**25. Void. Константы. Переменные. Объявление переменных. Границы переменных**

**Void**

Ключевое слово void используется при объявлении функций, если функция не воз­вращает никакого значения при ее вызове.

**Константы**

Константы — предопределенные значения. Они используются, чтобы делать про­граммы более легкими для чтения.

**true/false** это булевы константы, определяющие логические уровни. false лег­ко определяется как 0 (ноль), а true, как 1, но может быть и чем-то другим, отлич­ным от нуля. Поэтому -1, 2 и 200 — это все тоже определяется как true.

#define true 0x1

#define false 0x0

**high/low**— уровни сигналов порта high и low:

#define HIGH 0x1

#define LOW 0x0

**input/output** — настройка цифровых портов на ввод (input) и вывод (output) сиг­налов:

#define INPUT 0x0

#define OUTPUT 0x1

Цифровые порты могут использоваться на ввод или вывод сигналов. Изменение порта с ввода на вывод производится при помощи функции pinMode() :

pinMode(13, OUTPUT); // 13 вывод будет выходом

pinMode(12, INPUT); // 12 - входом

В программе можно создавать собственные константы:

#define LEFT 0x95

#define MESS\_LEFT "поворот влево"

**Переменные**

Переменные — это способ именовать и хранить числовые значения для последую­щего использования программой. Переменные — это значения, которые могут по­следовательно меняться, в отличие от констант, чье значение никогда не меняется. Переменные нужно декларировать (объявлять). Следующий код объявляет пере­менную inputVariable, а затем присваивает ей значение, полученное от второго анало­гового порта:

int inputVariable=0;

inputVariable=analogRead(2);

Переменные могут быть названы любыми именами, которые не являются ключе­выми словами языка программирования Arduino.

**Объявление переменных**

Все переменные должны быть задекларированы до того, как они могут использо­ваться. Объявление переменной означает определение типа ее значения: int, long, float и т. д., задание уникального имени переменной, и дополнительно ей можно присвоить начальное значение. Все это следует делать только один раз в програм­ме, но значение может меняться в любое время при использовании арифметических или других разных операций.

Следующий пример показывает, что объявленная переменная inputVariable имеет тип int, и ее начальное значение равно нулю. Это называется простым присваива­нием.

int inputVariable = 0;

Переменная может быть объявлена в разных местах программы, и то, где это сде­лано, определяет, какие части программы могут использовать переенную.

**Границы переменных**

Переменные могут быть объявлены в начале программы перед void setup(), ло­кально внутри функций и иногда в блоке выражений, таком как цикл for. То, где объявлена переменная, определяет ее границы (область видимости), т. е. возмож­ность некоторых частей программы ее использовать.

**Глобальные переменные** таковы, что их могут видеть и использовать любые функ­ции и выражения программы. Такие переменные декларируются в начале програм­мы перед функцией setup().

**Локальные переменные** определяются внутри функций или таких частей, как цикл for. Они видимы и могут использоваться только внутри функции, в которой объяв­лены. Таким образом, могут существовать несколько переменных с одинаковыми именами в разных частях одной программы, которые содержат разные значения. Уверенность, что только одна функция имеет доступ к ее переменной, упрощает программу и уменьшает потенциальную опасность возникновения ошибок.

**26. Цифровой ввод/вывод. Функция pinMode. Функция digitalWrite(), Void setup(), Void loop(), Функция digitalRead()**

1. **Цифровые ввод/вывод:**

* пины с 0-13
* логический уровень единицы 5V, 0-0V
* максимальный ток вывода 40мА
* к контактам подключены подтягивающие резисторы, которые по умолчанию выключены, но могут быть включены программно.
* на плате выведены 14 цифровых пинов, любой из которых может работать, как на вывод информации, так и на ввод. Для этого в ходе программы применяются специальные функции: (рассмотрим ниже)

**Цифровой ввод/вывод**

Рассмотрим функции цифрового ввода/вывода:

- pinMode();

- digitalWrite();

- digitalRead().

**Функция pinMode**

Устанавливает режим работы заданного входа/выхода (pin) как входа или как вы­хода.

Синтаксис:

pinMode(pin, mode);

Параметры:

 pin — номер входа/выхода (pin), который вы хотите установить;

 mode — режим. Одно из двух значений: input или output устанавливает на вход или выход соответственно.

Пример:

int ledPin = 13; // Светодиод, подключенный к входу/выходу 13

void setup()

{

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

**Функция digitalWrite()**

Подает high или low значение на цифровой вход/выход (pin).

Если вход/выход (pin) был установлен в режим выход (output) функцией pinMode(), то для значения high напряжение на соответствующем входе/выходе (pin) будет 5 В (3,3 В для плат 3,3 В) и 0 В (земля) для low.

