

# ARDUINO для ДЕТЕЙ

С помощью этой книги начинающие компьютерные гении смогут собрать на базе платформы Arduino световую гирлянду, аппарат азбуки Морзе, двигатель постоянного тока и другие полезные вещи и запрограммировать режимы их работы. В доступной и интересной форме с прекрасными примерами детям объясняются азы электроники, электротехники и программирования на диалекте языка C++ для Arduino.

Издание будет полезно детям от 10 лет, которые интересуются компьютерной техникой.

Для работы нужно иметь стартовый набор Arduino.

Интернет-магазин:  
[www.dmkpress.com](http://www.dmkpress.com)  
Книга – почтой:  
e-mail: [orders@alians-kniga.ru](mailto:orders@alians-kniga.ru)  
Оптовая продажа:  
КТК «Галактика»  
Тел./факс: (499) 782-3889  
e-mail: [books@alians-kniga.ru](mailto:books@alians-kniga.ru)

**DMK**  
ИЗДАТЕЛЬСТВО  
[www.dmk.pf](http://www.dmk.pf)

ISBN 978-5-97060-541-7



9 785970 605417 >

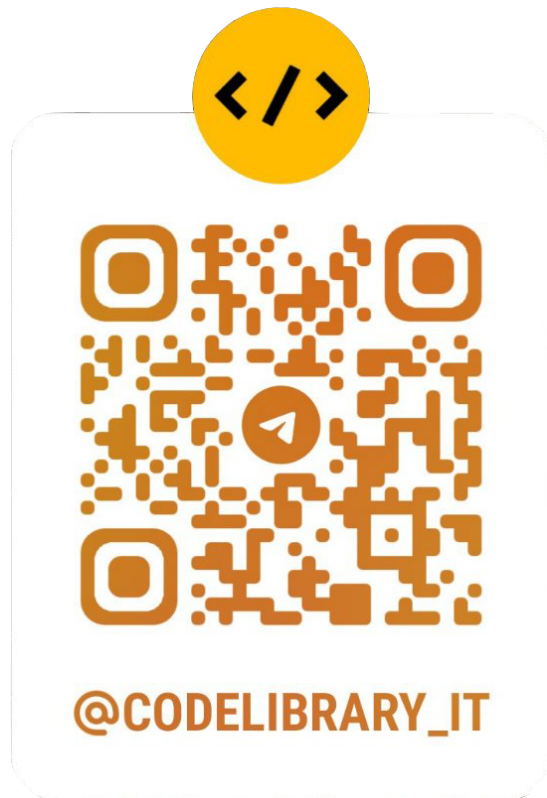
ARDUINO для ДЕТЕЙ

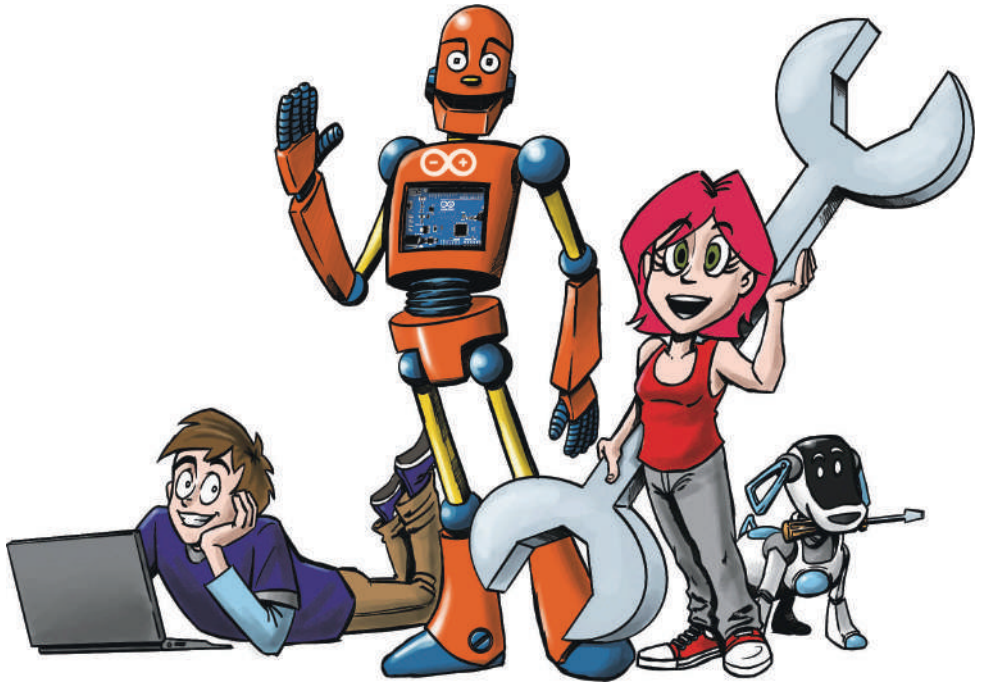


Эрик Шернич

Эрик Шернич

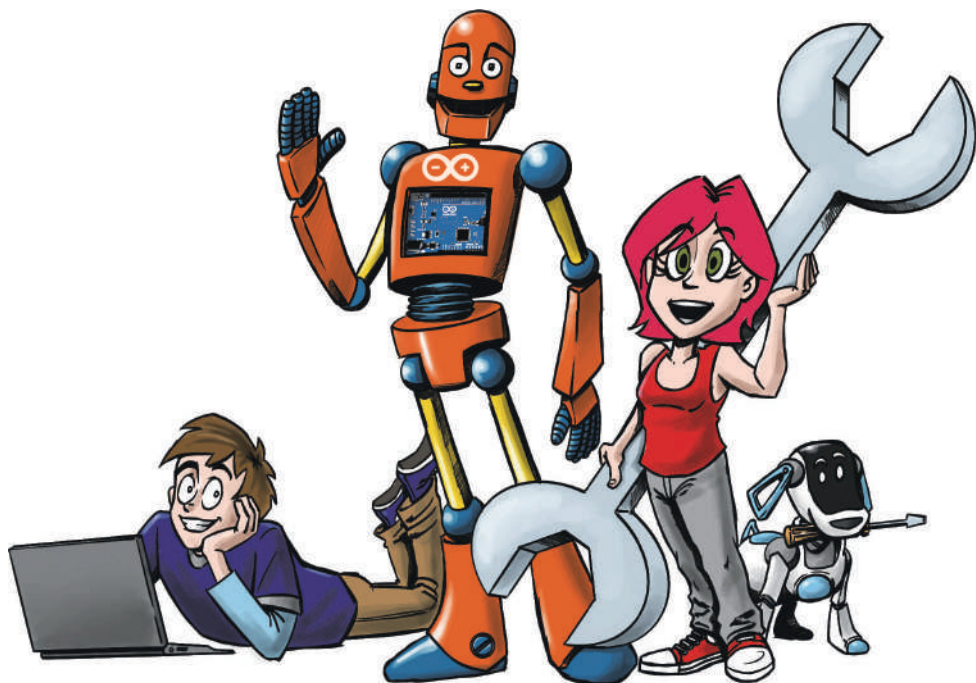
# Arduino для детей





# Arduino für Kids

Эрик Шернич



# Arduino для детей

**ДМК**  
издательство

Москва, 2019



**УДК 681.4:004.9Arduino**  
**ББК 32.816с515+32.965с515**  
**Ш49**

**Шернич Э.**

Ш49 **Arduino для детей / пер. с нем. М. М. Степаненковой. – М.: ДМК Пресс, 2019. – 170 с.: ил.**

**ISBN 978-5-97060-541-7**

С помощью этой книги начинающие компьютерные гении и смогут собрать на базе платформы Arduino световую гирлянду, аппарат азбуки Морзе, двигатель постоянного тока и другие полезные вещи и запрограммировать режимы их работы. В доступной и интересной форме с прекрасными примерами детям объясняются азы электроники, электротехники и программирования на диалекте языка C++ для Arduino.

Издание будет полезно детям от 10 лет, которые интересуются компьютерной техникой, для работы нужно иметь стартовый набор Arduino.

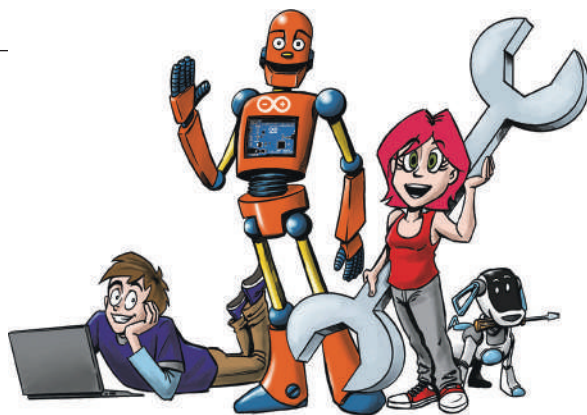
**УДК 681.4:004.9Arduino**  
**ББК 32.816с515+32.965с515**

First published as Arduino fur Kids by Erik Schernich. © 2nd edition 2017 by MITP Verlag GmbH&Co, KG Allrights reserved. Published with arrangements made by Maria Pinto-Peuckmann, Literary Agency-World Copyright Promotion, Kaufering, Germany.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

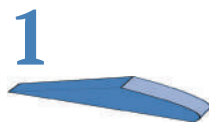
ISBN 978-3-95845-581-8 (нем.)  
ISBN 978-5-97060-541-7 (рус.)

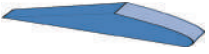
Copyright © 2017 mitp-Verlags GmbH & Co. KG  
© Издание, перевод, ДМК Пресс, 2019



# Содержание

<b>Введение</b> .....	9
Что такое микросхема?.....	9
Как пишутся программы? .....	10
Материалы.....	11
Начало работы.....	11
Заключение .....	16
Несколько заданий .....	17
<b>Мигай, мигай, огонек</b> .....	19
Установка программного обеспечения .....	19
Наша первая программа.....	22
Наша вторая программа: гирлянда со светодиодами .....	27
Наша третья программа: аппарат Морзе .....	36
Наша четвертая программа: игра «Горячий провод» .....	39
Заключение .....	48
Несколько вопросов.....	49
...и несколько заданий.....	49



	<b>2</b>	<b>Arduino говорит</b> ..... 51	51
		Отправка первого текста ..... 51	51
		Заключение ..... 59	59
		Вопрос..... 60	60
		...и задание на сегодня ..... 60	60
	<b>3</b>	<b>Сенсоры – интерфейсы для мира</b> ..... 61	61
		Что такое датчик? ..... 62	62
		Включить светодиоды ..... 63	63
		Заключение ..... 72	72
		Несколько вопросов..... 73	73
...и несколько заданий..... 73	73		
	<b>4</b>	<b>Моторы – движение с Arduino</b> ..... 75	75
		Двигатель постоянного тока – веселое вращение ..... 76	76
		Эффективно управлять двигателем ..... 79	79
		Сервоприводы ..... 81	81
		Заключение ..... 86	86
		Несколько вопросов..... 86	86
...и несколько заданий..... 86	86		
	<b>5</b>	<b>Чтение исходного кода других разработчиков</b> .... 87	87
		Документация ..... 87	87
		Загадочный исходный код ..... 88	88
		Заключение ..... 91	91
		Несколько вопросов..... 91	91
...и задание ..... 91	91		
	<b>6</b>	<b>ЖК-дисплей – отображение данных на самом Arduino</b> ..... 93	93
		Что такое ЖК-дисплей? ..... 94	94
		Заключение ..... 98	98

Несколько вопросов.....	98
...и задание .....	98
<b>Arduino и мультиметр .....</b>	<b>99</b>
Для начала история из жизни .....	99
Какие сведения нам нужны?.....	100
Изучаем потенциометр .....	105
Заключение .....	108
Несколько вопросов.....	108
...и несколько заданий.....	108
<b>Arduino online .....</b>	<b>109</b>
HTML – ворота в интернет .....	109
«Сеть, нам нужна сеть» .....	111
Заключение .....	114
Несколько вопросов.....	115
...и несколько заданий.....	115
<b>Клавиатура с Arduino Leonardo .....</b>	<b>117</b>
Первые шаги с Leonardo .....	118
Первая маленькая клавиатура .....	120
Ключ обеспечения секретности .....	124
Заключение .....	126
Несколько вопросов.....	126
...и несколько заданий.....	127
<b>Взгляд за пределы IDE.....</b>	<b>129</b>
C++, сердце Arduino.....	129
Перенос программы на Arduino.....	134
Программирование AVR.....	137
Заключение .....	142
Несколько вопросов.....	142
...и несколько заданий.....	142

7



8



9

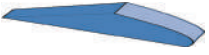


10





## 11

**Не забудь меня – использование EEPROM..... 145**

Общая информация о EEPROM..... 145

Что можно запрограммировать в EEPROM? ..... 147

Проект: черный ящик..... 147

Заключение ..... 156

Несколько вопросов..... 156

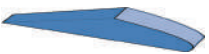
...и несколько заданий..... 156

## А

**Установка IDE..... 157**

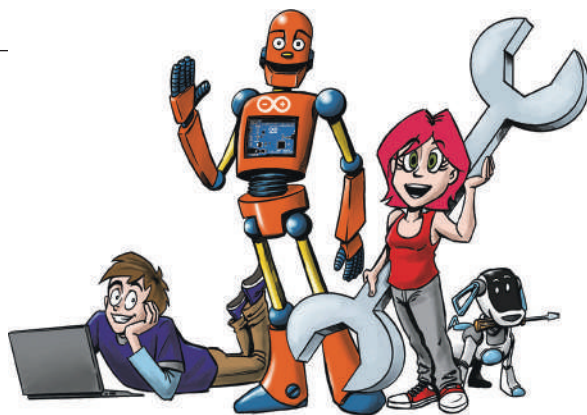
Установка..... 157

## Б

**Ответы..... 159**

## В

**Список материалов..... 165****Предметный указатель.....167**



## Введение

Ты уже давно мечтаешь начать программировать или разобраться, из каких деталей состоит компьютер? После прочтения этой книги ты вряд ли сможешь самостоятельно собрать компьютер, но некоторые вещи будут тебе по силам.

Во введении рассказывается:

- ⊙ что такое микросхема, микроконтроллер и Arduino;
- ⊙ как написать программу;
- ⊙ какие материалы тебе понадобятся для этой книги.

В процессе чтения ты легко освоишь основы электроники!

## Что такое микросхема?

Наверняка у тебя уже есть представление о том, что из себя представляет микросхема. По-английски их называют микрочипами (*microchip*) или просто чипами (*chips*), и я дальше иногда тоже буду их так называть. Обычно это маленький черный квадратик, расположенный на плате, например на системной плате компьютера. Микросхема, которую ты будешь программировать, выглядит немного иначе. Она прямоугольная, а не квадратная, и у нее намного меньше выводов (металлических ножек по краям чипа). Кроме того, эти выводы крупнее, чем у обычных микросхем, которые ты мог встречать раньше.

## Что такое микроконтроллер?

Микроконтроллер – это микросхема, которая уже содержит в себе все необходимые элементы (комплектующие). Если

провести аналогию, то системная плата компьютера – это микроконтроллер, а оперативная память на ней – одно из комплектующих. Точно так же оперативная память содержится внутри нашего микроконтроллера. Микроконтроллеры часто называют *computer-on-chip* – «однокристальный компьютер» по-русски.

## Что такое Arduino?

Чтобы тебе было легче освоить программирование микроконтроллера, существует так называемый проект Arduino. В нем есть готовые платы с микроконтроллером и собственным программным обеспечением для создания программ. Как ясно из заголовка книги, здесь пойдет речь о программировании микросхем на плате Arduino.

Проект Arduino предоставляет готовую плату с подходящей средой разработки программ на персональном компьютере (ПК). Раньше разработчикам приходилось самим мастерить платы, чтобы иметь возможность научиться их программировать.

## Как пишутся программы?

К сожалению, программирование осуществляется не голосом, а с помощью текста, набираемого на компьютере. Этот текст пишется не на русском языке, а с помощью специальных знаков и нескольких английских слов. Но пусть это тебя не пугает. Ты будешь учиться программировать с помощью довольно сложного языка программирования C++. Чтобы новичкам было легче им овладеть, создатели Arduino разработали упрощенный *диалект* (то есть вариант) этого языка программирования. C++ основан на нескольких словах и множестве знаков (символов), которые выглядят очень загадочно. В скобках указывается, что означает такой знак, как, например, ++ (приращение) или % (деление по модулю).

Со временем ты запомнишь, что значит каждый из этих символов, и сможешь безошибочно их использовать. В программном коде ниже показано, что можно сделать с помощью C++. Я написал эту программу для террариумного регулятора температуры, создание которого находится пока на начальной стадии:

```
#include "dimmen.h"  
#include "kern_temperatur.h"  
#include "terra_temperatur_class.h"
```

```
#include "class_cool.h"
void setup() {
  pinMode(13, OUTPUT);  pinMode(12,OUTPUT);
  Serial.begin(9600);  ADMUX = 0xC8;    delay(10);
}

void loop() {
  Cooler cooler(13); delay(100); bool hot = kern_temp(17);
  if (hot) cooler.start();
  else{
    cooler.stop();
  }
  delay(500); }
```

Также для программирования тебе понадобится программное обеспечение для ПК, которое можно найти на сайте [arduino.cc](http://arduino.cc). Подробную информацию по установке смотри в приложении А или в следующей главе.

Рассмотрим теперь устройство, которое нам необходимо для этой книги. Наверно, перед тобой сейчас лежит Arduino Uno – самая простая и удобная для изучения плата Arduino. Эта плата подключается к компьютеру через кабель USB. При составлении схемы плата должна быть отключена от кабеля USB, подключать кабель можно только тогда, когда схема готова. Пока мы не будем плату программировать, лишь использовать в качестве источника питания для схем.

## Материалы

Чтобы продолжить, нам потребуются некоторые материалы. Это светодиод (деталь, которая излучает свет), разноцветные проводники-перемычки (черный и красный проводники обычно применяются для подключения питания), плата Arduino, резистор и так называемая *макетная плата*, на которой это все соединяется.

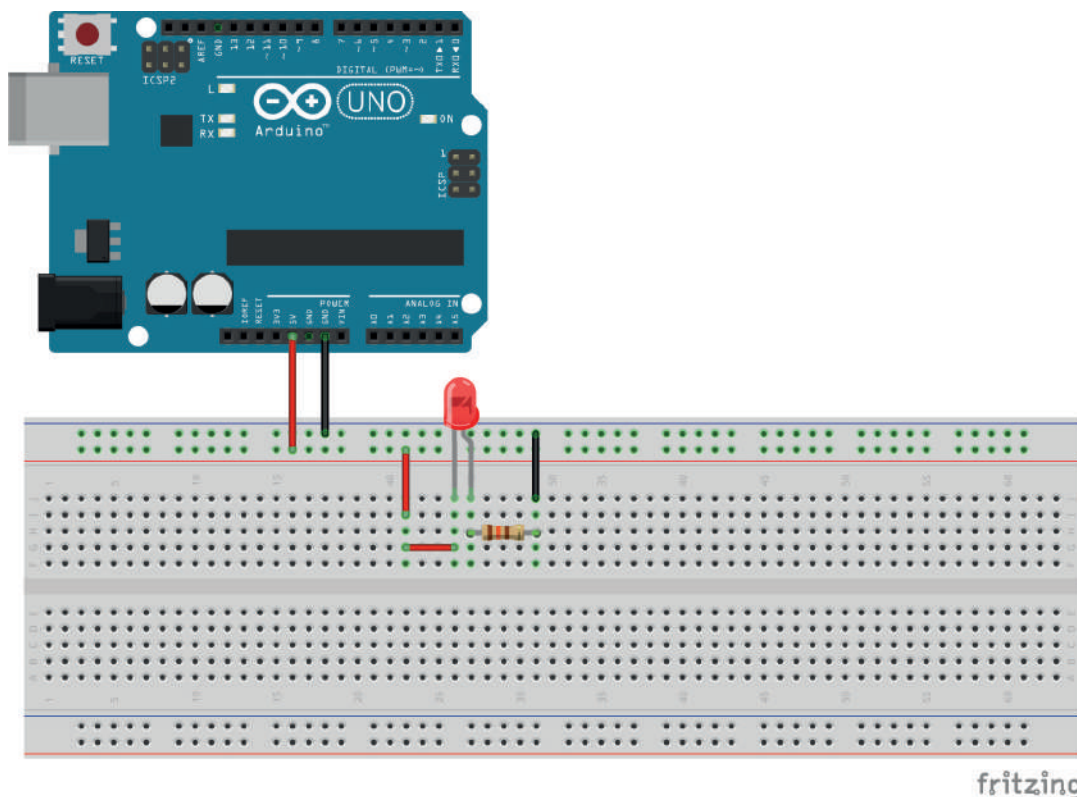
Все необходимые материалы перечислены в приложении Б в конце этой книги.

## Начало работы

Попробуем выполнить несложную задачу с Arduino. Предлагаю сначала просто зажечь светодиод. Это можно сделать двумя способами: используя в качестве источника тока либо Arduino, либо батарею. Мы будем пользоваться

первым способом, применяя, как говорили, здесь плату Arduino только как источник питания.

Для знакомства со светодиодом построй электрическую схему по первому рисунку в этой книге. При этом обрати внимание на следующее: красный проводник подключается к выводу Arduino с надписью VCC, а черный – к выводу с надписью GND, то есть к плюсу и минусу питания. И наконец: красный проводник (положительный полюс) присоединяется к длинному выводу светодиода, а черный (отрицательный полюс) – к резистору и, через него, к короткому выводу светодиода.



Таким образом, у тебя есть уже четыре важные детали для схемы подключения светодиода, которая будет часто встречаться в книге: Arduino, макетная плата, сам светодиод с резистором и, наконец, соединительные проводники (перемычки). И хотя последняя деталь самая простая, без нее ты не сможешь построить ни одну схему. Только при пайке плат можно обойтись без перемычек, но и здесь есть исключения, когда они бывают необходимы.

