



ШПОРА ARDUINO

Версия 1.0 от 01.03.2020

Краткая шпаргалка по основным функциям Arduino Wiring и языку C++. Оформлено с комментариями и примерами в виде кода: для большей наглядности и возможности сразу почувствовать код и запомнить, как он выглядит. Для полной информации по каждой главе обращайтесь по ссылкам.

[Оглавление](#)

Синтаксис	2
Переменные и типы данных	2
Область видимости	3
Строки.....	3
Serial.....	4
Условия и выбор.....	5
Циклы	6
Математика, вычисления	6
Функции	8
Входы/выходы.....	8
Цифровые.....	8
Аналоговые	8
ШИМ	9
Прерывания	9
Случайные числа	9
Функции времени.....	10
Структуры	10
Перечисления	10
Битовые операции	11
Указатели и ссылки	12

Синтаксис

```
// полный урок тут: https://alexgyver.ru/lessons/syntax/

// однострочный комментарий

/*
  многострочный
  комментарий
*/

// каждая команда оканчивается ;
// каждой скобке ( { < соответствует закрывающая > } )

// === ПРЕПРОЦЕССОР ===
#include <Servo.h>    // подключает библиотеку. Ищет в папке с библиотеками
#include "Servo.h"    // ищет в папке со скетчем, а потом в папке с библиотеками
#define MY_CONST 10   // объявить "жёсткую" константу MY_CONST равной 10

// эта функция обязательно должна быть в скетче в одном экземпляре
void setup() {
  // код выполнится 1 раз при старте программы
}

// эта функция обязательно должна быть в скетче в одном экземпляре
void loop() {
  // код будет выполняться циклично после setup
}
```

Переменные и типы данных

```
// полный урок тут: https://alexgyver.ru/lessons/syntax/

boolean flag1;           // объявить
boolean flag1, flag2;    // объявить несколько
boolean flag1 = true;    // объявить и инициализировать

// === ТИПЫ ДАННЫХ ===
boolean или bool        // 1 байт, логическая. true/false или 1/0
int8_t                  // 1 байт, целочисл., -128... 127
char                    // 1 байт, символьная, -128... 127 или 'a'
uint8_t, byte           // 1 байт, целочисл., 0... 255
int16_t, int, short     // 2 байта, целочисл., -32 768... 32 767
uint16_t, unsigned int, word // 2 байта, целочисл., 0... 65 535
int32_t, long           // 4 байта, целочисл., -2 147 483 648... 2 147 483 647
uint32_t, unsigned long // 4 байта, целочисл., 0... 4 294 967 295
float, double           // 4 байта, дробн., -3.4028235E+38... 3.4028235E+38
// примеч.: на других платформах double имеет размер 8 бит и большую точность

'a'                    // символ
"abc"                  // строка или массив символов

// === МАССИВЫ ===
int myInts[6];         // указываем количество ячеек
int myPins[] = {2, 4, 8, 3, 6}; // указываем содержимое ячеек
float Sens[3] = {0.2, 0.4, -8.5}; // указываем и то и то, количество ячеек должно совпадать
char message[6] = "hello"; // храним символы

// === СПЕЦИФИКАТОРЫ ===
const                 // константа, такую переменную нельзя изменить. const int val = 10;
static                // статическая переменная (см. ниже)
volatile              // не оптимизировать переменную. Использовать для работы в прерываниях
extern                // указывает компилятору, что эта переменная объявлена в другом файле программы
```

Область видимости

```
// === ГЛОБАЛЬНАЯ ===
// Глобальная переменная объявляется вне функций и доступна
// для чтения и записи в любом месте программы, в любой её функции.
byte var;
void setup() {
    // спокойно меняем глобальную переменную
    var = 50;
}
void loop() {
    // спокойно меняем глобальную переменную
    var = 70;
}

// === ЛОКАЛЬНАЯ ===
// Локальная переменная живёт внутри функции или внутри любого блока кода,
// заключённого в { фигурные скобки }, доступна для чтения и записи только внутри него.
void setup() {
    byte var; // локальная для setup переменная
    // спокойно меняем локальную переменную
    var = 50;
}
void loop() {
    // приведёт к ошибке, потому что в этом блоке кода var не объявлена
    var = 70;

    // сделаем тут отдельный блок кода
    {
        byte var2 = 10;
        // var2 существует только внутри этого блока!
    }
    // вот тут var2 уже будет удалена из памяти
}

// === СТАТИЧЕСКАЯ ЛОКАЛЬНАЯ ===
// статическая локальная переменная не удаляется из памяти
// после выхода из функции
void setup() {
    myFunc(); // вернёт 20
    myFunc(); // вернёт 30
    myFunc(); // вернёт 40
    myFunc(); // вернёт 50
}
void loop() {
}
byte myFunc() {
    static byte var = 10;
    var += 10;
    return var;
}
```

