound#about)

|  |
| --- |
| x += y;   // эквивалент записи x = x + y; |
| x -= y;   // эквивалент записи x = x - y; |

|  |
| --- |
| x \*= y;   // эквивалент записи x = x \* y; |
| x /= y;   // эквивалент записи x = x / y; |

**Параметры**

* x: переменная любого типа
* y: переменная любого типа или константа **27**

**Пример**

|  |
| --- |
| x = 2; |
| x += 4;      // x равно 6 | |

|  |
| --- |
| x -= 3;      // x равно 3 |
| x \*= 10;     // x равно 30 | |

|  |
| --- |
| x /= 2;      // x равно 15 |

[Справочник языка](https://arduino.ru/Reference)

**Данные**

**Константы**

* [HIGH](https://arduino.ru/Reference/Constants) | [LOW](https://arduino.ru/Reference/Constants)
* [INPUT](https://arduino.ru/Reference/Constants) | [OUTPUT](https://arduino.ru/Reference/Constants)
* [true](https://arduino.ru/Reference/Constants) | [false](https://arduino.ru/Reference/Constants)
* [Целочисленные константы](https://arduino.ru/Reference/IntegerConstants)
* [Константы с плавающей запятой](https://arduino.ru/Reference/Fpconstants)

Константы

Константами в языке Ардуино называют некоторые предопределенные значения. Они позволяют сделать код программы более легким для восприятия. Константы разделяют на три группы:

**Логические константы**

В языке Ардуино есть две константы для представления логических значений истина и ложь: **true** и**false**.

**false**

false определяется как 0, в логическом выражении.

**true**

Обычно говорят, что true — это 1, это корректно, но также корректно то, что для integer любой значение отличное от 0 будет также true в логическом выражение. Т.е -1, -2 и -200 будет также true в логическом выражении.

Обратите внимание, что true и false пишутся строчными буквами, в отличие от HIGH, LOW, INPUT и OUTPUT.

**Задание значение на входа/выходе, HIGH и LOW**

Возможны только два значения для считывания или записи на цифровой порт вход/выхода: **HIGH** и **LOW**. 28

**HIGH**

HIGH может обозначать несколько разное в зависимость от уставки режима порта как INPUT или OUTPUT. Когда порт вход/выхода установлен в режим INPUT с помощью функции [pinMode](https://arduino.ru/Reference/PinMode), и считывается функцией [digitalRead](https://arduino.ru/Reference/DigitalRead), микроконтроллер отдаст значение HIGH при напряжение 3В или выше на указанном порту.

Также порт может быть установлен как INPUT функцией pinMode, и затем установлен в  HIGH значение функцией [digitalWrite](https://arduino.ru/Reference/DigitalWrite). Это подключит к порту [внутренний подтягивающий резистор](https://arduino.ru/Tutorial/DigitalPins) 20K, что позволит получать постоянное значение HIGH при чтение этого порта, если только значение не будет приведено к LOW внешней цепью подключенной к этому порту.

Когда порт вход/выхода сконфигурирован как OUTPUT функцией pinMode, и установлено значение HIGH функцией digitalWrite, на порту будет постоянное напряжение 5В. От этого порта может быть запитан светодиод, подключенный через резистор на землю или к другому порту, сконфигурированному как OUTPUT и установленному в LOW.

**LOW**

Значение LOW также разное для режима INPUT и OUTPUT. Когда порт сконфигурирован как INPUT, и считывается функцие digitalRead, микроконтроллер вернет LOW если напряжение на данном порту меньше или равно 2В.

Если же порт установлен в OUTPUT и LOW, то напряжение на выходе порта будет 0 Вольт. К этому порту можно подключать нагрузку как к земле, как описано выше на примере светодиода.

**Задание режима порта вход/выхода, INPUT и OUTPUT**

[Цифровые порты](https://arduino.ru/Tutorial/DigitalPins) вход/выхода, могут быть установлены в режимы работы как вход или выход:**INPUT** или **OUTPUT**. Установка производится[функцией pinMode().](https://arduino.ru/Reference/PinMode)

**INPUT**

Порты Arduino установленные в режим **INPUT** находятся в высокоимпедансном состоянии. Это означает то, что порт ввода дает слишком малую нагрузки на схему, в которую он включен. Эквивалентом внутреннему сопротивлению будет резистор 100 МОм подключенный к выводу микросхемы. Таким образом, для перевода порта ввода из одного состояния в другое требуется маленькое значение тока. Это позволяет применять выводы микросхемы для подключения различных датчиков, но не питания.

**OUTPUT**

Порт установленный в режим выхода — **OUTPUT,** находится в низкоимпедансном состояние. Он может пропускать через себя довольно большой ток, до 40 mA, достаточный для запитывание внешней цепи, например, светодиода. В этом состоянии порт может быть поврежден как замыкании на землю так и на питание 5В. Тока с порта микроконтроллера не достаточно  для питания моторов и сервоприводов напрямую.

**Смотрите также**

* [pinMode()](https://arduino.ru/Reference/PinMode)
* [digitalWrite()](https://arduino.ru/Reference/DigitalWrite)
* [digitalRead()](https://arduino.ru/Reference/DigitalRead)
* [Цифровые входы](https://arduino.ru/Tutorial/DigitalPins)

29

Целочисленные константы

Целочисленные константы — это числа используемые напрямую в коде скетча, без определения переменной для их хранения. По умолчанию такие константы трактуются как тип **int**, но это может быть изменено директивами U и L (см. ниже)

Обычно такие константы считаются десятичными целыми числами, но специальные директивы позволяют задать отличный базис.

| **Базис** | **Пример** | **Директива** | **Комментарий** |
| --- | --- | --- | --- |
| 10 (decimal- десятеричный | 123 | нет |  |
| 2 ( binary - двоичный) | B1110111 | первая B | только для 8-бит значений (0-255), разрешенные знаки 0 и 1 |
| 8 (octal) | 073 | первая "0" | разрешенные знаки 0-7 |
| 16(hexadecimal) | 0x7B | первые "0x" | разрешенные знаки 0-9, A-F, a-f |

**Decimal (десятичные)** базис 10. Наиболее распространен. Константы без префиксов (директив) по умолчанию считаются десятичными

Пример:

|  |
| --- |
| 101     // 101 десятичные   ((1 \* 10^2) + (0 \* 10^1) + 1) |

**Binary** (двоичный или бинарный) базис 2. Только 0 и 1 разрешены для записи.

Пример:

|  |
| --- |
| B101    // тоже что 5 десятичных   ((1 \* 2^2) + (0 \* 2^1) + 1) |

Использование двоичного формата допускается только с 8-битными числами, принимающими значение от 0 (B0) до 255(B11111111). Если необходимо задать значение 16-битного целого )**int** бинарными константами, то это может быть сделано в два шага:

|  |
| --- |
| myInt = (B11001100 \* 256) + B10101010; |

**Octal** — базис 8. Допустимы 0-7. Форматирующий префикс "0".

Пример:

|  |
| --- |
| 0101    // то же, что десятичное 65   ((1 \* 8^2) + (0 \* 8^1) + 1) |

*Внимание!* Включение "0" перед константой по ошибке может привести к очень сложно-диагностируемой ошибке, из-за того, что компилятор будет считать константу в восьмеричной системе.

**Hexadecimal (or hex)** базис 16. Допустимы знаки 0-9, A-F и a-f. A — это десятеричное 10, B — 11, и т.д до F — 15. Предваряющий "0x" используется как форматирующая директива.

Пример: 30

|  |
| --- |
| 0x101   // same as 257 decimal   ((1 \* 16^2) + (0 \* 16^1) + 1) |

U и L директивы.

По умолчанию целочисленные константы относятся компилятором к типу **int**. Чтобы заставить компилятор использовать другой тип данных, используются следующие директивы.

* 'u' или 'U' для беззнакового целочисленного числа. Пример: 33u
* 'l' или 'L' для **long** типа данных. Пример: 100000L
* 'ul' или 'UL' для беззнакового числа **long**типа данных. Пример: 32767ul

**Смотрите также**

* [константы](https://arduino.ru/Reference/Constants)

## Константы с плавающей запятой (floating point constants)

* Очень похожи на [целочисленные константы](https://arduino.ru/Reference/IntegerConstants). Используются в коде программы для более удобного понимания и чтения кода. Значения записываются по зарубежному стандарту только через точку.
* **Пример**

|  |
| --- |
| n = .005; |

* Также принимаются значения, записанные с экспонентой вида «Е» и «е».

|  |  |  |
| --- | --- | --- |
| Формат записи константы с плавающей запятой | Значение константы: | либо: |
| 10.0 | 10 |  |
| 2.34Е5 | 2.34 \* 10^5 | 234000 |
| 67е-12 | 67.0 \* 10^-12 | .000000000067 |

* [Справочник языка](https://arduino.ru/Reference)

**31 31**

**Типы данных**

* [boolean](https://arduino.ru/Reference/BooleanVariables)
* [char](https://arduino.ru/Reference/Char)
* [byte](https://arduino.ru/Reference/Byte)
* [int](https://arduino.ru/Reference/Int)
* [unsigned int](https://arduino.ru/Reference/UnsignedInt)
* [word](https://arduino.ru/Reference/Word)
* [long](https://arduino.ru/Reference/Long)
* [unsigned long](https://arduino.ru/Reference/UnsignedLong)
* [float](https://arduino.ru/Reference/Float)
* [double](https://arduino.ru/Reference/Double)
* [string](https://arduino.ru/Reference/String) - массив символов
* [String](https://arduino.ru/Reference/StringObject) - объект класса
* [массив (array)](https://arduino.ru/Reference/Array)
* [void](https://arduino.ru/Reference/Void)

boolean

Логический (булевый) тип данных**— boolean.** Может принимать одно из двух значений [true](https://arduino.ru/Reference/Constants) или [false](https://arduino.ru/Reference/Constants). **boolean** занимает в памяти один байт.

**Пример**

[код](https://arduino.ru/Reference/BooleanVariables#viewSource)

[расечатать](https://arduino.ru/Reference/BooleanVariables#printSource)[?](https://arduino.ru/Reference/BooleanVariables#about)

|  |
| --- |
| int LEDpin = 5;       // Светодиод на входе 5 |
| int switchPin = 13;   // выключатель на порту 13, замыкает на землю | |

|  |
| --- |
|  |
| boolean running = false; | |

|  |
| --- |
|  |
| void setup() | |

|  |
| --- |
| { |
| pinMode(LEDpin, OUTPUT); | |

|  |
| --- |
| pinMode(switchPin, INPUT); |
| digitalWrite(switchPin, HIGH);      // включаем подтягивающий резистор | |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| void loop() | |
| { |

|  |
| --- |
| if (digitalRead(switchPin) == LOW) |
| {  // выключатель нажат, т.к. подтягивающий резистор будет давайть HIGH на входе, если не замкнут напрямую на землю | |

|  |
| --- |
| delay(100);                        // ждем 0.1сек |
| running = !running;                // меняем значение булевой переменной | |

|  |  |
| --- | --- |
| digitalWrite(LEDpin, running)      // включаем или выключаем светодиод. | |
| } |

|  |
| --- |
| } |

**Смотрите также**

* [константы](https://arduino.ru/Reference/Constants)
* [логические операторы](https://arduino.ru/Reference/Boolean)

[Справочник языка](https://arduino.ru/Reference)

char

Переменная типа **char** занимает 1 байт памяти и может хранить один алфавитно-цифровой символ (литеру). При объявление литеры используются одиночные кавычки: 'A' (двойные кавычки используется при объявлении строки символов - тип string: "ABC").

Символ хранится в памяти как число, соответствующее коду символа в [таблице кодировки символов ASCII](https://arduino.ru/Reference/ASCIIchart).  Т.к. символ хранится как число в памяти над ним возможно производить арифметические действия (например, 'A' + 1 будет 66, т.к. ASCII код для 'A' - 65).

Тип **char** знаковый тип, т.е. число (код) хранящийся в памяти может принимать значения от -128 до 127. Если необходим беззнаковая однобайтовая переменная, используйте тип **byte**.

**Пример**

|  |
| --- |
| char myChar = 'A'; |
| char myChar = 65;      //Варианты эквивалентны | |

**Смотрите также**

* [byte](https://arduino.ru/Reference/Byte)
* [int](https://arduino.ru/Reference/Int)
* [массивы](https://arduino.ru/Reference/Array)
* Serial.println

[Справочник языка](https://arduino.ru/Reference)

byte

**Описание типа**

Тип данных byte 8-ми битное беззнаковое целое число, в диапазоне 0..255.

**Пример**

|  |
| --- |
| byte c = B10010;  // "B" префикс двоичной системы счисления(B10010 = 18 в десятичной системе счисления) |

[Справочник языка](https://arduino.ru/Reference)

int

**Описание типа данных**

Тип данных**int** (от *англ.*integer - целое число) один их наиболее часто используемых типов данных для хранения чисел. **int** занимает 2 байта памяти, и может хранить числа от -32 768 до 32 767 (от -2^15 до 2^15-1)

Для размещения отрицательных значений int использует, так называемый, [дополнительный код представления числа](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BF%D0%BE%D0%BB%D0%BD%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4_(%D0%BF%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%B0)). Старший бит указывает на отрицательный знак числа, остальные биты инвертируются с добавлением 1.

Arduino компилятор сам заботиться о размещение в памяти и представление отрицательных чисел, поэтому арифметические действия над целыми числами производятся как обычно.

**Пример**

|  |
| --- |
| int ledPin = 13; |

**Синтаксис**

    int var = val;

* *var* - имя переменной;
* *val* - значение присваиваемое переменной;

**Замечание по использованию типа int**

Когда переменная типа **int** в следствие арифметической операции достигает своего максимального значения, она "перескакивает" на самое минимальное значение и наоборот:

|  |
| --- |
| int x |
| x = -32,768; | |

|  |  |
| --- | --- |
| x = x - 1;       // x теперь равно 32,767 - перескакивает на минимальное значение | |
|  |

|  |
| --- |
| x = 32,767; |
| x = x + 1;       // x теперь равно -32,768 | |

**Смотрите также**

* [byte](https://arduino.ru/Reference/Byte)
* [long](https://arduino.ru/Reference/Long)
* [Целочисленные константы](https://arduino.ru/Reference/IntegerConstants)

[Справочник языка](https://arduino.ru/Reference)

unsigned int

**Описание типа**

Тип данных **unsigned int** - беззнаковое целое число, также как и тип **int** (знаковое) занимает в памяти 2 байта. Но в отличие от **int**, тип **unsigned int** может хранить только положительные целые числа в диапазоне от 0 до 65535 (2^16)-1).

Отличие кроется в том как **unsigned int** использует старший бит, иногда называемый знаковый бит. Если старший бит равен 1, то для типа **int** компилятор Arduino считает, что это число отрицательное, а остальные 15 bit несут информацию о модуле целого числа в [дополнительного кода представления числа](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BF%D0%BE%D0%BB%D0%BD%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4_(%D0%BF%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%B0)), в то время как **unsigned int** использует все 16 бит для хранения модуля числа.

**Пример**

|  |
| --- |
| unsigned int ledPin = 13; |

**Синтаксис**

|  |
| --- |
| unsigned int var = val; |

* var - имя переменной
* val  - присваиваемое значение

**Замечание по использованию типа unsigned int**

Когда переменная типа **int** в следствие арифметической операции достигает своего максимального значения, она "перескакивает" на самое минимальное значение и наоборот:

|  |  |
| --- | --- |
| unsigned int x | |
| x = 0; |

|  |  |
| --- | --- |
| x = x - 1;       // x теперь равна 65535 | |
| x = x + 1;       // x теперь 0 |

**Смотрите также**

* [int](https://arduino.ru/Reference/Int)
* [byte](https://arduino.ru/Reference/Byte)
* [long](https://arduino.ru/Reference/Long)
* [Целочисленные константы](https://arduino.ru/Reference/IntegerConstants)

[Справочник языка](https://arduino.ru/Reference)

word

**Описание типа**

Тип данных word хранит 16-битное, не содержащее знака, число от 0 до 65535. Тоже самое, что [unsigned int](https://arduino.ru/Reference/UnsignedInt) — (беззнаковое целое число).

**Пример**

|  |
| --- |
| word w = 10000; |

**Смотрите также**

* [int](https://arduino.ru/Reference/Int)
* [byte](https://arduino.ru/Reference/Byte)
* [long](https://arduino.ru/Reference/Long)
* [Целочисленные константы](https://arduino.ru/Reference/IntegerConstants)

[Справочник языка](https://arduino.ru/Reference)

long

**Описание типа**

Тип данных **long** используется для хранения целых чисел в расширенном диапазоне от -2,147,483,648 до 2,147,483,647. **long** занимает 4 байта в памяти.

**Пример**

|  |
| --- |
| long speedOfLight = 186000L;  // См. значение постфикса 'L' в теме Целочислесленные константы |

**Синтаксис**

|  |
| --- |
| long var = val; |

* var - имя переменной
* val  - присваиваемое значение

**Смотрите также**

* [int](https://arduino.ru/Reference/Int)
* [byte](https://arduino.ru/Reference/Byte)
* [Целочисленные константы](https://arduino.ru/Reference/IntegerConstants)

unsigned long

**Описание типа**

**Unsigned** **long** используется для хранения положительных целых чисел в диапазоне от 0 до 4,294,967,295 (2^32 - 1)изанимает 32 бита (4 байта) в памяти.

**Пример**

|  |  |
| --- | --- |
| unsigned long time; | |
|  |

|  |  |
| --- | --- |
| void setup() | |
| { |

|  |  |
| --- | --- |
| Serial.begin(9600); | |
| } |

|  |
| --- |
|  |
| void loop() | |

|  |
| --- |
| { |
| Serial.print("Time: "); | |

|  |
| --- |
| time = millis(); |
| //выводит время прошедшее с момента начала выполнения программы | |

|  |
| --- |
| Serial.println(time); |
| // ожидаем (делаем пауза) 1 секунду | |

|  |  |
| --- | --- |
| delay(1000); | |
| } |

**Синтаксис**

|  |
| --- |
| unsigned long var = val; |

* var - имя переменной
* val  - присваиваемое значение

**Смотрите также**

* [int](https://arduino.ru/Reference/Int)
* [byte](https://arduino.ru/Reference/Byte)
* [long](https://arduino.ru/Reference/Long)
* [Целочисленные константы](https://arduino.ru/Reference/IntegerConstants)

float

**Описание типа**

Тип данных **float** служит для хранения чисел с плавающей запятой. Этот тип часто используется для операций с данными, считываемыми с аналоговых входов. Диапазон значений — от -3.4028235E+38 до 3.4028235E+38. Переменная типа **float** занимает 32 бита (4 байта) в памяти.

Тип float имеет точность 6-7 знаков, имеются ввиду все знаки, а не только мантисса. Обычно для увеличения точности используют другой тип - [double](https://arduino.ru/Reference/Double), но на платформе Arduino, double и float имеют  одинаковую точность.

Хранение в памяти чисел с плавающей точкой в двоичной системе обуславливает потерю точности. Так, например,  6.0 / 3.0 не обязательно равен 2.0. Сравнивая два числа с плавающей точкой следует проверять не точное равенство, а разницу между этими числами, меньше ли она некого выбранной малого порога.

Следует также учитывать, что арифметические операции над числами с плавающей запятой выполняются существенно медленнее, чем над целыми.

**Пример**

|  |
| --- |
| float myfloat; |
| float sensorCalbrate = 1.117; | |

**Синтаксис**

|  |
| --- |
| float var = val; |

* var - имя переменной
* val  - присваиваемое значение

**Пример использования в коде**

|  |
| --- |
| int x; |
| int y; |

|  |  |
| --- | --- |
| float z; | |
|  |

|  |
| --- |
| x = 1; |
| y = x / 2;            // y теперь равен 0, тип int не может хранить дробные числа | |

|  |
| --- |
| z = (float)x / 2.0;   // z равна .5 (следует использовать 2.0, а не 2) |

**Смотрите также**

* [int](https://arduino.ru/Reference/Int)
* [double](https://arduino.ru/Reference/Double)

[Справочник языка](https://arduino.ru/Reference)

double

**Описание типа данных**

Тип данных **double**, в отличие от большинства языков программирования, имеет ту же точность, что и тип **float** и занимает также 4 байта памяти.