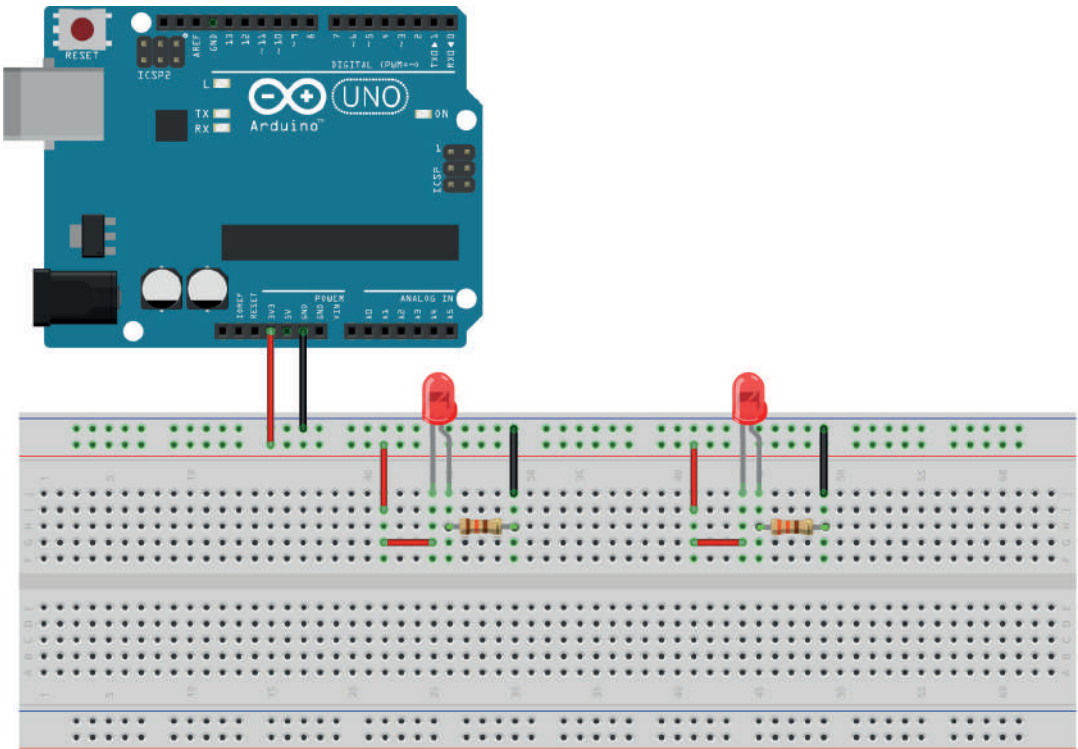
## Резистор

К сожалению, подключать светодиоды напрямую к источнику питания, подобно лампочке, нельзя. Из-за большой силы тока они бы просто сгорели. Поэтому необходима еще одна небольшая деталь – *резистор*. Он служит для того, чтобы ослабить ток и предотвратить перегрев чувствительного компонента. Резистор выглядит как очень маленькая трубка с длинными блестящими выводами по концам, на которую нанесено несколько цветных полосок. От многих других компонентов резисторы отличаются тем, что не важно, в каком направлении они подключены относительно плюса и минуса источника питания.

Резисторы стоят относительно недорого (около 2–3 рублей за штуку), так что на 100 рублей у тебя выйдет не менее 30–50 штук. Они бывают разной величины сопротивления (номинала), которое измеряется в омах.

В примере, который мы собрали выше, необходим резистор сопротивлением 130 Ом. Оставим один светодиод подключенным, как было ранее, а второй подключим через резистор с другим сопротивлением – 330 Ом.





fritzing

В этом примере левый светодиод горит ярче, чем правый, то есть чем меньше сопротивление резистора, тем яркость светодиода больше. Кстати, не важно, с какой стороны к светодиоду подключается резистор – их можно было бы включить между длинным концом и красным проводом.

А теперь решающий вопрос: как определить значение сопротивления резистора, не используя измерительные приборы? Если присмотреться, можно заметить у каждого резистора несколько разноцветных колец, которые образуют свой цветовой код. С помощью таблицы ниже можно определить цветной код для нужной величины сопротивления резистора.



Если ты правильно держишь резистор, золотое или серебряное кольцо всегда должно располагаться справа.

Цвет	1-е кольцо	2-е кольцо	3-е кольцо (множитель)	4-е кольцо (крайнее справа – допуск)
Серебряное			0,01	10%
Золотое			0,1	5%
Черное	–	0	1	
Коричневое	1	1	10	
Красное	2	2	100	
Оранжевое	3	3	1 К = 1000	
Желтое	4	4	10К	
Зеленое	5	5	100К	
Синее	6	6	1М	
Фиолетовое	7	7	10М	
Серое	8	8	100М	
Белое	9	9	1Г	

Везде, где в таблице стоит буква «К», подразумеваются три нуля. Потому что К значит «кило», то есть тысяча (1К соответствует 1000, 10К соответствуют 10 000). М(ега) и Г(ига) значат 1 000 000 и 1 000 000 000.

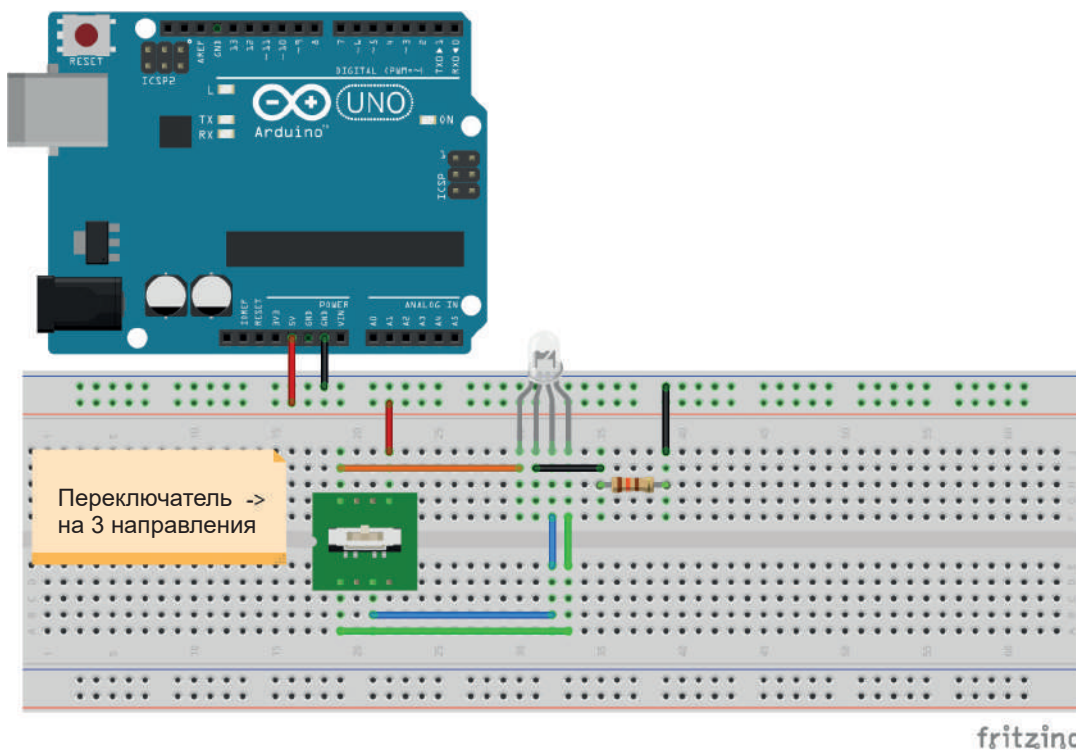
Золотое или серебряное кольцо обозначает точность (допуск). Допуск показывает, на сколько может отклоняться значение сопротивления. Чаще всего встречаются резисторы с допуском 5%, у которого кольцо золотого цвета.

Расчет примера:

Если нам нужен резистор на 130 Ом, у него будут следующие цвета: 1-е кольцо = коричневый, 2-е кольцо = оранжевый и 3-е кольцо = коричневый, то есть 13 (1-е и 2-е кольца) × 10 (множитель) равно (=) 130 (Ом).

## Схема с разноцветным светодиодом

Построй следующую схему.



Это схема со светодиодом, который может гореть тремя разными цветами (как правило, красным, зеленым и синим). В этой схеме тебе понадобится резистор на 130 Ом, так как мы подключаем схему к Arduino в качестве источника питания (вывод VCC  $\Rightarrow$  красный, вывод GND  $\Rightarrow$  черный). Если двигать ползунок размещенного на схеме переключателя на три направления, то светодиод будет последовательно загораться одним из трех цветов.

Здесь есть один общий отрицательный полюс и три положительных полюса для каждого цвета светодиода.

## Заключение

Теперь ты знаешь...

- как подключать светодиоды и что такое резистор;
- как подключать разноцветные светодиоды.

И ты знаешь...

- как сделать так, чтобы светодиод сгорел.

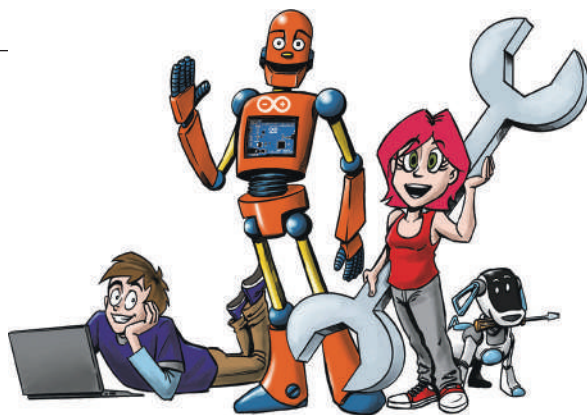
## Несколько заданий

Чтобы закрепить знания, выполни следующие задания.

1. Подключи три светодиода в ряд (+ 3 резистора на 330 Ом каждый).
2. Если ты знаешь, что такое мостовая схема включения диодов (обычных выпрямительных), то используй ее, чтобы зажечь светодиод, независимо от того, как он подключен к полюсу.
3. Если ты не справился с предыдущим заданием, изучи мостовой выпрямитель.

Ты познакомился с введением к этой книге. В следующих главах мы будем строить и применять гораздо более интересные схемы. Вперед, к новым открытиям!





# 1

## Мигай, мигай, огонек

В этой главе ты узнаешь, как запрограммировать Arduino. Кроме того, ты научишься использовать кнопки и узнаешь о возможностях их применения вместе с Arduino.

Мы разберем следующие вопросы:

- ⊙ установка программного обеспечения, подключение Arduino;
- ⊙ как включить и выключить светодиоды; как остановить Arduino на время;
- ⊙ чтение состояния кнопки (замкнута или разомкнута);
- ⊙ как составить план собственного проекта.

В конце главы мы спланируем и запрограммируем небольшой проект: мигающая гирлянда с различными функциями. Во второй главе мы продолжим работу над гирляндой и сможем управлять ей и менять ее функции через компьютер.

## Установка программного обеспечения

Установка программного обеспечения не должна вызывать у тебя никаких затруднений. Сначала нужно скачать

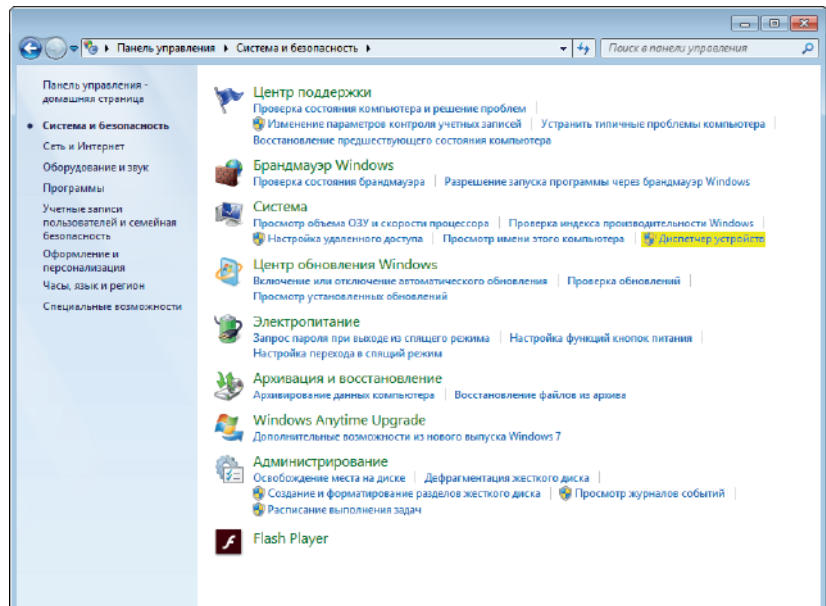


## 1

программное обеспечение здесь: <http://arduino.cc/en/Main/Software>. На момент написания этой книги актуальная версия 1.8.5.

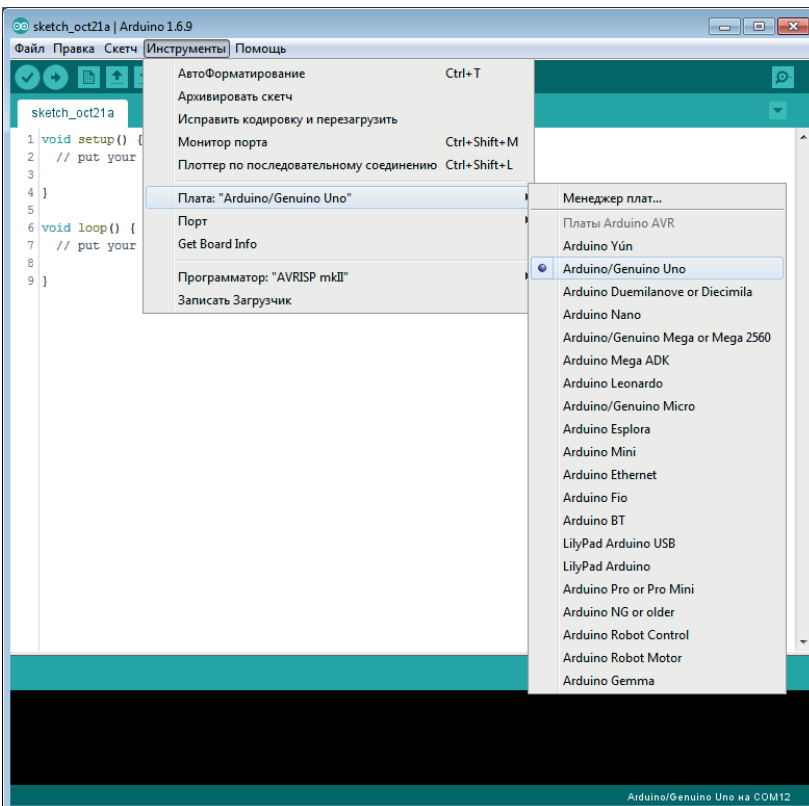
Для тех, кто пользуется Windows, на сайте имеется выбор: самоустанавливающийся EXE-архив или обычный архив в формате ZIP. Мы остановимся на втором варианте. Скачанный архив распакуй с помощью любого архиватора (например, Winrar или 7Zip, можно использовать и собственные возможности Windows). Содержащуюся там папку под названием **arduino** (маленькими буквами с добавлением номера версии) можно сохранить, например, на рабочем столе. В папке находится так называемая IDE, программа для создания исходного кода и его преобразования в «машинный язык». По сути, IDE представляет собой обычную программу для работы с текстом (текстовый редактор), которая различным цветом помечает команды программирования.

Теперь перейдем к самому сложному этапу – установке драйвера. Для операционной системы Windows нужно установить драйвер, который позволит опознавать подключенную плату Arduino и ее программировать. Подключи к USB-кабелю Arduino Uno и подожди. Через некоторое время должно появиться окошко, в котором предложат найти или обновить драйверы. Пройгнорируй его и открой панель управления. Затем нажми на раздел **Система и безопасность**. Там выбери пункт меню **Система | Диспетчер устройств**.



В открывшемся окне нужно выбрать пункт **Порты (COM и LPT)**. Теперь там должно отображаться название платы Arduino.

Нажатием правой кнопки мыши на этом названии выбери **Обновить драйвер**, а в открывшемся окне – **Выполнить поиск драйверов на этом компьютере**. Теперь тебе нужно найти папку на рабочем столе, в которую ты распаковал IDE (в дальнейшем мы будем называть ее просто **Arduino**, хотя обычно она еще дополняется номером версии IDE). В подпапке **Drivers (... \Arduino \Drivers)** ты найдешь файл с названием *Arduino.inf*. Выбери его, и установка драйвера завершена. (Но честно, быстрее можно установить всю операционную систему Linux, чем один драйвер для Windows.) Итак, драйвер установлен. Теперь ты можешь через USB-кабель подключить Arduino к компьютеру и приступить к программированию. Но сначала нам предстоит настроить в IDE нужную плату Arduino. Если ты используешь рекомендуемую здесь Arduino Uno, выбери ее в меню **Инструменты | Плата...**, см. рисунок:



## 1

Затем необходимо определить правильный порт: отключи плату Arduino Uno (если ты ее уже подключил), посмотри порты в **Инструменты | Последовательный порт...** и запиши их. Затем подключи Arduino Uno заново. Выбери порт, который появился. Теперь можно начать программировать.

## Наша первая программа

Наша первая программа поможет проверить, правильно ли подключен Arduino.

```
void setup() {}  
void loop() {}
```

Этот текст (исходный код) тебе нужно ввести в IDE и потом кликнуть на значок со стрелкой вправо в верхнем левом углу, под строкой меню. Если в строке над черным полем в нижней части IDE после всех преобразований появится надпись «Загрузка завершена», то Arduino настроен правильно, в противном случае фон окрасится оранжевым цветом с надписью «Проблема загрузки в плату...». Очень часто такая ошибка возникает, когда настроено все правильно, но плата или порт в IDE выбраны неверно, а также тогда, когда плату отключили от USB на время составления схемы и забыли включить обратно. В случае если все подключено, но ошибка все-таки возникает, повтори инструкции выше или см. приложение А.

Программа для Arduino, так называемый *скетч* (sketch), всегда состоит, как минимум, из двух частей: «setup()» и «loop()». Код, который записан в части setup, выполняется один раз при запуске или при сбросе контроллера. Код в части loop, напротив, выполняется в бесконечном цикле. Исходный код, который ты пишешь сам после заголовков «setup()» и «loop()», заключается в фигурные скобки. Почему это так, ты узнаешь в следующей главе в разделе «Как работают функции».

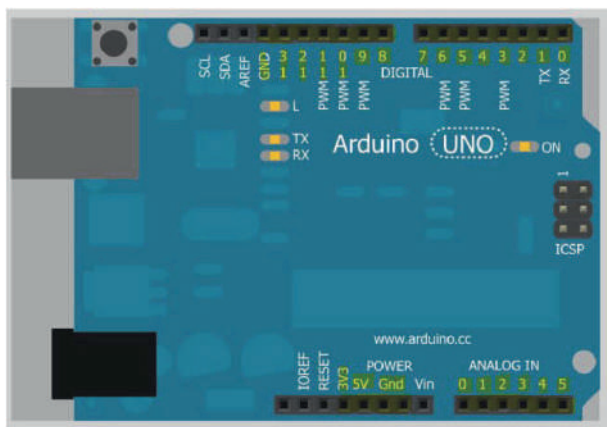
Наша первая программа должна управлять светодиодом и включать его, а позднее он будет мигать.

```
void setup() {  
  pinMode(13,OUTPUT); //Пин 13 на выход  
  digitalWrite(13,HIGH); //Пин 13 включить  
}  
void loop() {}
```

Все, что стоит за двумя косыми линиями, в программе всегда расценивается как комментарий, то есть написанный за ними текст не имеет функции, а служит только для того, чтобы код было проще читать.



В Arduino все выводы микросхемы контроллера выведены на небольшие металлические контакты в черных гнездах по бокам платы Arduino Uno (на рисунке ниже отмечены желтым цветом). У микросхем выводы часто называют английским словом «пины», потому что там они действительно похожи на иголочки (*pin* – иголка). Выводы платы Arduino по традиции тоже так иногда называют, и, встретив его в этой книге, помните, что *пин* означает то же самое, что просто *вывод*. Каждым выводом-пином платы Arduino программа может управлять по отдельности.



Рядом с каждым выводом на плате написан номер, по которому ты можешь его настроить. В примере я использую пин 13, потому что на нем уже установлен светодиод на плате. Этот код тебе необходимо загрузить в Arduino, как показано в примере выше, и светодиод сразу загорится зеленым, красным или желтым цветом (цвет зависит от изготовителя платы).

В коде выше ты можешь увидеть, что команда `pinMode` является *функцией*. В скобках указываются *параметры* функции – сведения, которые необходимы для ее работы. Здесь первым параметром является номер вывода, вторым – состояние. Это может быть OUTPUT (выход) или INPUT (вход). Состояние «вход» нам понадобится далее в разделе «Чтение входов», а пока будем работать с состоянием «выход».

## 1

```
void setup() {
  pinMode(13,OUTPUT); //Пин 13 на выходе
  digitalWrite(13,HIGH); //Пин 13 включить
}
void loop() {}
```

Выходы всегда что-то переключают, например светодиоды, входы всегда что-то читают, например состояние кнопки (нажата или не нажата). Функция всегда заканчивается точкой с запятой (;), чтобы транслятор знал, где конец команды. Транслятор (что по-английски означает «переводчик», иначе его называют компилятором) переводит (транслирует или компилирует) исходный код на машинный язык, то есть в двоичную систему счисления, чтобы его понял микроконтроллер. (Двоичная система – это единицы и нули, с которыми работает компьютер и о которых ты, вероятно, уже слышал.)

Команда `pinMode()` необходима, чтобы Arduino знал, как использовать вывод – на выход или на вход. Вторая команда, `digitalWrite()`, включает состояние вывода. В `digitalWrite()` во втором параметре ты можешь указать либо `HIGH` («высокий уровень», близкий к питанию 5 вольт), тогда пин будет выдавать ток, либо `LOW` («низкий уровень», близкий к нулю), тогда пин будет принимать ток от внешнего источника.

Ты, наверное, уже заметил, что в информатике используется много английских слов. Английский язык играет важную роль в программировании. И специальные термины Arduino, например `HIGH` (здесь в значении «высокий уровень»), и большинство языков программирования, а также технических описаний компонентов схем «составлено» на английском. Поэтому занятия английским для программиста и электронщика будут совсем не лишними.

## Как сделать так, чтобы светодиод мигал?

Ты можешь записать в цикл `loop` функцию `digitalWrite()` с чередующимися `HIGH` и `LOW`, но это не будет работать. Вот попробуй с этим кодом:

```
void setup() {
  pinMode(13,OUTPUT); //Только один раз, поскольку мы хотим
                      //использовать пин 13 только как выход
}
void loop() {
  digitalWrite(13,HIGH);
  digitalWrite(13,LOW);
}
```

Сам по себе код должен работать, и он работает! Но поскольку контроллер функционирует с частотой 16 миллионов герц (герц означает одно колебание в секунду; если команда длится один такт, вывод переключается 16 миллионов раз в секунду, соответственно, состояние вывода меняется каждые 0,0000000625 секунды), человеческий глаз не способен увидеть реакцию светодиода. Необходимо немного увеличить интервал. Для этого используется команда `delay()` (*delay* – задержка). Эта команда приостанавливает контроллер на заданное количество миллисекунд (1000 мс = 1 секунда). Чтобы светодиод менял состояние один раз в секунду, нам понадобится следующий код:

```
void setup() { pinMode(13,OUTPUT);
}
void loop() {
digitalWrite(13,HIGH); //Включить светодиод
delay(1000); //Подождать 1 секунду
digitalWrite(13,LOW); //Выключить светодиод
delay(1000); //И только через 1 с снова включить
                //и цикл начинается сначала
}
```

Итак, твой первый проект – мигающий светодиод – выполнен. Можешь сохранить его через **Файл | Сохранить...** под любым именем. Arduino IDE сохраняет все файлы проектов с исходным кодом в одном месте, так что ты сможешь использовать их позже.