Строки

// полный урок тут: <https://alexgyver.ru/lessons/strings/>

```
String string0 = "Hello String"; // заполняем словами в кавычках
String string1 = String("lol ") + String("kek"); // сумма двух строк
String string2 = String('a'); // строка из символа в одинарных кавычках
String string3 = String("This is string"); // конвертируем строку в String
String string4 = String(string3 + " more"); // складываем строку string3 с текстом в
кавычках
String string5 = String(13); // конвертируем из числа в String
String string6 = String(20, DEC); // конвертируем из числа с указанием
базиса (десятичный)
```

```

String string7 = String(45, HEX);           // конвертируем из числа с указанием
базиса (16-ричный)
String string8 = String(255, BIN);         // конвертируем из числа с указанием
базиса (двоичный)
String string9 = String(5.698, 3);        // из float с указанием количества знаков
после запятой (тут 3)

// длина строки
String textString = "Hello";
sizeof(textString); // вернёт 6
textString.length(); // вернёт 5
// полный набор инструментов String тут https://alexgyver.ru/lessons/strings/

// ===== МАССИВЫ СИМВОЛОВ =====
// объявить массив текста длиной 6 символов
// и задать текст
char helloArray[] = "Hello!";

// объявить массив текста длиной 100 символов
// и задать в его начало текст
char textArray[100] = "World";

// длина строки
char textArray[100] = "World";
sizeof(textArray); // вернёт 100
strlen(textArray); // вернёт 5

```

Serial

```

// полный урок тут: https://alexgyver.ru/lessons/serial/

// === СТАРТ/СТОП ===
Serial.begin(Speed); // открыть порт на скорости
Serial.end();        // закрыть порт
Serial.available();  // возвращает количество байт в буфере приёма

// === ПЕЧАТЬ ===
// Отправляет в порт значение val - число или строку
Serial.print(val);
Serial.print(val, format);

// Отправляет и переводит строку
Serial.println(val);
Serial.println(val, format);

Serial.print(78); // выведет 78
Serial.print(1.23456); // 1.23 (умолч. 2 знака)
Serial.print('N'); // выведет N
Serial.print("Hello world."); // Hello world.
Serial.print(78, BIN); // вывод "1001110"
Serial.print(78, OCT); // вывод "116"
Serial.print(78, DEC); // вывод "78"
Serial.print(78, HEX); // вывод "4E"
Serial.print(1.23456, 0); // вывод "1"
Serial.print(1.23456, 2); // вывод "1.23"
Serial.print(1.23456, 4); // вывод "1.2345"

// === ПАРСИНГ ===
Serial.setTimeout(value); // таймаут ожидания приёма данных для парсинга, мс. По
умолчанию 1000 мс (1 секунда)
Serial.readString(); // принять строку
Serial.parseInt(); // принять целочисленное
Serial.parseFloat(); // принять float

```

Условия и выбор

// полный урок тут: <https://alexgyver.ru/lessons/conditions/>

```
// === Сравнение и логика ===  
== , != , >= , <= ; // равно, не равно, больше или равно, меньше или равно  
! , && , || ; // НЕ, И, ИЛИ
```

```
// === if-else ===  
// при выполнении одного действия {} необязательны  
if (a > b) c = 10; // если a больше b, то c = 10  
else c = 20; // если нет, то c = 20
```

```
// вместо сравнения можно использовать лог. переменную  
boolean myFlag, myFlag2;  
if (myFlag) c = 10;
```

```
// сложные условия  
// если оба флага true - c = 10  
if (myflag && myFlag2) c = 10;
```

```
// при выполнении двух и более {} обязательны  
if (myFlag) {  
    c = 10;  
    b = c;  
} else {  
    c = 20;  
    b = a;  
}
```

```
// === else if ===  
byte state;  
if (state == 1) a = 10; // если state 1  
else if (state == 2) a = 20; // если нет, но если state 2  
else a = 30; // если и это не верно, то вот
```

```
// === Оператор ? ===  
// "Короткий" вариант if-else  
int c = (a > b) ? 10 : -20; // если a > b, то c = 10. Если нет, то c = -20  
Serial.println( (flag) ? ("флаг поднят") : ("флаг опущен") );
```

```
// === Оператор выбора ===  
switch (val) {  
case 1: // выполнить, если val == 1  
    break;  
case 2: // выполнить, если val == 2  
    break;  
default: // выполнить, если val ни 1 ни 2  
    // default опционален  
    break;  
}
```

```
// Оператор break очень важен, позволяет выйти из switch  
// Можно использовать так:  
switch (val) {  
case 1:  
case 2:  
case 3:  
case 4:  
    // выполнить, если val == 1, 2, 3 или 4  
    break;  
case 5:  
    // выполнить, если val == 5  
    break;  
}
```