Тип **double** поддерживается в Arduino для совместимости кода с другими платформами.

**Смотрите также**

* [float](https://arduino.ru/Reference/Float)

[Справочник языка](https://arduino.ru/Reference)

string - текстовые строки - массив символов

**Описание**

Текстовые строки в Ардуино объявляются как массив (array) типа [**char**](https://arduino.ru/Reference/Char) (символов - литер), оканчивающийся символом "конца строки".

**Синтаксис**

Ниже приведены варианты объявления и присвоения строк:

|  |
| --- |
| char Str1[15]; |
| char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'}; | |

|  |  |
| --- | --- |
| char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'}; | |
| char Str4[ ] = "arduino"; |

|  |
| --- |
| char Str5[8] = "arduino"; |
| char Str6[15] = "arduino"; | |

Возможны следующие варианты объявления текстовых строк:

* Объявить массив символов без присваивания значений — Str1
* Объявить массив символов и присвоить значения всем элементам, кроме последнего, компилятор Arduino автоматически добавит символ конца строки, как в Str2
* Явно объявить завершающий символ, как в Str3
* Инициализировать массив строковой константой в двойных кавычках. Компилятор автоматически задаст требуемый размер на массив, равный количеству символов плюс завершающий символ, как в варианте Str4
* Инициализировать массив с явным заданием размера и присвоением строковой константы, строка Str5
* Инициализировать массив с явным заданием дополнительного размера (с запасом), фактически превышающего размер строковой константы при начальном присвоение, как в варианте Str6

**Символ окончания строки**

Обычно строки оканчиваются нулевым символом (код 0 в ASCII). Это позволяет функциям (таким как Serial.print()) выявлять окончание строки. В противном случае могут считаться байты памяти, не принадлежащие переменной.

Массивы символов, выделяемый под строку, должен иметь один дополнительный элемент для символа конца строки. Именно поэтому Str2 и Str5 должны иметь 8 символом, хотя "arduino" 7 символов — последнему элементу автоматически присваивается символ окончания строки.

Технически возможно объявить строку без символа окончания строки, например, объявить длину массива равной 7 для Str2. Это приведет к некорректной работе функций, оперирующих строками.

**Одинарные и двойные кавычки**

Строки всегда объявляются внутри двойных кавычек ("Abc"). Символы — внутри одинарных ('A')

**Инициализация длинных строк**

Длинные строки могут быть объявлены так:

|  |  |
| --- | --- |
| char myString[] = "This is the first line" | |
| " this is the second line" |

|  |
| --- |
| " etcetera"; |

**Массивы строк**

При работе с большими объемами текстовой информации, например в проектах с LCD дисплеем, бывает удобно использовать массивы строк. Так как строки сами по себе массивы, массивы строк будет двумерным массивом.

В примере ниже, символ звездочки после объявления типа "**char\***" указывает на то, что это массив указателей. Это необходимо для задания двумерного массива. В данном случае не требуется понимания всех тонкостей работы с указателями.

**Пример**

|  |  |
| --- | --- |
| char\* myStrings[]={"This is string 1", "This is string 2", "This is string 3", | |
| "This is string 4", "This is string 5","This is string 6"}; |

|  |
| --- |
|  |
| void setup(){ | |

|  |  |
| --- | --- |
| Serial.begin(9600); | |
| } |

|  |
| --- |
|  |
| void loop(){ | |

|  |
| --- |
| for (int i = 0; i < 6; i++){ |
| Serial.println(myStrings[i]); | |

|  |  |
| --- | --- |
| delay(500); | |
| } |

|  |
| --- |
| } |

**Смотрите также**

* [массивы](https://arduino.ru/Reference/Array)

[Справочник языка](https://arduino.ru/Reference)

String - объект класса

**String** класс появился в версии Arduino 0019. Этот класс позволяет хранить и манипулировать текстовыми строками, по сравнению с [string](https://arduino.ru/Reference/String) (массивом символов) класс **String** предоставляет удобные функции для работы со строками, такие как поиск вхождения в строку, объединение строк и др. Класс **String** занимает несколько больше места в памяти, чем массив символов **string**.

Обратите внимание при обращение к классу **String** — он пишется с заглавной **S**. Массив символов **string** — с прописной **s**. Строковые константы, записанные в "двойных кавычка" интерпретируются компилятором как массив символов, а не объект класса String.

**Функции**

* [String](https://arduino.ru/Reference/StringConstructor)()
* [charAt()](https://arduino.ru/Reference/StringCharAt)
* [compareTo()](https://arduino.ru/Reference/StringCompareTo)
* concat()
* endsWith()
* equals()
* equalsIgnoreCase()
* getBytes()
* indexOf()
* lastIndexOf()
* length()
* replace()
* [setCharAt()](https://arduino.ru/Reference/StringSetCharAt)
* startsWith()
* [substring()](https://arduino.ru/Reference/StringSubstring)
* toCharArray()
* toLowerCase()
* toUpperCase()
* trim()

**Операторы**

* [[] (доступ к элементу строки)](https://arduino.ru/Reference/StringBrackets)
* + (объединение строк)
* == (сравнение строк)

**Смотрите также**

* [string](https://arduino.ru/Reference/String): массив символов

[Справочник языка](https://arduino.ru/Reference)

* Массивы [массив (array)](https://arduino.ru/Reference/Array)

Массивы (arrays) — именованный набор однотипных переменных, с доступом к отдельным элементам по их индексу.

**Объявление массивов**

Ниже приведены несколько корректных вариантов объявления массивов:

|  |
| --- |
| int myInts[6]; |
| int myPins[] = {2, 4, 8, 3, 6}; | |

|  |  |
| --- | --- |
| int mySensVals[6] = {2, 4, -8, 3, 2}; | |
| char message[6] = "hello"; |

Массив может быть объявлен без непосредственной инициализации элементов массива, как в случае массива myInts.

Массив myPins был объявлен без явного задания размера. Компилятор сам посчитает фактическое количество элементов и создаcт в памяти  массив необходимого размера.

Размер может быть задан явно, одновременно с инициализацией элементов массива. Обратите внимания, что при создании массива типа [char](https://arduino.ru/Reference/Char), необходим дополнительный элемент массива для нулевого символа. Подробнее см. [строки](https://arduino.ru/Reference/String).

**Доступ к элементам массива**

Индексация массива начинается с 0. Это значит, что для массива с 10-тью элементами, индекс 9 будет последним:

[код](https://arduino.ru/Reference/Array#viewSource)

[расечатать](https://arduino.ru/Reference/Array#printSource)[?](https://arduino.ru/Reference/Array#about)

|  |
| --- |
| int myArray[10]={9,3,2,4,3,2,7,8,9,11}; |
| // myArray[0]    первый элемент, содержит 9 | |

|  |
| --- |
| // myArray[9]    последний элемент, содержит 11 |
| // myArray[10]   это неверно возможно возвращение произвольного значения из области памяти не относящийся к массиву | |

**Присваиваем значение элементу массива:**

|  |
| --- |
| mySensVals[0] = 10; |

**Возвращаем значение элемента массива:**

|  |
| --- |
| x = mySensVals[4]; |

**Массивы и FOR циклы**

Чаще всего для перебора элементов цикла используется [цикл for](https://arduino.ru/Reference/For), счетчик цикла используется как индекс для доступа к каждому элементу массива. Например, для вывода массива через Serial порт можно использовать следующий код:

|  |
| --- |
| int i; |
| for (i = 0; i < 5; i = i + 1) { | |

|  |  |
| --- | --- |
| Serial.println(myPins[i]); | |
| } |

**Смотрите также**

* [цикл for](https://arduino.ru/Reference/For)
* [строки (string)](https://arduino.ru/Reference/String)

[Справочник языка](https://arduino.ru/Reference)

void

Ключевое слово **void** используется при объявлении функций, если функция не возвращает никакого значение при ее вызове (в некоторых языках программирования такие функции называют процедурами).

**Пример**

|  |  |
| --- | --- |
| // в функциях "setup" и "loop" производятся некоторые действия, | |
| // но ничего не возвращается во внешнюю программу |

|  |
| --- |
|  |
| void setup() | |

|  |
| --- |
| { |
| // ... | |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| void loop() | |
| { |

|  |  |
| --- | --- |
| // ... | |
| } |

**Смотрите также**

* [Функции в языке Ардуино](https://arduino.ru/Reference/FunctionDeclaration)

[Справочник языка](https://arduino.ru/Reference)

**Преобразование типов данных**

* [char()](https://arduino.ru/Reference/CharCast)
* [byte()](https://arduino.ru/Reference/ByteCast)
* [int()](https://arduino.ru/Reference/IntCast)
* [long()](https://arduino.ru/Reference/LongCast)
* [float()](https://arduino.ru/Reference/FloatCast)

char()

**char()** приводит значение к типу [char](https://arduino.ru/Reference/Char).

**Синтаксис**

char(x)

**Параметры**

* x: переменная любого типа

**Возвращаемое значение**

[char](https://arduino.ru/Reference/Char)

**Смотрите также**

* [char](https://arduino.ru/Reference/Char)

[Справочник языка Arduino](https://arduino.ru/Reference)

byte()

**byte()** приводит значение к типу [byte](https://arduino.ru/Reference/Byte).

**Синтаксис**

|  |
| --- |
| byte(x) |

**Параметры**

* x: переменная любого типа

**Возвращаемое значение**

[byte](https://arduino.ru/Reference/Byte)

**Смотрите также**

* [byte](https://arduino.ru/Reference/Byte)

[Справочник языка Arduino](https://arduino.ru/Reference)

int()

**int()** приводит значение к типу [int](https://arduino.ru/Reference/Int).

**Синтаксис**

|  |
| --- |
| int(x) |

**Параметры**

* x: переменная любого типа

**Возвращаемое значение**

[int](https://arduino.ru/Reference/Int)

**Смотрите также**

* [int](https://arduino.ru/Reference/Int)

[Справочник языка Arduino](https://arduino.ru/Reference)

long()

**long()** приводит значение к типу [long](https://arduino.ru/Reference/Long).

**Синтаксис**

|  |
| --- |
| long(x) |

**Параметры**

* x: переменная любого типа

**Возвращаемое значение**

[long](https://arduino.ru/Reference/Long)

**Смотрите также**

* [long](https://arduino.ru/Reference/Long)

[Справочник языка Arduino](https://arduino.ru/Reference)

float()

**float()** приводит значение к типу [float](https://arduino.ru/Reference/Float).

**Синтаксис**

|  |
| --- |
| float(x) |

**Параметры**

* x: переменная любого типа

**Возвращаемое значение**

[float](https://arduino.ru/Reference/Float)

**Замечание по использованию**

Обратите внимание на ограничения по значению для типа [float](https://arduino.ru/Reference/Float).

**Смотрите также**

* [float](https://arduino.ru/Reference/Float)

[Справочник языка Arduino](https://arduino.ru/Reference)

**Область видимости переменных и квалификаторы**

* [Область видимости](https://arduino.ru/Reference/Scope)
* [static](https://arduino.ru/Reference/Static)
* [volatile](https://arduino.ru/Reference/Volatile)
* [const](https://arduino.ru/Reference/Const)

Область видимости переменных

Переменные в языке программирования С, используемом Arduino, обладают свойством, которое называется область видимости, в отличие от языков, подобных BASIC, где каждая переменная является глобальной.

Глобальная переменная доступна для любой функции в программе. Локальные переменные видны только в той функции, в которой они объявлены. В среде Arduino любая переменная, объявленная вне функции (например, setup(), loop() и т.д.), является глобальной переменной.

Когда программа начинает разрастаться и усложняться, использование локальных переменных – удобный способ, гарантирующий, что только одна функция будет иметь доступ к своим собственным переменным. Тем самым предотвращаются ошибки программирования, когда одна функция непреднамеренно изменяет значения переменных, используемых другой функцией.

Также иногда бывает удобно объявить и инициализировать переменную внутри цикла. При этом создается переменная, доступная только внутри скобок цикла for.

**Пример**

|  |  |
| --- | --- |
| int gPWMval;  // глобальная переменная, видима в любой функции скетча | |
|  |

|  |  |
| --- | --- |
| void setup() | |
| { |

|  |  |
| --- | --- |
| // ... | |
| } |

|  |
| --- |
|  |
| void loop() | |

|  |
| --- |
| { |
| int i;    // "i" видима только внутри "loop" | |

|  |  |
| --- | --- |
| float f;  // "f" видима только внутри "loop" | |
| // ... |

|  |
| --- |
|  |
| for (int j = 0; j <100; j++){ | |

|  |  |
| --- | --- |
| // переменная j доступна только внутри этого цикла | |
| } |

|  |
| --- |
|  |
| } | |

[Справочник языка](https://arduino.ru/Reference)

Квалификатор static

Ключевое слово **static** используется для создания переменной, которая видна только одной функции. Однако в отличие от локальных переменных, которые создаются и уничтожаются при каждом вызове функции, статические переменные остаются после вызова функции, сохраняя свои значения между её вызовами.

**Пример**

[код](https://arduino.ru/Reference/Static#viewSource)

[расечатать](https://arduino.ru/Reference/Static#printSource)[?](https://arduino.ru/Reference/Static#about)

|  |
| --- |
| /\* RandomWalk |
| \* Paul Badger 2007 | |

|  |  |
| --- | --- |
| \* RandomWalk ("Случайное блуждание") программа "перемещается" между двумя точками последовательности | |
| \* Максимальное перемещение задается параметром "stepsize". |

|  |  |
| --- | --- |
| \* Static переменная изменяет свое значение на случайную велечину. | |
| \* Такое "блуждание" иногда называют "pink noise" (розовый шум). |

|  |  |
| --- | --- |
| \*/ | |
|  |

|  |
| --- |
| #define randomWalkLowRange -20 |
| #define randomWalkHighRange 20 |

|  |  |
| --- | --- |
| int stepsize; | |
|  |

|  |  |
| --- | --- |
| int thisTime; | |
| int total; |

|  |
| --- |
|  |
| void setup() | |

|  |
| --- |
| { |
| Serial.begin(9600); | |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| void loop() | |
| { |

|  |
| --- |
| stepsize = 5; |
| thisTime = randomWalk(stepsize); | |

|  |  |
| --- | --- |
| Serial.println(thisTime); | |
| delay(10); |

|  |  |
| --- | --- |
| } | |
|  |

|  |
| --- |
| int randomWalk(int moveSize){ |
| static int  place;     // переменная сохраняет значение между вызовами функции | |

|  |  |
| --- | --- |
| // другие функции не могут изменять значение этой переменной | |
| place = place + (random(-moveSize, moveSize + 1)); |

|  |
| --- |
|  |
| if (place < randomWalkLowRange){                    // проверяем выхождение за границы | |

|  |
| --- |
| place = place + (randomWalkLowRange - place);     // если "выскочили" то делаем разворот |
| // этот пример взят с arduino.cc, вообще? если раскрыть скобки, |

|  |  |
| --- | --- |
| // то получим просто place = randomWalkLowRange; | |
| } |

|  |
| --- |
| else if(place > randomWalkHighRange){ |
| place = place - (place - randomWalkHighRange);     // разворот | |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| return place; | |
| } |

[Справочник языка](https://arduino.ru/Reference)

Квалификатор volatile

Квалификатор **volatile**, используется перед типом переменной, чтобы изменить способ интерпретации и доступа к переменной компилятором и в дальнейшем программой.

Объявление переменной с квалификаторм **volatile** – это указание для компилятора. Компилятор – это программа, которая переводит текст вашей программы на С/С++ в машинный код, то есть в реальные команды для чипа Atmega в Arduino.

Он указывает компилятору загрузить переменную из ОЗУ, и не из запоминающего регистра – временной ячейки памяти, в которой хранятся переменные программы и производятся операции с ними. При определенных условиях значения переменных, хранящихся в регистрах, могут оказаться неточными.

Переменная должна быть объявлена **volatile**, когда её значение может быть изменено чем-либо за пределами того участка программы, где она объявлена, например, параллельно выполняющимся процессом. В Arduino единственным местом, где это может проявиться, является участок программы, ассоциированным с прерываниями, вызванный программой обработки прерываний. См. [AttachInterrupt()](https://arduino.ru/Reference/AttachInterrupt)

**Пример**

|  |  |
| --- | --- |
| // переключаем светодиод, когда вызывается прерывание, изменением состояния входного пина | |
|  |

|  |
| --- |
| int pin = 13; |
| volatile int state = LOW; | |

|  |
| --- |
|  |
| void setup() | |

|  |
| --- |
| { |
| pinMode(pin, OUTPUT); | |

|  |  |
| --- | --- |
| attachInterrupt(0, blink, CHANGE); | |
| } |

|  |
| --- |
|  |
| void loop() | |

|  |
| --- |
| { |
| digitalWrite(pin, state); | |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| void blink() | |
| { |

|  |  |
| --- | --- |
| state = !state; | |
| } |

**Смотрите также**

* [AttachInterrupt](https://arduino.ru/Reference/AttachInterrupt)

[Справочник языка](https://arduino.ru/Reference)

Квалификатор const

Ключевое слово **const** обозначает константы. Это квалификатор переменной, изменяющий её свойства, делающий её доступной только для чтения. Это означает, что переменная может использоваться так же как и любая ругая переменная этого типа, но её значение не может быть изменено. Если вы попытаетесь присвоить переменной типа константа значение, вы получите ошибку компилятора.

Константы, объявленные с помощью ключевого слова **const**, подчиняются тем же правилам определения их области видимости, что и другие переменные. Это, и подвохи в использовании директивы #define, делают ключевое слово const превосходным способом задания констант и более предпочтительным, чем использование #define.

**Пример**

|  |  |
| --- | --- |
| const float pi = 3.14; | |
| float x; |

|  |
| --- |
|  |
| // .... | |

|  |
| --- |
|  |
| x = pi \* 2;    // константы можно использовать в вычислениях | |

|  |
| --- |
|  |
| pi = 7;        // ошибка - нельзя присваивать значение константе | |

**#define или const**

Можно использовать как **const**, так и **#define** для создания численных и строковых констант. В [массивах](https://arduino.ru/Reference/Array) можно использовать только константы, объявленные с квалификаторм **const**. В общем случае лучше использовать **const**, а не *#define* для создания констант.

See also:

* #define
* [volatile](https://arduino.ru/Reference/Volatile)

[Справочник языка](https://arduino.ru/Reference)

Функции

**Цифровой ввод/вывод**

* [pinMode](https://arduino.ru/Reference/PinMode)()
* [digitalWrite](https://arduino.ru/Reference/DigitalWrite)()
* [digitalRead](https://arduino.ru/Reference/DigitalRead)()

Функция pinMode

pinMode()

**Описание**

Устанавливает режим работы заданного вход/выхода(pin) как входа или как выхода. Подробнее про [цифровые вход/выходы(pins)](https://arduino.ru/Tutorial/DigitalPins).

**Синтаксис**

pinMode(pin, mode)

**Параметры**

* pin: номер вход/выхода(pin), который Вы хотите установить
* mode: режим одно из двух значение - INPUT или OUTPUT, устанавливает на вход или выход соответственно.

**Возвращаемое значение**

нет

**Пример**

|  |  |
| --- | --- |
| int ledPin = 13;                 // Светодиод, подключенный к вход/выходу 13 | |
| void setup() |

|  |
| --- |
| { |
| pinMode(ledPin, OUTPUT);      // устанавливает режим работы - выход | |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| void loop() | |
| { |

|  |  |
| --- | --- |
| digitalWrite(ledPin, HIGH);   // включает светодиод | |
| delay(1000);                  // ждет секунду |

|  |  |
| --- | --- |
| digitalWrite(ledPin, LOW);    // выключает светодиод | |
| delay(1000);                  // ждет секунду |

|  |
| --- |
| } |

**Примечание**

Аналоговые входы (analog pins) могут быть использованы как цифровые вход/выходы (digital pins). Обращение к ним идет по номерам от 14 (для аналогового входа 0) до 19 (для аналогового входа 5).