## Что такое переменные и для чего они нужны

Итак, у тебя есть первый «работающий» код, который обеспечивает цикличное управление выводами. Наверняка тебе интересно, как можно упорядочить, например, 50 выводов (в Arduino Mega их более 50 штук)? В Arduino есть возможность дать каждому выводу свое имя. Для этого есть разные способы, здесь мы будем присваивать имена с помощью *переменных*. Содержание переменной может меняться, отсюда и название. Ее противоположность – *константа*, которая никогда не меняется. В переменных можно установить номера выводов, которыми мы хотим управлять. Для этого нужна числовая переменная, так называемая *целочисленная переменная* (*integer*, *int*). Предыдущий пример с переменными выглядит так:

```
int led = 13; //Определить переменные
int pause = 1000;
void setup() {pinMode(led,OUTPUT);
```



## 1

```

}
void loop() {
digitalWrite(led,HIGH); //Пин, который установлен для светодиода,
                        //переключается
delay(pause); //Подождать количество мс, которое установлено в паузе
digitalWrite(led,LOW);
delay(pause);
}

```

Если компилятор встречает выражение вида `тип_переменной имя_переменной = показатель;`, переменная инициализируется. В таблице приведены типы переменных, которые мы будем использовать.

Тип переменной	Содержание	Пример
int	Целое число от -32 768 до 32 767	9
float	Число с плавающей запятой	9.4
byte	Целое число от 0 до 255	50
char	Буква (символ)	'e'
bool (или boolean)	Булево значение, другими словами – «правда» (true) или «ложь» (false)	true/false или 1/0
long	Целое число от -2 147 483 648 до 2 147 483 647	262000

Важно, что имена переменных не могут начинаться с цифры и не содержат пробела (вместо него можно использовать символ подчеркивания, например `langer_variablename`). Помни, что переменная во всем тексте программы пишется одинаково, с учетом прописных и строчных букв (`led`, `LED` и `Led` – это три разные переменные!). Буквы в именах употребляются только из английского алфавита, из других языков не допускается.

Для отдельных переменных необходимо помнить следующие моменты. Переменные всегда имеют определенный диапазон значений, который нельзя превышать (например, от 0 до 255 у типа `byte`). Числа с плавающей запятой (в школе их называют вещественными, или действительными, числами) должны быть написаны с разделительной точкой (.) перед дробной частью, поскольку так принято в Америке (например, 3,14 должно выглядеть как 3.14). В переменной `char` буква всегда должна стоять в верхних одинарных кавычках ('A'), а в `bool` ты можешь использовать только указанные в таблице значения `true` или `false`. В следующей главе ты узнаешь, для чего используется `bool` (`boolean`).

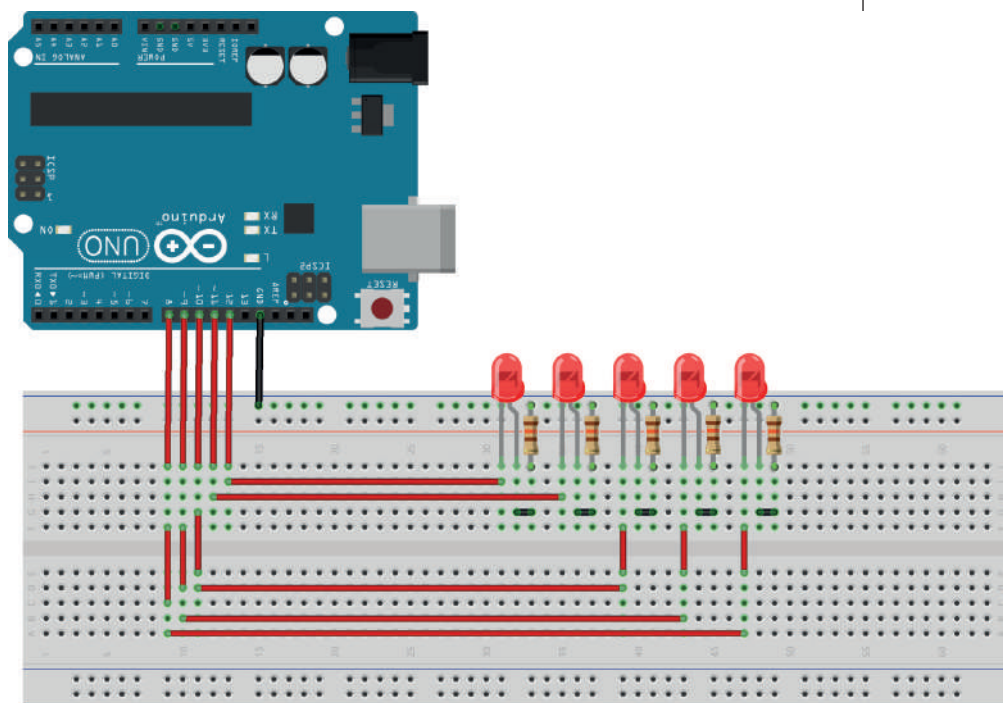
Как упорядочить переменные? Каждый микроконтроллер содержит определенное количество RAM-памяти, то есть оперативной памяти



микросхемы. Ее можно разделить на небольшие единицы-ячейки. В зависимости от размера переменной резервируется определенное количество ячеек памяти, и в них записывается конкретное число. Поэтому необходимо всегда иметь достаточный объем свободной оперативной памяти, потому что в противном случае программа может сработать неправильно. Небольшое утешение: все представленные в книге программы без проблем работают на Arduino Uno. Некоторые скетчи из-за собственных расширений могут заполнить всю память, но на это будет отдельное указание в тексте.

## Наша вторая программа: гирлянда со светодиодами

Теперь давай сделаем с помощью переменных кое-что полезное, а именно небольшую гирлянду со светодиодами. Для этого необходимо минимум пять светодиодов, чтобы можно было что-то заметить. Я предлагаю использовать выводы с 12-го по 8-й. Здесь решающее значение имеет соединение перемычками. Если у тебя макетная плата, можно сделать так:



## 1

Все это довольно просто: ты соединяешь вывод 8 с крайним слева светодиодом и поочередно следующий вывод со следующим светодиодом. Таким образом, у нас уже готово соединение, и сейчас мы проведем следующий эксперимент. Сначала давай разработаем цикл, который будет заставлять мигать один светодиод и затем переходить к следующему и т. д. В исходном коде для этого нет ничего нового, и он относительно прост:

```
int pin1 = 8;
int pin2 = 9;
int pin3 = 10;
int pin4 = 11;
int pin5 = 12;

void setup() {
  pinMode(pin1,OUTPUT)
  pinMode(pin2,OUTPUT)
  pinMode(pin3,OUTPUT)
  pinMode(pin4,OUTPUT)
  pinMode(pin5,OUTPUT) }
void loop() {
  //Заставь Pin 1 мигать
  digitalWrite(pin1,HIGH);
  delay(100);
  digitalWrite(pin1,LOW);
  delay(100);
  //Pin 2
  digitalWrite(pin2,HIGH);
  delay(100);
  digitalWrite(pin2,LOW);
  delay(100);
  //Pin 3
  digitalWrite(pin3,HIGH);
  delay(100);
  digitalWrite(pin3,LOW);
  delay(100);
  //Pin 4
  digitalWrite(pin4,HIGH);
  delay(100);
  digitalWrite(pin4,LOW);
  delay(100);
  //Pin 5
  digitalWrite(pin5,HIGH);
  delay(100);
  digitalWrite(pin5,LOW);
  delay(100);
}
```

## Как работают функции

Вот видишь, ничего нового, но, пожалуй, это слишком длинный код для такого простого эффекта. Как насчет того, чтобы его укоротить? Это можно сделать с помощью функций. Термин уже упоминался выше, а теперь мы самостоятельно напишем функцию. В нашем случае функция – это просто выполнение нескольких команд (по сути, тоже функций, например `digitalWrite()`), собранных в одном вызове. С помощью функций можно упорядочить код (кстати, `setup()` и `loop()` – тоже функции). Такой метод – техника так называемого *структурного программирования*. Также сюда относятся циклы и переходы на метку, причем в этой книге я переходы не использую, потому что на сегодняшний день они являются устаревшими и вредят коду (такой код еще называют «спагетти-код» – без шуток, это реальный термин!).

Вернемся к функциям. Рассмотрим одну функцию, которую ты уже знаешь: `void digitalWrite(int,int)`. Так можно коротко обобщить функции, их названия, а также типы передаваемых в нее параметров (в нашем случае два целых числа типа `int`). Перед функцией идет тип возвращаемого значения, в нашем случае `void` (в переводе «пустота» – функция ничего не возвращает, но указать это все равно требуется). Таким образом, функцию можно переписать так:

```
Тип_возвращаемого значения Название_функции(параметр){код_функции;}
```

Вот функция для сокращения нашего длинного кода:

```
void blink(int pin,int msec) {  
digitalWrite(pin,HIGH);  
delay(msek);  
digitalWrite(pin,LOW);  
delay(msek);  
}
```

Это наша первая функция. В круглых скобках указываются параметры, то есть какой вывод должен быть включен и как долго. В фигурных скобках идет код, который относится к функции.

После названия функции и фигурных скобок точка с запятой не ставится.



## 1

Будет очень интересно, если загрузить весь код в Arduino. Вот еще раз обзор текущего скетча, теперь с применением нашей функции:

```
void blink(int pin,int msec)
{ digitalWrite(pin,HIGH);
  delay(msec);
  digitalWrite(pin,LOW);
  delay(msec);
}

void setup() {
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
  pinMode(11,OUTPUT);
  pinMode(12,OUTPUT);
}

void loop() {
  blink(8,1000); //Продолжительность в миллисекундах
  blink(9,1000);
  blink(10,1000);
  blink(11,1000);
  blink(12,1000);
}
```

Исходный файл теперь значительно короче, но делает то же самое; здорово, правда? Определения функции всегда должны стоять за пределами другой функции, чтобы не было ошибки. Мы плавно подошли к довольно сложному, но очень важному вопросу: так называемые *области действия*. Предположим, ты определяешь переменную внутри функции `blink()`. Можно ли ее использовать в других функциях? К сожалению, нет. Это связано с областями действия. Вот небольшой исходный код для наглядности:

```
int dateiebene;
void setup() {
  int funktionsebene;
}
```

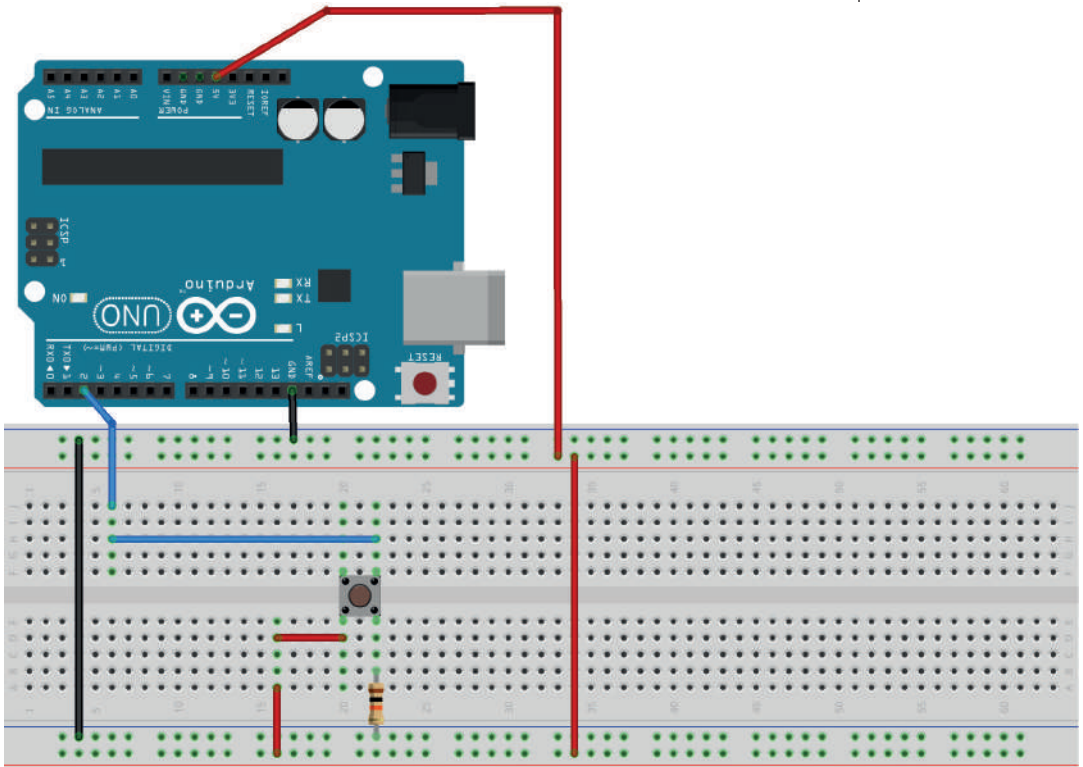
Переменные, которые были определены на уровне файла программы (`dateiebene`), можно вызвать где угодно. Переменные, которые были определены на уровне функции (`funktionsebene`), можно вызвать только в соответствующей функции. Это скучная теория, но полезно ее знать, иначе

можно вместо загрузки в Arduino все время получать сообщения об ошибках.

Теперь перейдем к теме чтения состояния выводов, в которой меньше теории и больше практики.

## Чтение входов

Чтение выводов, которые подключены на вход (INPUT), – это важная тема, которая понадобится тебе также в остальной части этой книги. Возьмем такую ситуацию: ты хочешь, чтобы светодиод мигал только тогда, когда нажимается кнопка. Для этого нам понадобится функция, противоположная `digitalWrite()`, а именно `digitalRead()` (по-английски *write* – писать, а *read* – читать). Прежде чем писать исходный код, сначала построим следующую схему:



Построй ее именно так, как показано на рисунке. Важно присоединить кнопку к проводникам питания так, как нарисовано, иначе может произойти короткое замыкание.

## 1

Теперь нужно подключить резистор со значением сопротивления 10 кОм.

Если все работает, начнем набирать исходный код. Как обычно, сначала скетч, а потом пояснение.

```
int ledpin = 13;
int buttonpin = 2;
int buttonstate;

void setup() {
  pinMode(ledpin,OUTPUT); //Output, так как светодиод включен
  pinMode(buttonpin,INPUT); //Input, чтобы мы могли это прочесть
}

void loop() {
  buttonstate = digitalRead(buttonpin);
  if (buttonstate == HIGH) {
    digitalWrite(ledpin,HIGH);
  } else {      digitalWrite(ledpin,LOW);      }
}
```

## If и else: если не получается это, сделай вот это

Здесь ты видишь сразу несколько новых приемов программирования и функций: запрос if-else, а также установка и чтение состояния вывода.

Сначала расширение функции pinMode(): указав слово INPUT (ввод), мы сигнализируем, что вывод внутри (в микросхеме) нужно читать (*read*), а не писать в него (*write*), как при включении светодиода. Затем идет числовая переменная buttonstate; мы назначаем ей в строке loop() не конкретное значение, а то, которое будет получено в ответ от функции digitalRead(), то есть высокий (HIGH = 1) или низкий (LOW = 0) уровень тока на выводе. Уже в данной главе ты научишься программировать это самостоятельно.

Следующая строка будет посложнее, потому что мы подошли к выражению if. Чтобы тебе было более понятно, я запишу эту часть кода в виде псевдокода (*псевдокод* – запись содержания функции словами, не программирование).

```
если нажата кнопка
-> тогда включить светодиод,
если нет
-> выключить светодиод
```



В фигурных скобках после имени функции указывается содержание запроса. Компьютер (Arduino) знает, что код в фигурных скобках после команды `if` означает команду, которая должна быть выполнена, если нажата кнопка.

Мы подошли к выражению `if`; оно имеет приблизительно такую структуру:

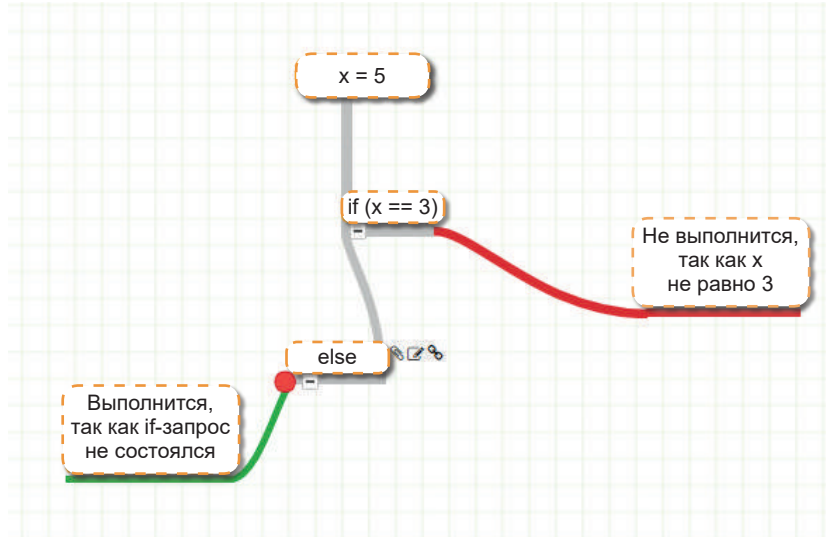
```
if (условие) {  
  //Тогда выполни код, который здесь стоит  
}
```

Условием при этом может быть сравнение переменной и константы, как в нашем примере, где указаны переменная `buttonstate` и константа `HIGH`. Важно, что здесь указываются два знака равенства (`==`), так Arduino понимает, что это сравнение (`buttonstate == HIGH`), а не назначение (`int xy = 9`). Конечно, ты можешь проверить не только на равенство, но и на другие операторы сравнения (в таблице предполагается, что `x = 5`):

Оператор	Пример	Результат
<code>==</code> (равно)	<code>x == 5</code>	Правильно
<code>!=</code> (не равно)	<code>x != 5</code>	Неправильно, потому что <code>x = 5</code>
<code>&gt;</code> (больше)	<code>x &gt; 3</code>	Правильно, потому что <code>x = 5</code> , то есть больше, чем 3
<code>&lt;</code> (меньше, чем)	<code>x &lt; 10</code>	Правильно, потому что <code>x = 5</code> , то есть меньше, чем 10
<code>&gt;=</code> (больше или равно)	<code>10 &gt;= x</code>	Правильно, потому что <code>x = 5</code> , то есть 10 больше
<code>&lt;=</code> (меньше или равно)	<code>x &lt;= 3</code>	Неправильно, потому что <code>x = 5</code> , то есть больше 3

Это может показаться сложным, но когда ты дочитаешь главу до конца, эта таблица станет для тебя более понятной. Теперь мы подошли к `else`: в переводе с английского *else* значит «в противном случае». Если выразить наш исходный код словами, он будет звучать так: если `buttonstate` равно `HIGH`, тогда включи светодиод, в противном случае выключи его. Более наглядно это показано на рисунке ниже, на так называемой *блок-схеме*.

## 1



Else-часть тоже заключается в фигурные скобки и может стоять только после выражения if (в другом месте она просто не будет иметь смысла). Теперь мы можем записать весь скетч в виде псевдокода:

Содержание строки loop:

- Прочти, поступает ли ток к выводу 11 (то есть нажата ли кнопка); если да, в переменной buttonstate окажется HIGH\*, в противном случае (то есть ток не поступает – кнопка не нажата) в переменной buttonstate окажется LOW\*.
- Теперь проверь, есть ли значение HIGH в переменной.
  - > Если там написано HIGH, включи светодиод.
  - > Если там написано что-то другое (else), выключи светодиод (в нашем случае там написано либо HIGH, либо LOW).
- Начни все сначала (цикл loop!).

\* Напоминаем, что HIGH и LOW внутри преобразовываются в целые числа 1 и 0, то есть это целочисленные переменные.

Теперь тебе должна быть понятна работа скетча. Но для закрепления материала выполни еще одно небольшое задание.

## Условия на практике

Напиши программу, которая отнимает  $y$  из  $x$  и сохраняет в переменной  $z$ . Светодиод должен включаться, когда  $z$  больше 10, в противном случае он не включается. Код можно записывать в части `setup`, тогда он будет работать один раз при запуске. Попробуй сначала сам разработать решение, а потом посмотри на предложенный мной вариант.

```
int x = 312; //Здесь всегда значения для задания x и y
int y = 9;
int z;

void setup() {
z = x - y; //Вычти y из x и сохрани в z
if (z > 10) { //Если z больше 10, то есть минимум 11
    digitalWrite(13,HIGH); //Включи светодиод на пин 13
    } else {
    digitalWrite(13,LOW); //В любом другом случае:
    //выключи светодиод (выражение else в этом случае можно
    //оставить пустым, поскольку светодиод
    //вначале все равно выключен)
    }
}

void loop(){ //Функция loop остается пустой,
            //поскольку код выполняется однократно
```

Совсем не сложно, правда?

Вот совет на случай, если ты забыл содержание функции (например, `digitalWrite()`). Выдели в исходном коде набранное оранжевым шрифтом название функции и выбери в контекстном меню (открывается нажатием правой кнопки) пункт «Найти в Справочнике». После этого в браузере откроется страница с коротким объяснением и примером кода с правильным применением. Если этот пункт в контекстном меню неактивен (отображается серым цветом), то это значит, что функция нестандартная и в Справочнике ее нет.