Если вход/выход (pin) был установлен в режим вход (input), то функция digitalwrite со значением high будет активировать внутренний нагрузочный рези­стор 20 K. Подача low в свою очередь отключает этот резистор. Нагрузочного рези­стора достаточно, чтобы светодиод, подключенный к входу, светил тускло. Если вдруг светодиод работает, но очень тускло, возможно необходимо установить режим выход (output) функцией pinMode() .

Синтаксис:

digitalWrite(pin, value);

Параметры:

 pin — номер входа/выхода (pin);

 value — значение high или low.

Пример:

int ledPin = 13;// Светодиод, подключенный к входу/выходу 13

**5) Функция Void 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.

**6) Функция Void 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.

**Функция digitalRead()**

Функция считывает значение с заданного входа: high или low.

Синтаксис:

digitalRead(pin);

Параметр: pin — номер входа/выхода (pin), который вы хотите считать.

Пример:

int ledPin = 13; // Светодиод, подключенный к входу/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 13

pinMode(inPin, INPUT); // устанавливает режим работы - вход для 7 }

void loop()

{

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

}

Если вход не подключен, то digitalRead может возвращать значения HIGH или LOW случайным образом. Аналоговые входы (analog pins) могут быть использованы как цифровые входы/выходы (digital pins). Обращение к ним идет по номерам от 14 (для аналогового входа 0) до 19 (для аналогового входа 5).

**27. Аналоговый ввод/вывод. Функция analogRead(). Функция analogReference(), Функция analogWrite()**

1. **Аналоговый ввод/вывод**

Входы: A0, A1, A2, A3, A4, A5 (6) каждый из которых может представить аналоговое напряжение в виде 10-битного числа.

**Аналоговый ввод/вывод**

Рассмотрим функции аналогового ввода/вывода:

- analogRead() ;

- analogReference() ;

- analogWrite() .

**Функция analogRead()**

Функция считывает значение с указанного аналогового входа.

Большинство плат Arduino имеют 6 каналов (8 каналов у платы Mini и Nano, 16 — у Mega) с 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 напряжения и шаг может быть изменен функцией analogReference (). Считывание значения с аналого­вого входа занимает примерно 100 микросекунд (0,0001 сек), т. е. максимальная частота считывания приблизительно 10 000 раз в секунду.

Синтаксис:

analogRead(pin);

Параметр: pin — номер порта аналогового входа, с которого будет производиться считывание: 0..5 для большинства плат, 0..7 для Mini и Nano и 0..15 для Mega.

Возвращаемое значение int (0 to 1023).

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

}

**Функция analogReference()**

Функция определяет опорное напряжение, относительно которого происходят ана­логовые измерения. Функция analogRead() возвращает значение с разрешением 8 битов (1024) пропорционально входному напряжению на аналоговом входе 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;

 external — внешний источник опорного напряжения, подключенный к выводу AREF.

Синтаксис:

analogReference(type);

Параметр: type — определяет используемое опорное напряжение (default, internal или external).

Внешнее напряжение рекомендуется подключать к выводу AREF 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.

**Функция analogWrite()**

Выдает аналоговую величину (ШИМ-волну) на порт входа/выхода. Функция может быть полезна для управления яркостью подключенного светодиода или скоростью вращения электродвигателя. После вызова analogWrite () на выходе будет генери­роваться постоянная прямоугольная волна с заданной шириной импульса до сле­дующего вызова analogWrite (или вызова digitalWrite или digitalRead на том же порту входа/выхода). Частота ШИМ-сигнала приблизительно 490 Гц.

На большинстве плат Arduino (на базе микроконтроллера ATmega168 или ATmega328) ШИМ поддерживают порты 3, 5, 6, 9, 10 и 11, на плате Arduino Mega — 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 входа/выхода функцией pinMode(). Функция analogWrite() никак не связана с аналоговыми вхо­дами и с функцией analogRead ().

Синтаксис:

analogWrite(pin, value);

Параметры:

 pin — порт входа/выхода, на который подается ШИМ-сигнал;

 value — период рабочего цикла: значение между 0 (полностью выключено) и 255 (сигнал подан постоянно).

Период ШИМ-сигнала на портах входа/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 задания яркости светодиода пропорционально значению, снимаемому с потенциометра:

int ledPin = 9; // Светодиод подключен к выходу 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,

// подключенного к потенциометру analogWrite(ledPin, val / 4); // analogRead возвращает значения от 0 // до 1023, analogWrite должно быть // в диапазоне от 0 до 255

}

**28. Дополнительные фунции ввода/вывода. Функция tone(). Функция noTone(). Функция shiftOut(). Функция pulseIn()**

**Функция tone()**

Генерирует на порту входа/выхода сигнал — прямоугольную "волну" заданной частоты и с 50 % рабочим циклом. Длительность может быть задана параметром, в противном случае сигнал генерируется до тех пор, пока не будет вызвана функ­ция noTone(). К порту входа/выхода может быть подключен пьезо- или иной 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 вхо­да/выхода 3 и 11 (кроме платы Arduino Mega).