**Смотрите также**

* [constants](https://arduino.ru/Reference/IntegerConstants)
* [digitalWrite](https://arduino.ru/Reference/DigitalWrite)()
* [digitalRead](https://arduino.ru/Reference/DigitalRead)()
* Описание цифровых вход/выходов

[Справочник языка](https://arduino.ru/Reference)

Функция digitalWrite()

digitalWrite()

**Описание**

Подает HIGH или LOW значение на цифровой вход/выход (pin).

Если вход/выход (pin) был установлен в режим выход (OUTPUT) функцией [pinMode()](https://arduino.ru/Reference/PinMode), то для значение HIGH напряжение на соответствующем вход/выходе (pin) будет 5В (3.3В для 3.3V плат), и 0В(земля) для LOW.

Если вход/выход (pin) был установлен в режим вход (INPUT), то функция digitalWrite со значением HIGH будет активировать внутренний 20K нагрузочный резистор. Подача LOW в свою очередь отключает этот резистор.  Нагрузочного резистра достаточно чтобы светодиод, подключенный к входу, светил тускло. Если вдруг светодиод работает, но очень тускло, возможно необходимо установить режим выход (OUTPUT) функцией pinMode().

*Замечание.* Вход/выход 13 сложнее использовать как цифровой вход, т.к. он имеет встроенный в плату резистор и светодиод. Если вы активируете еще внутренний нагрузочный резистор 20K, то напряжение на этом входе будет около 1.7В, вместо ожидаемых 5В, т.к. светодиод и добавочный резистор снижает напряжение, т.е. Вы всегда будете получать LOW. Если же Вам все же необходимо использовать 13ый вход/выход, то  используйте внешний нагрузочный резистор.

**Синтаксис**

digitalWrite(pin, value)

**Параметры**

* pin: номер вход/выхода(pin)
* value: значение HIGH или LOW

**Возвращаемое значение**

нет

**Пример**

|  |  |
| --- | --- |
| int ledPin = 13;                 // Светодиод подключенный к вход/выходу 13 | |
| void setup() |

|  |
| --- |
| { |
| pinMode(ledPin, OUTPUT);      // устанавливает режим работы - выход | |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| void loop() | |
| { |

|  |  |
| --- | --- |
| digitalWrite(ledPin, HIGH);   // включает светодиод | |
| delay(1000);                  // ждет секунду |

|  |  |
| --- | --- |
| digitalWrite(ledPin, LOW);    // выключает светодиод | |
| delay(1000);                  // ждет секунду |

|  |
| --- |
| } |

**Примечание**

Аналоговые входы (analog pins) могут быть использованы как цифровые вход/выходы (digital pins). Обращение к ним идет по номерам от 14 (для аналогового входа 0) до 19 (для аналогового входа 5).

**Смотрите также**

* [pinMode()](https://arduino.ru/Reference/PinMode)
* [digitalRead](https://arduino.ru/Reference/DigitalRead)()
* Описание цифровых вход/выходов

[Справочник языка](https://arduino.ru/Reference)

Функция digitalRead()

digitalRead()

**Описание**

Функция считывает значение с заданного входа - HIGH или LOW.

**Синтаксис**

digitalRead(pin)

**Параметры**

pin: номер вход/выхода(pin) который Вы хотите считать

**Возвращаемое значение**

HIGH или LOW

**Пример**

[код](https://arduino.ru/Reference/DigitalRead#viewSource)

[расечатать](https://arduino.ru/Reference/DigitalRead#printSource)[?](https://arduino.ru/Reference/DigitalRead#about)

|  |  |
| --- | --- |
| int ledPin = 13;                 // Светодиод подключенный к вход/выходу 13 | |
| int inPin = 7;                   // кнопка на входе 7 |

|  |  |
| --- | --- |
| int val = 0;                     // переменная для хранения значения | |
|  |

|  |  |
| --- | --- |
| void setup() | |
| { |

|  |
| --- |
| pinMode(ledPin, OUTPUT);       // устанавливает режим работы - выход для 13го вход/выхода (pin) |
| pinMode(inPin, INPUT);         //  устанавливает режим работы - вход для 7го вход/выхода (pin) |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| void loop() | |
| { |

|  |
| --- |
| val = digitalRead(inPin);      // считываем значение с входа |
| digitalWrite(ledPin, val);     // устанавливаем значение на светодиоде равным значению входа кнопки | |

|  |
| --- |
| } |

**Примечание**

Если вход не подключен, то digitalRead может возвращать значения HIGH или LOW случайным образом.

Аналоговые входы (analog pins) могут быть использованы как цифровые вход/выходы (digital pins). Обращение к ним идет по номерам от 14 (для аналогового входа 0) до 19 (для аналогового входа 5).

**Смотрите также**

* [pinMode](https://arduino.ru/Reference/PinMode)()
* [digitalWrite](https://arduino.ru/Reference/DigitalWrite)()
* Описание цифровых вход/выходов

[Справочник языка](https://arduino.ru/Reference)

**Аналоговый ввод/вывод**

* [analogRead](https://arduino.ru/Reference/AnalogRead)()
* [analogReference()](https://arduino.ru/Reference/AnalogReference)
* [analogWrite](https://arduino.ru/Reference/AnalogWrite)()

Функция analogRead()

Функция считывает значение с указанного[аналогового входа](https://arduino.ru/Tutorial/AnalogInputPins). Большинство плат Arduino имеют 6 каналов (8 каналов у платы Mini  и Nano, 16 у Mega) c 10-битным аналого-цифровым преобразователем (АЦП). Напряжение  поданное на аналоговый вход, обычно от 0 до 5 вольт будет преобразовано в значение от 0 до 1023, это 1024 шага с разрешением 0.0049 Вольт. Разброс напряжение и шаг может быть изменен функцией [analogReference()](https://arduino.ru/Reference/AnalogReference).

Считывание значение с аналогового входа занимает примерно 100 микросекунд (0.0001 сек), т.е. максимальная частота считывания приблизительно 10,000 раз в секунду.

**Синтаксис**

analogRead(pin)

**Параметры**

pin: номер порта аналогового входа с которого будет производиться считывание (A0..A5 для большинства плат, 0..7 для Mini и Nano и 0..15 для Mega)

| **Плата** | **Рабочее напряжение** | **Выход** | **Возвращаемое значение** |
| --- | --- | --- | --- |
| Uno | 5 Вольт | A0 .. A5 | 10 бит |
| Mini, Nano | 5 Вольт | A0 .. A7 | 10 бит |
| Mega, Mega2560, MegaADK | 5 Вольт | A0 .. A14 | 10 бит |
| Micro | 5 Вольт | A0 .. A11\* | 10 бит |
| Leonardo | 5 Вольт | A0 .. A11\* | 10 бит |
| Zero | 3.3 Вольт | A0 .. A5 | 12 бит\*\* |
| Due | 3.3 Вольт | A0 .. A11 | 12 бит\*\* |
| MKR Family boards | 3.3 Вольт | A0 to A6 | 12 бит\*\* |

\* - A0 до A5 обозначены на платах, А6 до А11 находятся на пинах 4,6,8,9,10 и 12  
\*\* - По умолчаниею **analogRead()** возвращает значение с разрешением 10 бит, для увеличения битности до 12 используйте команду **analogReadResolution()**

**Замечание**

Если аналоговый вход не подключен, то значения возвращаемые функцией analogRead() могут принимать случайные значения.

**Пример**

|  |
| --- |
| int analogPin = 3;     // номер порта к которому подключен потенциометр |
| int val = 0;           // переменная для хранения считываемого значения |

|  |
| --- |
|  |
| void setup() | |

|  |
| --- |
| { |
| Serial.begin(9600);              //  установка связи по serial | |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| void loop() | |
| { |

|  |
| --- |
| val = analogRead(analogPin);     // считываем значение |
| Serial.println(val);             // выводим полученное значение | |

|  |
| --- |
| } |

**Смотрите также**

* [analogReference()](https://arduino.ru/Reference/AnalogReference)
* [Аналоговые входы](https://arduino.ru/Tutorial/AnalogInputPins)

[Справочник языка](https://arduino.ru/Reference)

Функция analogReference()

**Описание**

Функция определяет опорное напряжение относительно которого происходят аналоговые измерения. Функция analogRead() возвращает значение с разрешением 10 бит пропорционально входному напряжению на [аналоговом входе](https://arduino.ru/Tutorial/AnalogInputPins), и в зависимости от опорного напряжения.

Возможные настройки:

* DEFAULT: стандартное опорное напряжение 5 В (на платформах с напряжением питания 5 В) или 3.3 В (на платформах с напряжением питания 3.3 В)
* INTERNAL: встроенное опорное напряжение 1.1 В на микроконтроллерах ATmega168 и ATmega328, и 2.56 В на ATmega8.
* INTERNAL1V1: встроенное опорное напряжение 1.1 В (Arduino Mega)
* INTERNAL2V56: встроенное опорное напряжение 2.56 (Arduino Mega)
* EXTERNAL: внешний источник опорного напряжения, подключенный к выводу AREF

**Синтаксис**

|  |
| --- |
| analogReference(type) |

**Параметры**

*type*: определяет используемое опорное напряжение (DEFAULT, INTERNAL или EXTERNAL).

**Возвращаемое значение**

нет

**Внимание**

Внешнее напряжение рекомендуется подключать к выводу AREF через резистор 5 кОм.

Таким образом уменьшается риск повреждения микросхемы Atmega если настройки *analogReference* не совпадают с возможностями платформы. Однако при этом произойдет небольшая просадка напряжения, вследствие того, что имеется встроенный резистор 32 кОм, подключенный к выводу AREF. В этом случае оба резистора работают как делитель напряжения. Подсоединение внешнего резистора позволяет быстро переключаться на напряжение 3.3 В вывода AREF с напряжения 5 В DEFAULT без конфигурации аппаратной части и АЦП.

**Использование вывода AREF**

Напряжение, подключенное к выводу AREF, конвертируется АЦП и, затем, определяется значение напряжения, при котором АЦП выдает самое высокое цифровое значение, т.е 1023. Другие значения напряжения, поступающие в АЦП, конвертируются пропорционально. Таким образом, при настройке DEFAULT 5 В значение напряжения 2.5 В в АЦП будет конвертироваться в 512.

В стандартной конфигурации платформ Arduino вывод AREF (вывод 21 Atmega) не задействован. В этом случае при настройке DEFAULT к выводу подключается внутреннее напряжение AVCC. Соединение является низко-импедансным и любое напряжение подведенное к выводу в этот момент может повредить микросхему ATMEGA.

Настройкой INTERNAL к выводу AREF подключается внутреннее напряжение 1.1 В (или 2.56 микросхемы ATmega8). При этом напряжение соответствующее или превышающее 1.1 В будет конвертироваться АЦП в 1023. Другие значения напряжения конвертируются пропорционально.

Внутреннее подключение источника 1.1 В к выводу является высоко-импедансным, что означает, что для измерение напряжения на выводе может быть произведено только мультиметром с высоким сопротивлением. Ошибочное подключение напряжения к выводу AREF  при этой настройке функции analogReference не повредит микросхему, но превысит значение 1.1 В. В этом случае АЦП будет конвертировать напряжение внешнего источника. Во избежание вышеописанных проблем настоятельно рекомендуется подключать внешнее напряжение через резистор 5 кОм.

Рекомендуемой настройкой для вывода AREF является EXTERNAL. При этом происходит отключение обоих внутренних источников, и внешнее напряжение будет являться опорным для АЦП.

**Смотрите также**

* [analogRead()](https://arduino.ru/Reference/AnalogRead)
* [Аналоговые входы](https://arduino.ru/Tutorial/AnalogInputPins)

[Справочник языка](https://arduino.ru/Reference)

Функция analogWrite()

**Описание**

Выдает аналоговую величину ([ШИМ волну](https://arduino.ru/Tutorial/PWM)) на порт вход/выхода. Функция может быть полезна для управления яркостью подключенного светодиода или скоростью электродвигателя. После вызова analogWrite() на выходе будет генерироваться постоянная прямоугольная волна с заданной шириной импульса до следующего вызова analogWrite (или вызова [digitalWrite](https://arduino.ru/Reference/DigitalWrite) или [digitalRead](https://arduino.ru/Reference/DigitalRead) на том же порту вход/выхода). Частота ШИМ сигнала приблизительно 490 Hz.

На большинстве плат Arduino (на базе микроконтроллера ATmega168 или ATmega328) ШИМ поддерживают порты 3, 5, 6, 9, 10 и 11, на [плате Arduino Mega](https://arduino.ru/Hardware/ArduinoBoardMega) порты с 2 по 13. На более ранних версиях плат Arduino analogWrite() работал только на портах 9, 10 и 11.

Для вызова analogWrite() нет необходимости устанавливать тип вход/выхода функцией [pinMode()](https://arduino.ru/Reference/PinMode).

Функция analogWrite никак не связана с аналоговыми входами и с функцией analogRead.

**Синтаксис**

analogWrite(pin, value)

**Параметры**

* pin: порт вход/выхода на который подаем ШИМ сигнал.
* value: период рабочего цикла значение между 0 (полностью выключено) and 255 (сигнал подан постоянно).

**Возвращаемое значение**

нет

**Замечание**

Период ШИМ сигнала на портах вход/выхода 5 и 6 будет несколько длиннее. Это связано с тем, что таймер для данных выходов также задействован функциями millis() и delay(). Данный эффект более заметен при установке коротких периодов ШИМ сигнала (0-10).

**Пример**

Задание яркости светодиода пропорционально значению, снимаемому с потенциометра

|  |
| --- |
| int ledPin = 9;    // Светодиод подключен к выходы 9 |
| int analogPin = 3; // потенциометр подключен к выходу 3 | |

|  |  |
| --- | --- |
| int val = 0;       // переменная для хранения значения | |
|  |

|  |  |
| --- | --- |
| void setup() | |
| { |

|  |  |
| --- | --- |
| pinMode(ledPin, OUTPUT);      // установка порта на выход | |
| } |

|  |
| --- |
|  |
| void loop() | |

|  |
| --- |
| { |
| val = analogRead(analogPin);  // считываем значение с порта, подключенному к потенциометру | |

|  |  |
| --- | --- |
| analogWrite(ledPin, val / 4); // analogRead возвращает значения от 0 до 1023, analogWrite должно быть в диапозоне от 0 до 255 | |
| } |

**Смотрите также**

* [analogRead()](https://arduino.ru/Reference/AnalogRead)
* [ШИМ](https://arduino.ru/Tutorial/PWM)

[Справочник языка](https://arduino.ru/Reference)

**Дополнительные фунции ввода/вывода**

* [tone](https://arduino.ru/Reference/Tone)()
* [noTone](https://arduino.ru/Reference/NoTone)()
* [shiftOut](https://arduino.ru/Reference/ShiftOut)()
* [pulseIn](https://arduino.ru/Reference/PulseIn)()

tone()

Генерирует на порту вход/выхода сигнал — прямоугольную "волну", заданной частоты и с 50% рабочим циклом. Длительность может быть задана параметром, в противном случае сигнал генерируется пока не будет вызвана функция [noTone()](https://arduino.ru/Reference/NoTone). К порту вход/выхода может быть подключен к  пьезо или другой динамик для воспроизведения сигнала.

Воспроизводиться одновременно может только один сигнал. Если сигнал уже воспроизводится на одном порту, то вызов **Tone()** с номером другого порта в качестве параметра ни к чему не приведет, если же **Tone()** будет вызвана с тем же номером порта, то будет установлена новая частота сигнала.

Использование функции **Tone()** помешает использовать ШИМ на портах вход/выхода 3 и 11 (кроме платы [Arduino Mega](https://arduino.ru/Hardware/ArduinoBoardMega)).

**Синтаксис**

|  |
| --- |
| tone(pin, frequency) |
| tone(pin, frequency, duration) | |

**Параметры**

* pin: номер порта вход/выхода, на котором будет генерироваться сигнал
* frequency: частота сигнала в Герцах
* duration: длительность сигнала в миллисекундах

**Возвращаемое значение**

нет

**Смотрите также**

* [noTone()](https://arduino.ru/Reference/NoTone)

[Справочник языка Arduino](https://arduino.ru/Reference)

noTone()

Останавливает сигнал, генерируемый на порту вход/выхода вызывом функции [Tone()](https://arduino.ru/Reference/Tone). Если сигнал не генерировался, то вызов **noTone()** ни к чему не приводит.

**Замечание:**если необходимы сигналы на разных порты, то следует сначала остановить один сигнал функцией **noTone()**, а лишь затем создавать новый сигнал на другом порту функцией**Tone()**.

**Синтаксис**

|  |
| --- |
| noTone(pin) |

**Параметры**

* pin: номер порта вход/выхода, на котором прекращаем сигнал

**Возвращаемое значение**

нет

**Смотрите также**

* [Tone()](https://arduino.ru/Reference/Tone)

[Справочник языка Arduino](https://arduino.ru/Reference)

shiftOut()

Выводит байт информации на порт вход/выхода последовательно (побитно). Вывод может осуществляться как с первого (левого), так и с последнего (правого) бита. Каждый бит последовательно подается на заданный порт, после чего подается сигнал на синхронизирующий порт вход/выход, информируя о доступности к считыванию бита.

Такой способ передачи данных называют последовательный протокол с синхронизацией. Он часто используется для взаимодействия микроконтроллеров с датчиками и сенсорами, а также другими микроконтроллерами. Последовательная передача с синхронизацией позволяет устройствам связываться на максимальной скорости. Смотрите также документацию (англ.) по [SPI (Serial Peripheral Interface Protocol) - протокол последовательного периферийного интерфейса](http://out.arduino.ru/?redirect=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FSerial_Peripheral_Interface_Bus&baseU=https%3A%2F%2Farduino.ru%2FReference%2FShiftOut).

**Синтаксис**

|  |
| --- |
| shiftOut(dataPin, clockPin, bitOrder, value) |

**Параметры**

* dataPin: номер порта вход/выхода, на который выводятся биты ([int](https://arduino.ru/Reference/Int))
* clockPin: номер порта по которому производится синхронизация ([int](https://arduino.ru/Reference/Int))
* bitOrder: используемая последовательность вывода бит. **MSBFIRST** (Most Significant Bit First) — слева или **LSBFIRST** (Least Significant Bit First) — справа.
* value: значение (байт) для вывода ([byte](https://arduino.ru/Reference/Byte))

**Возвращаемое значение**

нет

**Замечание по использованию**

Порт вывода (**dataPin**) и синхронизирующий порт (**clockPin**) должны быть предварительно сконфигурированы как порты вывода с помощью функции [pinMode()](https://arduino.ru/Reference/PinMode).