## 1

## Наша третья программа: аппарат Морзе

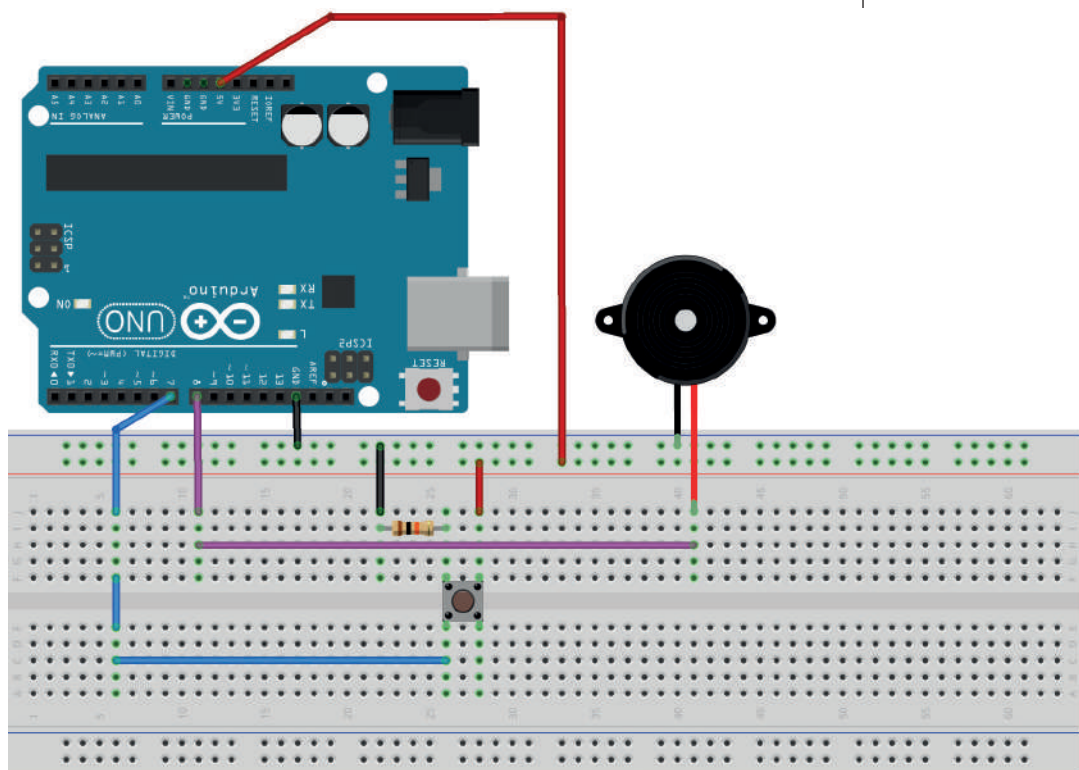
А сейчас мы разработаем кое-что, имеющее конкретную практическую значимость. Это небольшое устройство может пригодиться для передачи данных по телеграфному аппарату Морзе. Азбуку Морзе придумал Самюэл Морзе, чтобы передавать знаки по изобретенному им телеграфу. После некоторых усовершенствований (в результате совместной работы А. Вайла, Ф. Герке и др.) появилась используемая сегодня азбука. Если ты хочешь узнать больше об истории этого кода, можешь почитать статью в Википедии: [https://ru.wikipedia.org/wiki/Азбука\\_Морзе](https://ru.wikipedia.org/wiki/Азбука_Морзе).

Я предлагаю сделать ключ Морзе, который при нажатии на кнопку издает акустический сигнал. Он поможет в тренировке восприятия азбуки Морзе. Потом ты сможешь использовать эту процедуру в разработке собственных проектов.

### Подключить аппаратное обеспечение

Сначала подумаем, какие компоненты нам понадобятся. Наряду с Arduino, макетной платой и проводниками-перемычками (нашей стандартной комплектацией) для этого проекта нам нужны кнопка и средство для передачи сигналов Морзе. Последнее состоит из светодиодов, динамика, кабеля для подключения к последовательному порту (см. следующую главу). Я использую здесь динамик, просто чтобы показать тебе, что его легко можно заменить на светодиод. Поэтому нам вместо настоящего динамика понадобится небольшой маломощный пьезоизлучатель звука («пищалка», как говорят электронщики).

Если тебе сложно найти подходящий пьезоизлучатель, можешь просто использовать второй вариант со светодиодом. Сначала нам нужно собрать схему, чтобы у нас было с чем работать. При этом включение пьезоизлучателя осуществляется через Arduino, то есть нам нужно собрать две схемы (подключить к Arduino кнопку, а также пьезоизлучатель). Вот обещанная электрическая схема:



fritzing

Проводники расположены немного угловато, но принцип должен быть понятен. Сначала необходимо соединить токопроводящие шины (сплошные ряды контактов на краю макетной платы, которые помечены синей и красной линиями) с плюсом и минусом (на двух шинах), а затем с кнопкой. Возможно, тебе понадобятся еще перемычки в середине токопроводящих шин, поскольку там у некоторых макетных плат есть обрыв. Синий проводник от кнопки при этом идет на входной контакт 7 на плате Arduino.

## Программное обеспечение

И вот мы уже подошли к исходному коду, нашему скетчу. Ты можешь попробовать найти в интернете функцию, чтобы пьезоизлучатель издавал звуки (функция `tone()`), так ты будешь делать при создании собственного проекта. Но здесь можешь просто взять небольшой исходный код:

## 1

```

int buttonstate;
int note = 720; //Высота звука

void setup() {
  pinMode(7,INPUT);
}
void loop()
{

  buttonstate = digitalRead(7);
  if (buttonstate == HIGH) {
    tone(8,note); //Издаем звук
  } else {
    noTone(8); //В противном случае без звука :)
  }
}

```

В этом примере я исхожу из того, что пьезоизлучатель подключен к выводу 8, кнопка – к выводу 7. В переменной `int note` указана высота звука, которая должна подаваться на пьезоизлучатель (если точнее, это частота электрических колебаний на выводе 8). Ниже снова идет выражение `if` и единственная новая функция: `tone()`. У нее относительно простая структура. Первый параметр – номер вывода (у нас пин 8), а второй – частота (записана в переменной `note`).

Теперь мы можем заменить пьезоизлучатель на светодиод. Попробуй сначала сделать это самостоятельно. Это очень просто, но на всякий случай вот решение.

```

int buttonstate;
void setup() {
  pinMode(7,INPUT);
  pinMode(8,OUTPUT); //Светодиод на пин 8,
                    //схема с кнопкой остается такой же
}
void loop() {
  buttonstate = digitalRead(7);
  if (buttonstate == HIGH) {
    digitalWrite(8,HIGH);
  } else {
    digitalWrite(8,LOW);
  }
}

```

Это было легко. Тяжелее будет, если оформить код более компактно. Вот пример кода с пьезоизлучателем, который

сложен для восприятия новичками, и я не советую его использовать, поскольку это чревато ошибками.

```
void setup() {pinMode(7,INPUT);}
void loop()
{digitalRead(7) == 1 ? tone(8,720) : noTone(8);}
```

## Наша четвертая программа: игра «Горячий провод»

Теперь у тебя есть ключ Морзе в двух вариантах исполнения. Сейчас я предлагаю выполнить одно теоретическое задание, чтобы потом собрать игру.

Под теорией я имею в виду программирование функций, на этот раз с возвращаемым значением. Также ты познакомишься с функцией `switch()`. Это так называемая «функция-переключатель» (*switch* – «переключатель» по-русски), или, если совсем официально, «оператор множественного выбора». После этого мы приступим к большому проекту, а именно к новой модели игры «Горячий провод». Смысл игры в том, чтобы провести кольцо через изогнутый провод, не коснувшись при этом самого провода. Если провод задевается, звучит сигнал. Но мы дополним эту игру еще одним шагом: светодиоды будут показывать, сколько времени игрок затратил на прохождение маршрута. Вот список того, что нужно купить:

- неизолированная медная проволока (сними изоляцию с отрезка обычного провода диаметром 0,5–1 мм);
- деревянная подставка (для закрепления игры);
- клеевой пистолет;
- 3 светодиода (красный, желтый и зеленый);
- олово для пайки и паяльник;
- пьезоэлектрический громкоговоритель (посолиднее, чем «пищалка» в прошлом проекте; далее будем называть его просто «динамик»);
- и конечно, наш Arduino Uno.

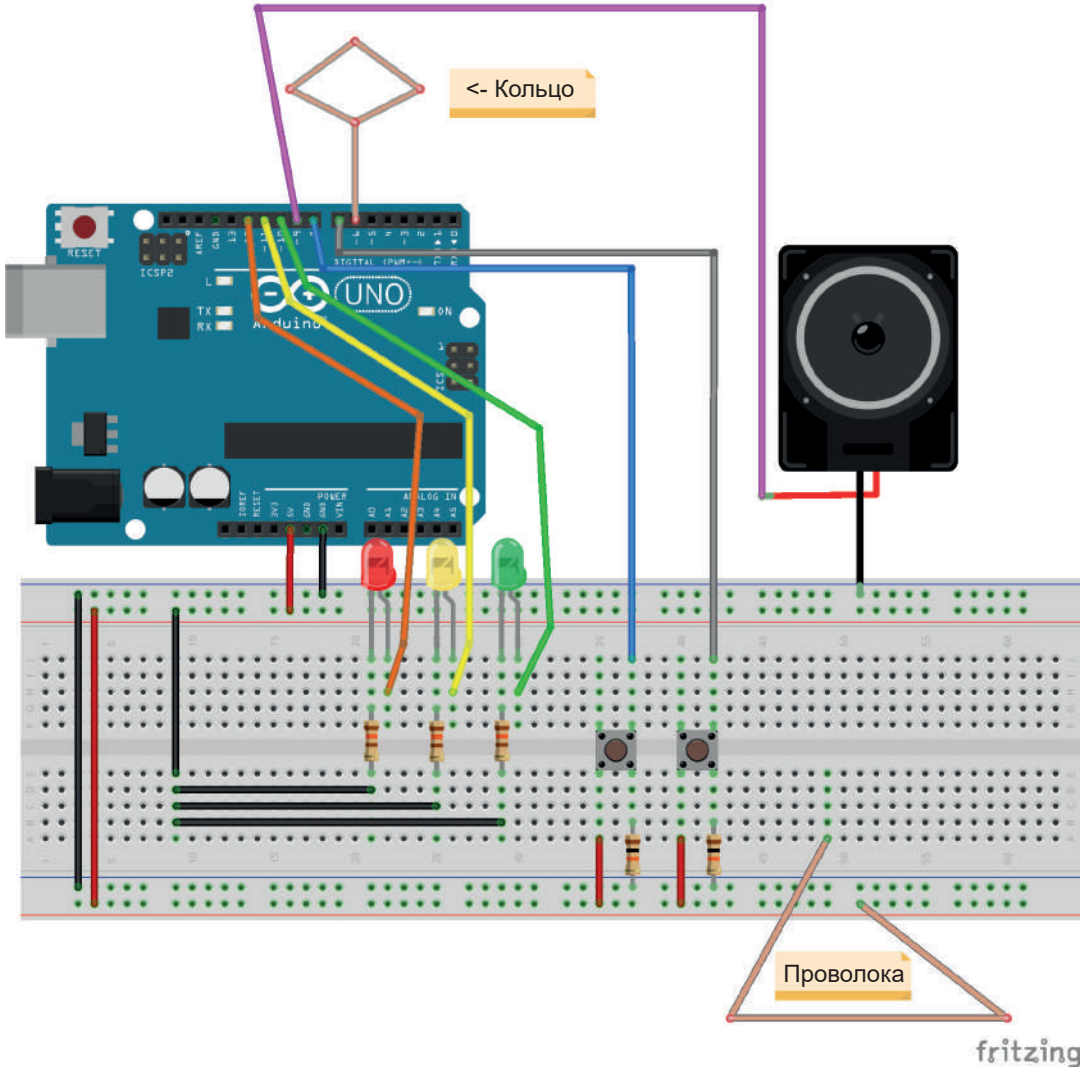
Перейдя по ссылке <http://fb.ru/article/253821/kak-nauchitsya-payat-poshagovaya-instruktsiya-osobennosti-i-rekomendatsii-professionalov>, ты найдешь хорошую инструкцию о том, как правильно паять.

Тебе потребуется довольно много времени, чтобы собрать всю конструкцию, но если ты это сделаешь, впоследствии



## 1

тебе будет легче выполнять такие задания. Для соединения: светодиоды подключаются к выводам 12, 11 и 10 (красный = 12, желтый = 11 и зеленый = 10). Динамик идет на вывод 9, две кнопки для старта и окончания – на выводы 8 и 7, «кольцо» из проволоки – на вывод 6, а еще кусок проволоки – на положительный полюс источника питания.



## Сначала переменные и функции

Вот список переменных, которые мы используем:

```
int red = 12;
int green = 10;
int yellow = 11;
int alert = 9;
int button_start = 8;
int button_end = 7;
int touch = 6;
//Здесь функции, которые мы будем писать
int play_game();
void show_result();
bool game_stop();
bool check_touch();
void clear_game();
```

Как ты заметил, здесь есть указание имен функций, которые будут расписаны отдельно, – это так называемый *прототип функции*. Он служит для того, чтобы можно было вызвать функцию из функции, не обращая внимания на последовательность их задания. Вот пример, чтобы было более понятно (код не компилируется, а служит только для наглядности):

```
int start() {
ende();
}
int ende() {
start();
}
```

Это не будет работать, ведь `ende` вызывается до того, как определяется. А значит, транслятор не знает, где он должен взять конец функции. Если определить сначала `ende()`, то же самое произойдет с функцией `start()`. Чтобы этого не произошло, функции необходимо определять заранее.

```
int start();
int ende();

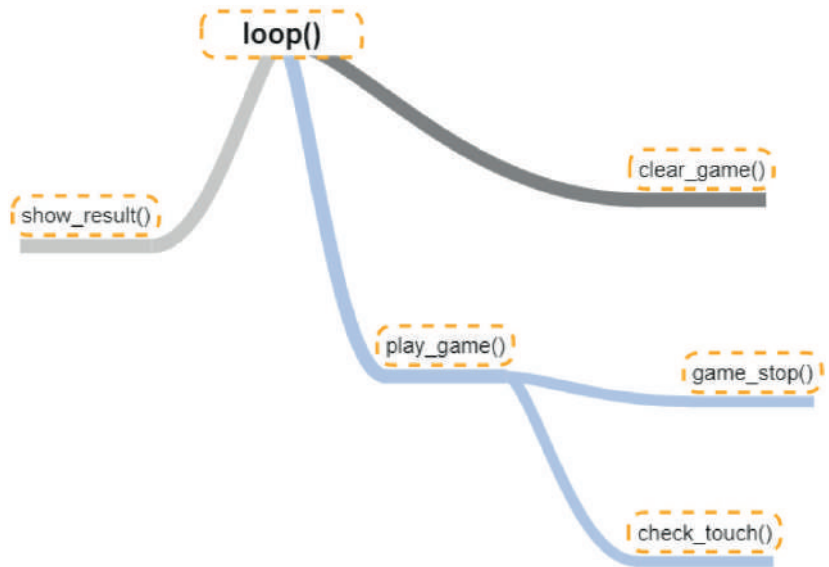
int start() {
ende();
}
int ende() {
```

## 1

```
start();
}
```

Так программа будет работать, поскольку уже указана ссылка на функцию `ende()`. Не спрашивай меня, почему это именно так: таковы правила.

Вернемся к нашей программе. Теперь, когда мы приблизительно знаем, какие функции нам нужны, мы должны найти место, с которого начнем писать. Посмотри на следующий рисунок: здесь я в виде *структуры дерева* перечисляю, какую из какой функции мы вызываем.



Я думаю, мы начнем с `clear_game()`, потому что она самая простая. Эта функция должна просто выключать светодиоды, если мы начинаем новую игру:

```
void clear_game() {
digitalWrite(red,LOW);
digitalWrite(yellow,LOW);
digitalWrite(green,LOW);
}
```

Это самая легкая функция. Уверен, ты смог бы написать ее и с закрытыми глазами (но лучше не надо, а то вдруг ты запишешь код поверх уже набранного текста). А теперь перейдем к функции `game_stop()`. Она тоже очень простая, ведь мы просто хотим проверить, нажата ли кнопка для окончания игры:

```
bool game_stop() {
int state = digitalRead(button_end);
if (state == 1) {return true;
} else {
return false;
}
}
```

В первой строке ты видишь, что перед названием функции стоит `bool`, а не `void`! Это потому, что функция должна возвращать значение типа `bool`. Если ты согласишься посмотреть список типов переменных, то узнаешь, что `bool` – это тип переменной, который содержит в себе либо `true`, либо `false`. Эти значения возвращаются с оператором `return`. Чтобы лучше понять, как работает этот механизм, вот еще один код для наглядности:

```
int addiere(int a,int b) {
int c = a + b;
return c;
}
int foo = addiere(5,4); //foo содержит теперь значение
                        //9, так как 5 + 4 = 9, и рассчитанное
                        //значение возвращается функцией, чтобы
                        //присвоить его переменной foo
```

Думаю, теперь тебе понятен принцип. Для чего это все нужно и зачем мы взяли тип данных `bool`, я поясню тебе в реализации функции `play_game()`. А сейчас мы подошли к функции `check_touch()`. По принципу работы она схожа с `game_stop()`, поскольку обе они заканчивают игру.

```
bool check_touch() {
if (digitalRead(touch) == 1) {
return true;
} else {
return false;
}
}
```

Чтобы было проще, мы не проводим здесь различия между ситуациями «Касание провода» и «Ты выиграл», я думаю, эту адаптацию ты сможешь выполнить и самостоятельно.

# 1

## Измерять время

А сейчас кое-что чуточку посложнее: мы измеряем время с начала запуска Arduino, чтобы получить продолжительность игры. Вот еще одна небольшая функция, которая это выполняет:

```
long play_game() {
  long time = 0;
  while(!game_stop() && !check_touch()) {
    time = millis();
  }
  return time;
}
```

Если ты был внимателен, то знаешь, что переменные `long` отражают больший диапазон чисел, чем `int`. Еще здесь есть новая функция `millis()`, которая отображает количество миллисекунд с момента запуска Arduino. Содержание части `while()` я объясню тебе позже. Здесь ты можешь сказать, что было бы здорово играть несколько игр подряд, не запуская каждый раз Arduino. Для этой задачи есть решение, но мы построим его только в финальной (то есть окончательной) версии нашего скетча.

## Показать результат

Посмотрев еще раз на рисунок, ты заметишь, что у нас не хватает еще одной функции, `show_result()`. Она будет измерять продолжительность и анализировать ее. Если она будет меньше 3 секунд, то пусть загорается зеленый светодиод. Если меньше 10, но больше 3 секунд, то загорится желтый светодиод. Если игра длилась больше 10 секунд, то горит красный светодиод. При этом, приступая к игре, нужно каждый раз перезапускать Arduino заново – для этого не надо выдергивать кабель USB, проще это делать с помощью кнопки, расположенной на плате Arduino около разъема USB.

При необходимости ты можешь задать другие цифры в зависимости от сложности твоего «маршрута» вдоль проволоки. Сначала идет функция с выражением `if`:

```
void show_result(long time) {
  if (time < 3000) { //Время меньше 3 сек.
    digitalWrite(green,HIGH);
  }
}
```

```
if (time > 3000 && time < 10000) { //Время больше 3 секунд,  
                                     //но меньше 10 секунд  
    digitalWrite(yellow,HIGH);  
}  
if (time > 10000) { //Время больше 10 секунд  
    digitalWrite(red,HIGH);  
}  
}
```

## Переключение: запустить необходимую реакцию

У задачи, которую выполняет функция `show_result()`, есть более изящное решение с использованием функции-переключателя (`switch`). При этом ты можешь запускать нужное действие в зависимости от значения переменной-ключа (в данном случае `time`):

```
void show_result(long time){  
switch(time) {  
    case 0-3000 :  
        digitalWrite(green,HIGH);  
        break;  
    case 4000-10000 :  
        digitalWrite(yellow,HIGH);  
        break;  
    default:  
        digitalWrite(red.HIGH); break;  
}  
}
```

Как видишь, код значительно сократился. Функцию структуры `switch` можно предусмотреть заранее. С помощью записи `switch(переменная)` функция инициализируется. Затем всегда следует строка выбора с ключевым словом `case`, в котором проверяется, имеет ли переменная заданное значение: например, в первом случае у нас будет проверяться, содержит ли переменная показатель от 0 до 3 секунд (3000 миллисекунд). За двоеточием далее всегда следует код, который должен быть исполнен, если переменная находится в нужном диапазоне. Части, содержащие строки выбора `case`, заканчиваются оператором `break` (то есть указанием на прерывание работы функции, если одно из значений выбора было реализовано). Затем может следовать либо следующее разветвление `case`, либо оператор `default` (что значит «по умолчанию»), который ведет себя подобно

## 1

else: указывает, что делать, если переменная оказалась вне диапазона всех указанных значений выбора.

Еще одно замечание по возможным ошибкам в некоторых программах: в выражении switch нельзя определять новые переменные.

Для полноты картины – вот весь скетч на данный момент:

```
int red = 12;
int green = 10;
int yellow = 11;
int alert = 9;
int button_start = 8;
int button_end = 7;
int touch = 6;
int play_game();
void show_result(long time);
bool game_stop();
bool check_touch();
void clear_game();

void setup() {}

void loop() {
  int start = digitalRead(button_start);
  if(start == 1) {
    clear_game();
    long time = play_game();
    show_result(time);
  }
}

int play_game() {
  long time = 0;
  while(!game_stop() && !check_touch()) {
    time = millis();
  }
  return time;
}

bool game_stop() {
  int state = digitalRead(button_end);
  if (state == 1) {return true;
} else {
  return false;
}
}
```



```
void clear_game() {
digitalWrite(red,LOW);
digitalWrite(yellow,LOW);
digitalWrite(green,LOW);
}

bool check_touch() {
if (digitalRead(touch) == 1) {
return true;
} else {
return false;
}
}

void show_result(long time){
switch(time) {
case 0-3000 : digitalWrite(green,HIGH); break;
case 4000-10000 : digitalWrite(yellow,HIGH); break;
default: digitalWrite(red,HIGH); break;
}
}
```

## While: исполнить цикл

Здесь в функции `play_game()` ты видишь цикл `while`. Он ведет себя так же, как цикл `loop()`, но цикл `while` исполняется до достижения условия, указанного в заголовке цикла (в круглых скобках). На этот раз мы работаем с переменными `bool`, чтобы избежать постоянных «!=» («не равно»). Кстати, то же самое касается выражений `if`. В них ты также можешь записать просто имя переменной, если речь идет о булевом типе. Если она содержит «true», исполняется выражение `if`, в противном случае нет.

С помощью символов «!» мы отрицаем содержание. Это значит, что `if(!variable)` исполняется только в том случае, если содержание не равно `true` (то есть равно `false`). Также мы можем соединить два условия, используя `&&` («и»). В примере выше это значит: цикл исполняется, только когда `game_stop()` и `check_touch()` возвращают `false`.

Возможно, тебе захочется, чтобы совпадал только один из параметров. Если мы запишем

```
if (!game_stop() || !check_touch()),
```

цикл исполняется, даже если только один из параметров возвращает `false` («||» означает «или»). Точную функцию

## 1

этих знаков («логических операторов») мы подробно рассмотрим в одной из следующих глав.

Итак, мы проделали уже большую работу. Но есть одна проблема: время измеряется с момента запуска Arduino, а не начала игры. Чтобы это исправить, нам нужно слегка подправить `loop()`.

```
void loop() {
  int start = digitalRead(button_start);
  if(start == 1) {
    clear_game();
    long cur = millis();
    long time = play_game();
    time = time - cur;
    show_result(time);
  }
}
```

В переменной `cur` (от англ. *current*, то есть «текущий») измеряется текущее время с момента запуска Arduino. После этого мы отнимаем предшествующее игре время от общего времени. Рассмотрим на словах:

- до нажатия кнопки старта прошло 3000 миллисекунд;
- после окончания игры прошло 10 000 миллисекунд, то есть 3000 мс до старта и 7000 чистого игрового времени;
- чистое игровое время (`long time`, 10 000 мс) минус время до того, как началась игра (`long cur`, 3000 мс). Таким образом, для прохождения маршрута понадобилось 7000 мс (= 7 секунд).