Циклы

// полный урок тут: <https://alexgyver.ru/lessons/loops/>

```
// === for ===
for (int i = 0; i < 10; i++) {
    Serial.println(i);    // вывод в порт 0, 1.. 9
}

// === while ===
while (a < b) {
    // выполняется, пока a меньше b
}

// === do while ===
// Отличается от while тем, что выполнится хотя бы один раз
do {
    // выполняется, пока a меньше b
} while (a < b);

// === Дополнительно ===
continue;    // перейти к след. итерации цикла
break;      // выйти из цикла
```

Математика, вычисления

// полный урок тут: <https://alexgyver.ru/lessons/compute/>

```
+ , - , * , / , % ;           // сложить, вычесть, умножить, разделить, остаток от
деления
a = b + c / d;

++ , -- , += , -= , *= , /= ; // прибавить 1, вычесть 1, прибавить, вычесть,
умножить, разделить
a++;           // ~ a = a + 1;
a /= 10;       // ~ a = a / 10;

// === БОЛЬШИЕ ВЫЧИСЛЕНИЯ ===
// ВАЖНО! Для арифметических вычислений по умолчанию используется ячейка long (4 байта)
// но при умножении и делении используется int (2 байта)
// Если при умножении чисел результат превышает 32'768, он будет посчитан некорректно.
// Для исправления ситуации нужно писать (long) перед умножением, что заставит МК
выделить дополнительную память
long val;
val = 2000000000 + 6000000;    // посчитает корректно (т.к. сложение)
val = 25 * 1000;              // посчитает корректно (умножение, меньше 32'768)
val = 35 * 1000;              // посчитает НЕКОРРЕКТНО! (умножение, больше
32'768)
val = (long)35 * 1000;         // посчитает корректно (выделяем память (long) )
val = 1000 + 35 * 10 * 100;    // посчитает НЕКОРРЕКТНО! (в умножении больше
32'768)
val = 1000 + 35 * 10 * 100L;    // посчитает корректно! (модификатор L)
val = (long)35 * 1000 + 35 * 1000; // посчитает НЕКОРРЕКТНО! Второе умножение всё
портит
val = (long)35 * 1000 + (long)35 * 1000; // посчитает корректно (выделяем память
(long) )

// === ВЫЧИСЛЕНИЯ FLOAT ===
// если при вычислении двух целочисленных нужен дробный результат - пишем (float)
float val;
val = 100 / 3;                // посчитает НЕПРАВИЛЬНО (результат 3.0)
val = (float)100 / 3;         // посчитает правильно (указываем (float))
val = 100.0 / 3;              // посчитает правильно (есть число float)

// при присваивании float числа целочисленному типу данных дробная часть отсекается
```

```

int val;
val = 3.25;          // val принимает 3
val = 3.92;          // val принимает 3
val = round(3.25);  // val принимает 3
val = round(3.92);  // val принимает 4

// === МАТЕМАТИЧЕСКИЕ ФУНКЦИИ ===
// Ограничить диапазон числа val между low и high
val = constrain(val, low, high);

// Перевести диапазон числа val (от inMin до inMax) в новый диапазон (от outMin до
outMax)
val = map(val, inMin, inMax, outMin, outMax);

min(a, b);          // Возвращает меньшее из чисел a и b
max(a, b);          // Возвращает большее из чисел
abs(x);             // Модуль числа
round(x);           // Математическое округление
radians(deg);      // Перевод градусов в радианы
degrees(rad);      // Перевод радиан в градусы
sq(x);             // Квадрат числа
cos(x)             // Косинус (радианы)
sin(x)             // Синус (радианы)
tan(x)             // Тангенс (радианы)
fabs(x)            // Модуль для float чисел
fmod(x, y)         // Остаток деления x на y для float
sqrt(x)           // Корень квадратный
sqrtf(x)          // Корень квадратный для float чисел
cbrt(x)           // Кубический корень
hypot(x, y)       // Гипотенуза ( корень(x*x + y*y) )
square(x)         // Квадрат ( x*x )
floor(x)          // Округление до целого вниз
ceil(x)           // Округление до целого вверх
exp(x)            // Экспонента (e^x)
cosh(x)           // Косинус гиперболический (радианы)
sinh(x)           // Синус гиперболический (радианы)
tanh(x)           // Тангенс гиперболический (радианы)
acos(x)           // Арккосинус (радианы)
asin(x)           // Арксинус (радианы)
atan(x)           // Арктангенс (радианы)
atan2(y, x)       // Арктангенс (y / x) (позволяет найти квадрант, в котором находится
точка)
log(x)            // Натуральный логарифм x ( ln(x) )
log10(x)          // Десятичный логарифм x ( log_10 x)
pow(x, y)         // Степень ( x^y )
fma(x, y, z)      // Возвращает x*y + z
fmax(x, y)        // Возвращает большее из чисел
fmin(x, y)        // Возвращает меньшее из чисел
trunc(x)          // Возвращает целую часть числа с дробной точкой
round(x)          // Математическое округление

// === КОНСТАНТЫ ===
F_CPU             // частота тактирования в Гц (16000000 для 16 МГц)
INT8_MAX          // 127 Максимальное значение для char, int8_t
UINT8_MAX         // 255 Максимальное значение для byte, uint8_t
INT16_MAX         // 32767 Максимальное значение для int, int16_t
UINT16_MAX        // 65535 Максимальное значение для unsigned int, uint16_t
INT32_MAX         // 2147483647 Максимальное значение для long, int32_t
UINT32_MAX        // 4294967295 Максимальное значение для unsigned long, uint32_t
M_E               // 2.718281828 Число e
M_LOG2E           // 1.442695041 log_2 e
M_LOG10E          // 0.434294482 log_10 e
M_LN2            // 0.693147181 log_e 2
M_LN10           // 2.302585093 log_e 10