Текущая реализация функции **shiftOut()** может выводить только один байт (8 бит) информации, поэтому необходимо произвести несколько действий чтобы  вывести значения больше 255.

|  |  |
| --- | --- |
| // Вывод будет MSBFIRST с первого (левого) бита | |
| int data = 500; |

|  |
| --- |
| // выводим старший байт |
| shiftOut(dataPin, clock, MSBFIRST, (data >> 8)); | |

|  |
| --- |
| // выводим младший бит |
| shiftOut(dataPin, clock, MSBFIRST, data); | |

|  |
| --- |
|  |
| // вывод LSBFIRST с последнего бита | |

|  |
| --- |
| data = 500; |
| // выводим младший бит | |

|  |  |
| --- | --- |
| shiftOut(dataPin, clock, LSBFIRST, data); | |
| // выводим старший бит |

|  |
| --- |
| shiftOut(dataPin, clock, LSBFIRST, (data >> 8)); |

**Пример**

Пример вывода счетчика от 0 до 255 на сдвиговый регистр с последовательным вводом 74HC595.

|  |
| --- |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*// |
| //  Name    : shiftOutCode, Hello World                         // |

|  |
| --- |
| //  Author  : Carlyn Maw,Tom Igoe                               // |
| //  Date    : 25 Oct, 2006                                      // |

|  |
| --- |
| //  Version : 1.0                                               // |
| //  Notes   : Code for using a 74HC595 Shift Register           // |

|  |
| --- |
| //          : to count from 0 to 255                            // |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

|  |
| --- |
|  |
| //Порт подключенный к ST\_CP 74HC595 | |

|  |
| --- |
| int latchPin = 8; |
| //Порт подключенный к SH\_CP  74HC595 | |

|  |
| --- |
| int clockPin = 12; |
| //Порт подключенный к DS 74HC595 | |

|  |  |
| --- | --- |
| int dataPin = 11; | |
|  |

|  |
| --- |
| void setup() { |
| //устанавливаем режим порта выхода | |

|  |
| --- |
| pinMode(latchPin, OUTPUT); |
| pinMode(clockPin, OUTPUT); |

|  |  |
| --- | --- |
| pinMode(dataPin, OUTPUT); | |
| } |

|  |
| --- |
|  |
| void loop() { | |

|  |
| --- |
| for (int j = 0; j < 256; j++) { |
| //устанавливаем LOW на latchPin пока не окончена передача байта | |

|  |
| --- |
| digitalWrite(latchPin, LOW); |
| shiftOut(dataPin, clockPin, LSBFIRST, j); | |

|  |  |
| --- | --- |
| //устанавливаем HIGH на latchPin, чтобы проинформировать регистр, что передача окончена. | |
| digitalWrite(latchPin, HIGH); |

|  |  |
| --- | --- |
| delay(1000); | |
| } |

|  |
| --- |
| } |

[Справочник языка Arduino](https://arduino.ru/Reference)

pulseIn()

Считывает длину сигнала на заданном порту (**HIGH** или **LOW**). Например, если задано считывание **HIGH** функцией **pulseIn()**, функция ожидает пока на заданном порту не появиться **HIGH**. Когда **HIGH** получен, включается таймер, который будет остановлен когда на порту вход/выхода будет **LOW**. Функция **pulseIn()** возвращает длину сигнала в микросекундах. Функция возвращает 0, если в течение заданного времени (таймаута) не был зафиксирован сигнал на порту.

Возможны некоторые погрешности в измерение длинных сигналов. Функция может измерять сигналы длиной от 10 микросекунд до 3 минут.

**Синтаксис**

|  |
| --- |
| pulseIn(pin, value) |
| pulseIn(pin, value, timeout) | |

**Параметры**

* pin: номер порта вход/выхода, на котором будет ожидаться сигнал. ([int](https://arduino.ru/Reference/Int))
* value: тип ожидаемого сигнала — [**HIGH**](https://arduino.ru/Reference/Constants) или [**LOW**](https://arduino.ru/Reference/Constants)**.**
* timeout (опционально): время ожидания сигнала (таймаут) в микросекундах; по умолчанию - одна секунда. ([unsigned long](https://arduino.ru/Reference/UnsignedLong))

**Возвращаемое значение**

Длина сигнала в микросекундах или 0, если сигнал не получен до истечения таймаута. ([unsigned long](https://arduino.ru/Reference/UnsignedLong))

**Пример**

|  |
| --- |
| int pin = 7; |
| unsigned long duration; | |

|  |
| --- |
|  |
| void setup() | |

|  |
| --- |
| { |
| pinMode(pin, INPUT); | |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| void loop() | |
| { |

|  |  |
| --- | --- |
| duration = pulseIn(pin, HIGH); | |
| } |

[Справочник языка Arduino](https://arduino.ru/Reference)

**Работа со временем**

* [millis](https://arduino.ru/Reference/Millis)()
* [micros](https://arduino.ru/Reference/Micros)()
* [delay](https://arduino.ru/Reference/Delay)()
* [delayMicroseconds](https://arduino.ru/Reference/DelayMicroseconds)()

millis()

Возвращает количество миллисекунд с момента начала выполнения текущей программы на плате Arduino. Это количество сбрасывается на ноль, в следствие переполнения значения, приблизительно через 50 дней.

**Параметры**

нет

**Возвращаемое значение**

Количество миллисекунд с момента начала выполнения программы. ([unsigned long](https://arduino.ru/Reference/UnsignedLong))

**Пример**

|  |  |
| --- | --- |
| unsigned long time; | |
|  |

|  |
| --- |
| void setup(){ |
| Serial.begin(9600); | |

|  |
| --- |
| } |
| void loop(){ | |

|  |  |
| --- | --- |
| Serial.print("Time: "); | |
| time = millis(); |

|  |  |
| --- | --- |
| //выводит количество миллисекунд с момента начала выполнения программы | |
| Serial.println(time); |

|  |  |
| --- | --- |
| // ждет секунду, перед следующей итерацией цикла. | |
| delay(1000); |

|  |
| --- |
| } |

**Смотрите также**

* [delay()](https://arduino.ru/Reference/Delay)
* [delayMicroseconds()](https://arduino.ru/Reference/DelayMicroseconds)
* [micros()](https://arduino.ru/Reference/Micros)

[Справочник языка Arduino](https://arduino.ru/Reference)

micros()

Возвращает количество микросекунд с момента начала выполнения текущей программы на плате Arduino. Значение переполняется и сбрасывается на ноль, приблизительно через 70 минут. На 16MHz платах Ардуино ([Duemilanove](https://arduino.ru/Hardware/ArduinoBoardDuemilanove) и [Nano](https://arduino.ru/Hardware/ArduinoBoardNano)) функция **micros()** имеет разрешение 4 микросекунды (возвращаемое значение всегда кратно 4). На 8MHz платах ([Arduino Lilypad](https://arduino.ru/Hardware/ArduinoBoardLilyPad)) разрешение функции 8 микросекунд.

(в одной секунде 1 000 миллисекунд и 1 000 000 микросекунд)

**Параметры**

нет

**Возвращаемое значение**

Количество микросекунд с момента начала выполнения программы. ([unsigned long](https://arduino.ru/Reference/UnsignedLong))

**Пример**

|  |  |
| --- | --- |
| unsigned long time; | |
|  |

|  |
| --- |
| void setup(){ |
| Serial.begin(9600); | |

|  |
| --- |
| } |
| void loop(){ | |

|  |  |
| --- | --- |
| Serial.print("Time: "); | |
| time = micros(); |

|  |  |
| --- | --- |
| //выводит количество микросекунд с момента начала выполнения программы | |
| Serial.println(time); |

|  |  |
| --- | --- |
| // ждет секунду, перед следующей итерацией цикла. | |
| delay(1000); |

|  |
| --- |
| } |

**Смотрите также**

* [delay()](https://arduino.ru/Reference/Delay)
* [delayMicroseconds()](https://arduino.ru/Reference/DelayMicroseconds)
* [millis()](https://arduino.ru/Reference/Millis)

[Справочник языка Arduino](https://arduino.ru/Reference)

delay()

Останавливает выполнение программы на заданное в параметре количество миллисекунд (1000 миллисекунд в 1 секунде).

**Синтаксис**

|  |
| --- |
| delay(ms) |

**Параметры**

ms: количество миллисекунд, на которое приостанавливается выполнение программы. ([unsigned long](https://arduino.ru/Reference/UnsignedLong))

**Возвращаемое значение**

Нет

**Пример**

[код](https://arduino.ru/Reference/Delay#viewSource)

[расечатать](https://arduino.ru/Reference/Delay#printSource)[?](https://arduino.ru/Reference/Delay#about)

|  |  |
| --- | --- |
| int ledPin = 13;                 // светодиод подключен на порт 13 | |
|  |

|  |  |
| --- | --- |
| void setup() | |
| { |

|  |  |
| --- | --- |
| pinMode(ledPin, OUTPUT);      // устанавливается режим порта - выход | |
| } |

|  |
| --- |
|  |
| void loop() | |

|  |
| --- |
| { |
| digitalWrite(ledPin, HIGH);   // включаем светодиод | |

|  |
| --- |
| delay(1000);                  // ожидаем секунду |
| digitalWrite(ledPin, LOW);    // выключаем светодиод | |

|  |  |
| --- | --- |
| delay(1000);                  // ожидаем секунду | |
| } |

**Замечания по использования функции**

Не рекомендуется использовать эту функцию для событий длиннее 10 миллисекунд, т.к. во время останова, не могут быть произведены манипуляции с портам, не могут быть считаны сенсоры или произведены математические операции. В качестве альтернативного подхода возможно контролирование времени выполнения тех или иных функций с помощью [millis()](https://arduino.ru/Reference/Millis).

Большинство активности платы останавливается функцией **delay()**. Тем не менее работа прерываний не останавливается, продолжается запись последовательно (serial) передаваемых данных на RX порту, ШИМ сигнал ([analogWrite](https://arduino.ru/Reference/AnalogWrite)) продолжает генерироваться на портах.

**Смотрите также**

* [delayMicroseconds()](https://arduino.ru/Reference/DelayMicroseconds)
* [millis()](https://arduino.ru/Reference/Millis)
* [micros()](https://arduino.ru/Reference/Micros)

[Справочник языка Arduino](https://arduino.ru/Reference)

delayMicroseconds()

Останавливает выполнение программы на заданное в параметре количество микросекунд (1 000 000 микросекунд в 1 секунде).

В данной версии Ардуино максимальная пауза воспроизводимая корректно — 16383. Возможно это будет изменено в следующих версиях Arduino. Для остановки выполнения программы более чем на несколько тысяч микросекунд рекомендуется использовать функцию [delay()](https://arduino.ru/Reference/Delay).

**Синтаксис**

|  |
| --- |
| delayMicroseconds(us) |

**Параметры**

us: количество микросекунд, на которое приостанавливается выполнение программы. ([unsigned int](https://arduino.ru/Reference/UnsignedInt))

**Возвращаемое значение**

Нет

**Пример**

|  |  |
| --- | --- |
| int outPin = 8;                 // цифровой порт вход/выхода 8 | |
|  |

|  |  |
| --- | --- |
| void setup() | |
| { |

|  |  |
| --- | --- |
| pinMode(outPin, OUTPUT);      // устанавливается режим порта - выход | |
| } |

|  |
| --- |
|  |
| void loop() | |

|  |
| --- |
| { |
| digitalWrite(outPin, HIGH);   // подаем HIGH на выход | |

|  |
| --- |
| delayMicroseconds(50);        // ожидаем 50 микросекунд |
| digitalWrite(outPin, LOW);    // устанавливаем LOW на выходе | |

|  |  |
| --- | --- |
| delayMicroseconds(50);        // ожидаем 50 микросекунд | |
| } |

**Смотрите также**

* [delay()](https://arduino.ru/Reference/Delay)
* [millis()](https://arduino.ru/Reference/Millis)
* [micros()](https://arduino.ru/Reference/Micros)

[Справочник языка Arduino](https://arduino.ru/Reference)

**Математические функции**

* [min](https://arduino.ru/Reference/Min)()
* [max](https://arduino.ru/Reference/Max)()
* [abs](https://arduino.ru/Reference/Abs)()
* [constrain](https://arduino.ru/Reference/Constrain)()
* [map](https://arduino.ru/Reference/Map)()
* [pow](https://arduino.ru/Reference/Pow)()
* [sq](https://arduino.ru/Reference/Sq)()
* [sqrt](https://arduino.ru/Reference/Sqrt)()

min(x,yx)

Возвращает наименьшее из двух значений.

**Параметры**

* x: первое число, любой тип
* y: второе число, любой тип

**Возвращаемое значение**

Возвращает меньшее из двух сравниваемых значений.

**Пример**

|  |
| --- |
| sensVal = min(sensVal, 100); // проверяем если sensVal больше 100, то senseVal будет присвоено 100 |

**Замечания по использованию функции**

Функция **min()** зачастую используется для ограничения верхней границы значений переменной. Функцией [max()](https://arduino.ru/Reference/Max) ограничивают нижнюю границу переменной.

В силу специфики реализации функции **min()** следует избегать использования других функций в качестве параметров.

|  |  |
| --- | --- |
| min(a++, 100);   // может привести к некорректным результатам | |
|  |

|  |
| --- |
| a++; |
| min(a, 100);    // так корректно | |

**Смотрите также**

* [max()](https://arduino.ru/Reference/Max)
* [constrain()](https://arduino.ru/Reference/Constrain)

[Справочник языка Arduino](https://arduino.ru/Reference)

max(x, y)

Возвращает большее из двух значений.

**Параметры**

* x: первое число, любой тип
* y: второе число, любой тип

**Возвращаемое значение**

Возвращает большее из двух сравниваемых значений.

**Пример**

|  |
| --- |
| sensVal = max(sensVal, 20); // проверяем если sensVal меньше 20, то senseVal будет присвоено 20 |

**Замечания по использованию функции**

Функция **max()** зачастую используется для ограничения нижней границы значений переменной. Функцией [min()](https://arduino.ru/Reference/Min) ограничивают верхнюю границу переменной.

В силу специфики реализации функции **max()** следует избегать использования других функций в качестве параметров.

|  |  |
| --- | --- |
| max(a--, 0);   // может привести к некорректным результатам | |
|  |

|  |
| --- |
| a--; |
| max(a, 0);    // так корректно | |

**Смотрите также**

* [min()](https://arduino.ru/Reference/Min)
* [constrain()](https://arduino.ru/Reference/Constrain)

[Справочник языка Arduino](https://arduino.ru/Reference)

abs()

Возвращает модуль числа.

**Параметры**

x: число

**Возвращаемое значение**

* **x**: если **x** больше или равен 0.
* **-x**: если **x** меньше 0.

**Замечания по использованию функции**

В силу специфики реализации функции **abs()** следует избегать использования других функций в качестве параметров.

|  |  |
| --- | --- |
| abs (a++);   // может привести к некорректным результатам | |
|  |

|  |
| --- |
| a++; |
| abs(a, 0);    // так корректно | |

[Справочник языка Arduino](https://arduino.ru/Reference)

constrain(x, a, b)

Функция проверяет и если надо задает новое значение, так чтобы оно была в области допустимых значений, заданной параметрами.

**Параметры**

* x: проверяемое значение, любой тип
* a: нижняя граница области допустимых значений, любой тип
* b: верхняя граница области допустимых значений, любой тип

**Возвращаемое значение**

* **x**: если **x** входит в область допустимых значений [**a**..**b**]
* **a**: если **x**  меньше **a**
* **b**: если **x** больше **b**

**Пример**

|  |
| --- |
| sensVal = constrain(sensVal, 10, 150); |
| // ограничиваем значения sensVal диапазоном от 10 до 150 | |

**Смотрите также**

* [min()](https://arduino.ru/Reference/Min)
* [max()](https://arduino.ru/Reference/Max)

[Справочник языка Arduino](https://arduino.ru/Reference)

map(value, fromLow, fromHigh, toLow, toHigh)

Функция пропорционально переносит значение (**value**) из текущего диапазона значений (**fromLow** .. **fromHigh**) в новый диапазон (**toLow** .. **toHigh**), заданный параметрами.

Функция **map()** не ограничивает значение рамками диапазона, как это делает функция [constrain()](https://arduino.ru/Reference/Constrain). **Contrain()**может быть использован до или после вызова **map()**, если необходимо ограничить допустимые значения заданным диапазоном.

Обратите внимание, что "нижняя граница" может быть как меньше, так и больше "верхней границы". Это может быть использовано для того чтобы "перевернуть" диапазон:

|  |
| --- |
| y = map(x, 1, 50, 50, 1); |

Возможно использование отрицательных значений:

|  |
| --- |
| y = map(x, 1, 50, 50, -100); |

Функция **map()** оперирует целыми числами. При пропорциональном переносе дробная часть не округляется по правилами, а просто отбрасывается.

**Параметры**

* *value*: значение для переноса
* *fromLow*: нижняя граница текущего диапазона
* *fromHigh*: верхняя граница текущего диапазона
* *toLow*: нижняя граница нового диапазона, в который переноситься значение
* *toHigh*: верхняя граница нового диапазона

**Возвращаемое значение**

Значение в новом диапазоне

**Пример**

|  |  |
| --- | --- |
| /\* Переносим значение с аналогового входа (возможные значения от 0 до 1023) в 8 бит (0..255) \*/ | |
| void setup() {} |

|  |
| --- |
|  |
| void loop() | |

|  |
| --- |
| { |
| int val = analogRead(0); | |

|  |  |
| --- | --- |
| val = map(val, 0, 1023, 0, 255); | |
| analogWrite(9, val); |

|  |
| --- |
| } |

**Дополнительно**

Математически функция **map()** может быть записана так:

|  |  |
| --- | --- |
| long map(long x, long in\_min, long in\_max, long out\_min, long out\_max) | |
| { |

|  |  |
| --- | --- |
| return (x - in\_min) \* (out\_max - out\_min) / (in\_max - in\_min) + out\_min; | |
| } |

**Смотрите также**

* [constrain()](https://arduino.ru/Reference/Constrain)

[Справочник языка Arduino](https://arduino.ru/Reference)

pow(base, exponent)

Вычисляет значение возведенное в заданную степень. **pow()** может возводить в дробную степень.

**Параметры**

* *base*: число ([float](https://arduino.ru/Reference/Float))
* *exponent*: степень, в которую будет возводиться число ([float](https://arduino.ru/Reference/Float))

**Возвращаемое значение**

Результат возведения в степень, число ([double](https://arduino.ru/Reference/Double)).

**Смотрите также**

* [double](https://arduino.ru/Reference/Double)
* [float](https://arduino.ru/Reference/Float)
* [sq()](https://arduino.ru/Reference/Sq)

[Справочник языка Arduino](https://arduino.ru/Reference)

sq(x)

Функция возвращает квадрат числа, заданного параметром.

**Параметры**

* x: число, любой тип

**Возвращаемое значение**

Квадрат числа

**Смотрите также**

* [pow()](https://arduino.ru/Reference/Pow)
* [sqrt()](https://arduino.ru/Reference/Sqrt)

[Справочник языка Arduino](https://arduino.ru/Reference)

sqrt(x)

Функция вычисляет квадтратный корень числа, заданного параметром.

**Параметры**

* x: число, любой тип

**Возвращаемое значение**

Квадратный корень числа ([double](https://arduino.ru/Reference/Double))

**Смотрите также**

* [pow()](https://arduino.ru/Reference/Pow)
* [sq()](https://arduino.ru/Reference/Sq)

[Справочник языка Arduino](https://arduino.ru/Reference)

**Тригонометрические функции**

* [sin](https://arduino.ru/Reference/Sin)()
* [cos](https://arduino.ru/Reference/Cos)()
* [tan](https://arduino.ru/Reference/Tan)()

sin(rad)

Возвращает синус угла, заданного в радианах. Результат функции всегда в диапазоне -1..1.

**Параметры**

* rad: угол в радианах ([float](https://arduino.ru/Reference/Float))

**Возвращаемое значение**

Синус угла ([double](https://arduino.ru/Reference/Double))

**Замечания по использованию функции**

Функции Serial.print() и Serial.println() в текущей версии не могут корректно выводить [float](https://arduino.ru/Reference/Float).

**Смотрите также**

* [cos()](https://arduino.ru/Reference/Cos)
* [tan()](https://arduino.ru/Reference/Tan)

[Справочник языка Arduino](https://arduino.ru/Reference)

cos(rad)

Возвращает косинус угла, заданного в радианах в параметре. Результат функции всегда находится в диапазоне -1..1.

**Параметры**

* rad: угол в радианах ([float](https://arduino.ru/Reference/Float))

**Возвращаемое значение**

Косинус угла ([double](https://arduino.ru/Reference/Double))

**Замечания по использованию функции**

Функции Serial.print() и Serial.println() в текущей версии не могут корректно выводить [float](https://arduino.ru/Reference/Float).

**Смотрите также**

* [sin()](https://arduino.ru/Reference/Sin)
* [tan()](https://arduino.ru/Reference/Tan)

[Справочник языка Arduino](https://arduino.ru/Reference)

tan(rad)

Возвращает тангенс угла, заданного в радианах в передаваемом параметре. Результат функции в диапазоне от минус бесконечность до плюс бесконечность.

