**Типы данных, переменные**



**Переменная** - это ячейка в [оперативной памяти](https://alexgyver.ru/lessons/basics/) микроконтроллера, которая имеет своё уникальное название (а также адрес в памяти) и хранит значение соответственно своему размеру. К переменной мы можем обратиться по её имени или адресу и получить это значение, либо изменить его. Зачем это нужно? В переменной могут храниться промежуточные результаты вычислений, полученные "снаружи" данные (с датчиков, Интернета, интерфейсов связи) и так далее.

**Измерение информации**

Прежде чем перейти к переменным и их типам, нужно вспомнить школьный курс информатики, а именно - как хранятся данные в "цифровом" мире. Любая память состоит из элементарных ячеек, которые имеют всего **два** состояния: 0 и 1. Эта единица информации называется **бит**(**bit**). Минимальным блоком памяти, к которому можно обратиться из программы по имени или адресу, является **байт** (**byte**), который в Arduino (и в большинстве других платформ и процессоров) состоит из **8 бит**, таким образом любой тип данных будет кратен 1 байту.

Максимальное количество значений, которое можно записать в один байт, составляет 2^8 = 256. В программировании счёт всегда начинается с нуля, поэтому один байт может хранить число от **0 до 255**. Более подробно о двоичном представлении информации и битовых операциях мы поговорим в [отдельном уроке](https://alexgyver.ru/lessons/bitmath/).

Стандартные типы переменных в Arduino по своему размеру кратны степени двойки, давайте их распишем:

* 1 байт = 8 бит = 256
* 2 байта = 16 бит = 65 536
* 4 байта = 32 бита = 4 294 967 296

**Типы данных**

Переменные разных типов имеют разные особенности и позволяют хранить числа в разных диапазонах.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Альт. название** | **Вес** | **Диапазон** | **Особенность** |
| boolean | bool | 1 байт **\*** | 0 или 1,  **true**   или  **false** | Логический тип |
| char | - | 1 байт | -128… 127 (AVR), 0.. 255 (esp) | Символ (код символа) из таблицы ASCII |
| - | int8\_t | 1 байт | -128… 127 | Целые числа |
| byte | uint8\_t | 1 байт | 0… 255 | Целые числа |
| int  **\*\*** | int16\_t  ,  short | 2 байта | -32 768… 32 767 | Целые числа. На ESP8266/ESP32 - 4 байта! См. ниже |
| unsigned int  **\*\*** | uint16\_t  ,  word | 2 байта | 0… 65 535 | Целые числа. На ESP8266/ESP32 - 4 байта! См. ниже |
| long | int32\_t | 4 байта | -2 147 483 648…    2 147 483 647 | Целые числа |
| unsigned long | uint32\_t | 4 байта | 0… 4 294 967 295 | Целые числа |
| float | - | 4 байта | -3.4E+38… 3.4E+38 | Числа с плавающей точкой, точность: 6-7 знаков |
| double | - | 4/8 байт | -1.7E+308.. 1.7E+308 | Для AVR то же самое, что  float  .  На ESP и прочих 32-бит МК - 8 байт, точность - 15-16 знаков |
| - | int64\_t | 8 байт **\*\*\*** | -(2^64)/2... (2^64)/2-1 | Целые числа |
| - | uint64\_t | 8 байт **\*\*\*** | 2^64-1 | Целые числа |

* (**\***) - да,

bool

 занимает 1 байт (8 бит), так как это минимальная адресуемая ячейка памяти. Есть способы запаковать логические переменные в 1 бит, о них поговорим в [другом уроке](https://alexgyver.ru/lessons/code-optimisation/).

* (**\*\***) - на ESP8266/ESP32

int

 и

unsigned int

 занимает 4 байта, то есть является аналогами типов

long

 и

unsigned long

!

* (**\*\*\***) - Компилятор также поддерживает **64 битные** числа. Стандартные Arduino-библиотеки с переменными этого типа не работают, поэтому можно использовать только в своём коде.

**Целочисленные типы**

Переменные целочисленных типов нужны для хранения целых чисел. В своей программе рекомендуется использовать альтернативное название типов (второй столбец в таблице выше), потому что:

* Проще ориентироваться в максимальных значениях
* Легче запомнить
* Название более короткое
* Проще изменить один тип на другой
* Размер переменной задан жёстко и не зависит от платформы (например

int

 на AVR это 2 байта, а на esp8266 - 4 байта)

Максимальные значения хранятся в константах, которые можно использовать в коде. Иногда это помогает избавиться от лишних вычислений:

* UINT8\_MAX

 - 255

* INT8\_MAX

 - 127

* UINT16\_MAX

 - 65 535

* INT16\_MAX

 - 32 767

* UINT32\_MAX

- 4 294 967 295

* INT32\_MAX

 - 2 147 483 647

* UINT64\_MAX

 - 18 446 744 073 709 551 615

* INT64\_MAX

 -

**Логический тип**

bool

 - логический, он же **булевый** (придуман [Джорджем Булем](https://ru.wikipedia.org/wiki/%D0%91%D1%83%D0%BB%D1%8C,_%D0%94%D0%B6%D0%BE%D1%80%D0%B4%D0%B6)) тип данных, принимает значения

0

 и

1

 или

**false**

 и

**true**

 - ложь и правда. Используется для хранения состояний, например включено/выключено, а также для работы в [условных конструкциях](https://alexgyver.ru/lessons/conditions/).