## Заключение

Это была одна из самых объемных глав, поэтому заключение мне захотелось сделать лаконичным:

- ты установил IDE Arduino и написал свой первый скетч;
- ты можешь использовать функции `digitalWrite()` и `digitalRead()` для выводов;
- ты знаешь, как использовать переменные и писать функции;
- ты познакомился с управляющими конструкциями `if()` и `switch()`;
- ты собрал ключ Морзе и игру «Горячий провод».

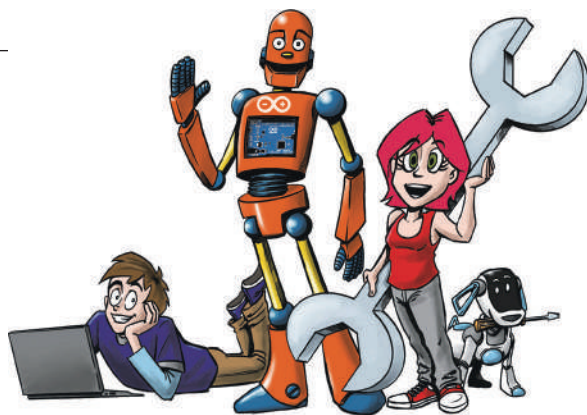
## Несколько вопросов...

1. Как в IDE Arduino настроить используемую плату?
2. Для чего нужны переменные `bool`?
3. С какими типами циклов ты познакомился?
4. Какой результат у функции `millis()`?

## ...и несколько заданий

1. Перепиши программу для ключа Морзе так, чтобы одновременно и мигал светодиод, и звучал сигнал.
2. Найди по ссылке команд Arduino справочную страницу для функции `setup`.
3. Напиши программу, которая заставит мигать гирлянду из 10 лампочек.
4. Напиши функцию, которая может рассчитать квадрат числа ( $x^2 = x * x$ ).
5. Отдохни немного после этой длинной главы.





# 2

## Arduino говорит

В этой главе ты узнаешь, как использовать последовательный интерфейс и с его помощью посылать сообщения компьютеру. Кроме того, ты познакомишься с процедурой отладки. При этом ты сможешь находить логические ошибки в коде и исправлять их. В конце мы сделаем еще одну небольшую игрушку, и ты сможешь включать и выключать ее через компьютер.

В общем и целом мы разберем следующие вопросы:

- ⊙ использование последовательного интерфейса;
- ⊙ основная идея отладки;
- ⊙ простая процедура отладки.

В этой главе дается несколько полезных приемов программирования на Arduino, но ты можешь пропустить ее и прочитать позже.

### Отправка первого текста

Что такое *последовательный интерфейс*? Возможно, ты уже где-то слышал термин *COM-порт*. Последовательный интерфейс раньше был в некоторой степени тем, чем сейчас является USB-порт. Сейчас редко можно найти разъем COM-порта на задней стенке системного блока. Тем не ме-

## 2

нее сегодня мы можем эмулировать этот разъем через USB, то есть обмануть Arduino и сказать ему, что он взаимодействует с последовательным портом. По-английски последовательный порт называется *Serial port*, и ты часто можешь встретить термин «serial» применительно к Arduino, в том числе и в названиях связанных с ним функций.

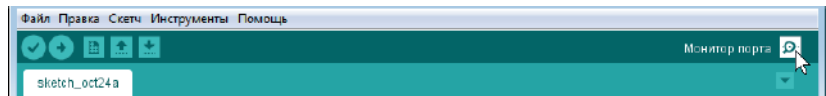
Загрузи в Arduino следующий код:

```
void setup() {
  Serial.begin(9600);
  Serial.println("Hello, world"); } //"Привет, мир"
void loop() {}
```

Загрузив эту программу на Arduino, ты увидишь, что произойдет: ничего.

Чтобы можно было что-то прочесть через последовательный интерфейс, нужен специальный инструмент на ПК – так называемый *терминал*. К счастью, простейший терминал уже встроен в Arduino IDE.

Кликни на значок лупы в верхнем правом углу.



Должно появиться еще одно окно Монитора порта (терминала). Если внизу справа указано «9600 baud», в окне должен появиться текст «Hello, world». К сожалению, для правильного отображения символов в такой примитивной программе терминала, как Монитор порта, приходится обходиться сообщениями на английском языке (можешь попробовать: вместо русских будут выводиться «кракозябры»).

## Инициализация порт на частоте

Вернемся к исходному коду. Все команды, которые относятся к последовательному порту, имеют вид `Serial.function()`. В начале каждого скетча, который должен использовать порт, необходимо инициализировать его на определенной скорости передачи. Команда для этого выглядит так: `Serial.begin(скорость);`. К сожалению, скорость нельзя установить, какую захочется, а можно использовать только одно из предложенных значений. Просто посмотри на список, открывающийся там, где в окошке терминала указано

«9600 baud», и ты найдешь все значения скоростей, с которыми работает Arduino.

В Arduino принято измерять скорость передачи в «бодах» (baud), что показывает количество отправленных единиц информации (например, бит) в секунду. 9600 baud здесь означает 9600 бит в секунду. Так как один символ (char) равен одному байту, то есть восьми битам, то такая скорость примерно соответствует передаче 1000 символов (букв) в секунду (с учетом некоторого количества «лишних» служебных битов, всегда сопровождающих передачу через последовательный порт).

## Выведи что-нибудь!

Следующая команда `println()` выводит что-нибудь на терминале. *Println* – это сокращение для *print line*, что по-русски значит «печатать строку» (иными словами, «выведи что-нибудь»). В простых двойных кавычках («Привет, мир» в примере выше) для этой функции указывается так называемый *строковый тип данных* (англ. *string*), то есть кусок текста. Он выводится на терминале. Теперь мы можем отправлять пользователю целые предложения, а не только кодировать сигналы с помощью светодиодов. Ты можешь, например, переписать игру «Горячий провод» из предыдущей главы так, чтобы не только включался светодиод определенного цвета, но и на терминале появлялось время, затраченное игроком на прохождение маршрута.

## Отладка

А сейчас ты узнаешь, как осуществляется процедура отладки. К сожалению, в Arduino IDE обычно нет встроенного отладчика. Поэтому нам придется импровизировать. Но что же такое отладчик? *Отладчик* – это программное обеспечение, которое проверяет выполняющуюся программу и при желании останавливает ее в тех местах, где предположительно есть ошибка. В этих местах можно просмотреть переменные и их содержание. Кроме того, нажатием определенных клавиш на клавиатуре можно вручную проверить каждую строку. Это значит, можно не исполнять автоматически 1000 строк, например в `setup()`, а остановиться на любом месте, вручную запустить программу дальше и отобразить содержание переменных после выполнения каждой строки (именно поэтому предпочтительно записывать каждую команду в отдельной строке).



## 2

Конечно, это очень упрощенное объяснение основной идеи. Речь идет преимущественно о логических ошибках (то есть, например,  $1*1$  вместо  $1+1$ ), а не синтаксических (`println()` вместо `println()`). Кстати, *синтаксические ошибки* отображаются в черном поле внизу IDE при попытке скомпилировать программу или записать ее в контроллер.

Для отладки нам также понадобится скетч. Я предлагаю взять такой простой исходный код:

```
int led = 13;
void setup() {
  pinMode(led,OUTPUT);
}
void loop() {
  digitalWrite(led,HIGH);
  delay(1000);
  digitalWrite(led,LOW);
  delay(1000);
}
```

В таком виде все работает. Но если мы пропустим предпоследнюю строку, мигания не будет. Это потому, что сначала светодиод включен, потом одну секунду остается включенным и затем выключается, но `loop()` сразу начинает цикл сначала, и светодиод снова включается. Все из-за опоздавшего `delay()`:

```
int led = 13;
void setup() {
  pinMode(led,OUTPUT);
}
void loop() {
  digitalWrite(led,HIGH);
  delay(1000);
  digitalWrite(led,LOW);
}
```

Предположим, что ты этого не знаешь и пропустил ошибку. Такое случается даже чаще, чем кажется, у меня, например, уже однажды возникала вышеназванная ошибка. Тогда были каникулы, и я какое-то время не работал с Arduino. Чтобы найти эту ошибку, мы можем сейчас начать отладку. Результат должен будет отобразиться на терминале (то есть передаваться через последовательный порт в компьютер).

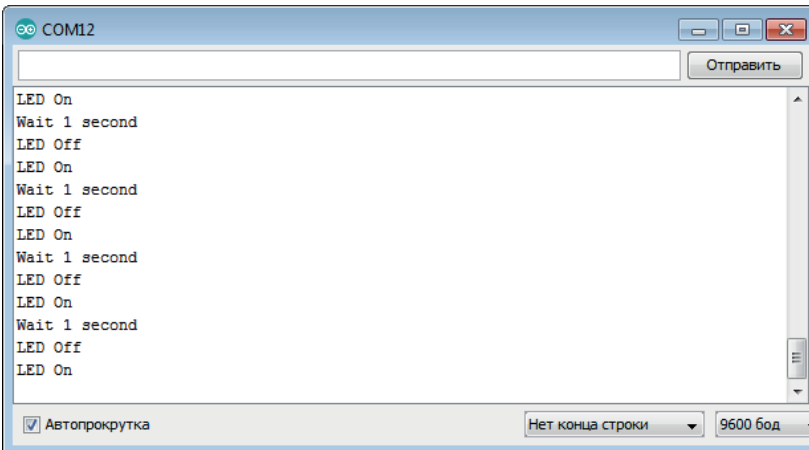
Мы перестраиваем программу так, чтобы после каждой команды отображалось, что именно было сделано. Для больших программ это слишком трудоемко (там нужно обращать внимание только на частные проблемы), но в нашем импровизированном случае достаточно этой процедуры. В результате получается следующий код:

```
int led = 13;

void setup() {
  pinMode(led,OUTPUT);
  Serial.begin(9600);
}

void loop() {
  digitalWrite(led,HIGH);
  Serial.println("LED On"); //"Светодиод включен"
  Serial.println("Wait 1 second"); //"Подождать 1 секунду"
  delay(1000);
  digitalWrite(led,LOW);
  Serial.println("LED Off"); //"Светодиод выключен"
}
```

В нашем случае в окне Монитора порта должно отображаться примерно следующее:



```
COM12
LED On
Wait 1 second
LED Off
LED On
Wait 1 second
LED Off
LED On
Wait 1 second
LED Off
LED On
Wait 1 second
LED Off
LED On
```

Как видишь, здесь есть небольшой «оптический обман». Светодиод снова включается сразу после выключения, и надпись «LED On» («Светодиод включен») возникнет немедленно после «LED Off» («Светодиод выключен»), без паузы. Так ты можешь понять, что происходит, и испра-

## 2

вить ошибку. Вставка задержки `delay(1000);` предпоследней строкой (перед закрывающей фигурной скобкой) решит эту проблему. Таким способом ты сможешь отладить и более крупные проблемы. Например, можно вывести переменные:

```
Serial.println ("Variable xy: " + variablename)
```

При этом в верхних кавычках нужно записать только текстовую часть сообщения, а не имя переменной. Иначе будет ошибка. Если отбросить `ln` у команды `println`, то даже несколько коротких предложений можно написать друг за другом без разрыва строки, например:

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print("Text ... "); // "Текст..."
  Serial.print("on one line"); // "в одну линию"
  Serial.println(); // Разрыв строки
  Serial.print("And moves on to the next");
  // "И переходит на следующую "
  Serial.print(" string");// "строку"
}
```

## Отправить команду

Но давай продолжим и перейдем к одной очень интересной теме. До этого момента мы выводили через Монитор порта Arduino просто знаки, а сейчас отправим обратно команду, чтобы переключить светодиоды. (Посмотри на пустую строку над окошком вывода.) Начнем с малого: с чтения и вывода знаков. Код для этого выглядит так:

```
void setup() {
  Serial.begin(9600);
  Serial.println("Send a new text "); // "Отправить новый текст"
}

void loop() { if(Serial.available()) {
  char incoming = Serial.read();
  Serial.println(incoming);
}
}
```

Если ты сейчас исполнишь этот код, то сможешь отправить тексты через верхнее текстовое поле Монитора порта и отобразить их потом снова в Мониторе порта. То есть схема отправки текста выглядит так: компьютер (текстовое поле в Мониторе порта) ⇒ Arduino ⇒ компьютер (Монитор порта). Как видишь, у нас здесь есть еще один новый тип переменных/данных, `char`. В переменных `char` записываются отдельные буквы. Также наверху в выражении `if` ты видишь условие `Serial.available()`. Это значит, что код в выражении `if` выполняется только тогда, когда в последовательном интерфейсе еще остались переданные знаки (*available* – доступный). Представь интерфейс в виде таблицы с одним столбцом.

Сначала она пуста. Но если ты через компьютер отправишь в Arduino слово «cat», таблица будет выглядеть так:

	c
	a
	t

Теперь мы записали там слово «cat». При вызове `Serial.read()`; всегда выводится последняя принятая строка. Поскольку это имеет смысл только тогда, когда перед этим в память порта (называемую буфером) записывается что-то новое, используется защитный механизм с `Serial.available()`. *Available* – английское слово, которое означает «доступный». Если тебя удивляет, почему я использовал `println()`, а не `print()`, то я делаю это, чтобы иметь возможность отправлять отдельные буквы, поскольку переменные типа `char` могут представлять только одну букву. Следовательно, «cat» выводится как «с а t». Это только один наглядный пример.

А теперь я хочу тебе показать, как сделать разрыв строки через 30 знаков. Для этого мы внедряем счетчик, который, насчитав 30 знаков, активирует разрыв строки. Это может выглядеть, например, так:

```
int i = 0;
void loop() {
  if(Serial.available()) {
    i = i + 1;
    char incoming = Serial.read();
    Serial.print(incoming);
  }
  if(i == 30) {
```

## 2

```
Serial.println();
}
}
```

Просто, но эффективно. Здесь мы видим новую структуру:  $i = i + 1$ . Она заботится о том, чтобы элемент  $i$  получал содержание от  $i$ , но одновременно увеличивался на 1. Это так называемая операция *инкремента*. Ее противоположность, которая уменьшает переменную на единицу, называется *декремент*. Есть и соответствующие глаголы – *инкрементировать* и *декрементировать*. Так как эти структуры чаще всего используются в счетчиках, существует и более короткая синтаксическая структура:  $i++$  для инкрементирования и  $i--$  для декрементирования. Двойной плюс ( $++$ ) обозначает оператор инкремента, а двойной минус ( $--$ ) – оператор декремента. С этим оператором предыдущий скетч будет выглядеть так:

```
int i = 0;
void loop() {
  if(Serial.available()) {
    i++; //Вместо i = i + 1;
        char incoming = Serial.read();
    Serial.print(incoming);
  }
  if(i == 30) {
    Serial.println();
  }
}
```

Не намного короче, но все же. После отправки 30 знаков `Serial.println()` активирует разрыв строки.

Вот тебе еще одно упражнение: напиши функцию, с которой ты можешь отдельно вызвать счетчик. Пример, как это можно сделать:

```
int count(int i);

void setup() {
  Serial.begin(9600);
  Serial.println(„Send a new text „);
}
int i = 0;
void loop() {
  if(Serial.available()) {
```

```
i = count(i);
char incoming = Serial.read();
Serial.print(incoming);
}
}

int count(int i) {
i++;
if(i == 30) {
  Serial.println();
}
}
return i;
```

А сейчас мы разработаем небольшую систему управления выводами Arduino:

```
int pin;
char mode;

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println("Enter level (h->HIGH, l->LOW):");
  //"Введите уровень ((h->HIGH, l->LOW))"
  mode = Serial.read();
  Serial.println("Enter pin"); //"Введите пин"
  pin = Serial.read();
  //Устанавливаем:
  if(mode == 'h') { //Можно также указать в двойных кавычках "h"
    digitalWrite(pin,HIGH); //Если mode = h, то высокий уровень
  } else if (mode == 'l') { //Если mode != h,
    digitalWrite(pin,LOW); //то низкий уровень
  } else {
    Serial.println("Error");
  }
}
```

## Заключение

В этой главе ты познакомился с использованием последовательного порта и процедурой отладки. Кстати, по-английски ошибку в программе называют словом «баг» (*bug* –

# 2

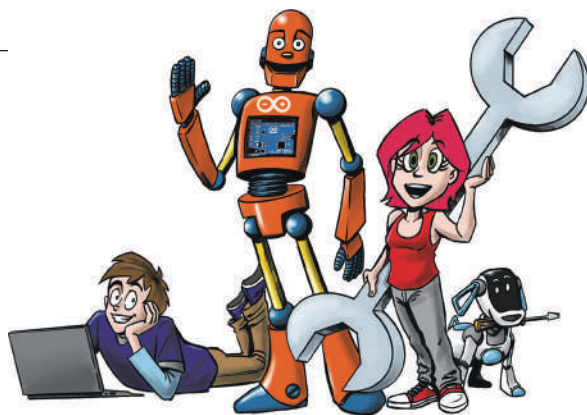
жучок), и это слово очень часто можно встретить во всех языках мира (отладчик тогда будет называться «дебагер»). Это связано с тем, что именно насекомые вызывали ошибки в переключающих схемах первых компьютеров, поэтому нужно было избавляться от этих вредителей.

## Вопрос...

1. Где можно получить данные по установленной скорости передачи последовательного порта?

## ...и задание на сегодня

1. Напиши программу с несколькими светодиодами и попроси знакомого, который тоже умеет программировать, изменить ее так, чтобы в одном из кодов была ошибка. Исправь ошибку с помощью отладки.



# 3

## Сенсоры – интерфейсы для мира

В этой главе мы будем заниматься *сенсорами* – устройствами для измерения каких-то величин. «Сенсоры» – английский термин, а по-русски их называют датчиками. Мы, конечно, будем чаще употреблять русский термин, но может встретиться и иностранный, и пусть тебя это не смущает: сенсоры и датчики – это одно и то же, только на разных языках.

Ты научишься выполнять измерения с помощью датчиков и применять их в схемах. Ты научишься:

- ⊙ считывать данные нескольких датчиков;
- ⊙ анализировать полученные от датчиков значения;
- ⊙ мастерить светодиодную гирлянду, которая светится только в темноте;
- ⊙ использовать широтно-импульсную модуляцию для управления яркостью свечения светодиодов.

Эта глава – одна из самых важных, поскольку получение информации с датчиков относится к самым частым операциям во время работы с Arduino. Я советую тебе не спеша прочитать эту главу два раза.



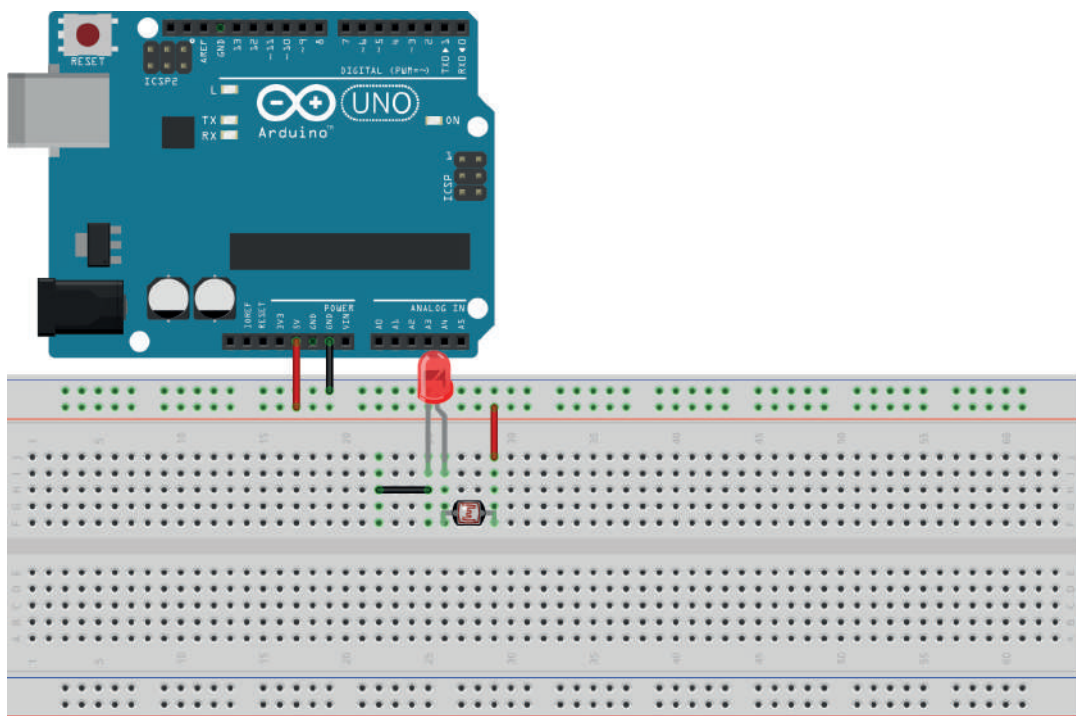
## 3

## Что такое датчик?

Держу пари, ты видел уже много приборов с датчиками, но они не вызывали у тебя особого интереса, или я ошибаюсь?

С точки зрения программиста Arduino, датчик – это электрическая деталь, которая на основании внешних воздействий (например, света или температуры) подает определенный сигнал. Arduino может этот сигнал распознать и интерпретировать.

Построй, например, следующую схему:



fritzing

Здесь Arduino используется только в качестве источника питания (вместо него можно подключить три батарейки AA или AAA). Светодиод с источником тока соединяется через новую деталь – *светочувствительный резистор*. На профессиональном языке он называется *фоторезистор* (фото с его помощью, к сожалению, сделать нельзя).

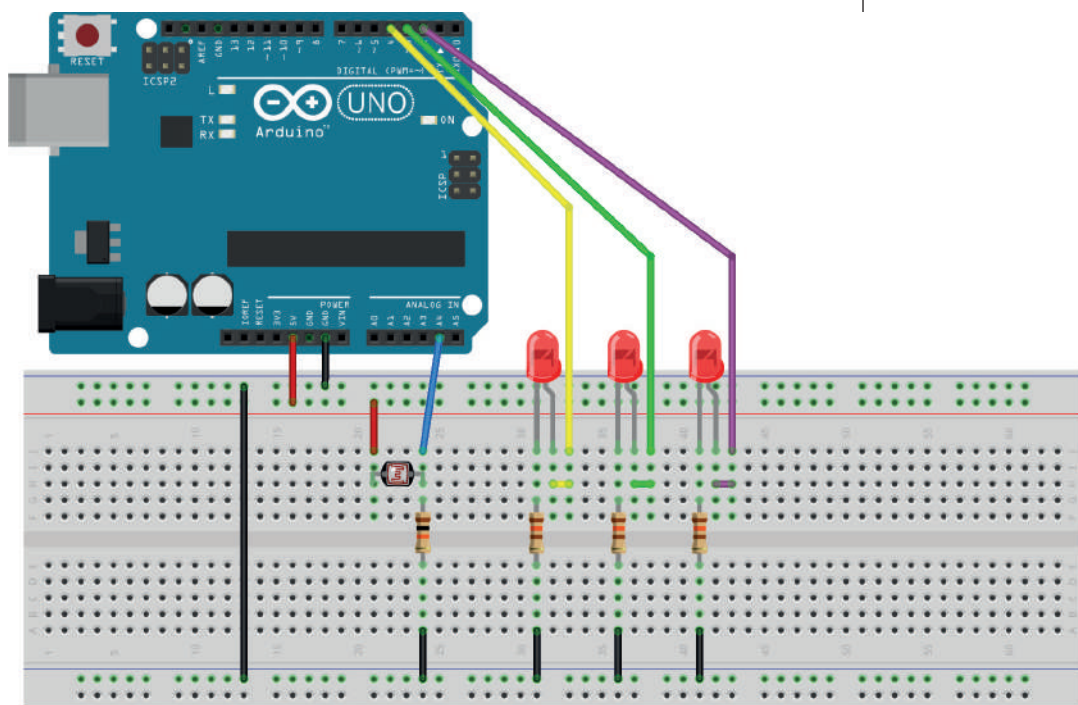
Фоторезистор – это специальная деталь, которая изменяет сопротивление в зависимости от того, как много света по-

падает на датчик. Если датчик стоит на солнце и соединен со светодиодом, светодиод горит ярче всего.

Если подержать над ним руку, сопротивление станет больше, и светодиод будет гореть слабее. Попробуй, это работает даже без управления от Arduino.

## Включить светодиоды

Но такая очень простая схема не имеет смысла – светодиод ярче, когда ярче внешнее освещение, а надо бы наоборот, чтобы светодиод горел, когда освещения нет. Построй следующую схему для Arduino:



fritzing

С ее помощью мы через Arduino измерим уровень освещенности, а потом в темноте включим небольшую гирлянду.

Светодиоды по отдельности подключаются к цифровым пинам 2, 3 и 4 по обычной схеме, с резисторами 130 Ом. Датчик подключается к 5 вольтам и к GND (отрицательному полюсу) через резистор на 10 кОм и потом к аналоговому входу Arduino A4.

## 3



Название *аналоговый* можно объяснить так (конечно, немного упрощенно): у цифрового вывода есть только два состояния, вкл и выкл (1 и 0). Например, я могу либо включить, либо выключить светодиод, а цифровой датчик возвращает либо HIGH, либо LOW. Это не проблема, когда датчику нужны только два состояния, как, например, датчику дыма: у него два состояния, есть дым (1, HIGH) или нет дыма (0, LOW).

Аналоговый вывод работает по-другому: он измеряет величину напряжения на входе. Это хорошо подходит, например, для датчика света, потому что солнечный свет бывает разным по силе (в зависимости от того, что за окном: солнечная или облачная погода, сумерки или ночь), датчик плавно меняет свое сопротивление и подает на вход Arduino напряжение разной величины. Цифровой пин не может это зарегистрировать, так как не способен измерять плавно меняющиеся (аналоговые) сигналы.

## Аналоговые входы

Сначала давай посмотрим, как работают аналоговые входы. Возьми для этого предыдущую схему, но без светодиодов. Теперь выясним, какие показатели читает аналоговый вход.

Вот скетч (объяснение потом):

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  int get = analogRead(A4);
  Serial.println(get);
  delay(1000);
}
```