**Параметры**

* rad: угол в радианах ([float](https://arduino.ru/Reference/Float))

**Возвращаемое значение**

Тангенс угла ([double](https://arduino.ru/Reference/Double))

**Замечания по использованию функции**

Функции Serial.print() и Serial.println() в текущей версии не могут корректно выводить [float](https://arduino.ru/Reference/Float).

**Смотрите также**

* [sin()](https://arduino.ru/Reference/Sin)
* [cos()](https://arduino.ru/Reference/Cos)

[Справочник языка Arduino](https://arduino.ru/Reference)

**Генераторы случайных значений**

* [randomSeed](https://arduino.ru/Reference/RandomSeed)()
* [random](https://arduino.ru/Reference/Random)()

randomSeed(seed)

Функция **RandomSeed()** инициализирует генератор псевдослучайных чисел. Генерируемая последовательность случайных чисел очень длинная, и всегда одна и та же. Точка в этой последовательности, с которой начинается генерация чисел, зависит от параметра *seed*.

Если при каждом запуске программы необходимо получать разные последовательности значений, генерируемых функцией [random()](https://arduino.ru/Reference/Random), то необходимо инициализировать генератор псевдослучайных чисел со случайным параметром. Например, можно использовать значение, отдаваемое функцией [analogRead()](https://arduino.ru/Reference/AnalogRead) c неподключенного порта вход/выхода.

В некоторых случаях необходимо получать одинаковую последовательность при каждом запуске программы на Arduino. В этом случае инициализировать генератор псевдослучайных чисел следует вызовом функции **randomSeed()** с фиксированным параметром.

**Параметры**

* *seed*: параметр, задающий начало выдачи псевдослучайных значений на последовательности (см. выше). ([int](https://arduino.ru/Reference/Int), [long](https://arduino.ru/Reference/Long))

**Возвращаемое значение**

нет

**Пример**

|  |  |
| --- | --- |
| long randNumber; | |
|  |

|  |
| --- |
| void setup(){ |
| Serial.begin(9600); | |

|  |  |
| --- | --- |
| randomSeed(analogRead(0)); | |
| } |

|  |
| --- |
|  |
| void loop(){ | |

|  |
| --- |
| randNumber = random(300); |
| Serial.println(randNumber); | |

|  |
| --- |
|  |
| delay(50); | |

|  |
| --- |
| } |

**Смотрите также**

* [random()](https://arduino.ru/Reference/Random)

[Справочник языка Arduino](https://arduino.ru/Reference)

random()

Функция **random()** возвращает псевдослучайное число.

**Синтаксис**

|  |
| --- |
| random(max) |
| random(min, max) | |

**Параметры**

* *min*: нижняя граница случайных значений, включительно. (опционально)
* *max*: верхняя граница случайных значений, не включительно.

**Возвращаемое значение**

Случайное число между *min* и *max*-1. ([long](https://arduino.ru/Reference/Long))

**Замечание по использованию**

Если при каждом запуске программы необходимо получать разные последовательности значений, генерируемых функцией **random()**, то необходимо инициализировать генератор псевдослучайных чисел со случайным параметром. Например, можно использовать значение, отдаваемое функцией [analogRead()](https://arduino.ru/Reference/AnalogRead) c неподключенного порта вход/выхода.

В некоторых случаях необходимо получать одинаковую последовательность при каждом запуске программы на Arduino. В этом случае инициализировать генератор псевдослучайных чисел следует вызовом функции [randomSeed()](https://arduino.ru/Reference/RandomSeed) с фиксированным параметром.

**Пример**

|  |  |
| --- | --- |
| long randNumber; | |
|  |

|  |
| --- |
| void setup(){ |
| Serial.begin(9600); | |

|  |
| --- |
|  |
| // если порт 0 неподключен, то генератор псевдослучайных чисел | |

|  |  |
| --- | --- |
| // будет инициализироваться функцией randomSeed() со случайного | |
| // значения при каждом запуске программы из-за "шума" на порту |

|  |  |
| --- | --- |
| randomSeed(analogRead(0)); | |
| } |

|  |
| --- |
|  |
| void loop() { | |

|  |  |
| --- | --- |
| // выводим случайное число из диапазона 0..299 | |
| randNumber = random(300); |

|  |  |
| --- | --- |
| Serial.println(randNumber); | |
|  |

|  |  |
| --- | --- |
| // выводим случайное число из диапазона 10..19 | |
| randNumber = random(10, 20); |

|  |  |
| --- | --- |
| Serial.println(randNumber); | |
|  |

|  |  |
| --- | --- |
| delay(50); | |
| } |

**Смотрите также**

* [randomSeed()](https://arduino.ru/Reference/RandomSeed)

[Справочник языка Arduino](https://arduino.ru/Reference)

**Внешние прерывания**

* [attachInterrupt()](https://arduino.ru/Reference/AttachInterrupt)
* [detachInterrupt()](https://arduino.ru/Reference/DetachInterrupt)

attachInterrupt(interrupt, function, mode)

Задает функцию обработки внешнего прерывания, то есть функция, которая будет вызвана по внешнему прерыванию. Если до это была задана другая функция, то назначается новая. Большинство контроллеров Arduino умеют обрабатывать до двух внешних прерываний, пронумерованных так: 0 (на цифровом порту 2) и 1 (на цифровом порту 3). [Arduino Mega](https://arduino.ru/Hardware/ArduinoBoardMega2560) обрабатывает дополнительно еще четыра прерывания: 2 (порт 21), 3 (порт 20), 4 (порт 19) и 5 (порт 18).

| **Плата** | **int.0** | **int.1** | **int.2** | **int.3** | **int.4** | **int.5** |
| --- | --- | --- | --- | --- | --- | --- |
| UNO, Ethernet | 2 | 3 |  |  |  |  |
| Mega2560 | 2 | 3 | 21 | 20 | 19 | 18 |
| Leonardo | 3 | 2 | 0 | 1 | 7 |  |

[Arduino Due](https://arduino.ru/Hardware/ArduinoDue) имеет мощные возможности обработки прерываний, что позволяет прикрепить функцию прерывания для всех доступных контактов. Вы можете напрямую указать порт(пин) в attachInterrupt ().

**Параметры**

**interrupt**: номер прерывания ([*int*](https://arduino.ru/Reference/IntCast))   
или **pin**: номер цифрового порта (только для [Arduino Due](https://arduino.ru/Hardware/ArduinoDue))

**function**: функция, вызваемая прерыванием, функция должна быть без параметров и не возвращать значений. В англоязычной документации употребляется термин *interrupt service routine для такой функции.*

**mode** задает режим обработки прерывания. Допустимо использование следующих констанст:

* **LOW** вызывает прерывание, когда на порту [LOW](https://arduino.ru/Reference/Constants)
* **CHANGE** прерывание вызывается при смене значения на порту, с LOW на HIGH и наоборот
* **RISING** прерывание вызывается только при смене значения на порту с LOW на HIGH
* **FALLING** прерывание вызывается только при смене значения на порту с HIGH на LOW

**Возвращаемое значение**

Нет

**Замечание по использованию**

*Внутри функции обработки прерывания не работает*[*delay()*](https://arduino.ru/Reference/Delay)*, значения возвращаемые*[*millis()*](https://arduino.ru/Reference/Millis)*не изменяются. Возможна потеря данный передаваемых по последовательному соединению (Serial data) в момент выполнения функциии обработки прерывания. Переменные, изменяемые в функции, должным быть объявлены как volatile.*

**Использование перерываний**

Прерывания обычно используют для задач, которые должны быть выполнены автоматически при наступление какого либо внешнего воздействия. Например, считывание значения энкодера (датчика угла) или реакция на действия пользователя.

Так, если вы хотите считывать значения энкодера без использования прерываний, то написать программу, которая бы не пропускала ни одного импульса, практически невозможно, такая программа должна будет практически все время считывать значения с датчика, чтобы не пропусть импульс. На другие операции просто не останется процессорного времени. Это же относится и к другим датчикам, выдающим короткий импульс. В таких задачах использования внешних прерываний позволяет разгрузить процессор для других операций и не пропустить ожидаемый сигнал.

**Пример**

|  |
| --- |
| int pin = 13; |
| volatile int state = LOW; | |

|  |
| --- |
|  |
| void setup() | |

|  |
| --- |
| { |
| pinMode(pin, OUTPUT); | |

|  |  |
| --- | --- |
| attachInterrupt(0, blink, CHANGE); | |
| } |

|  |
| --- |
|  |
| void loop() | |

|  |
| --- |
| { |
| digitalWrite(pin, state); | |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| void blink() | |
| { |

|  |  |
| --- | --- |
| state = !state; | |
| } |

**Смотрите также**

* [detachInterrupt](https://arduino.ru/Reference/DetachInterrupt)

[Справочник языка](https://arduino.ru/Reference)

detachInterrupt(interrupt)

Выключает обработку внешнего прерывания.

**Параметры**

interrupt: номер прерывания (0 или 1). Для Arduino Mega еще 2, 3, 4 или 5

**Смотрите также**

* [attachInterrupt](https://arduino.ru/Reference/AttachInterrupt)()

[Справочник языка](https://arduino.ru/Reference)

**Функции передачи данных**

* [Serial](https://arduino.ru/Reference/Serial)

Serial

Набор функций **Serial** служит для связи устройства Ардуино с компьютером или другими устройствами, поддерживающими последовательный интерфейс обмена данными. Все платы Arduino имеют хотя бы один последовательный порт ([UART](https://ru.wikipedia.org/wiki/UART), иногда называют USART). Для обмена данными **Serial** используют цифровые порты ввод/вывода 0 (RX) и 1 (TX), а также USB порт. Важно учитывать, что если вы используете функции **Serial**, то нельзя одновременно с этим использовать порты 0 и 1 для других целей.

[Среда разработки Arduino](https://arduino.ru/Arduino_environment) имеет встроенный монитор последовательного интерфейса (Serial monitor). Для начала обмена данными необходимо запустить монитор нажатием кнопки Serial monitor и выставить ту же скорость связи (baud rate), с которой вызвана функция begin().

Плата [Arduino Mega](https://arduino.ru/Hardware/ArduinoBoardMega) имеет три дополнительных последовательных порта: **Serial1** на портах 19 (RX) и 18 (TX), **Serial2** на портах на портах 17 (RX) и 16 (TX), **Serial3** на портах на портах 15 (RX) и 14 (TX). Чтобы использовать эти порты для связи с компьютером понадобится дополнительные адаптеры USB-to-serial, т.к. они не подключены к встроенному адаптеру платы Mega. Для связи с внешним устройством через последовательный интерфейс соедините TX порт вашего устройства с RX портом внешнего устройства и RX порт вашего устройства с портом TX внешнего и соедините "землю" на устройствах. (Важно! Не подключайте эти порты напрямую к RS232 порту, это может повредить плату).

**Функции**

* [begin](https://arduino.ru/Serial/Begin)()
* [end](https://arduino.ru/Serial/End)()
* [available](https://arduino.ru/Reference/Serial/Available)()
* [read](https://arduino.ru/Reference/Serial/Read)()
* [flush](https://arduino.ru/Reference/Serial/Flush)()
* [print](https://arduino.ru/Reference/Serial/Print)()
* [println](https://arduino.ru/Reference/Serial/Println)()
* [write](https://arduino.ru/Reference/Serial/Write)()
* [peek](https://arduino.ru/Reference/Serial/Peek)()

[Справочник языка Arduino](https://arduino.ru/Reference)

**Библиотеки Arduino**

[Servo](https://arduino.ru/Reference/Library/Servo) — библиотека управления сервоприводами.  
[EEPROM](https://arduino.ru/Reference/Library/EEPROM) — чтение и запись энергонезависимой памяти микроконтроллера.  
[SPI](https://arduino.ru/Reference/Library/SPI) — библиотека, реализующая передачу данных через интерфейс SPI.  
[Stepper](https://arduino.ru/Reference/Library/Stepper) — библиотека управления шаговыми двигателями.

Библиотека Servo

Пт, 18/03/2011 - 17:36 | by **Alexander**

Эта библиотека функций для Arduino контроллера предоставляет набор функций для управления сервоприводами. Стандартные сервоприводы позволяют поворачивать привод на опредленный угол от 0 до 180 градусов обычно. Некоторые сервоприводы позволяют совершать полные обороты на заданной скорости.

Библиотека Servo позволяет одновременно управлять 12-ю сервоприводами на большинстве плат Arduino и 48-ю на [Arduino Mega](https://arduino.ru/Hardware/ArduinoBoardMega2560). На контроллерах отличных от Mega использование библиотеки отключает возможность использовать выходы 9 и 10 в [режиме ШИМ](https://arduino.ru/Tutorial/PWM) даже если привод не подключен к этим выводам. На плате Mega могут быть использованы до 12 сервоприводов без потери функционала ШИМ. При использовании Mega для управления от 12 до 23 сервоприводов нельзя будет использовать выходы 11 и 12 для ШИМ.

**Подключение**

В общем случае сервопривод подключается 3-мя проводами : питание, земля и сигнальный. Обычно питание - красный провод и может быть подключен к выводу +5V на плате Arduino. Черный провод земля подключается к GND выводу Arduino, сигнальный, обычно желты, провод подключается к цифровому выводу котроллера Arduino. Следует отметить, что мощные сервоприводы могут создавать большую нагрузку, в этом случает он должен быть запитан отдельно (не через выход +5V Arduino). Тоже самое верно для случая подключения сразу нескольких сервоприводов. Убедитесь, что привод и контроллер подключены к общей земле.

**Функции**

* [attach()](https://arduino.ru/Reference/Library/Servo/attach)
* [write()](https://arduino.ru/Reference/Library/Servo/write)
* [writeMicroseconds()](https://arduino.ru/Reference/Library/Servo/writeMicroseconds)
* [read()](https://arduino.ru/Reference/Library/Servo/read)
* [attached()](https://arduino.ru/Reference/Library/Servo/attached)
* [detach()](https://arduino.ru/Reference/Library/Servo/detach)

## Библиотека EEPROM

Вс, 27/03/2011 - 04:51 | by **Alexander**

Микрокотроллеры  ATmega  имеют свою энергонезависимую память, то есть у пользователей Ардуино есть возможность сохранять данные в этой памяти и они могут быть использованы после выключения-включения или перезагрузки контроллера. Arduino библиотека EEPROM предоставляет удобный и простой интерфейс работы с этой памятью.

Разные модели микроконтроллеров различаются объемом EEPROM памяти, так ATMega328 имеет 1024 байт, 512 байт у ATmega168 и ATmega8 и по 4Кб (4096 байт) у ATmega1280 и ATmega2560.

#### Функции

* [read](https://arduino.ru/Reference/Library/EERPOM/read)()
* [write](https://arduino.ru/Reference/Library/EERPOM/write)()

Библиотека SPI

Сб, 02/04/2011 - 01:57 | by **Alexander**

Библиотека SPI позволяет контроллеру Arduino взаимодействовать с устройствами поддерживающими SPI протокол. Arduino в данном случае выступает в качестве ведущего устройтва.

**Коротко о Serial Peripheral Interface (SPI).**

Последоваетельный периферийный интерфейс (SPI) — это последовательный синхроный протокол передачи данных используемый микроконтроллерами для обмена данными с одним или несколькими периферийными устройствами на небольших растояниях.

Для организации соединения SPI необходимо одно ведущее устройство, обычно это микроконтроллер, которое управляет соединением с ведомыми устройствами. Обычно подключение осуществляется тремя общими линиями и линией выбора переферийного(ведомого) устройства:

* Master In Slave Out (MISO), переводится как  "вход ведущего выход ведомого", используется для передачи данных от ведомого к ведущему.
* Master Out Slave In (MOSI) — выход ведущего вход ведомого, для передачи данных от ведущего к периферийным устройствам.
* Serial Clock (SCK) — синхронизирующая линия, синхросигнал генерируется ведущим устройством.
* Slave Select pin — вход на ведомых устройствах с помощью которого ведущий может инициировать обмен данными с периферийным устройством. Если на этом входе [LOW,](https://arduino.ru/Reference/Constants) то ведомый взаимодействует с ведущим, если [HIGH](https://arduino.ru/Reference/Constants), то ведомый игнорирует сигналы от ведущего.

 При работе с SPI устройствами надо учитывать следующие моменты:

* Какой порядок вывода данных используется:  Most Significant Bit (MSB - старший бит (разряд)) or Least Significant Bit (LSB - младший бит) первый. Порядок может быть изменен функцией [SPI.setBitOrder](https://arduino.ru/Reference/Library/SPI/setBitOrder)().
* Уровень сигнала синхронизации — по какому синхронизирующему сигналу (HIGH или LOW) передаются данные.
* Фаза синхронизации — влияет на последовательность установки и выборки данных. Фаза синхронизации SPI и уровень сигнала задается функцией [SPI.setDataMode](https://arduino.ru/Reference/Library/SPI/setDataMode)().
* Скорость на которой работает SPI устанавливается функцией [SPI.setClockDivider](https://arduino.ru/Reference/Library/SPI/setClockDivider)().

Производители SPI устройств несколько по разному реализуют протокол, поэтому необходимо внимательно ознакомиться с техническим описанием к устройству. Комбинация фазы синхронизиции (CPHA) и уровня сигнала синхронизации (CPOL) задают режим логики работы интерфейса SPI. Режим устанавливается функцией [SPI.setDataMode](https://arduino.ru/Reference/Library/SPI/setDataMode)().

|  |  |  |
| --- | --- | --- |
| **Режим** | **Уровень сигнала (CPOL)** | **Фаза (CPHA)** |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |

**Подключение**

На контроллерах Arduino Duemilanove и других на базе ATmega168 /328, шина SPI использует выходы 10 (SS), 11 (MOSI), 12 (MISO), и 13 (SCK). На Arduino Mega — 50 (MISO), 51 (MOSI), 52 (SCK), и 53 (SS). Обратите внимание, что даже если вы не используете выход SS, он должен быть установлен как выход, в противном случае интерфейс может оказаться в режиме ведомого и библиотека не будет работать как надо.

В качестве SS выхода может быть использован выход отличный от 10-го. Например, при работе с Arduino Ethernet shield контроллер использует выход 4 для взаимодействия с SD картой по SPI и выход 10 для работы с Ethernet контроллером.

**Функции**

* [begin](https://arduino.ru/Reference/Library/SPI/begin)()
* [end](https://arduino.ru/Reference/Library/SPI/end)()
* [setBitOrder](https://arduino.ru/Reference/Library/SPI/setBitOrder)()
* [setClockDivider](https://arduino.ru/Reference/Library/SPI/setClockDivider)()
* [setDataMode](https://arduino.ru/Reference/Library/SPI/setDataMode)()
* [transfer](https://arduino.ru/Reference/Library/SPI/transfer)()

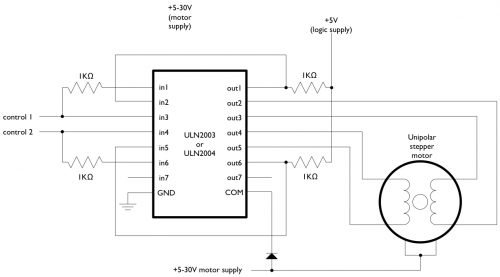
Выражаем благодарность Modular за помощь в переводе библиотеки

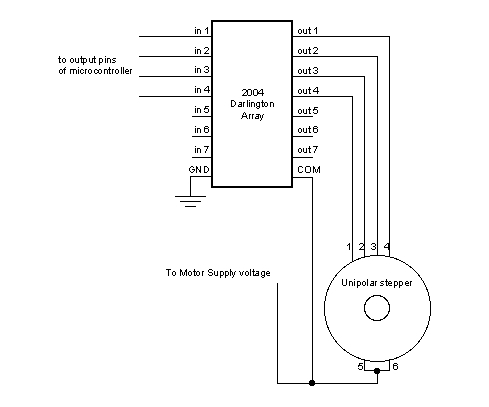
Библиотека Stepper

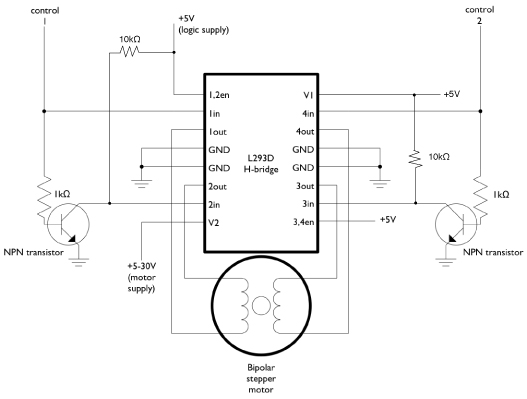
Вс, 10/04/2011 - 02:45 | by **Alexander**

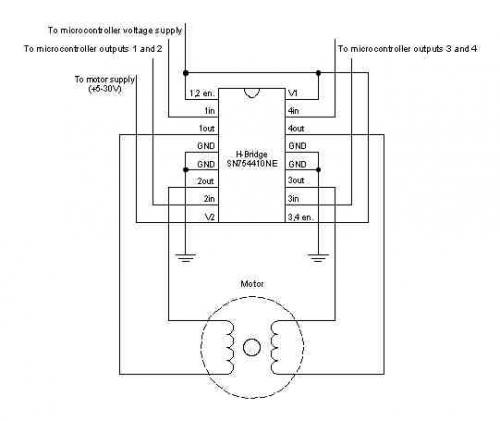
Библиотека Stepper предоставляет удобный интерфейс управления биполярными и униполяными шаговыми двигателями. Для управления шаговым двигателем, в зависимости от его типа (биполярный или униполярный) и выбранного способа подключения, понадобятся некоторые дополнительные электронные компоненты.