Также переменная типа

bool

 принимает значение

**true**

, если присвоить ей любое **отличное от нуля**число.

bool a = 0; // false

bool b = 1; // true

bool c = 25; // true

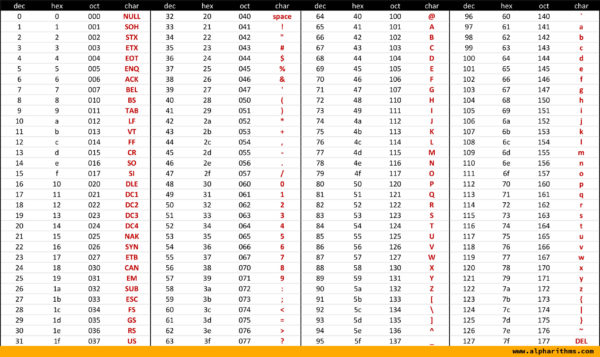
**Символьный тип**

char

 - тип данных для хранения символов, символ указывается в **одинарных кавычках**:

char var = 'a';

. По факту это целочисленный тип данных, а переменная хранит номер (код) символа в таблице ASCII:

[](https://alexgyver.ru/wp-content/uploads/2021/07/ascii-table-alpharithms-scaled-1.jpg)

Отдельный символьный тип данных нужен для удобства работы, чтобы программа могла понять разницу между числом и символом, например для вывода на дисплей (чтобы вывести именно букву **A**, а не число **65**). Из символов можно составлять строки, об этом более подробно поговорим в уроках про [символьные строки](https://alexgyver.ru/lessons/cstring/) и [String-строки](https://alexgyver.ru/lessons/strings/" \t "_blank).

**Символы и числа**

Несмотря на то, что в языке Си символ это по сути целое число, значения например

'3'

 и

3

 не равны между собой, потому что символ

'3'

 с точки зрения программы является числом

51

. На практике иногда бывает нужно конвертировать символы чисел в соответствующие им целые числа и наоборот (при работе со строками и буферами вручную), для этого распространены следующие алгоритмы:

* Из символа в число - взять младший ниббл (4 бита):

symbol & *0xF*

* Из символа в число - вычесть символ 0:

symbol - '0'

* Из числа в символ - прибавить символ 0:

symbol + '0'

**Дробные числа**

float

*(англ. float - плавающий)* - тип данных для чисел с плавающей точкой, т.е. десятичных дробей. Arduino поддерживает три типа ввода чисел с плавающей точкой:

|  |  |  |
| --- | --- | --- |
| **Тип записи** | **Пример** | **Чему равно** |
| Десятичная дробь | 20.5 | 20.5 |
| Научный | 2.34E5 | 2.34\*10^5 или 234000 |
| Инженерный | 67e-12 | 67\*10^-12 или 0.000000000067 |

Выше в таблице есть пометка "точность: 6-7 знаков" - это означает, что в этом типе можно хранить числа, размер которых не больше 6-7 цифр, остальные цифры будут утеряны! Причём целой части отдаётся приоритет. Вот так это выглядит в числах (в комментарии - реальное число, которое записалось в переменную):

float v;

v = 123456.654321; // 123456.656250

v = 0.0123456789; // 0.0123456788

v = 0.0000123456789; // 0.0000123456788

v = 123456789; // 123456792.0

Другие особенности float чисел и работу с ними мы рассмотрим в уроках про [математические операции](https://alexgyver.ru/lessons/compute/) и [условия](https://alexgyver.ru/lessons/conditions/).

**Объявление и инициализация**

* Объявление переменной - резервирование ячейки памяти указанного типа на имя:

тип\_данных имя;

* Присваивание - задание переменной значения при помощи оператора = (равно):

имя = значение;

* Инициализация переменной - объявление и присваивание начального значения:

тип\_данных имя = значение;

Можно объявить и инициализировать несколько переменных через запятую:

byte myVal;

int sensorRead = 10;

byte val1, val2, val3 = 10;

* Переменная должна быть объявлена **до** использования, буквально **выше по коду**. Иначе вы получите ошибку *Not declared in this scope* - переменная не объявлена.
* Нельзя объявить две и более переменных с одинаковым именем в одной области определения.