Синтаксис:

tone(pin, frequency);

tone(pin, frequency, duration);

Параметры:

- pin — номер порта входа/выхода, на котором будет генерироваться сигнал;

- frequency — частота сигнала в герцах;

- duration — длительность сигнала в миллисекундах.

**Функция noTone()**

Останавливает сигнал, генерируемый на порту входа/выхода, вызовом функции tone () . Если сигнал не генерировался, то вызов noTone() ни к чему не приводит.

Если необходимы сигналы на разных портах, то следует сначала остановить один сигнал функцией noTone(), а лишь затем создавать новый сигнал на другом порту функцией Tone().

Синтаксис:

noTone(pin);

Параметр: pin — номер порта входа/выхода, на котором прекращается сигнал.

**Функция shiftOut()**

Выводит байт информации на порт входа/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 входа/выхода, информируя о доступности к считыванию бита.

Такой способ передачи данных называется последовательным протоколом 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 (на англ. языке) по протоколу последова­тельного периферийного интерфейса (SPI, Serial Peripheral Interface Protocol).

Синтаксис:

shiftOut(dataPin, clockPin, bitOrder, value);

Параметры:

- dataPin — номер порта входа/выхода, на который выводятся биты (int);

- clockPin — номер порта, по которому производится синхронизация (int);

- bitOrder — используемая последовательность вывода битов. msbfirst (Most Significant Bit First) — слева или lsbfirst (Least Significant Bit First) — справа;

-  value — значение (байт) для вывода (byte).

Порт вывода (dataPin) и синхронизирующий порт (clockPin) должны быть предвари­тельно сконфигурированы как порты вывода с помощью функции pinMode ().

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

Пример вывода:

// 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, LSBFIRST, data);

// выводим старший бит

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

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

// 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);

}

}

**Функция pulseIn()**

Считывает длину сигнала на заданном порту (high или low). Например, если задано считывание high функцией pulsein(), функция ожидает, пока на заданном порту не появится high. Когда high получен, включается таймер, который будет остановлен, когда на порту входа/выхода будет low. Функция pulsein() возвращает длину сиг­нала в микросекундах. Функция возвращает 0, если в течение заданного времени (тайм-аута) не был зафиксирован сигнал на порту.

Возможны некоторые погрешности в измерении 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 входа/выхода, на котором будет ожидаться сигнал;

- value — тип ожидаемого сигнала: high или low;

- timeout время ожидания сигнала (тайм-аут) в секундах (unsigned long).

Возвращаемые значения: длина сигнала в микросекундах или 0, если сигнал не по­лучен до истечения тайм-аута (тип unsigned long).

Пример использования функции:

int pin = 7; unsigned long duration; void setup()

{

pinMode(pin, INPUT);

}

void loop()

{

duration = pulseIn(pin, HIGH);

}

**29. Работа со временем. Функция millis(). Функция micros(). Функция delay(). Функция delayMicroseconds()**

**Работа со временем**

**Функция millis()**

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

Параметров нет.

Возвращаемое значение — количество миллисекунд с момента начала выполнения программы (тип unsigned long).

Пример использования функции:

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);

}

**Функция micros()**

Возвращает количество микросекунд с момента начала выполнения текущей про­граммы на плате Arduino. Значение переполняется и сбрасывается на ноль прибли­зительно через 70 минут. На платах Arduino с 16 МГц (Duemilanove и Nano) функ­ция micros() имеет разрешение 4 секунды (возвращаемое значение всегда крат­но 4). На платах с 8 МГц (Arduino Lilypad) — разрешение функции 8 секунд.

Параметров нет.

Возвращаемое значение — количество микросекунд с момента начала выполнения программы (unsigned long).

Пример использования функции:

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);

}

**Функция delay()**

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

Синтаксис:

delay(ms);

Параметр: ms — количество миллисекунд, на которое приостанавливается выпол­нение программы ( тип unsigned long).

Пример использования функции:

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

}

Не рекомендуется использовать эту функцию для событий длиннее 10 миллисе­кунд, т. к. во время останова не могут быть произведены манипуляции с портами, не могут быть считаны сенсоры или произведены математические операции. В ка­честве альтернативного подхода возможно контролирование времени выполнения тех или иных функций с помощью millis(). При использовании функции delay() работа прерываний не останавливается, продолжается запись последовательно (serial) передаваемых данных на RX‑порту, ШИМ-сигнал (analogWrite) продолжает генерироваться на портах.