**Подключение**

1) **Униполярный шаговый двигатель, управление через 2 выхода**. В данном и следующем подключение используется микросхема драйвера нагрузки ULN2003 или ULN2004 (Darlington array).  
[](https://arduino.ru/sites/default/files/Reference/stepper/unipolar_stepper_2.png)

2) **Униполярный шаговый двигатель, управление через 4 выхода**.  


3) **Биполярный шаговый двигатель, управление через 2 выхода**. Подключение на примере микросхемы L293D.  


4) **Биполярный шаговый двигатель, управление через 4 выхода**. Подключение с помощью микросхемы SN754410NE.  
[](https://arduino.ru/sites/default/files/Reference/stepper/bipolar_stepper_4.jpg)

**Функции**

* [Stepper](https://arduino.ru/Reference/Library/Stepper/StepperConstructor)(steps, pin1, pin2)
* [Stepper](https://arduino.ru/Reference/Library/Stepper/StepperConstructor)(steps, pin1, pin2, pin3, pin4)
* [setSpeed](https://arduino.ru/Reference/Library/Stepper/StepperSetSpeed)(rpm)
* [step](https://arduino.ru/Reference/Library/Stepper/StepperStep)(steps)

**Пример**

* Motor Knob (Cкоро будет)

[Справочник языка](https://arduino.ru/Reference)

**N76E003 макетная плата, MS51FB9AE макетная плата, Nuvoton MCU.**

51 scheda di sviluppo MCU scheda di apprendimento STC89C52 sistema minimo AVR AT89S51 scheda di prova per auto intelligente

// Односерийные комментарии

/\*многострочные комментарии

\*/

#incluide // подключить файл (библиотеку)

void setup() {

  // put your setup code here, to run once:

}

    //всё наладится внутрь{} будет выполнено один раз при нагрузке Ардуина

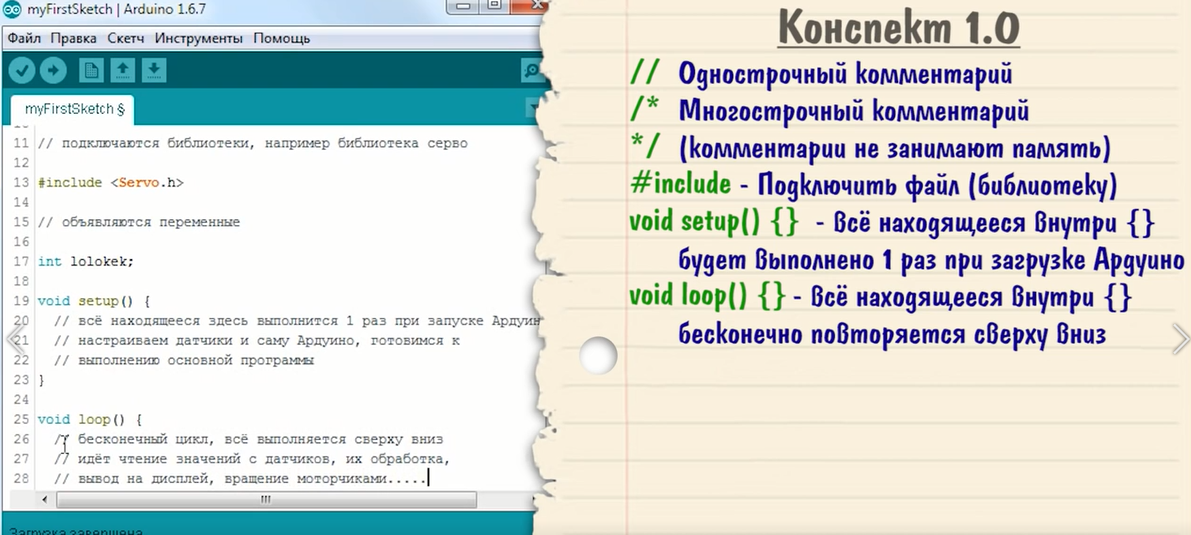
void loop() {

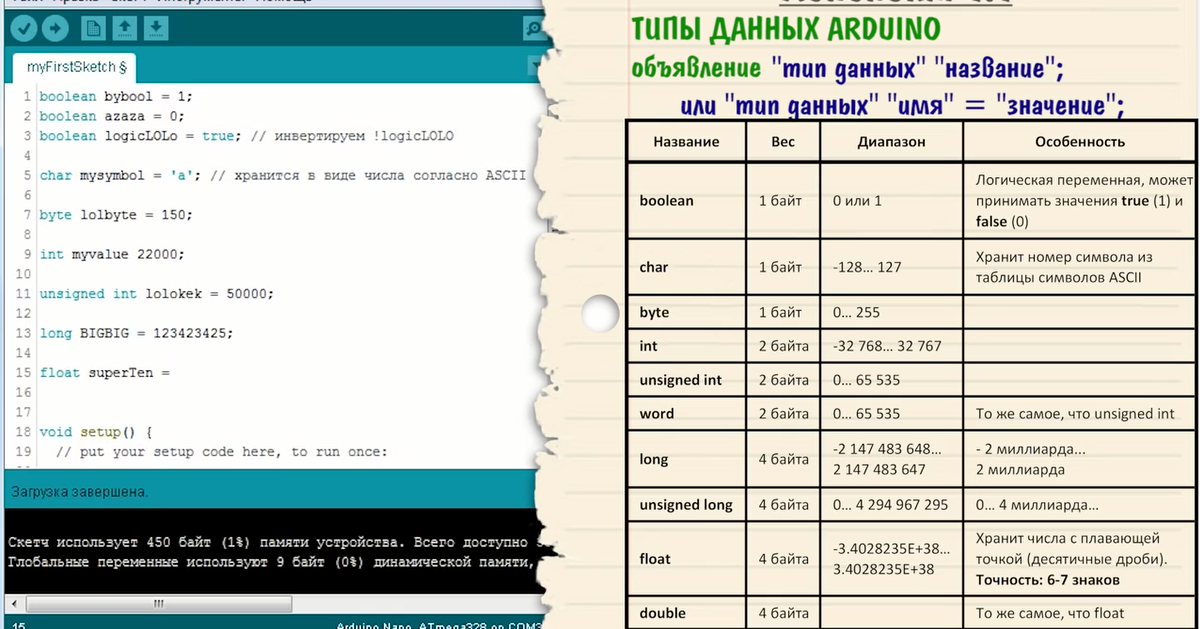
  // put your main code here, to run repeatedly:

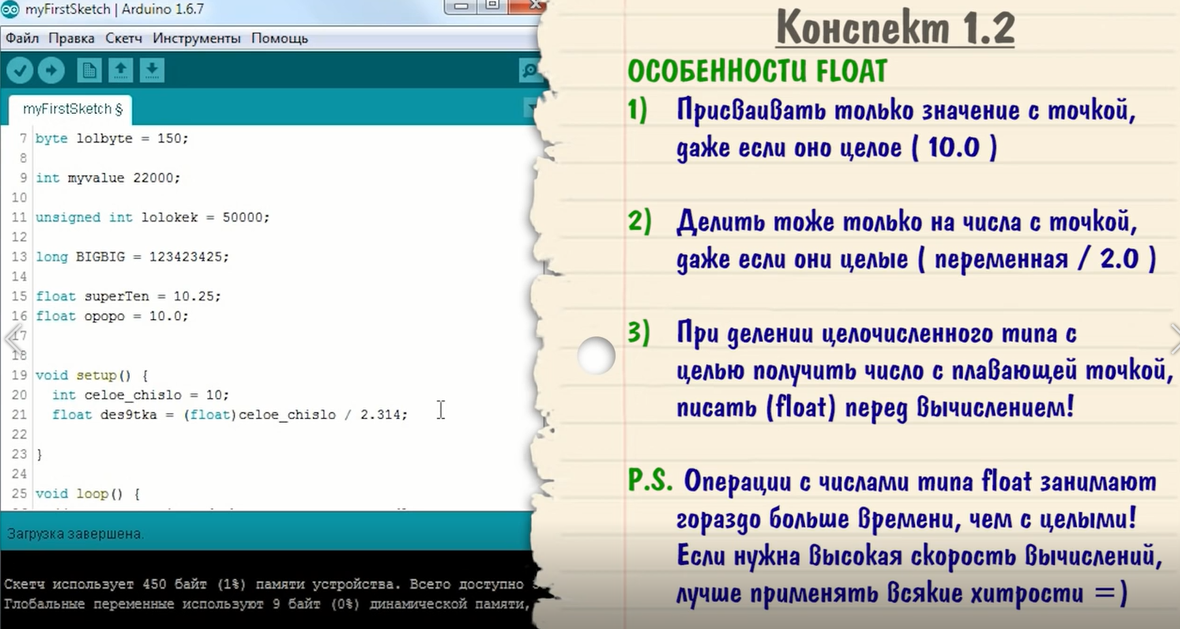
}

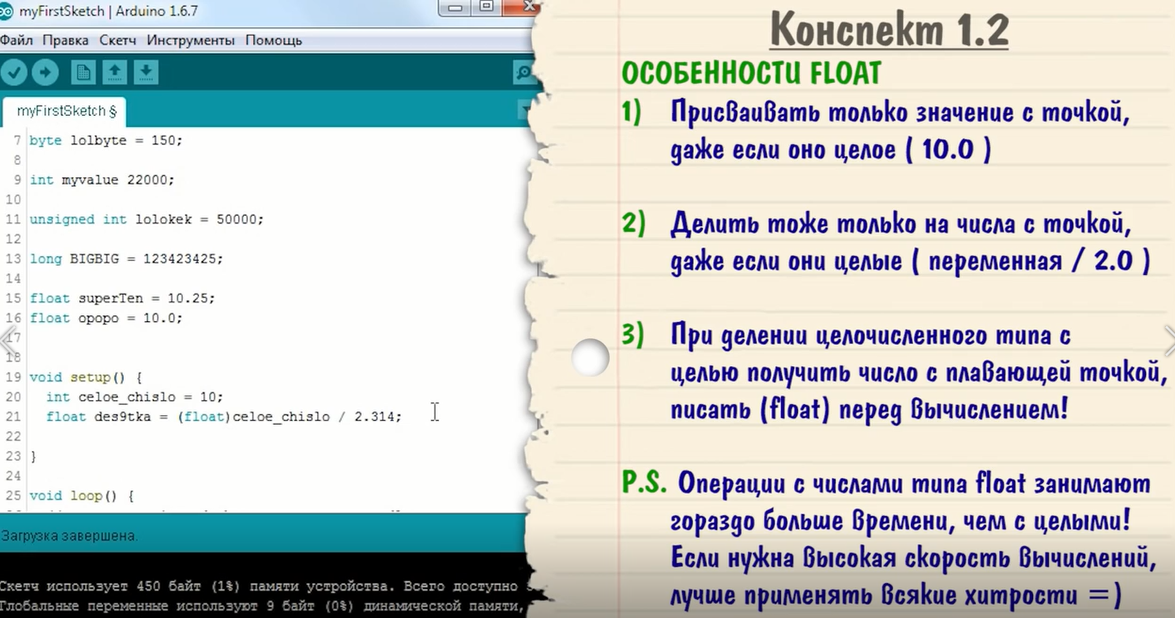
//всё наладится внутрь{} бесконечно повторяется сверху вниз

int myword=100;//это  объявления тип данных. названиеи. имя . значение. и т. д.









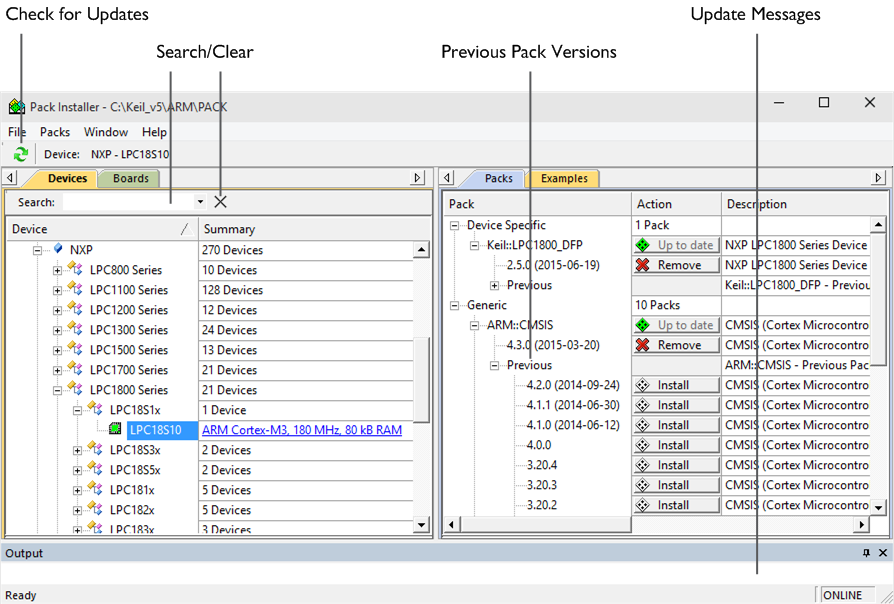
**Pack Installer**

Pack Installer takes care of downloading, installing, updating, and managing the different Software Packs. They can be added any time to [MDK Core](https://www.keil.com/mdk5/core). This makes new device support and middleware updates independent from the toolchain.

Using Pack Installer, you can choose from [devices](https://www.keil.com/dd2) and [boards](https://www.keil.com/boards2) to get quick access to example projects that can be used as a reference for your own development.

Pack Installer обеспечивает загрузку, установку, обновление и управление различными пакетами программного обеспечения. Их можно добавить в **MDK Core** в любое время. Это делает поддержку новых устройств и обновления промежуточного программного обеспечения независимыми от цепочки инструментов.

Используя **Pack Installer**, вы можете выбирать **устройства** и **платы**, чтобы получить быстрый доступ к примерам проектов, которые можно использовать в качестве эталона для собственной разработки.



* Using the **Devices** tab you can filter for microcontrollers supported by MDK.
* The **Boards** tab offers the same functionality to look for development boards.
* Enter any device or board name in the **Search** box to quickly find what you are looking for. Remove all filtering using the **X** next to it.
* The **Packs** tab shows supporting Software Packs depending on the device/board selection. Install an appropriate Pack version using either the latest one or using a **Previous** version.
* The **Examples** tab shows ready-to-run projects that can be used for setting up your own application.
* Use the **Check for Updates** button and observe update messages in the **Output** window.

Используя вкладку «**Устройства**», вы можете отфильтровать микроконтроллеры, поддерживаемые MDK.

 Вкладка «**Доски**» предлагает те же функции для поиска плат разработки.

 Введите любое имя устройства или платы в **поле поиска**, чтобы быстро найти то, что вы ищете. Удалите всю фильтрацию, нажав **X** рядом с ней.

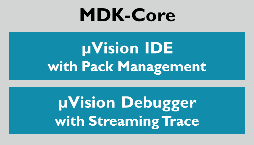
 На вкладке **Packs** отображаются поддерживаемые пакеты программного обеспечения в зависимости от выбора устройства/платы. Установите соответствующую версию пакета, используя последнюю или предыдущую версию.

 На вкладке «**Примеры**» показаны готовые к запуску проекты, которые можно использовать для настройки собственного приложения.

 Используйте кнопку «**Проверить наличие обновлений**» и наблюдайте за сообщениями об обновлениях в окне «**Вывод**».

[**https://www2.keil.com/mdk5/core/**](https://www2.keil.com/mdk5/core/)

**MDK Core & Software Packs**

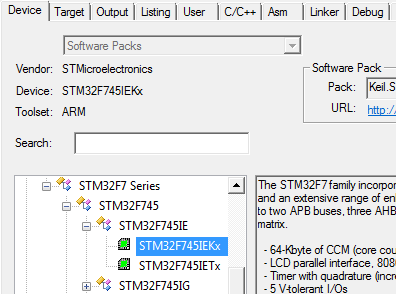


**MDK-Core** is based on [µVision](https://www2.keil.com/uvision) with leading support for Cortex-M devices including the new ARMv8-M architecture. [Pack Installer](https://www2.keil.com/mdk5/packinstaller) is used to download, install, and manage **Software Packs**. The [µVision Debugger](https://www2.keil.com/debug) enables you to test, verify, and optimize your application code. It fully supports **streaming trace** for debugging of historical sequences, execution profiling, performance optimization, and code coverage analysis.

Software Packs make new device support and middleware updates independent from the toolchain.

* A list of all public [Software Packs](https://www.keil.com/dd2/pack) is available. These Packs provide additional middleware or device support.
* The [Device Database](https://www.keil.com/dd2) shows all microcontroller devices that are supported by Software Packs.
* [Third-Party Software Packs](https://www2.keil.com/mdk5/partnerpacks/) provide additional middleware from ARM partners.

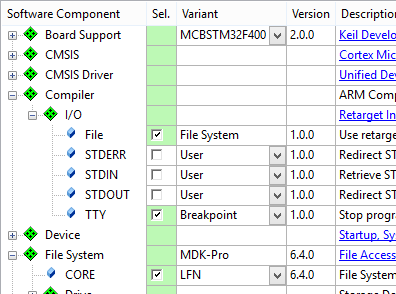
**Benefits of Software Packs**



**Access Device Information**

The installation of a **Device Family Pack (DFP)** adds the support for a complete microcontroller series. Selecting a device in the [µVision Project Manager](https://www2.keil.com/mdk5/uvision) shows only relevant tool options and pre-configures the parameters for assembler, compiler, linker, and debugger with flash programming algorithms.

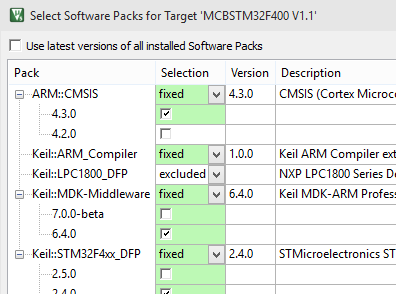
This selection gives you device-specific access to documentation, system/startup code, source code templates, and software components for peripheral access from within the IDE.



**Use Many Proven Software Components**

Software Packs install software components with building blocks that help you to efficiently utilize the features of modern microcontrollers. The **CMSIS Software Pack** gives you core peripheral access, DSP functions, and the CMSIS-RTOS RTX real-time operating system. A **DFP** adds frequently used software components for device peripheral access, and MDK-Professional's **Middleware Software Pack** includes complete software stacks for communication, file storage, and graphic. Additional software components are available from many [third-party companies](https://www2.keil.com/mdk5/partnerpacks/).