Скетч, на первый взгляд, выглядит очень простым. Мы просто сохраняем в переменной *get* величину, полученную через *analogRead()*, и потом отображаем ее через последовательный интерфейс.



Ты наверняка спросишь, зачем я подключил второй (постоянный) резистор к фоторезистору? Это нужно, чтобы реализовать делитель напряжения, который меняет свое значение при изменении освещенности.

Открой сейчас Монитор порта и поэкспериментируй с тенью и настольной лампой. Я получил такие величины:

Яркость	Показатель
Освещение настольной лампой	1009
Дневной свет (зимний месяц)	655
Прикрыл рукой	277

Максимальная величина числа, которое может выдать аналоговый вход, равна 1023. Теперь нам нужно подумать, как эту информацию использовать, чтобы включить гирлянду. Я предлагаю включать гирлянду при величине ниже 400 (но, конечно, у тебя может быть другое значение). Так как мы уже раньше строили гирлянду, вот общий код:

```
int schwelle = 400; //Момент, когда светодиод начинает гореть

void blinke() {
digitalWrite(4,HIGH);
delay(100);
digitalWrite(4,LOW);
delay(100);
digitalWrite(3,HIGH);
delay(100);
digitalWrite(3,LOW);
delay(100);
digitalWrite(2,HIGH);
delay(100);
digitalWrite(2,LOW);
delay(100);
}

void setup() {
pinMode(4,OUTPUT);
pinMode(3,OUTPUT);
pinMode(2,OUTPUT);
}

void loop() {
int get = analogRead(A4);
if(get <= 400) {
  blinke();
}
}
```

## 3

Здесь мы сначала устанавливаем пороговую величину, а потом включаем лампочки, если величина, получаемая с порта, ниже этого порога. Чтобы сделать функцию `loop()` более наглядной, я заключил световой эффект в отдельную функцию.

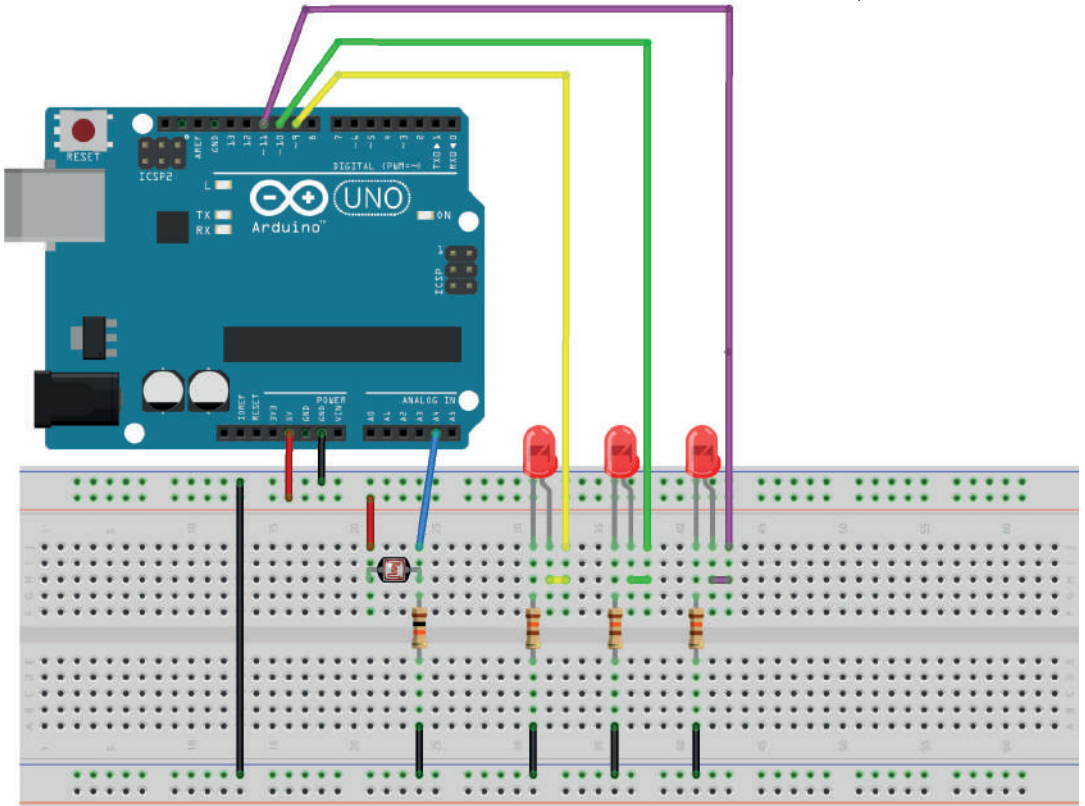
Теперь ты можешь еще немного расширить эту функцию – например, взять разноцветные светодиоды или добавить новые эффекты (чтобы лампочки включались сначала слева направо, потом справа налево).

### Аналоговый выход

Сейчас ты познакомишься с другой аналоговой функцией `analogWrite()` (аналоговый выход). Для этого мы доработаем скетч выше. Пусть мы хотим, чтобы светодиоды горели с различной яркостью, в зависимости от того, насколько светло вокруг. Если только смеркается, они горят слабо, а в полной темноте они горят максимально ярко.

Но сначала вопрос: как можно сделать так, чтобы светодиоды горели с разной яркостью?

Это возможно благодаря ШИМ – *широотно-импульсной модуляции* (англ. Pulse-Width-Modulation, PWM). Это не что иное, как очень быстрое автоматическое включение и выключение светодиодов, в результате чего достигается эффект, который называется *диммирование*. Включение происходит в разное время, и в среднем кажется, что светодиод горит ярче или слабее. Для этого порту необходимо указать значение от 0 (всегда выключен) до 255 (всегда включен). Чтобы продолжать использовать предыдущее соединение, нам нужно слегка изменить электрическую схему: светодиоды придется переключить к выводам 9, 10 и 11, которые поддерживают режим ШИМ (можно использовать и другие выводы, которые обозначены на плате Arduino Uno дополнительным значком «~»).



fritzing

Сначала попробуем, как работает функция `analogWrite()`. Загрузи следующий простой скетч:

```
void setup()
{
  analogWrite(9,128);
  analogWrite(10,255);
}

void loop() {
}
```

Обрати внимание, что мы здесь не использовали функцию `pinMode()`. Для работы `analogWrite()` она не нужна, так как направление работы вывода (на выход, OUTPUT) здесь устанавливается автоматически. В зависимости от типа и цвета светодиода ты можешь поэкспериментировать с указанными в функции величинами. Используй следующий код, чтобы наблюдать еще один замечательный эффект:

## 3

```

void setup() {
}

void loop() {
for(int i = 150; i != 255; i++) {
    analogWrite(9,i);
    delay(70);
}
for(int i = 255; i != 150;i--) {
    analogWrite(9,i);
    delay(70);
}
}

```

По этой программе светодиод, подключенный к выводу 9, будет менять яркость до максимальной, а потом опять до средней. То, насколько сильным будет эффект, тоже зависит от типа светодиода.

А теперь перейдем непосредственно к нашему проекту. Ты, конечно, понял, что для `analogWrite()` необходимы показатели от 0 до 255, в то время как `analogRead()` возвращает показатели от 0 до 1023.  $1023 / 4 \approx 255$  (если брать только целое число результата деления), и нам не составит труда найти решение для проекта.

### Уменьшить яркость светодиодов

Еще раз краткое описание проекта: мы хотим сделать так, чтобы чем ярче был свет вокруг, тем слабее горели светодиоды, то есть чтобы светодиоды были выключены при нормальной освещенности и горели тем ярче, чем темнее становилось в комнате.

Попробуй сначала самостоятельно найти путь решения и разработать свою программу.

Если тебе не удалось самому найти решение, вот мой вариант:

```

void setup() {
}

void loop() {
int get = analogRead(A4);
get = get / 4;
int set = 255-get;
analogWrite(9,set);
analogWrite(10,set);
}

```



```
analogWrite(11,set);  
}
```

Здесь при любом типе светодиодов будем замечен эффект диммирования. Код при этом работает так: сначала в переменной `get` сохраняется текущее значение из аналогового порта и делится на 4, чтобы получилась цифра от 0 до 255. В переменной `set` мы сохраняем разницу между максимумом и полученным значением, то есть устанавливаем, насколько ярко должен гореть светодиод. Потом мы записываем это для всех трех светодиодов одновременно.

Теперь нам нужно сделать гирлянду с регулируемой яркостью, для чего мы слегка изменяем функцию `blinke()`.

```
void setup() {  
}  
  
void blinke(int set) {  
  analogWrite(9,set);  
  delay(100);  
  analogWrite(9,LOW);  
  delay(100);  
  analogWrite(10,set);  
  delay(100);  
  analogWrite(10,LOW);  
  delay(100);  
  analogWrite(11,set);  
  delay(100);  
  analogWrite(11,LOW);  
  delay(100);  
}  
  
void loop() {  
  int get = analogRead(A4);  
  get = get / 4;  
  int set = 255-get;  
  blinke(set);  
}
```

Этот скетч тоже очень простой и понятный. На этом мы закончили тему функций `analogWrite()/analogRead()` и фоторезисторов.

## Транзисторы

Мы подошли к следующему компоненту. Благодаря этому компоненту, называемому *транзистором*, ты можешь даже пальцем замкнуть контакт через два провода, не получив при этом удара током.



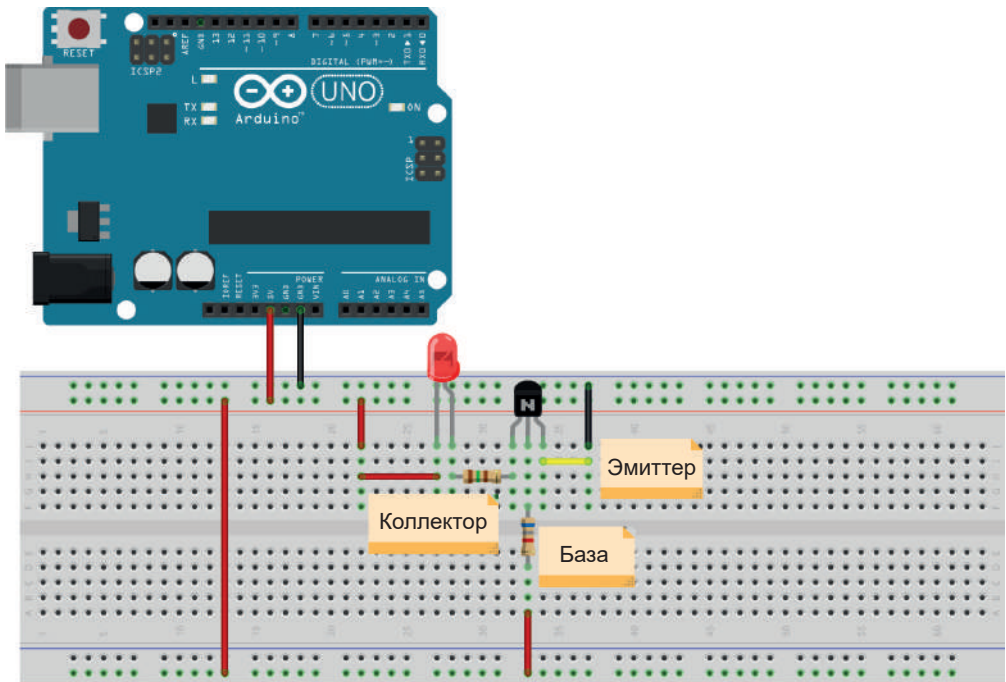
## 3

Но сначала разберем принцип работы транзистора. Транзистор – это деталь, которая усиливает электрический ток. Ты подключаешь к одному из выводов транзистора электрическую цепь, и через другой вывод протекает значительно более сильный ток.



Транзистор – это деталь с тремя ножками, имеющими специальные названия. Если перед тобой плоская сторона, то справа эмиттер, в середине база, а слева коллектор. Это стандартная разводка выводов для маломощных транзисторов (например, BC337 или российского КТ3102), но все равно лучше читать сначала техническое описание, потому что у других типов она может отличаться. Если заставить ток течь через базу к эмиттеру, то более сильный ток потечет через коллектор к эмиттеру.

В следующей схеме ты видишь желтый провод, который выходит из эмиттера. Если ты соединишь его с «землей» (GND), то светодиод загорится. При этом светодиод подключен через резистор на 150 Ом, а управляющая схема базы транзистора через 6,8 кОм. Таким образом, мы можем управлять схемой в коллекторе через переключающую схему, подключенную к базе.



**Внимание!** Неправильная полярность может повредить транзистор. Поэтому всегда следи за тем, чтобы транзистор был правильно включен относительно источника питания.

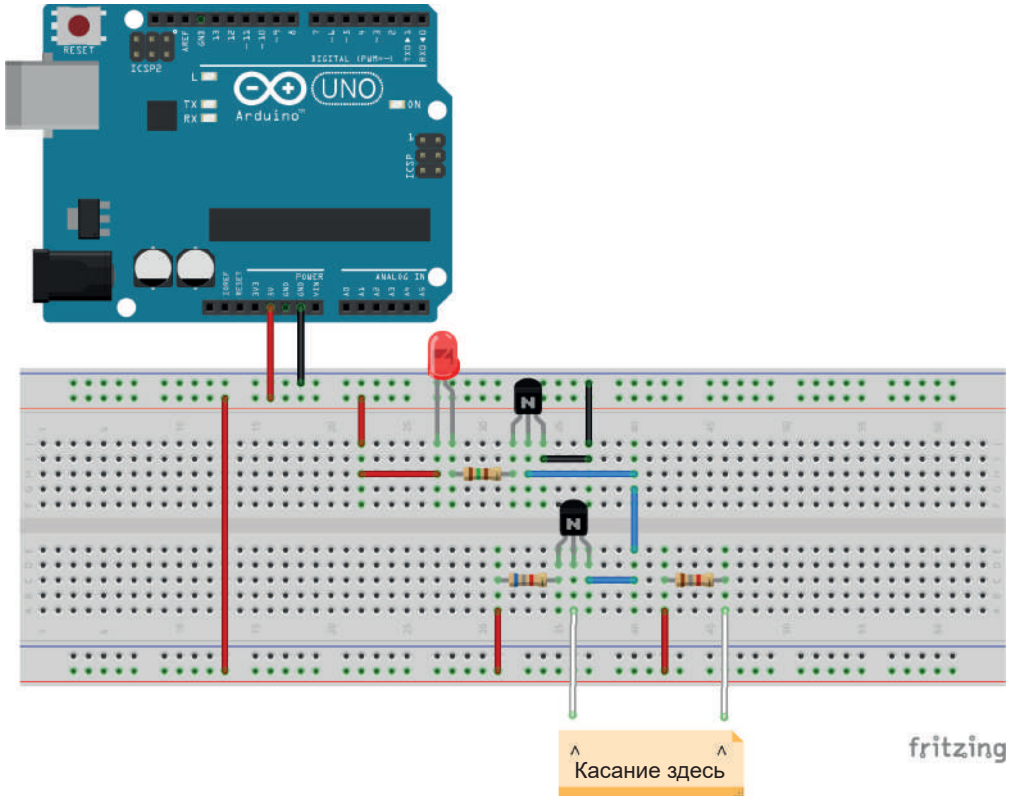


Сейчас тебе наверняка хочется узнать, для чего же нужен транзистор. Ведь можно же переключать непосредственно с помощью функции `digitalWrite()`?

Транзистор предлагает несколько преимуществ – в первую очередь это усиление тока в десятки и сотни раз (в предыдущей схеме ток через светодиод примерно в 45 раз больше управляющего тока через базу). Без транзисторов нельзя представить современную технику, поэтому они есть в большинстве устройств. Например, в микропроцессоре тысячи очень маленьких транзисторов. Раньше все радиоприемники были большими, потому что нужны были лампы, которые позже заменили транзисторами. Только благодаря транзисторам стало возможным уменьшить устройства и разработать вездесущую сегодня микроэлектронику.

Пусть нам нужно усилить очень-очень маленький ток. Для этого мы подключим несколько транзисторов так, чтобы они усиливали друг друга (построим вариант так называемой *схемы Дарлингтона*). Построй такую схему, как показано на рисунке ниже. Сопротивление на правом белом кабеле составляет 1,8 кОм. Сопротивление в коллекторе нижнего транзистора составляет 6,8 кОм. Верхнее сопротивление у светодиода остается тем же, что и раньше, – 150 Ом. Теперь нужно коснуться пальцами двух белых кабелей, и светодиод начнет гореть:

3



Это происходит потому, что транзисторы могут усилить очень слабый ток на базе. Если ток течет на белый кабель от источника питания через пальцы (и небольшой, по сравнению с ними, резистор 1,8 кОм), то электрическая цепь базы первого транзистора замыкается, ток через него поступает на базу второго транзистора, и светодиод горит.

## Заключение

В этой главе ты познакомился с основными функциями некоторых датчиков и можешь читать их через аналоговые выходы. Также ты познакомился с функциями `analogRead()` и `analogWrite()`. Кроме того, теперь ты знаешь, например, что такое фоторезистор и как его использовать.

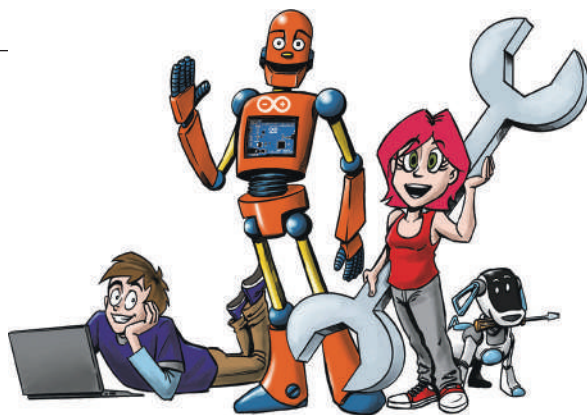
## Несколько вопросов...

1. Как называется первый датчик из этой главы?
2. Назови выводы транзистора.
3. Какая цепь в транзисторе усиливает другую?

## ...и несколько заданий

1. С помощью последней монтажной схемы этой главы проверь, какие предметы, кроме пальцев, проводят достаточно тока.
2. Найди другие датчики, которые работают как фоторезистор (например, термистор или термометр сопротивления), и построй с ними соответствующую переключающую схему.





# 4

## Моторы – движение с Arduino

В этой главе мы будем заниматься моторами. Например, мы построим небольшой вентилятор и секундомер. В основном мы будем работать с двумя разными моторами, но я буду останавливаться и на других конструкциях. Под моторами здесь понимается, конечно же, не двигатель внутреннего сгорания.