**Функция delayMicroseconds()**

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

В данной версии Arduino максимальная пауза, воспроизводимая корректно, — 16 383. Возможно, это будет изменено в следующих версиях Arduino. Для останов­ки выполнения программы, более чем на несколько тысяч микросекунд, рекомен­дуется использовать функцию delay().

Синтаксис:

delayMicroseconds(us);

Параметр: us — количество микросекунд, на которое приостанавливается выпол­нение программы (unsigned int).

Пример использования функции:

int outPin = 8; // цифровой порт входа/выхода 8

void setup()

{

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

void loop()

{

digitalWrite(outPin, HIGH); // подаем HIGH на выход delay(50); // ожидаем 50 микросекунд

digitalWrite(outPin, LOW); // устанавливаем LOW на выходе delay(50); // ожидаем 50 микросекунд

}

**30. Математические функции: map() ; pow() ; sq(); sqrt (); min(); max(); abs(); constrain();**

**Математические функции**

В языке представлены следующие математические функции:

- map() ;

- pow() ;

- sq();

- sqrt () .

- min() ;

- max() ;

- abs() ;

- constrain();

**Функция 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 использования функции:

sensVal = min(sensVal, 100);

// проверяем, если sensVal больше 100, то senseVal будет присвоено 100

**Функция 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 использования функции:

sensVal = max(sensVal, 20);

// проверяем, если sensVal меньше 20, то senseVal будет присвоено 20

Функция max() зачастую используется для ограничения нижней границы значений переменной. Функцией min() ограничивают верхнюю границу переменной. В силу специфики реализации функции max() следует избегать использования других функций в качестве параметров.

Например:

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

a-- ;

max(a, 0); // так корректно

**Функция abs()**

Возвращает модуль числа.

Параметр: x — число.

Возвращаемые значения:

-  x — если x больше или равен 0;

-  -x — если x меньше 0.

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

**Функция constrain(x, a, b)**

Функция проверяет и, если надо, задает новое значение так, чтобы оно было 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

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(). Функция contsrain () может быть использована до или после вы­зова map (), если необходимо ограничить допустимые значения заданным диапазо­ном.

Обратить внимание, что "нижняя граница" может быть как меньше, так и больше "верхней границы". Это может быть использовано, 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 правилам, 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 переносится значение;

- toHigh — верхняя граница нового диапазона.

Возвращаемое значение — значение в новом диапазоне.

Пример использования функции:

// 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);

}

**Функция pow(base, exponent)**

Вычисляет значение, возведенное в заданную степень. Функция pow() может воз­водить и в дробную степень.

Параметры:

-  base — число (тип float);

-  exponent — степень, в которую будет возводиться число (тип float). Возвращаемое значение — результат возведения в степень, число (тип double).

**Функция sq(x)**

Функция возвращает квадрат числа, заданного параметром.

Параметр: x — число, любой тип.

Возвращаемое значение — квадрат числа.

**Функция sqrt(x)**

Функция вычисляет квадратный корень числа, заданного параметром.

Параметры: x — число, любой тип.

Возвращаемое значение — квадратный корень числа (тип double).

 Тригонометрические функции

В языке представлены следующие тригонометрические функции: sin(); cos(); tan().

**Функция sin( rad)**

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

Параметр: rad — угол в радианах (float).

Возвращаемое значение: синус угла (тип double).

**Функция cos(rad)**

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

Параметр: rad — угол в радианах (тип float).

Возвращаемое значение: косинус угла (тип double).

**Функция tan(rad)**

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

Параметр: rad — угол в радианах (тип float).

Возвращаемое значение: тангенс угла (тип double).

**31. Генераторы случайных значений: randomSeed(); random() .**

**Генераторы случайных значений**

Функции формирования случайных чисел:

- randomSeed() ;

- random() .

**Функция 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.

Параметр: seed — параметр, задающий начало выдачи псевдослучайных значений на последовательности (тип int, long).

**Функция 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, включительно.

Возвращаемое значение: случайное число между min и max - 1 (тип long).

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() c неподключенного порта входа/выхода. В некоторых случаях необходимо получать одинаковую по­следовательность при каждом запуске программы на Arduino. Тогда инициализиро­вать генератор псевдослучайных чисел следует вызовом функции randomSeed() с фиксированным параметром.

Пример использования функции:

long randNumber; void setup()

{

Serial.begin(9600);

}

void loop()

{

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

Serial.println(randNumber);

// выводим случайное число из диапазона 0..19 randNumber = random(10, 20);

Serial.println(randNumber); delay(50);

}