```

```

M_PI           // 3.141592654 pi
M_PI_2        // 1.570796327 pi/2
M_PI_4        // 0.785398163 pi/4
M_1_PI        // 0.318309886 1/pi
M_2_PI        // 0.636619772 2/pi
M_2_SQRTPI    // 1.128379167 2/корень(pi)
M_SQRT2       // 1.414213562 корень(2)
M_SQRT1_2     // 0.707106781 1/корень(2)
PI            // 3.141592654 Пи
HALF_PI       // 1.570796326 пол Пи
TWO_PI        // 6.283185307 два Пи
EULER         // 2.718281828 Число Эйлера e
DEG_TO_RAD    // 0.01745329  Константа перевода град в рад
RAD_TO_DEG    // 57.2957786

```

Функции

// полный урок тут: <https://alexgyver.ru/lessons/functions/>

// Функция, которая ничего не принимает и ничего не возвращает. Пример - сумма

```

void sumFunction() {
    c = a + b;
}

```

// Функция, которая ничего не принимает и возвращает результат. Пример - сумма

```

int sumFunction() {
    return (a + b);
}

```

// Функция, которая принимает параметры и возвращает результат. Пример - сумма

```

int sumFunction(byte paramA, byte paramB) {
    return (paramA + paramB);
}

```

// оператор return завершает выполнение функции и возвращает результат

// в void функции он вернёт void, всё верно

Входы/выходы

Цифровые

// урок: <https://alexgyver.ru/lessons/digital/>

```
pinMode(pin, mode);
```

// Устанавливает режим работы пина pin (ATmega 328: D0-D13, A0-A5) на режим mode:

// INPUT - вход (все пины сконфигурированы так по умолчанию)

// OUTPUT - выход (при использовании analogWrite ставится автоматически)

// INPUT_PULLUP - подтяжка к питанию (например для обработки кнопок)

```
digitalRead(pin);
```

// Читает состояние пина pin и возвращает :

// 0 или LOW - на пине 0 Вольт (точнее 0-2.5В)

// 1 или HIGH - на пине 5 Вольт (точнее 2.5-опорное В)

```
digitalWrite(pin, value);
```

// Подаёт на пин pin сигнал value:

// 0 или LOW - 0 Вольт (GND)

// 1 или HIGH - 5 Вольт (точнее, напряжение питания)