Если конкретно, мы сделаем следующее:

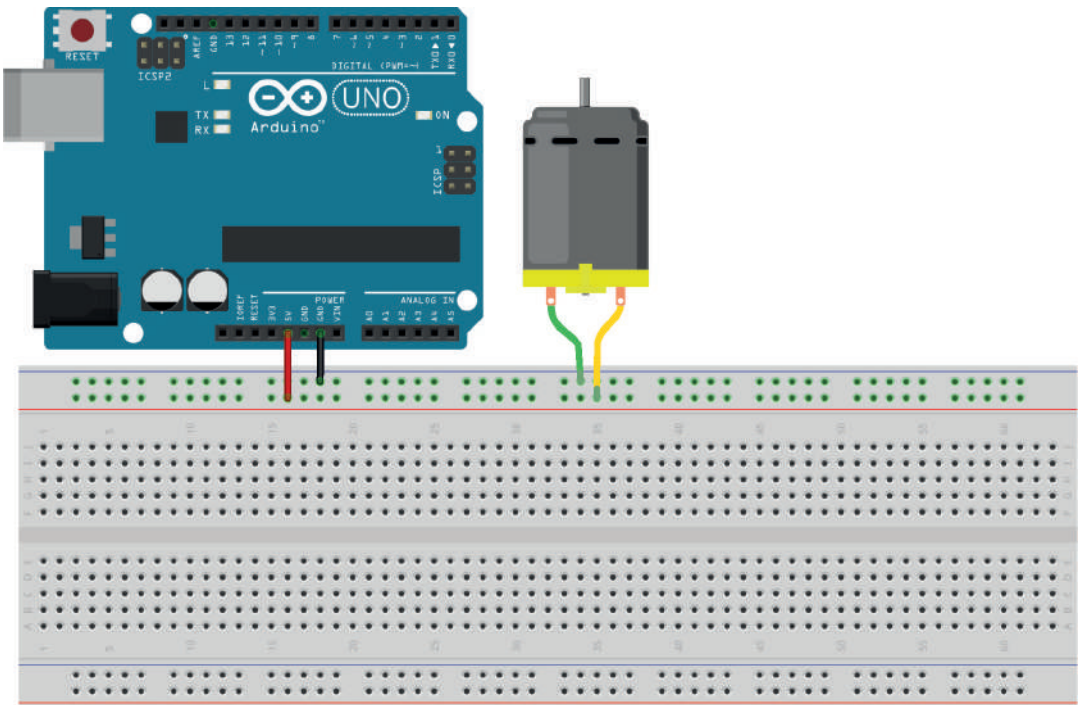
- ⊙ познакомимся с подключением двигателя постоянного тока;
- ⊙ построим небольшой вентилятор;
- ⊙ настроим двигатель на два направления вращения;
- ⊙ познакомимся с сервомотором;
- ⊙ построим таймер (секундомер).

В этой главе тебе нужно будет соорудить небольшую крыльчатку, а если у тебя нет соответствующих навыков, то просто купить ее. В любом случае, этот проект будет тебе интересен.

# 4

## Двигатель постоянного тока – веселое вращение

Сначала мы займемся *двигателями постоянного тока* (англ. DC Motor). У обычного двигателя постоянного тока два выхода. От того, как подключить входы по отношению к полярности источника питания, зависит направление вращения двигателя. Мы используем это позже, когда будем разрабатывать скетч, чтобы двигатель вращался в обоих направлениях. Но для начала построй следующую пробную схему:



fritzing

### Ручной вентилятор

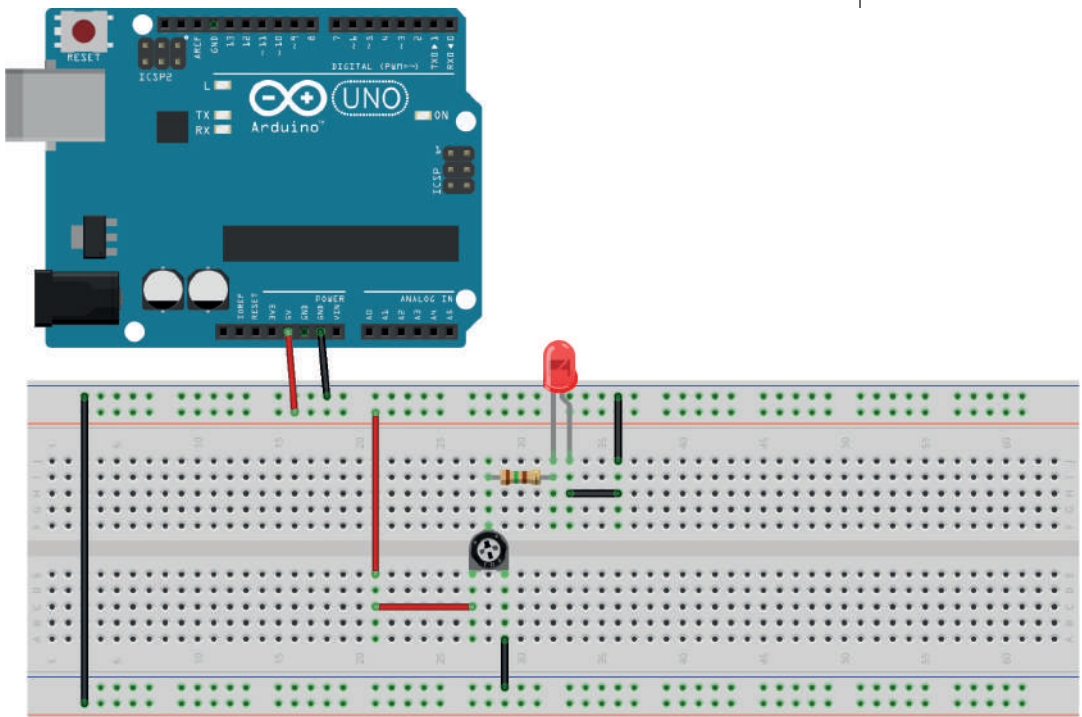
Если подключить желтый кабель от двигателя к плюсу питания, а зеленый – к минусу, двигатель начнет очень быстро вращаться. Если поменять кабели местами, двигатель будет вращаться в другую сторону. Такие двигатели устанавливаются на игрушечные машинки. Двигатели большего размера и немного другой формы используются в настольных вен-



тиляторах. Сейчас мы будем использовать такой двигатель для постройки игрушечного вентилятора. Для этого нам нужно смастерить крыльчатку, которая устанавливается на валу двигателя. Инструкции, как это сделать, ты можешь найти в интернете, там же можно купить и готовый пропеллер.

Сначала мы попробуем использовать еще одну новую деталь – так называемый *потенциометр* (переменный резистор). Это резистор с изменяемым значением сопротивления. С его помощью мы можем регулировать количество оборотов двигателя вручную.

Чтобы лучше понять принцип работы потенциометра, построим следующую схему со светодиодом:



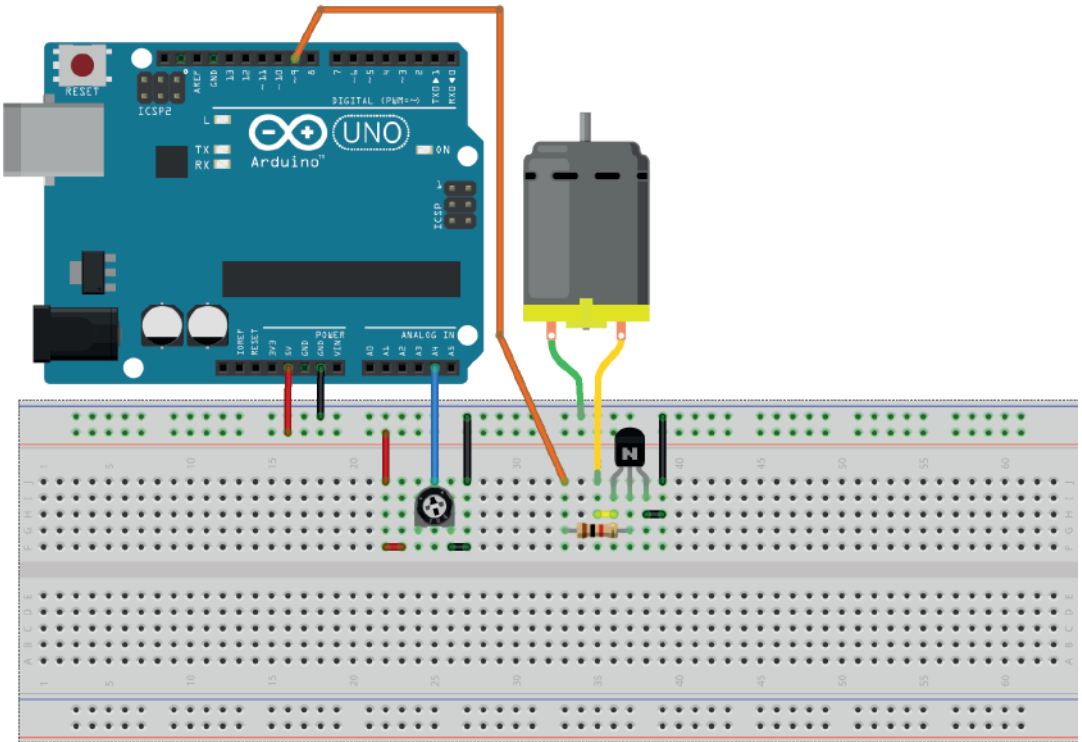
fritzing

Номинальное значение сопротивления переменного резистора на схеме – 1,5 кОм. Если изменить положение регулятора, изменится яркость светодиода. Это связано с тем, что между левым и средним выводами (ползунками) получается переменный резистор. Левый вывод при этом подключен к плюсу питания 5 В, а правый – к минусу (GND). Так мы снова построили делитель напряжения.



# 4

Можно ли заменить в схеме светодиод на двигатель постоянного тока, чтобы получить регулируемый вентилятор? Нет, просто так этого сделать нельзя по одной простой причине: светодиод потребляет очень маленький ток, а двигатель, даже игрушечный – гораздо больший. Нам понадобится бы очень большой потенциометр, чтобы регулировать обороты двигателя напрямую, и все равно он мог бы перегреться и в конце концов выйти из строя.



fritzing

Мы попробуем регулировать обороты вентилятора с помощью уже известной нам ШИМ (функция `analogWrite()`), управляемой на этот раз потенциометром вручную, а не фотодатчиком, как для гирлянды с регулируемой яркостью.

На рисунке приведена такая схема. Обрати внимание, что двигатель подключен через транзистор, чтобы не повредить вывод Arduino (сопротивление резистора, подключенного к выводу базы, равно 1 кОм). Загрузи в Arduino следующий код (посмотри на код для гирлянды с регулируемой яркостью, и ты увидишь, что они очень похожи):

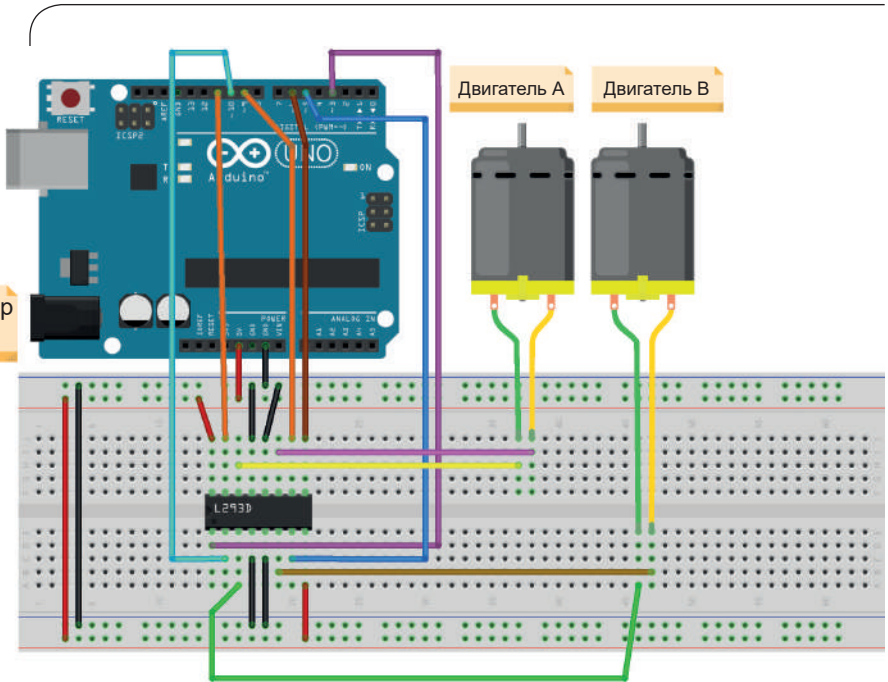
```
void setup() {  
}  
  
void loop() {  
  int get = analogRead(A4);  
  int set = get / 4;  
  analogWrite(9,set);  
}
```

При вращении потенциометра обороты двигателя будут меняться. Без дополнительного питания схема будет работать только с самыми маленькими двигателями. Если имеющийся у тебя двигатель все-таки не будет вращаться, необходимо отключить схему от кабеля USB и подключить к плате Arduino внешний источник питания (адаптер) через большой круглый разъем на той же стороне платы, где подключается USB.

## Эффективно управлять двигателем

Выше я упомянул, что управлять двигателями напрямую с Arduino не получится. И уж наверняка это не выйдет, если двигатели еще более мощные, чем игрушечные. В таких случаях можно подключить специальную микросхему – так называемый *драйвер двигателя*, который специально предназначен для управления двигателем. Мы подключим популярный драйвер L293D. Он может параллельно управлять сразу двумя двигателями:

Внешний адаптер питания ->



fritzing

Для управления двигателями загрузи в Arduino следующий код:

```
int motor_a_tempo = 6; //Выходы управления
int motor_b_tempo = 3; //включением двигателей
int motor_a_con_a = 11; //Управление направлением для двигателя А
int motor_a_con_b = 9; //смотри таблицу ниже
int motor_b_con_a = 10; //Управление направлением для двигателя В
int motor_b_con_b = 5; //смотри таблицу ниже

void setup() {
  pinMode(motor_a_con_a,OUTPUT);
  pinMode(motor_a_con_b,OUTPUT);
  pinMode(motor_b_con_a,OUTPUT);
  pinMode(motor_b_con_b,OUTPUT);
}

void loop() {
  digitalWrite(motor_a_con_a,HIGH); //Направление двигателя А
  digitalWrite(motor_a_con_b,LOW); //смотри таблицу ниже
  analogWrite(motor_a_tempo, 50); //Показатель 0-255 для задания
  //скорости вращения, двигатель В
  //на четверть оборотов
  digitalWrite(motor_b_con_a,LOW); //Направление двигателя В
```

```
digitalWrite(motor_b_con_b,HIGH); //смотри таблицу ниже
analogWrite(motor_b_temp,255); //Двигатель А на максимум

delay(1000);

analogWrite(motor_a_temp.0); //Стоп оба
analogWrite(motor_b_temp.0);
delay(1000);
}
```

Для двух двигателей питания от USB наверняка не хватит, потому после загрузки программы отключи плату Arduino от USB-кабеля и подключи внешний адаптер питания. В таблице показано, как нужно управлять выводами, чтобы двигатель вращался в нужном направлении.

Вывод А управления направлением (_con_a)	Вывод В управления направлением (_con_b)	Направление вращения
LOW	HIGH	Например, вправо*
HIGH	LOW	И наоборот, влево

\* В зависимости от подключения выводов двигателя к схеме.

## Сервоприводы

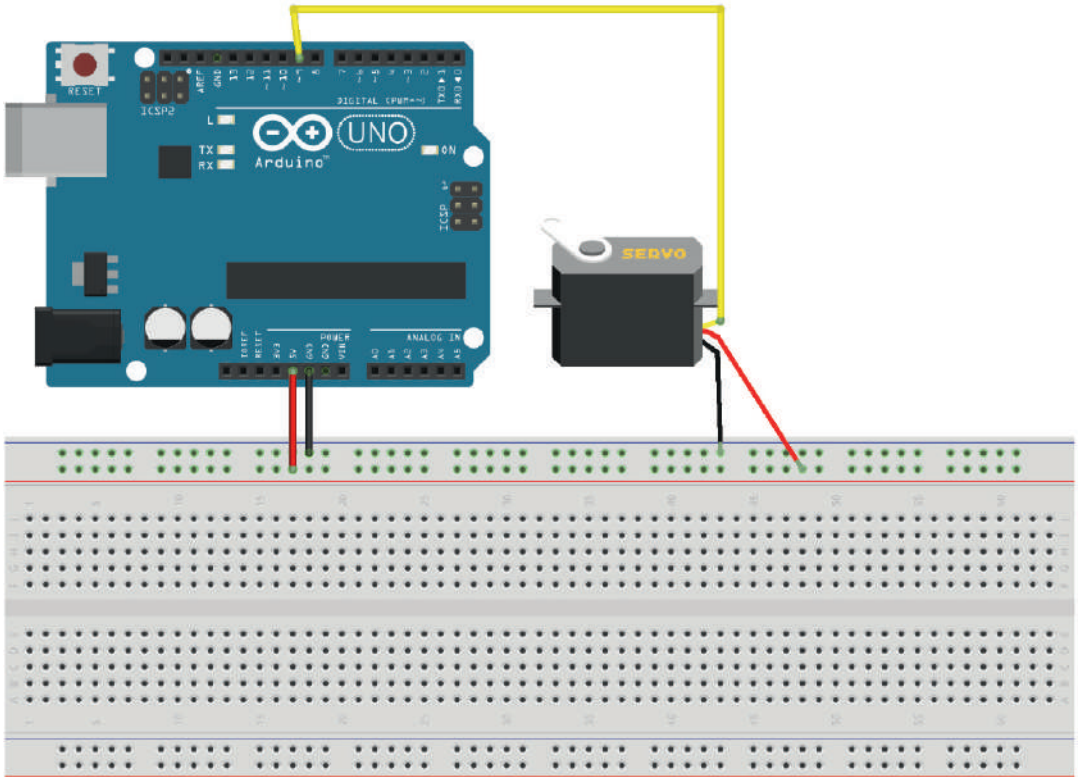
Мы подошли к одной интересной теме: использование *сервоприводов*. Сервопривод похож на двигатель постоянного тока, но имеет три вывода, в результате чего может поворачиваться на определенный угол. Мы используем это, чтобы запрограммировать 30-секундный таймер. Внутри сервопривода находится собственный контроллер, который обрабатывает сигналы от Arduino и управляет поворотом двигателя.

При покупке необходимо учитывать, что существует несколько видов сервоприводов. Нам нужны те, которые могут бесконечно вращаться в обоих направлениях, так как существуют сервоприводы, которые вращаются максимум на 180°. Стандартными считаются сервоприводы, которые не могут делать полный оборот, а полнооборотные (360°) сервоприводы (Continuous rotation Servos) – это модернизация стандартного сервопривода. При желании сервопривод с ограничением можно самому переделать так, чтобы он совершал полный круг вращения.



# 4

Что касается конструкции сервопривода, у него есть три вывода, один на VCC, один на GND, и тот, через который мы управляем движением с помощью подачи импульсов определенной длительности. Цвета выводов сервопривода – коричневый, красный, оранжевый. Коричневый идет на GND, красный – на VCC, а оранжевый – это канал данных, у нас он идет на вывод 9 платы Arduino. Возьми следующий скетч и подключи сервопривод, как показано на следующей схеме.



fritzing

```
#include <Servo.h>
Servo servo; int pos = 0;
int temp = 15;
void setup() {
  servo.attach(9);
}

void loop() {
```

```
for(pos = 0; pos < 180; pos += 1) {
  servo.write(pos);
  delay(temp);
}
for(pos = 180; pos >= 1; pos -= 1) {
  servo.write(pos);
  delay(temp);
}
```

Это основа наших последующих схем с сервоприводом. Желтый здесь – канал данных, который у сервоприводов часто бывает оранжевого цвета. Если сейчас ввести программу в Arduino, сервопривод будет быстро вращаться в диапазоне 180°. При этом программа работает так: сначала мы подключаем к скетчу библиотеку Servo.h, которая содержит необходимые функции. Затем объявляем объект типа Servo с именем servo (поскольку существует различие между прописными и строчными буквами, можно взять в качестве имени объекта имя типа, но только со строчной начальной буквой). В переменных мы сохраняем положение (pos) и скорость вращения (temp). После этого мы подключаем сервопривод к пину 9 (servo.attach(9)). В обоих циклах for мы вращаем сервопривод с помощью функции write до положения, которое в первом цикле увеличивается, а во втором уменьшается. Так мы получаем вращение на 180° и обратно. Чтобы лучше понять команду write, возьми следующий код:

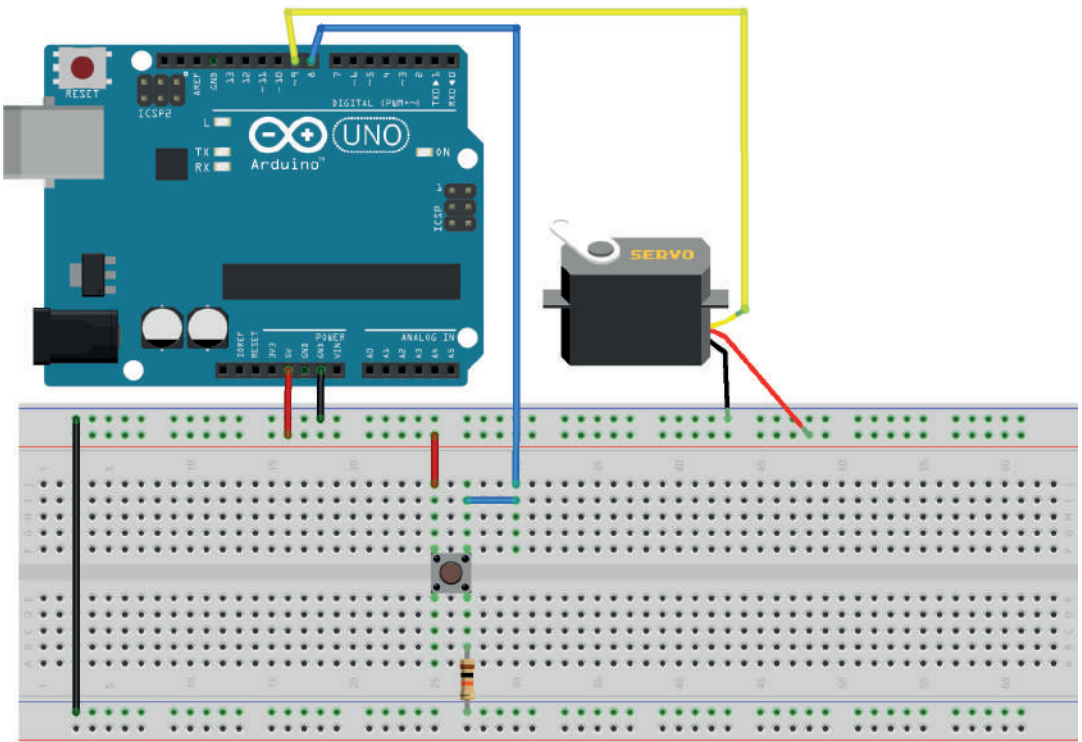
```
#include <Servo.h>
Servo servo;
int pos = 0;
void setup() {
  servo.attach(9);
}
void loop() {
  while(pos != 99); {
    servo.write(pos);
    delay(15);
    pos++;
  }
}
```

С этим кодом сервопривод вращается до положения 99° и остается там, пока ты не сбросишь Arduino или заново не подключишь питание.