Select the software components for your application in the Run-Time Environment (RTE). You can choose between variants (for example debug or release) and quickly access related documentation.

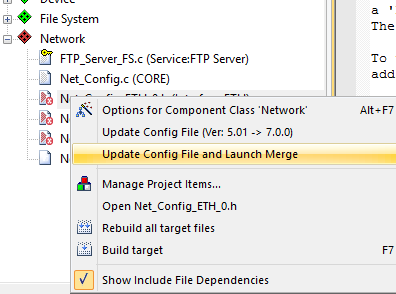


**Choose Software Pack Versions**

MDK allows you to select specific versions for the Software Packs that you are using in your embedded application. Typically, you want to use latest versions of all installed Software Packs during development – this is the default for new projects.

Once your project is in the validation or release phase, you can fix the versions of the Software Packs that are using. MDK will then no longer update Software Packs and the related software components for this project.

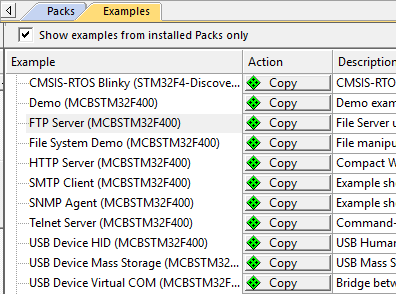
MDK also creates a build log that lists all the Software Packs along with the software components used in the application code. Saving the Software Packs along with the MDK Core installer enables you to recreate the application code even after years.



**Update Software Components**

Software components simplify the update to new versions since fixed files (marked with a key symbol in the project window) and configuration files (that contain application specific setup information) are clearly separated from the application code. For both file types, the RTE tracks version information.

When you use a new Software Pack, the related software components are also updated. Fixed files of the component are automatically included. For updated configuration files, MDK shows three different compatibility levels: fully compatible, compatible extensions, and incompatible extensions. The project window's menu command **Update Config File and Launch Merge** allows you incorporate a new version of a configuration file.



**Get Started with Example Projects**

Software Packs also include example projects that help you to learn how to use devices, peripherals, and software components. You may use parts of the example in your embedded application or the complete example as a starting point for a new project.

[Pack Installer](https://www.keil.com/mdk5/packinstaller) helps you to search the extensive list of available example projects in all public Software Packs. You may list examples that are available for a device, a device series, or an evaluation board. To explore an example, copy the project to your workspace. Then build and execute the example code.

C:\Users\mitil\AppData\Local\Arm\Packs

E:\Radiotecnica\Radiootehnica\Program\Keil\Keil\Programirovca\Novo MDK

[www.keil.com/pack](http://www.keil.com/pack)

<https://arduino.ru/Hardware/ArduinoBoardUno>

Arduino Uno

|  |  |
| --- | --- |
| [Arduino Uno](https://arduino.ru/sites/default/files/Hardware/updated/uno_front.jpg)[Ардуино Уно](https://arduino.ru/sites/default/files/Hardware/updated/uno_back.jpg) |  |

**Общие сведения**

**Arduino Uno** контроллер построен на **ATmega328** ([техническое описание](http://out.arduino.ru/?redirect=http%3A%2F%2Fwww.atmel.com%2Fdyn%2Fresources%2Fprod_documents%2Fdoc8161.pdf&baseU=https%3A%2F%2Farduino.ru%2FHardware%2FArduinoBoardUno), pdf). Платформа имеет 14 цифровых вход/выходов (6 из которых могут использоваться как выходы ШИМ), 6 аналоговых входов, кварцевый генератор 16 МГц, разъем USB, силовой разъем, разъем ICSP и кнопку перезагрузки. Для работы необходимо подключить платформу к компьютеру посредством кабеля USB, либо подать питание при помощи адаптера AC/DC или батареи.

В отличие от всех предыдущих плат, использовавших FTDI USB микроконтроллер для связи по USB, новый **Ардуино Uno** использует микроконтроллер **ATmega8U2** ([техническое описание, pdf](http://out.arduino.ru/?redirect=http%3A%2F%2Fwww.atmel.com%2Fdyn%2Fresources%2Fprod_documents%2Fdoc7799.pdf&baseU=https%3A%2F%2Farduino.ru%2FHardware%2FArduinoBoardUno)).

"Uno" переводится как один с итальянского и разработчики тем самым намекают на грядущий выход Arduino 1.0. Новая плата стала флагманом линейки плат Ардуино. Для сравнения с предыдущими версиями можно обратиться к [полному списку плат Arduino](https://arduino.ru/Hardware).

**Характеристики**

|  |  |
| --- | --- |
| Микроконтроллер | ATmega328 |
| Рабочее напряжение | 5 В |
| Входное напряжение (рекомендуемое) | 7-12 В |
| Входное напряжение (предельное) | 6-20 В |
| Цифровые Входы/Выходы | 14 (6 из которых могут использоваться как выходы [ШИМ](https://arduino.ru/Tutorial/PWM)) |
| Аналоговые входы | 6 |
| Постоянный ток через вход/выход | 40 мА |
| Постоянный ток для вывода 3.3 В | 50 мА |
| Флеш-память | 32 Кб (ATmega328) из которых 0.5 Кб используются для загрузчика |
| ОЗУ | 2 Кб (ATmega328) |
| EEPROM | 1 Кб (ATmega328) |
| Тактовая частота | 16 МГц |

**Схема и исходные данные**

Файлы EAGLE: [arduino-duemilanove-reference-design.zip](https://arduino.ru/Schematic/arduino-uno-reference-design.zip)

Принципиальная схема: [arduino-duemilanove-schematic.pdf](https://arduino.ru/Schematic/arduino-uno-schematic.pdf)

**Питание**

**Arduino Uno** может получать питание через подключение USB или от внешнего источника питания. Источник питания выбирается автоматически.

Внешнее питание (не USB) может подаваться через преобразователь напряжения AC/DC (блок питания) или аккумуляторной батареей. Преобразователь напряжения подключается посредством разъема 2.1 мм с центральным положительным полюсом. Провода от батареи подключаются к выводам Gnd и Vin разъема питания.

Платформа может работать при внешнем питании от 6 В до 20 В. При напряжении питания ниже 7 В, вывод 5V может выдавать менее 5 В, при этом платформа может работать нестабильно. При использовании напряжения выше 12 В регулятор напряжения может перегреться и повредить плату. Рекомендуемый диапазон от 7 В до 12 В.

Выводы питания:

* **VIN**. Вход используется для подачи питания от внешнего источника (в отсутствие 5 В от разъема USB или другого регулируемого источника питания). Подача напряжения питания происходит через данный вывод.
* **5V**. Регулируемый источник напряжения, используемый для питания микроконтроллера и компонентов на плате. Питание может подаваться от вывода VIN через регулятор напряжения, или от разъема USB, или другого регулируемого источника напряжения 5 В.
* **3V3**. Напряжение на выводе 3.3 В генерируемое встроенным регулятором на плате. Максимальное потребление тока 50 мА.
* **GND**. Выводы заземления.

**Память**

Микроконтроллер ATmega328 располагает 32 кБ флэш памяти, из которых 0.5 кБ используется для хранения загрузчика, а также 2 кБ ОЗУ (SRAM) и 1 Кб EEPROM.(которая читается и записывается с помощью [библиотеки EEPROM](https://arduino.ru/Reference/Library/EERPOM)).

**Входы и Выходы**

Каждый из 14 цифровых выводов Uno может настроен как вход или выход, используя функции[pinMode()](https://arduino.ru/Reference/PinMode), [digitalWrite()](https://arduino.ru/Reference/DigitalWrite), и [digitalRead()](https://arduino.ru/Reference/DigitalRead), . Выводы работают при напряжении 5 В. Каждый вывод имеет нагрузочный резистор (по умолчанию отключен) 20-50 кОм и может пропускать до 40 мА. Некоторые выводы имеют особые функции:

Функция pinMode

pinMode()

**Описание**

Устанавливает режим работы заданного вход/выхода(pin) как входа или как выхода. Подробнее про [цифровые вход/выходы(pins)](https://arduino.ru/Tutorial/DigitalPins).

**Синтаксис**

pinMode(pin, mode)

**Параметры**

* pin: номер вход/выхода(pin), который Вы хотите установить
* mode: режим одно из двух значение - INPUT или OUTPUT, устанавливает на вход или выход соответственно.

**Возвращаемое значение**

нет

**Пример**

|  |  |
| --- | --- |
| int ledPin = 13;                 // Светодиод, подключенный к вход/выходу 13 | |
| void setup() |

|  |
| --- |
| { |
| pinMode(ledPin, OUTPUT);      // устанавливает режим работы - выход | |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| void loop() | |
| { |

|  |  |
| --- | --- |
| digitalWrite(ledPin, HIGH);   // включает светодиод | |
| delay(1000);                  // ждет секунду |

|  |  |
| --- | --- |
| digitalWrite(ledPin, LOW);    // выключает светодиод | |
| delay(1000);                  // ждет секунду |

|  |
| --- |
| } |

**Примечание**

Аналоговые входы (analog pins) могут быть использованы как цифровые вход/выходы (digital pins). Обращение к ним идет по номерам от 14 (для аналогового входа 0) до 19 (для аналогового входа 5).

**Смотрите также**

* [constants](https://arduino.ru/Reference/IntegerConstants)
* [digitalWrite](https://arduino.ru/Reference/DigitalWrite)()
* [digitalRead](https://arduino.ru/Reference/DigitalRead)()
* Описание цифровых вход/выходов

[Справочник языка](https://arduino.ru/Reference)

* **Последовательная шина: 0 (RX) и 1 (TX)**. Выводы используются для получения (RX) и передачи (TX) данных TTL. Данные выводы подключены к соответствующим выводам микросхемы последовательной шины ATmega8U2 USB-to-TTL.
* **Внешнее прерывание: 2 и 3**. Данные выводы могут быть сконфигурированы на вызов прерывания либо на младшем значении, либо на переднем или заднем фронте, или при изменении значения. Подробная информация находится в описании функции [attachInterrupt()](https://arduino.ru/Reference/AttachInterrupt).
* **ШИМ: 3, 5, 6, 9, 10, и 11.** Любой из выводов обеспечивает [ШИМ](https://arduino.ru/Tutorial/PWM) с разрешением 8 бит при помощи функции [analogWrite()](https://arduino.ru/Reference/AnalogWrite).
* **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).**Посредством данных выводов осуществляется связь SPI, для чего используется [библиотека SPI](https://arduino.ru/Reference/Library/SPI).

Библиотека SPI

Сб, 02/04/2011 - 01:57 | by **Alexander**

Библиотека SPI позволяет контроллеру Arduino взаимодействовать с устройствами поддерживающими SPI протокол. Arduino в данном случае выступает в качестве ведущего устройтва.

**Коротко о Serial Peripheral Interface (SPI).**

Последоваетельный периферийный интерфейс (SPI) — это последовательный синхроный протокол передачи данных используемый микроконтроллерами для обмена данными с одним или несколькими периферийными устройствами на небольших растояниях.

Для организации соединения SPI необходимо одно ведущее устройство, обычно это микроконтроллер, которое управляет соединением с ведомыми устройствами. Обычно подключение осуществляется тремя общими линиями и линией выбора переферийного(ведомого) устройства:

* Master In Slave Out (MISO), переводится как  "вход ведущего выход ведомого", используется для передачи данных от ведомого к ведущему.
* Master Out Slave In (MOSI) — выход ведущего вход ведомого, для передачи данных от ведущего к периферийным устройствам.
* Serial Clock (SCK) — синхронизирующая линия, синхросигнал генерируется ведущим устройством.
* Slave Select pin — вход на ведомых устройствах с помощью которого ведущий может инициировать обмен данными с периферийным устройством. Если на этом входе [LOW,](https://arduino.ru/Reference/Constants) то ведомый взаимодействует с ведущим, если [HIGH](https://arduino.ru/Reference/Constants), то ведомый игнорирует сигналы от ведущего.

 При работе с SPI устройствами надо учитывать следующие моменты:

* Какой порядок вывода данных используется:  Most Significant Bit (MSB - старший бит (разряд)) or Least Significant Bit (LSB - младший бит) первый. Порядок может быть изменен функцией [SPI.setBitOrder](https://arduino.ru/Reference/Library/SPI/setBitOrder)().
* Уровень сигнала синхронизации — по какому синхронизирующему сигналу (HIGH или LOW) передаются данные.
* Фаза синхронизации — влияет на последовательность установки и выборки данных. Фаза синхронизации SPI и уровень сигнала задается функцией [SPI.setDataMode](https://arduino.ru/Reference/Library/SPI/setDataMode)().
* Скорость на которой работает SPI устанавливается функцией [SPI.setClockDivider](https://arduino.ru/Reference/Library/SPI/setClockDivider)().

Производители SPI устройств несколько по разному реализуют протокол, поэтому необходимо внимательно ознакомиться с техническим описанием к устройству. Комбинация фазы синхронизиции (CPHA) и уровня сигнала синхронизации (CPOL) задают режим логики работы интерфейса SPI. Режим устанавливается функцией [SPI.setDataMode](https://arduino.ru/Reference/Library/SPI/setDataMode)().

|  |  |  |
| --- | --- | --- |
| **Режим** | **Уровень сигнала (CPOL)** | **Фаза (CPHA)** |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |

**Подключение**

На контроллерах Arduino Duemilanove и других на базе ATmega168 /328, шина SPI использует выходы 10 (SS), 11 (MOSI), 12 (MISO), и 13 (SCK). На Arduino Mega — 50 (MISO), 51 (MOSI), 52 (SCK), и 53 (SS). Обратите внимание, что даже если вы не используете выход SS, он должен быть установлен как выход, в противном случае интерфейс может оказаться в режиме ведомого и библиотека не будет работать как надо.

В качестве SS выхода может быть использован выход отличный от 10-го. Например, при работе с Arduino Ethernet shield контроллер использует выход 4 для взаимодействия с SD картой по SPI и выход 10 для работы с Ethernet контроллером.

**Функции**

* [begin](https://arduino.ru/Reference/Library/SPI/begin)()
* [end](https://arduino.ru/Reference/Library/SPI/end)()
* [setBitOrder](https://arduino.ru/Reference/Library/SPI/setBitOrder)()
* [setClockDivider](https://arduino.ru/Reference/Library/SPI/setClockDivider)()
* [setDataMode](https://arduino.ru/Reference/Library/SPI/setDataMode)()
* [transfer](https://arduino.ru/Reference/Library/SPI/transfer)()
* **LED: 13.**Встроенный светодиод, подключенный к цифровому выводу 13. Если значение на выводе имеет высокий потенциал, то светодиод горит.

На платформе Uno установлены 6 аналоговых входов (обозначенных как A0 .. A5), каждый разрешением 10 бит (т.е. может принимать 1024 различных значения). Стандартно выводы имеют диапазон измерения до 5 В относительно земли, тем не менее имеется возможность изменить верхний предел посредством вывода AREF и функции [analogReference()](https://arduino.ru/Reference/AnalogReference). Некоторые выводы имеют дополнительные функции:

* **I2C: 4 (SDA) и 5 (SCL).** Посредством выводов осуществляется связь I2C (TWI), для создания которой используется [библиотека Wire.](http://out.arduino.ru/?redirect=http%3A%2F%2Farduino.cc%2Fen%2FReference%2FWire&baseU=https%3A%2F%2Farduino.ru%2FHardware%2FArduinoBoardUno)
* Wire
* [Communication]
* Description
* This library allows you to communicate with I2C devices, a feature that is present on all Arduino boards. I2C is a very common protocol, primarly used for reading/sending data to/from external I2C components. To learn more, visit [this article for Arduino & I2C](https://docs.arduino.cc/learn/communication/wire).
* Due to the hardware design and various architectural differences, the I2C pins are located in different places. The pin map just below highlights the default pins, as well as additional ports available on certain boards.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Board | I2C Default | I2C1 | I2C2 | Notes |
| UNO R3, UNO R3 SMD, UNO Mini Ltd | A4(SDA), A5(SCL) |  |  | I2C also available on the SDA / SCL pins (digital header). |
| UNO R4 Minima, UNO R4 WiFi | A4(SDA), A5(SCL) | Qwiic: D27(SDA), D26(SCL) |  | I2C also available on the SDA / SCL pins (digital header). |
| UNO WiFi Rev2, Zero | 20(SDA), 21(SCL) |  |  |  |
| Leonardo, Micro, Yùn Rev2 | D2(SDA), D3(SCL) |  |  |  |
| Nano boards | A4(SDA), A5(SCL) |  |  |  |
| MKR boards | D11(SDA), D12(SCL) |  |  |  |
| GIGA R1 WiFi | 20(SDA), 21(SCL) | D102(SDA1), D101 (SCL1) | D9(SDA2), D8 (SCL2) | Use Wire1.begin() for I2C1, and Wire2.begin() for I2C2. |
| Due | 20(SDA), 21(SCL) | D70(SDA1), D71(SCL1) |  | Use Wire1.begin() for I2C1 |
| Mega 2560 Rev3 | D20(SDA), D21(SCL) |  |  |  |

* This library inherits from the Stream functions, making it consistent with other read/write libraries. Because of this, send() and receive() have been replaced with read() and write().
* Recent versions of the Wire library can use timeouts to prevent a lockup in the face of certain problems on the bus, but this is not enabled by default (yet) in current versions. It is recommended to always enable these timeouts when using the Wire library. See the Wire.setWireTimeout function for more details.
* **Note:** There are both 7 and 8-bit versions of I2C addresses. 7 bits identify the device, and the eighth bit determines if it’s being written to or read from. The Wire library uses 7 bit addresses throughout. If you have a datasheet or sample code that uses 8-bit address, you’ll want to drop the low bit (i.e. shift the value one bit to the right), yielding an address between 0 and 127. However the addresses from 0 to 7 are not used because are reserved so the first address that can be used is 8. Please note that a pull-up resistor is needed when connecting SDA/SCL pins. Please refer to the examples for more information. MEGA 2560 board has pull-up resistors on pins 20 and 21 onboard.
* **The Wire library implementation uses a 32 byte buffer, therefore any communication should be within this limit. Exceeding bytes in a single transmission will just be dropped.**
* To use this library:
* #include <Wire.h>
* Functions
* [begin()](https://www.arduino.cc/reference/en/language/functions/communication/wire/begin)  
  [end()](https://www.arduino.cc/reference/en/language/functions/communication/wire/end)  
  [requestFrom()](https://www.arduino.cc/reference/en/language/functions/communication/wire/requestfrom)  
  [beginTransmission()](https://www.arduino.cc/reference/en/language/functions/communication/wire/begintransmission)  
  [endTransmission()](https://www.arduino.cc/reference/en/language/functions/communication/wire/endtransmission)  
  [write()](https://www.arduino.cc/reference/en/language/functions/communication/wire/write)  
  [available()](https://www.arduino.cc/reference/en/language/functions/communication/wire/available)  
  [read()](https://www.arduino.cc/reference/en/language/functions/communication/wire/read)  
  [setClock()](https://www.arduino.cc/reference/en/language/functions/communication/wire/setclock)  
  [onReceive()](https://www.arduino.cc/reference/en/language/functions/communication/wire/onreceive)  
  [onRequest()](https://www.arduino.cc/reference/en/language/functions/communication/wire/onrequest)  
  [setWireTimeout()](https://www.arduino.cc/reference/en/language/functions/communication/wire/setwiretimeout)  
  [clearWireTimeoutFlag()](https://www.arduino.cc/reference/en/language/functions/communication/wire/clearwiretimeoutflag)  
  [getWireTimeoutFlag()](https://www.arduino.cc/reference/en/language/functions/communication/wire/getwiretimeoutflag)

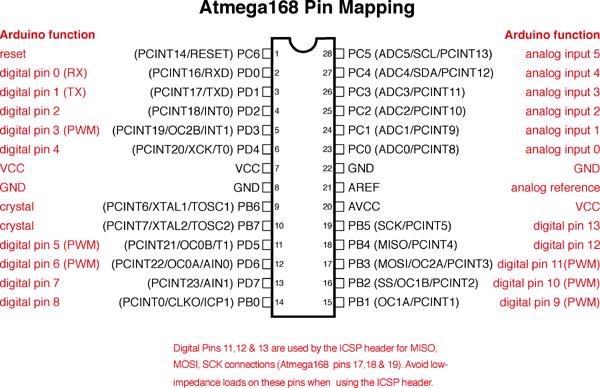
Дополнительная пара выводов платформы:

* **AREF.** Опорное напряжение для аналоговых входов. Используется с функцией [analogReference()](https://arduino.ru/Reference/AnalogReference).
* **Reset.**Низкий уровень сигнала на выводе перезагружает микроконтроллер. Обычно применяется для подключения кнопки перезагрузки на плате расширения, закрывающей доступ к кнопке на самой плате Arduino.