Аналоговые

// урок: <https://alexgyver.ru/lessons/analog-pins/>

```
analogRead(pin);
```

// Читает и возвращает оцифрованное напряжение с пина pin. 0-1023

// Перевести значение в напряжение:


```
float volt = (float)(analogRead(pin) * 5.0) / 1024;
// именно /1024, потому что АЦП сам отнимает 1 бит при вычислении

analogReference(mode);
// Устанавливает режим работы АЦП согласно mode:
// DEFAULT: опорное напряжение равно напряжению питания МК
// INTERNAL: встроенный источник опорного на 1.1V для ATmega168 или ATmega328P и 2.56V
на ATmega8
// INTERNAL1V1: встроенный источник опорного на 1.1V (только для Arduino Mega)
// INTERNAL2V56: встроенный источник опорного на 2.56V (только для Arduino Mega)
// EXTERNAL: опорным будет считаться напряжение, поданное на пин AREF
```

ШИМ

// урок: <https://alexgyver.ru/lessons/pwm-signal/>

```
analogWrite(pin, value);
// Запускает генерацию ШИМ сигнала на пине pin со значением value.
// Для стандартного 8-ми битного режима это значение 0-255, соответствует скважности 0-
100%.
// ШИМ пины:
// ATmega 328/168 (Nano, UNO, Mini): D3, D5, D6, D9, D10, D11
// ATmega 32U4 (Leonardo, Micro): D3, D5, D6, D9, D10, D11, D13
// ATmega 2560 (Mega): D2 - D13, D44 - D46
```

Прерывания

// полный урок тут: <https://alexgyver.ru/lessons/interrupts/>

```
attachInterrupt(pin, ISR, mode);
// Подключить прерывание на номер прерывания pin,
// назначить функцию ISR как обработчик и
// установить режим прерывания mode:
// LOW - срабатывает при сигнале LOW на пине
// RISING - срабатывает при изменении сигнала на пине с LOW на HIGH
// FALLING - срабатывает при изменении сигнала на пине с HIGH на LOW
// CHANGE - срабатывает при изменении сигнала (с LOW на HIGH и наоборот)
```

```
volatile int counter = 0; // переменная-счётчик
void setup() {
  Serial.begin(9600); // открыли порт для связи
  // подключили кнопку на D2 и GND
  pinMode(2, INPUT_PULLUP);

  // D2 это прерывание 0
  // обработчик - функция buttonTick
  // FALLING - при нажатии на кнопку будет сигнал 0, его и ловим
  attachInterrupt(0, buttonTick, FALLING);
}
void buttonTick() {
  counter++; // + нажатие
}
void loop() {
  Serial.println(counter); // выводим
  delay(1000); // ждём
}
```

Случайные числа

// полный урок тут: <https://alexgyver.ru/lessons/random/>

```
random(max); // возвращает случайное число в диапазоне от 0 до (max - 1)
random(min, max); // возвращает случайное число в диапазоне от min до (max - 1)
randomSeed(value); // дать генератору случайных чисел новую опорную точку для счёта
```

Функции времени

// полный урок тут: <https://alexgyver.ru/lessons/time/>

```
delay(period);  
// Приостанавливает" выполнение кода на time миллисекунд.  
// Дальше функции delay выполнение кода не идёт, за исключением прерываний.
```

```
delayMicroseconds(period);  
// Аналог delay(), но в микросекундах
```

```
millis(); // Возвращает количество миллисекунд, прошедших со старта программы  
micros(); // Возвращает количество микросекунд, прошедших со старта программы
```

Структуры

// полный урок тут: <https://alexgyver.ru/lessons/variables-types/>

```
struct myStruct { // создаём ярлык myStruct  
  boolean a;  
  byte b;  
  int c;  
  long d;  
  byte e[5];  
} kek; // и сразу создаём структуру kek  
  
// создаём массив структур cheburek типа myStruct  
myStruct cheburek[3];  
  
void setup() {  
  // присвоим членам структуры значения вручную  
  kek.a = true;  
  kek.b = 10;  
  kek.c = 1200;  
  kek.d = 789456;  
  kek.e[0] = 10; // e у нас массив!  
  kek.e[1] = 20;  
  kek.e[2] = 30;  
  
  // присвоим структуру kek структуре cheburek номер 0  
  cheburek[0] = kek;  
  
  // присвоим элемент массива из структуры kek  
  // структуре cheburek номер 1  
  cheburek[0].e[1] = kek.e[1];  
  
  // забьём данными структуру cheburek номер 2  
  cheburek[2] = (myStruct) {  
    false, 30, 3200, 321654, {1, 2, 3, 4, 5}  
  };  
}
```