**Константы**

Что такое константа понятно из её названия - что-то, значение чего мы можем **только прочитать** и не можем изменить: при попытке изменить получим ошибку компиляции. Задать константу можно двумя способами:

Как переменную, указав перед типом данных слово **const**:

const тип\_данных имя = значение;

. Пример:

const byte myConst = 10;

. По сути это будет обычная переменная, но её значение нельзя поменять. Особенности:

* **Занимает место** в оперативной памяти, но может быть оптимизирована (вырезана) компилятором, если используется просто как значение.
* **Имеет адрес** в памяти, по которому к ней можно обратиться.
* Вычисления с ней не оптимизируются и чаще всего выполняются точно так же, как с обычными переменными.
* Компилятор выдаст ошибку, если имя константы совпадает с именем другой переменной в программе.

При помощи директивы **#define**, *без знака равенства и точки с запятой в конце*:

#define имя значение

. Пример:

#define BTN\_PIN 10

. Работает так: указанное **имя** буквально заменяется в тексте программы на указанное **значение**. Такая дефайн-константа:

* **Не занимает места в оперативной памяти**, а хранится во Flash памяти как часть кода программы.
* Не имеет адреса в оперативной памяти.
* Вычисления с такими константами оптимизируются и выполняются быстрее, так как это просто цифры.
* Если имя дефайн-константы совпадёт с именем другого "объекта" в программе или даже в библиотеке - работа может быть непредсказуемой: можно получить невнятную ошибку компиляции, либо программа может просто работать некорректно! Дефайн буквально **заменяет текст** в коде программы, это довольно опасная штука.

Во избежание проблем нужно называть дефайн-константы максимально уникальными именами. Можно добавлять к ним префиксы, например вместо PERIOD сделать MY\_PERIOD и так далее.

**Область видимости**

Переменные, константы

const

 и другие создаваемые пользователем данные имеют такое важное понятие, как **область видимости**. Она бывает глобальной и локальной.

**Глобальная**

Глобальная переменная:

* Объявляется **вне функций**, например просто в начале программы.
* Доступна для чтения и записи **в любом месте программы**.
* Находится в оперативной памяти **на всём протяжении работы программы**, то есть не теряет своё значение.
* При объявлении имеет **нулевое значение**.

byte var; // глобальная переменная

void setup() {

var = 50;

}

void loop() {

var = 70;

}

**Локальная**

Локальная переменная:

* Объявляется внутри любого **блока кода**, заключённого в

{ фигурные скобки }

.

* Доступна для чтения и записи только **внутри своего блока кода** (и во всех вложенных в него).
* Находится в оперативной памяти с момента объявления и до закрывающей фигурной скобки, то есть удаляется из памяти и её значение стирается.
* При объявлении имеет **случайное значение**.

Важный момент: если имя локальной переменной совпадает с одной из глобальных, то приоритет обращения отдаётся локальной переменной (в её области определения).

byte var; // глобальная переменная

void setup() {

byte var; // локальная переменная

var = 50; // меняем локальную var

}

void loop() {

var = 70; // меняем глобальную var

}

**Статические переменные**

Вспомним, как работает обычная локальная переменная: при входе в свой блок кода локальная переменная **создаётся заново**, а при выходе - **удаляется из памяти** и теряет своё значение. Если локальная переменная объявлена как

static

 - она будет **сохранять своё значение на всём протяжении работы программы**, но область видимости останется локальной: взаимодействовать с переменной можно будет **только внутри блока кода, где она создана** (и во всех вложенных в него).

void setup() {

}

void loop() {

byte varL = 0;

varL++;

static byte varS = 0;

varS++;

// здесь varL всегда будет равна 1

// а varS - постоянно увеличиваться

}

Статические переменные позволяют более красиво организовывать свой код, избавляясь от лишних глобальных переменных.

**Преобразование типов**

Иногда требуется преобразовать один тип данных в другой: например, функция принимает int, а вы хотите передать ей byte. В большинстве случаев компилятор сам разберётся и преобразует byte в int, но иногда вылетает ошибка в стиле "*попытка передать byte туда, где ждут int*". В таком случае можно преобразовать тип данных, для этого достаточно указать нужный тип данных в скобках перед преобразуемой переменной (тип\_данных)переменная, иногда можно встретить запись тип\_данных(переменная). Результат вернёт переменную с новым типом данных, сам же тип данной у переменной не изменится. Например:

// переменная типа byte

byte val = 10;

// передаём какой-то функции, которая ожидает int

sendVal( (int)val );

И всё! Val будет обрабатываться как int, а не как byte.