Обратите внимание на соединение между выводами [Arduino и портами ATmega328](https://arduino.ru/Hardware/ATMega168pins).

ATmega168/328-Arduino распиновка

*Обратите внимание, что приведенная ниже распиновка соответствует только DIP чипам. Ардуино Mini базируется на более компактном чипе (IC package) и имеет два дополнительных ADC пина*.



Цифровые входы/выходы 11,12 и 13 используются ICSP коннектором для MISO, MOSI, SCK соединений (контакты 17, 18 и 19 на ATMega 168). Избегайте низкоомной нагрузки на этих к входах при использовании ICSP коннектора

**Связь**

На платформе **Arduino Uno** установлено несколько устройств для осуществления связи с компьютером, другими устройствами Arduino или микроконтроллерами. ATmega328 поддерживают последовательный интерфейс UART TTL (5 В), осуществляемый выводами 0 (RX) и 1 (TX). Установленная на плате микросхема ATmega8U2 направляет данный интерфейс через USB, программы на стороне компьютера "общаются" с платой через виртуальный COM порт. Прошивка ATmega8U2 использует стандартные драйвера USB COM, никаких стороних драйверов не требуется, но на Windows для подключения потребуется файл ArduinoUNO.inf.  Мониторинг последовательной шины (Serial Monitor) программы Arduino позволяет посылать и получать текстовые данные при подключении к платформе. Светодиоды RX и TX на платформе будут мигать при передаче данных через микросхему FTDI или USB подключение (но не при использовании последовательной передачи через выводы 0 и 1).

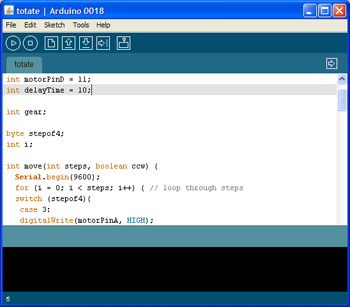
Библиотекой SoftwareSerial возможно создать последовательную передачу данных через любой из цифровых выводов Uno.

ATmega328 поддерживает интерфейсы I2C (TWI) и SPI. В Arduino включена библиотека Wire для удобства использования шины I2C.

**Программирование**

Платформа программируется посредством ПО Arduino. Из меню**Tools > Board** выбирается «Arduino Uno» (согласно установленному микроконтроллеру). Подробная информация находится в [справочнике](https://arduino.ru/Reference) и [инструкциях](https://arduino.ru/Arduino_environment).

Среда разработки Arduino

Среда разработки Arduino состоит из встроенного текстового редактора программного кода, области сообщений, окна вывода текста(консоли), панели инструментов с кнопками часто используемых команд и нескольких меню. Для загрузки программ и связи среда разработки подключается к аппаратной части Arduino.

**Скачать**

Последнюю версию можно найти по этой ссылке  
[http://www.arduino.cc/en/Main/Software](http://out.arduino.ru/?redirect=https%3A%2F%2Fwww.arduino.cc%2Fen%2FMain%2FSoftware&baseU=https%3A%2F%2Farduino.ru%2FArduino_environment)

Полезные ссылки:

* [Начало работы в Windows](https://arduino.ru/Guide/Windows)
* [Установка Arduino IDE на Ubuntu Linux](http://out.arduino.ru/?redirect=http%3A%2F%2Fubuntovod.ru%2Fsoft%2Farduino-ubuntu.html&baseU=https%3A%2F%2Farduino.ru%2FArduino_environment)

Программа, написанная в среде Arduino, называется скетч. Скетч пишется в текстовом редакторе, имеющем инструменты вырезки/вставки, поиска/замены текста. Во время сохранения и экспорта проекта в области сообщений появляются пояснения, также могут отображаться возникшие ошибки. Окно вывода текста(консоль) показывает сообщения Arduino, включающие полные отчеты об ошибках и другую информацию. Кнопки панели инструментов позволяют проверить и записать программу, создать, открыть и сохранить скетч, открыть мониторинг последовательной шины:

компиляция кода*Verify/Compile*  
Проверка программного кода на ошибки, компиляция.

остановка мониторинга*Stop*  
Остановка мониторинга последовательной шины(Serial monitor) или затемнение других кнопок.

[создание нового скетча](https://arduino.ru/sites/default/files/aruino_environment/new.gif)*New*  
Создание нового скетча.

[открыть файл скетча](https://arduino.ru/sites/default/files/aruino_environment/open.gif)*Open*  
Открытие меню доступа ко всем скетчам в блокноте. Открывается нажатием в текущем окне.

*Примечание*: из-за наличия ошибки в Java данное меню не может прокручиваться; при необходимости открыть скетч из этого списка проследуйте в меню File | Sketchbook.

[сохранение скетча](https://arduino.ru/sites/default/files/aruino_environment/save.gif)*Save*  
Сохранение скетча.

[загрузка скетча в контроллер](https://arduino.ru/sites/default/files/aruino_environment/export.gif)*Upload to I/O Board*  
Компилирует программный код и загружает его в устройство Arduino. Описание загрузки приведено ниже.

[Serial monitor](https://arduino.ru/sites/default/files/aruino_environment/serial_monitor.gif)*Serial Monitor*  
Открытие мониторинга последовательной шины (Serial monitor).

Дополнительные команды сгруппированы в пять меню: File, Edit, Sketch, Tools, Help. Доступность меню определяется работой, выполняемой в данный момент.

**Edit**

* *Copy for Discourse*  
  Копирует в буфер обмена подходящий для размещения на форуме код скетча с выделением синтаксиса.
* *Copy as HTML*  
  Копирует код скетча в буфер обмена как HTML код, для размещения на веб-страницах.

**Sketch**

* *Verify/Compile*  
  Проверка скетча на ошибки.
* *Import Library*  
  Добавляет библиотеку в текущий скетч, вставляя директиву #include в код скетча. Подробная информация в описании библиотек ниже (Libraries).
* *Show Sketch Folder*  
  Открывает папку, содержащую файл скетча, на рабочем столе.
* *Add File...*  
  Добавляет файл в скетч (файл будет скопирован из текущего места расположения). Новый файл появляется в новой закладке в окне скетча. Файл может быть удален из скетча при помощи меню закладок.

**Tools**

* *Auto Format*  
  Данная опция оптимизирует код, например, выстраивает в одну линию по вертикали открывающую и закрывающую скобки и помещает между ними утверждение.
* *Board*  
  Выбор используемой платформы. Список с описанием платформ приводится ниже.
* Serial Port  
  Меню содержит список последовательных устройств передачи данных (реальных и виртуальных) на компьютере. Список обновляется автоматически каждый раз при открытии меню Tools.
* *Burn Bootloader*  
  Пункты данного меню позволяют записать Загрузчик (Bootloader) в микроконтроллер на платформе Arduino. Данное действие не требуется в текущей работе с Arduino, но пригодится, если имеется новый ATmega (без загрузчика). Перед записью рекомендуется проверить правильность выбора платформы из меню. При использовании AVR ISP необходимо выбрать соответствующий программатору порт из меню Serial Port.

**Блокнот (Sketchbook)**

Средой Arduino используется принцип блокнота: стандартное место для хранения программ (скетчей). Скетчи из блокнота открываются через меню File > Sketchbook или кнопкой Open на панели инструментов. При первом запуске программы Arduino автоматически создается директория для блокнота. Расположение блокнота меняется через диалоговое окно Preferences.

**Закладки, Файлы и Компиляция**

Позволяют работать с несколькими файлами скетчей (каждый открывается в отдельной закладке). Файлы кода могут быть стандартными Arduino (без расширения), файлами С (расширение \*.с), файлами С++ (\*.срр) или головными файлами (.h).

**Загрузка скетча в Arduino**

Перед загрузкой скетча требуется задать необходимые параметры в меню **Tools > Board**и **Tools > Serial Port**. Платформы описываются далее по тексту. В ОС Mac последовательный порт может обозначаться как dev/tty.usbserial-1B1 (для платы USB) или /dev/tty.USA19QW1b1P1.1 (для платы последовательной шины, подключенной через адаптер Keyspan USB-to-Serial). В ОС Windows порты могут обозначаться как COM1 или COM2 (для платы последовательной шины) или COM4, COM5, COM7 и выше (для платы USB). Определение порта USB производится в поле Последовательной шины USB Диспетчера устройств Windows. В ОС Linux порты могут обозначаться как /dev/ttyUSB0, /dev/ttyUSB1.

После выбора порта и платформы необходимо нажать кнопку загрузки на панели инструментов или выбрать пункт меню File > Upload to I/O Board. Современные платформы Arduino перезагружаются автоматически перед загрузкой. На старых платформах необходимо нажать кнопку перезагрузки. На большинстве плат во время процесса будут мигать светодиоды RX и TX. Среда разработки Arduino выведет сообщение об окончании загрузки или об ошибках.

При загрузке скетча используется Загрузчик (Bootloader) Arduino, небольшая программа, загружаемая в микроконтроллер на плате. Она позволяет загружать программный код без использования дополнительных аппаратных средств. Загрузчик (Bootloader) активен в течении нескольких секунд при перезагрузке платформы и при загрузке любого из скетчей в микроконтроллер. Работа Загрузчика (Bootloader) распознается по миганию светодиода (13 пин) (напр.: при перезагрузке платы).

**Библиотеки**

Библиотеки добавляют дополнительную функциональность скетчам, например, при работе с аппаратной частью или при обработке данных. Для использования библиотеки необходимо выбрать меню **Sketch > Import Library**. Одна или несколько директив **#include** будут размещены в начале кода скетча с последующей компиляцией библиотек и вместе со скетчем. Загрузка библиотек требует дополнительного места в памяти Arduino. Неиспользуемые библиотеки можно удалить из скетча убрав директиву **#include**.

На Arduino.cc имеется [список библиотек](http://out.arduino.ru/?redirect=http%3A%2F%2Farduino.cc%2Fen%2FReference%2FLibraries&baseU=https%3A%2F%2Farduino.ru%2FArduino_environment). Некоторые библиотеки включены в среду разработки Arduino. Другие могут быть загружены с различных ресурсов. Для установки скачанных библиотек необходимо создать директорию «libraries» в папке блокнота и затем распаковать архив. Например, для установки библиотеки DateTime ее файлы должны находится в подпапке **/libraries/DateTime** папки блокнота.

Смотрите [данную инструкцию](http://out.arduino.ru/?redirect=http%3A%2F%2Farduino.cc%2Fen%2FHacking%2FLibraryTutorial&baseU=https%3A%2F%2Farduino.ru%2FArduino_environment) для написания собственной библиотеки.

**Аппаратные средства других разработчиков**

Поддерживаемые аппаратные средства других производителей добавляются в соответствующую подпапку папки блокнота. Устанавливаемые платформы могут включать собственные характеристики (в меню платформы), корневые библиотеки, загрузчик(Bootloader) и характеристики программатора. Для установки требуется распаковать архив в созданную папку.  (Запрещено использовать наименование папки "arduino", т.к. могут быть перезаписаны встроенные данные платформы Arduino.) Для деинсталляции данных удаляется соответствующая директория.

Подробная информация по созданию сборок описаний аппаратных средств других производителей находится на [страницах сайта Google Code](http://out.arduino.ru/?redirect=https%3A%2F%2Fcode.google.com%2Fp%2Farduino%2Fwiki%2FPlatforms&baseU=https%3A%2F%2Farduino.ru%2FArduino_environment).

**Мониторинг последовательной шины (Serial Monitor)**

Отображает данные посылаемые в платформу Arduino (плата USB или плата последовательной шины). Для отправки данных необходимо ввести текст и нажать кнопку Send или Enter. Затем выбирается скорость передачи из выпадающего списка, соответствующая значению **Serial.begin** в скетче. На ОС Mac или Linux платформа Arduino будет перезагружена (скетч начнется сначала) при подключении мониторинга последовательной шины.

Имеется возможность обмена информацией с платформой через программы Processing, Flash, MaxMSP и т.д. (см. подробности на странице описаний интерфейсов).

**Настройки**

Некоторые настройки изменяются в окне **Preferences** (меню Arduino в ОС Mac или File в ОС Windows и Linux). Остальные настройки находятся в файле, месторасположение которого указано в окне Preferences.

**Платформы**

Выбор платформы влияет на: параметры (напр.: скорость ЦП и скорость передачи данных), используемые при компиляции и загрузке скетчей и на настройки записи загрузчика (Bootloader) микроконтреллера. Некоторые характеристики платформ различаются только по последнему параметру (загрузка Bootloader), таким образом, даже при удачной загрузке с соответствующим выбором может потребоваться проверка различия перед записью загрузчика (Bootloader).

* [*Arduino Duemilanove*](https://arduino.ru/Hardware/ArduinoBoardDuemilanove)*или*[*Nano с ATmega328*](https://arduino.ru/Hardware/ArduinoBoardNano)  
  Тактовая частота ATmega328 16 МГц с возможностью автоматической перезагрузки. Используется для версий Arduino Pro или Pro Mini с ATmega328 на частоте 16 МГц (5 В).
* [*Arduino Diecimila*](https://arduino.ru/Hardware/ArduinoBoardDiecimila)*,*[*Duemilanove*](https://arduino.ru/Hardware/ArduinoBoardDuemilanove)*, или*[*Nano с ATmega168*](https://arduino.ru/Hardware/ArduinoBoardNano)  
  Тактовая частота ATmega168 16 МГц с возможностью автоматической перезагрузки. Компиляция и загрузка соответствует Arduino NG или старым версиям с ATmega168, но загрузка Bootloader имеет короткий таймаут (при перезагрузке светодиод пина 13 мигает один раз). Используется для версий Arduino Pro и Pro Mini с ATmega168 на частоте 16 МГц (5 В).
* [*Arduino Mega*](https://arduino.ru/Hardware/ArduinoBoardMega)  
  Тактовая частота ATmega1280 16 МГц с возможностью автоматической перезагрузки.
* [*Arduino Mini*](https://arduino.ru/Hardware/ArduinoBoardMini)  
  Соответствует Arduino NG или старым версиям с ATmega168 (напр.: тактовая частота ATmega168 16 МГц без возможности автоматической перезагрузки).
* *Arduino BT*  
  Тактовая частота ATmega168 16 МГц. Загрузка Bootloader происходит совместно с кодами для инициализации модуля Bluetooth.
* [*LilyPad Arduino*](https://arduino.ru/Hardware/ArduinoBoardLilyPad) с ATmega328  
  Тактовая частота ATmega328 8 МГц (3.3 В) с возможность автоматической перезагрузки. Соответствует Arduino Pro или Pro Mini (3.3 В, 8 МГц) с ATmega328.
* [*LilyPad Arduino*](https://arduino.ru/Hardware/ArduinoBoardLilyPad)*с ATmega168*  
  Тактовая частота ATmega168 8 МГц.

Компиляция и загрузка соответствует Arduino Pro или Pro Mini (8 МГц) с ATmega168.

Загруженный Bootloader имеет длинный таймаут (при перезагрузке светодиод пина 13 мигает три раза), т.к. оригинальные версии LilyPad не поддерживают автоматическую перезагрузку. Также не поддерживаются внешние часы и, следовательно, Bootloader конфигурирует загрузку внутренних 8 МГц часов в ATmega168.

При наличии поздних версий LilyPad (с 6-контакным программным вводом) перед загрузкой Bootloader требуется выбрать Arduino Pro или Pro Mini (8 MHz) с ATmega168.

* *Arduino Pro или Pro Mini (3.3 В, 8 МГц) с ATmega328*  
  Тактовая частота ATmega328 8 МГц (3.3 В) с возможность автоматической перезагрузки. Соответствует LilyPad Arduino с ATmega328.
* *Arduino Pro или Pro Mini (3.3 В, 8 МГц) с ATmega168*  
  Тактовая частота ATmega168 8 МГц (3.3 В) с возможность автоматической перезагрузки.
* *Arduino NG или предыдущие версии с ATmega168*  
  Тактовая частота ATmega168 16 МГц без возможности автоматической перезагрузки. Компиляция и загрузка соответствует Arduino Diecimila или Duemilanove с ATmega168, но загрузка Bootloader имеет длинный таймаут (при перезагрузке светодиод пина 13 мигает три раза).
* *Arduino NG или предыдущие версии с ATmega8*  
  Тактовая частота ATmega8 16 МГц без возможности автоматической перезагрузки.

Микроконтроллер ATmega328 поставляется с записанным загрузчиком, облегчающим запись новых программ без использования внешних программаторов. Связь осуществляется оригинальным протоколом STK500.

Имеется возможность не использовать загрузчик и запрограммировать микроконтроллер через выводы ICSP (внутрисхемное программирование). Подробная информация находится в [данной инструкции](http://out.arduino.ru/?redirect=http%3A%2F%2Farduino.cc%2Fen%2FHacking%2FProgrammer&baseU=https%3A%2F%2Farduino.ru%2FHardware%2FArduinoBoardUno).

**Автоматическая (программная) перезагрузка**

**Uno** разработана таким образом, чтобы перед записью нового кода перезагрузка осуществлялась самой программой Arduino на компьютере, а не нажатием кнопки на платформе. Одна из линий DTR микросхемы ATmega8U2, управляющих потоком данных (DTR), подключена к выводу перезагрузки микроконтроллеру ATmega328 через 100 нФ конденсатор. Активация данной линии, т.е. подача сигнала низкого уровня, перезагружает микроконтроллер. Программа Arduino, используя данную функцию, загружает код одним нажатием кнопки Upload в самой среде программирования. Подача сигнала низкого уровня по линии DTR скоординирована с началом записи кода, что сокращает таймаут загрузчика.

Функция имеет еще одно применение. Перезагрузка Uno происходит каждый раз при подключении к программе Arduino на компьютере с ОС Mac X или Linux (через USB). Следующие полсекунды после перезагрузки работает загрузчик. Во время программирования происходит задержка нескольких первых байтов кода во избежание получения платформой некорректных данных (всех, кроме кода новой программы). Если производится разовая отладка скетча, записанного в платформу, или ввод каких-либо других данных при первом запуске, необходимо убедиться, что программа на компьютере ожидает в течение секунды перед передачей данных.

На Uno имеется возможность отключить линию автоматической перезагрузки разрывом соответствующей линии.   Контакты микросхем с обоих концов линии могут быть соединены с целью восстановления. Линия маркирована «RESET-EN». Отключить автоматическую перезагрузку также возможно подключив резистор 110 Ом между источником 5 В и данной линией.

**Токовая защита разъема USB**

В **Arduino Uno** встроен самовостанавливающийся предохранитель (автомат), защищающий порт USB компьютера от токов короткого замыкания и сверхтоков. Хотя практически все компьютеры имеют подобную защиту, тем не менее, данный предохранитель обеспечивает дополнительный барьер. Предохранитель срабатыват при прохождении тока более 500 мА через USB порт и размыкает цепь до тех пока нормальные значения токов не будут востановлены.

**Физические характеристики**

Длина и ширина печатной платы Uno составляют 6.9 и 5.3 см соответственно. Разъем USB и силовой разъем выходят за границы данных размеров. Четыре отверстия в плате позволяют закрепить ее на поверхности. Расстояние между цифровыми выводами 7 и 8 равняется 0,4 см, хотя между другими выводами оно составляет 0,25 см.