Перечисления

// полный урок тут: <https://alexgyver.ru/lessons/variables-types/>

```
// создаём перечисление modes, не создавая ярлык  
enum {  
  NORMAL,  
  WAITING,  
  SETTINGS_1,  
  SETTINGS_2,  
  CALIBRATION,  
  ERROR_MODE,  
} modes;
```

```

void setup() {
  Serial.begin(9600); // для отладки
  modes = CALIBRATION; // присваивание значения
  // можем сравнивать
  if (modes == CALIBRATION) {
    Serial.println("calibr");
  } else if (modes == ERROR_MODE) {
    Serial.println("error");
  }

  // присваиваем числом
  modes = 3; // по нашему порядку это будет SETTINGS_2
}

```

Битовые операции

// полный урок тут: <https://alexgyver.ru/lessons/bitmath/>

```

// & - битовое И
// << - битовый сдвиг влево
// >> - битовый сдвиг вправо
// ^ - битовое исключающее ИЛИ (аналогичный оператор - xor)
// | - битовое ИЛИ
// ~ - битовое НЕ

bit(val); // возвращает 2 в степени val (0 будет 1, 1 будет 2, 2 будет 4, 3
будет 8 и т.д.)
bitClear(x, n); // устанавливает на 0 бит, находящийся в числе x под номером n
bitSet(x, n); // устанавливает на 1 бит, находящийся в числе x под номером n
bitWrite(x, n, b); // устанавливает на значение b (0 или 1) бит , находящийся в числе
x под номером n
bitRead(x, n); // возвращает значение бита (0 или 1), находящегося в числе x под
номером n
highByte(x); // извлекает и возвращает старший (крайний левый) байт переменной
типа word (либо второй младший байт переменной, если ее тип занимает больше двух байт).
lowByte(x); // извлекает и возвращает младший (крайний правый) байт переменной
(например, типа word).

// ===== Битовое И =====
// 0 & 0 == 0
// 0 & 1 == 0
// 1 & 0 == 0
// 1 & 1 == 1
myByte = 0b11001100;
myBits = myByte & 0b10000111;
// myBits теперь равен 0b10000100

// ===== Битовое ИЛИ =====
// 0 | 0 == 0
// 0 | 1 == 1
// 1 | 0 == 1
// 1 | 1 == 1
myByte = 0b11001100;
myBits = myByte | 0b00000001; // ставим бит №0
// myBits теперь равен 0b11001101

// ===== Битовое НЕ =====
~0 == 1
~1 == 0
myByte = 0b11001100;
myByte = ~myByte; // инвертируем
// myByte теперь 00110011

```

```

// ===== Битовое исключающее ИЛИ =====
// 0 ^ 0 == 0
// 0 ^ 1 == 1
// 1 ^ 0 == 1
// 1 ^ 1 == 0
myByte = 0b11001100;
myByte ^= 0b10000000; // инвертируем 7-ой бит
// myByte теперь 01001100

// ===== Битовый сдвиг =====
myByte = 0b00011100;
myByte = myByte << 3; // двигаем на 3 влево
// myByte теперь 0b11100000

myByte >>= 5;
// myByte теперь 0b00000111

myByte >>= 2;
// myByte теперь 0b00000001
// остальные биты потеряны!

```

Указатели и ссылки

```

// полный урок тут: https://alexgyver.ru/lessons/pointers/

// & - возвращает адрес данных в памяти (адрес первого блока данных)
// * - управляет значением по указанному адресу

// === указатели ===
// управление переменной через указатель
byte b; // просто переменная типа byte
b = 10; // b теперь 10
byte* ptr; // ptr - переменная "указатель на объект типа byte"
ptr = &b; // указатель ptr хранит адрес переменной b
*ptr = 24; // b теперь равна 24 (записываем по адресу &b)
byte s; // переменная s
s = *ptr; // s теперь тоже равна 24 (читаем по адресу &b)

// === ссылки ===
// управление переменной через ссылку
byte b; // просто переменная типа byte
b = 10; // b теперь 10
byte &link = b; // link - переменная "ссылка на объект типа byte"
link = 24; // b теперь равна 24 (записываем через ссылку)
byte s; // переменная s
s = link; // s теперь тоже равна 24 (читаем по ссылке)

```