# 4

## 30-секундный таймер

А сейчас мы смастерим 30-секундный таймер. Для этого нужно сначала вырезать небольшой круг из картона и нанести на него по краю 30 черточек на одинаковом расстоянии в расчете на полкруга. В середине круга сделай отверстие, чтобы установить круг на сервопривод. Это наша заготовка для таймера. Теперь ты можешь попробовать по памяти построить на макетной плате схему с кнопкой. Если не получается, посмотри на электрическую схему – там я подключил кнопку вместе с сервоприводом (циферблат на схеме не показан). Схема такая (сервопривод подключить к выводу 9 платы, кнопку – к выводу 8):



Если не учитывать кнопку, то все это будет работать со следующим исходным кодом:

```
#include <Servo.h>

Servo myservo;
```



```
int pos = 0;

void setup() {
  myservo.attach(9);
}

void loop() {
  for(pos = 0; pos < 180; pos += 6) {
    myservo.write(pos);
    delay(1000);
  }
  for(pos = 180; pos >= 1; pos -= 1) {
    myservo.write(pos);
    delay(15);
  }
}
```

Нижняя часть в цикле Loop та же, что и в предыдущем коде, только были адаптированы скорость и угол поворота. Теперь сервопривод вращается медленно ( $6^\circ$  в секунду, всего за 30 шагов он сделает половину оборота) вверх и быстро возвращается снова вниз. Единственное, что нам сейчас осталось сделать, – это добавить код для кнопки:

```
#include <Servo.h>

Servo myservo;
int pos = 0; int buttonpin = 12;
void setup() {
  pinMode(buttonpin, OUTPUT);
  myservo.attach(9);
}

void loop() {
  if (digitalRead(buttonpin) == HIGH) {
    for(pos = 0; pos < 180; pos += 6) {
      myservo.write(pos);
      delay(1000);
    }
    for(pos = 180; pos >= 1; pos -= 1) {
      myservo.write(pos);
      delay(15);
    }
  }
}
```



## 4

Если сейчас нажать кнопку, таймер заработает и будет считать секунды до 30; затем он снова вернется в исходное положение. Если ты хочешь доработать таймер, вот несколько предложений, которые ты можешь реализовать позднее. Можно сделать так, чтобы по завершении цикла таймер издавал звук; или вывести время через последовательный порт, чтобы таймер показывал через Монитор порта прошедшее время в процентах. А еще ты можешь сделать время срабатывания переменным и задавать его вручную с помощью потенциометра.

## Заключение

В этой главе ты узнал немного об электродвигателях, которыми можно управлять с помощью Arduino:

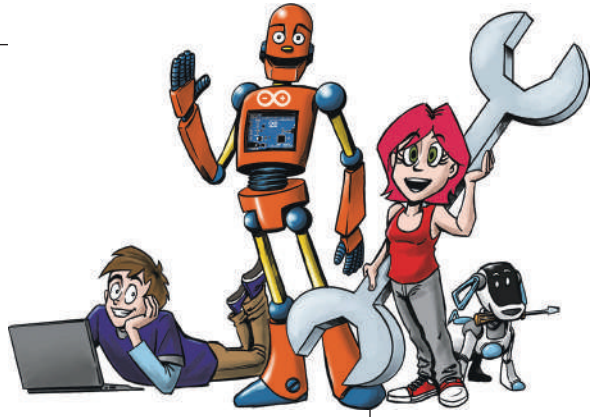
- что такое двигатель постоянного тока;
- как управлять им и построить с ним вентилятор;
- что такое сервопривод;
- как им управлять
- и построить с ним секундомер.

## Несколько вопросов...

1. Каким двигателем управляют заданием угла поворота в градусах?
2. На сколько градусов вращается стандартный сервопривод?
3. Можно ли управлять двигателем постоянного тока?
4. Что такое секундомер?

## ...и несколько заданий

1. Когда прочтешь еще несколько глав, дополни секундомер новыми усовершенствованиями.
2. Попробуй построить машинку с четырьмя двигателями постоянного тока, питающимися от батареек. Для этого тебе нужно поставить на концы двигателей небольшие колеса, например от игрушечной машинки.



# 5

## Чтение исходного кода других разработчиков

Это действительно сложная глава!

Я хочу сделать небольшой экскурс в теорию: тебе нужно будет объяснять функцию и содержание исходного кода, не имея документации. Ты научишься:

- ⦿ получать информацию из технических описаний;
- ⦿ обращать внимание на операторов `include`;
- ⦿ объяснять смысл кода (теория);
- ⦿ использовать программу и схему (практика).

Ты можешь пока отложить эту главу, так как она будет интересна только тогда, когда ты захочешь понять и применить чужой код или другие проекты (конечно, не помешает научиться этому заранее).

### Документация

Обычно каждая программа поставляется с документацией, пусть даже очень краткой и малоинформативной. Иногда она может состоять даже из одного предложения, например «Программа читает яркость через светодиод». Сюда прилагается еще подходящий исходный код и, возможно, электрическая схема.

## 5

На этом завершим разговор о документации к программам, потому что в этой главе мы будем обходиться без нее.

## Загадочный исходный код

Вот небольшой исходный код, который я разработал после вопроса на одном форуме.

```
/*Функции, чтобы одновременно включить несколько пинов (с)
2013,2014,2015,2016,2017 Э. Шерних*/
void an(int pin,int pin2=13,int pin3=13,int pin4=13,int
pin5=13,int pin6=13,int pin7=13,int pin8=13,int pin9=13) {
digitalWrite(pin,HIGH);
digitalWrite(pin2,HIGH);
digitalWrite(pin3,HIGH);
digitalWrite(pin4,HIGH);
digitalWrite(pin5,HIGH);
digitalWrite(pin6,HIGH);
digitalWrite(pin7,HIGH);
digitalWrite(pin8,HIGH);
digitalWrite(pin9,HIGH);
}
void aus(int pin,int pin2=13,int pin3=13,int pin4=13,int
pin5=13,int pin6=13,int pin7=13,int pin8=13,int pin9=13) {
digitalWrite(pin,LOW);
digitalWrite(pin2,LOW);
digitalWrite(pin3,LOW);
digitalWrite(pin4,LOW);
digitalWrite(pin5,LOW);
digitalWrite(pin6,LOW);
digitalWrite(pin7,LOW);
digitalWrite(pin8,LOW);
digitalWrite(pin9,LOW);
}

void setup() {
an(3,4,5);
}
void loop() {}
```

Если ты сейчас согласишься посмотреть на этот исходный код, то наверняка не увидишь функцию. Поэтому я предлагаю тебе просто загрузить его в Arduino и протестировать. Для этого нужно подключить светодиоды к пинам 3, 4 и 5.

Если ты сейчас введешь этот код, загорятся только эти све-

тодиоды и светодиод 13. Если хочешь попробовать проникнуть в суть функции, измени параметры при вызове функции. После таких попыток можно со временем разобраться, что значит тот или иной код.

Этот небольшой код несложно понять: он должен одной-единственной командой включать и выключать несколько пинов. Но чтобы все было интереснее, вот такой исходный код:

```
#include <EEPROM.h>
int address = 0;
int analog_address = 513;
byte value;
bool start = true;
int led_1 = 6;
int led_2 = 5;
int button = 7;
int buttonstate = 0;

void digitalWrite_p(int pin,int state) {
    if(state == 1) {
        EEPROM.write(address,1);
        digitalWrite(pin,HIGH);
        ++address;
    }
    if(state == 0) {
        EEPROM.write(address,2);
        digitalWrite(pin,LOW);
        ++address;
    }
}

void digitalRead_p(int pin) {
    int state = digitalRead(pin);
    if(state == HIGH) {
        EEPROM.write(address,3);
    }
}

void setup() {
    Serial.begin(9600);
    pinMode(led_1,OUTPUT);
    pinMode(led_2,OUTPUT);
    pinMode(button,INPUT);
}
```

## 5

```

void loop() { if (start) {
  value = EEPROM.read(address);
  while(value != 255){
    value = EEPROM.read(address);
    Serial.print(address);
    Serial.print ("\t");
    Serial.print(value,DEC);
    Serial.println();
    ++address;
    if (address == 512) {
      Serial.println("Memory full, manual reset");
      //"Память заполнена, сброс вручную"
      address = 0;
    }
  }
  while(value != 255){
    value = EEPROM.read(address);
    Serial.print(analog_address);
    Serial.print ("\t");
    Serial.print(value,DEC);
    Serial.println();
    ++analog_address; if (analog_address == 1000) {
      Serial.println("Memory full, manual reset");
      //"Память заполнена, сброс вручную"
      address = 513;
    }
  }
  address--;
  Serial.print("Address: ");
  Serial.print(address);
  Serial.println(=
  Serial.println()Start Attempts"); //"Начать попытки"
  delay(1000);
  digitalWrite_p(led_1,1);
  delay(1000);
  digitalWrite_p(led_2,1); delay(1000);
  digitalWrite_p(led_2,0); delay(1000);
  digitalWrite_p(led_1,0);
  start = false; //Адрес обнаружен
}
}

```



Еще один важный совет: всегда обращай внимание на то, какие были использованы библиотеки в заголовке, поскольку они могут содержать незнакомые тебе команды.

## Заключение

В этой главе ты начал толковать исходный код, не зная его функции. Так как это очень маленькая глава, здесь нет ее краткого изложения.

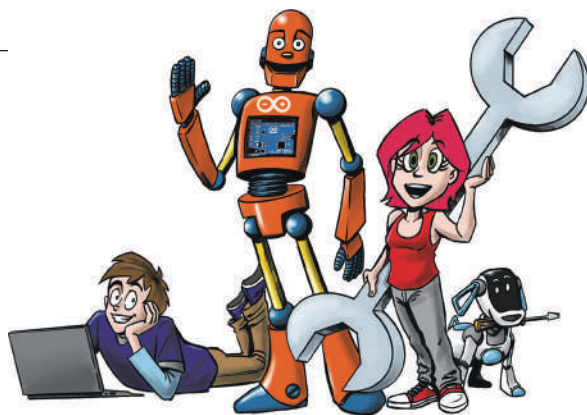
## Несколько вопросов...

1. Для чего нужна документация?
2. Может ли документация быть очень краткой?
3. Какова функция последнего исходного кода этой главы?

## ...и задание

1. Если ты не можешь ответить на последний вопрос, ответь на него после прочтения всей книги.





# 6

## ЖК-дисплей – отображение данных на самом Arduino

Во второй главе мы уже занимались демонстрацией текста через последовательный порт. В этой главе мы будем делать это на ЖК-дисплее, который напрямую связан с Arduino.

Знания, которые ты получишь:

- ⦿ как устроен и функционирует ЖК-дисплей;
- ⦿ отправка текста на ЖК-дисплей;
- ⦿ как подключить дисплей;
- ⦿ идеи по практическому использованию.

В этой главе мы рассмотрим принцип работы ЖК-дисплея и потом на основе материала главы 3 сможем выводить показатели различных датчиков на дисплее.

Эта глава будет легкой, так как тебе нужно будет только выводить текст.



## 6

## Что такое ЖК-дисплей?



ЖК-дисплей расшифровывается как *жидкокристаллический дисплей* (Liquid Crystal Display). Это можно понимать буквально, поскольку в дисплее действительно жидкий кристалл. С помощью небольшого контроллера на плате дисплея мы немного экономим силы, поскольку мы не управляем дисплеем вручную, а обмениваемся сообщениями с контроллером дисплея. Чтобы дисплей работал с программами из этой книги, нужно, чтобы у него был установлен контроллер hd44780 (или совместимый с ним – смотри описание или спрашивай продавцов при покупке).

Плата дисплея соединяется с Arduino несколькими проводниками. У дисплея, который я использую для этой книги, 16 выводов. Если на плате дисплея установлен указанный выше тип контроллера, то тебе понадобится только техническое описание с разводкой выводов, чтобы подключить дисплей к Arduino. Нам нужны следующие выводы на дисплее: RS, Enable, D4, D5, D6 и D7, а также выводы электропитания.

У этого ЖК-дисплея все выводы вверху слева:



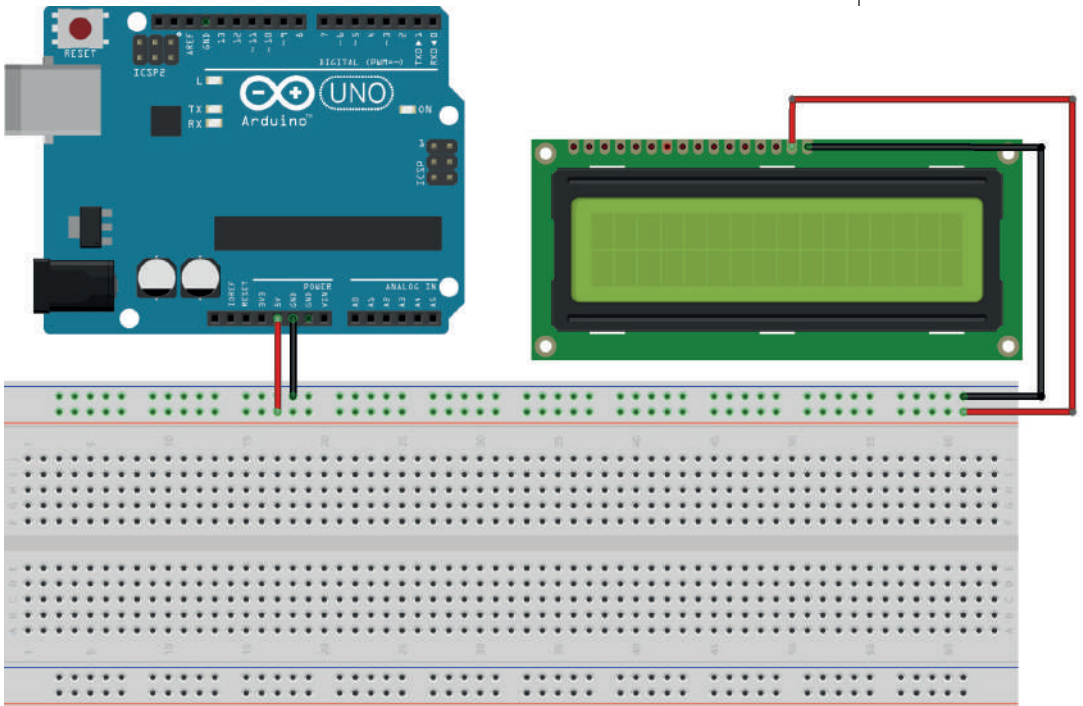
Все будет для нас даже еще проще, поскольку существует готовая стандартная библиотека для Arduino под названием LiquidCrystal, с помощью которой нам будет легче управлять дисплеем. С ее помощью мы можем писать на дисплее текст, перемещать курсор и совершать другие действия. Чтобы можно было соединить дисплей с макетной платой, нужно к выводам на плате дисплея припаять проводники или игольчатый разъем.

Затем подключи дисплей проводниками или этим разъемом к макетной плате. Замечательной альтернативой

## Что такое ЖК-дисплей?

припаянным проводникам будет припаять гнездовой разъем, как в Arduino. Тогда ты можешь подключать проводники произвольно, как в обычном соединении Arduino с макетной платой, и они никогда не отломятся в месте пайки.

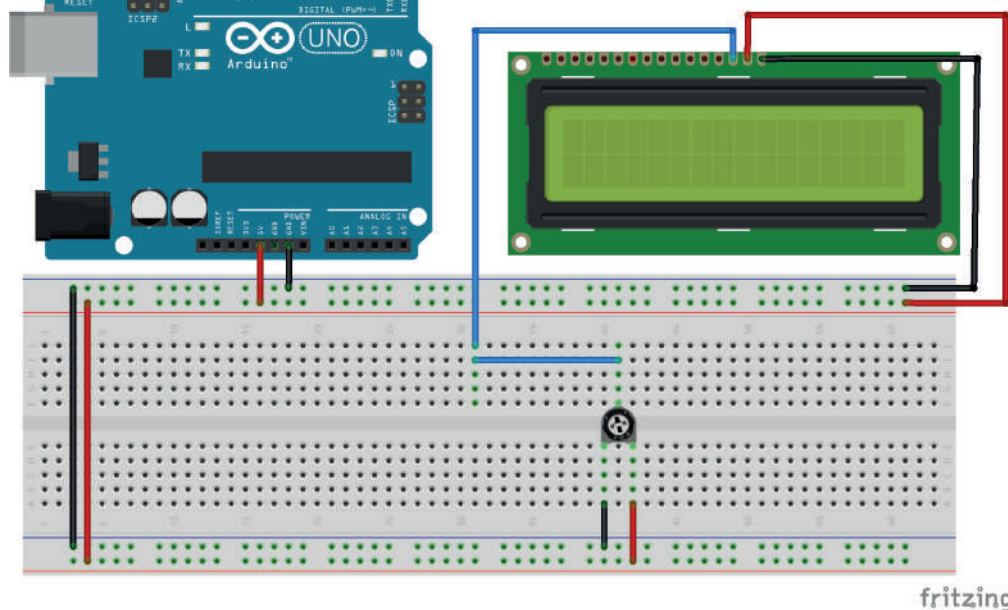
Построй для начала такую схему. Возможно, тебе придется исправить подключение выводов, отыскав в техническом описании твоего ЖК-дисплея выводы с такими же названиями. Красный кабель идет на VCC, а черный – на GND дисплея.



fritzing

Построив эту схему, ты можешь заметить, что *яркость фона* ЖК-дисплея довольно слабая. Ее можно менять, но для этого нужно добавить в схему потенциометр с номинальным сопротивлением 10 кОм. В результате можно будет настроить контраст между шрифтом и фоном (после настройки при определенном значении напряжения питания можно заменить потенциометр на два постоянных резистора, измерив сопротивление его плечей):

6



Вот исходный код для первого текста дисплея. Если ты введешь код, должен появиться текст «Hello, world!» («Привет, мир!»). К сожалению, выводить текст по-русски на такой дисплей напрямую не получится. И притом у дисплеев производителей из разных стран разные способы вывода символов национальных языков, так что нам на первых порах придется ограничиться английским алфавитом.

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  lcd.begin(16, 2); //Укажи количество знаков и строк
                  //для данного типа дисплея
  lcd.print("Hello, world!"); //Выведи текст
}

void loop() {
}
```

В скобках при создании объекта lcd типа LiquidCrystal указаны выходы Arduino, к которым нужно подключать определенные выходы дисплея (по порядку: RS, Enable, D4, D5,

D6 и D7). Чтобы сделать код более понятным, мы немного его перепишем:

```
#include <LiquidCrystal.h>
#define RS 12
#define Enable 11
#define D4 5
#define D5 4
#define D6 3
#define D7 2

LiquidCrystal lcd(RS, Enable, D4, D5, D6, D7);

void setup() {
  lcd.begin(16, 2); //Укажи количество знаков и строк
                  //для данного типа дисплея
  lcd.print("Hello, world!"); //Выведи текст
}

void loop() {}
```

## Команда для компилятора: define

Здесь также используется новая команда, `#define`. Это команда не для Arduino, как раньше, а команда напрямую компилятору. С ее помощью ты настроишь имя константы, которую компилятор должен заменить на ее заданное значение. Например:

```
#define var 7
```

Команда означает, что каждый «var» в исходном тексте заменяется на число 7. Теперь на электрической схеме ты сможешь легко найти, как все подключается. Определяемые в исходном коде номера выводов нужно изменить по описанию твоего дисплея, так как они могут быть разными у разных дисплеев.

На этом мы закончим тему дисплеев. В завершение я только подскажу тебе еще одну команду:

```
lcd.clear()
```

С ее помощью можно удалить все содержимое с дисплея, чтобы можно было отобразить что-то новое.

# 6

## Заключение

В этой главе мы немного поработали с ЖК-дисплеем. Ты познакомился с важными техниками, которые помогут сделать твои проекты на Arduino более удобными в эксплуатации. Ты узнал:

- что такое ЖК-дисплей;
- какие выводы определены для питания и управления контрастом;
- как управлять дисплеем.

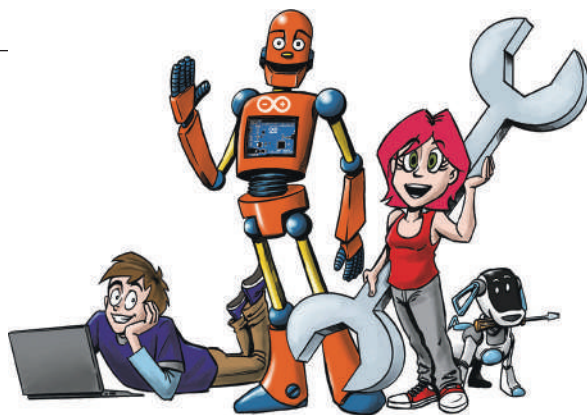
В целом мы разобрали темы, к которым я еще буду возвращаться в следующих главах.

## Несколько вопросов...

1. Что значит сокращение ЖК-дисплей?
2. Какая библиотека нужна, чтобы управлять ЖК-дисплеем?
3. Обязательно ли нужен потенциометр, чтобы изменить контраст?

## ...и задание

1. Построй схему, с помощью которой можно поприветствовать каждого человека лично по имени. Это можно сделать, например, с использованием последовательного порта или различными кнопками.



# 7

## Arduino и мультиметр

В этой главе мы будем в большей степени заниматься не Arduino, а компонентами схем. Это значит, что мы попробуем проверить неправильные схемы, спаянные или собранные на макетной плате.

При этом я планирую далее рассмотреть такие вопросы:

- ⊙ устройство мультиметра;
- ⊙ какой лучше: цифровой или аналоговый;
- ⊙ назначение отдельных функций;
- ⊙ измерение силы тока;
- ⊙ измерение напряжения;
- ⊙ измерение сопротивления резисторов;
- ⊙ «прозвонка» схемы.

### Для начала история из жизни

Чтобы начать главу, я решил рассказать небольшую историю на тему сбоя схем. Давно, когда я еще учился в школе, мы паяли на уроке одну схему, и у моего одноклассника возникла ошибка, которую было трудно обнаружить. Чтобы найти ошибку, я взял мультиметр и быстро смог установить, что у одноклассника разрыв в месте пайки, в результате чего ток не проходил. Это реальный пример, когда

## 7

мультиметр сэкономил время на длительную повторную пайку, потому что мы смогли быстро обнаружить ошибку, а не проверять все детали.

## Какие сведения нам нужны?

Сначала нам нужно решить, какой тип изображений мы хотим видеть на мультиметре: на дисплее (цифровой) или на цифровой шкале (аналоговый стрелочный). Я предпочитаю цифровые мультиметры, поэтому именно их я использую в книге. Преимуществом цифрового мультиметра является то, что возможно отображение значения с установленной единицей измерения, в то время как со стрелочным мультиметром нужно всегда держать в голове эту единицу (например, ампер или миллиампер). С помощью мультиметра, который я использую в книге, можно измерить сопротивление, напряжение и силу тока.

Вот изображение мультиметра, который я использую.



Если ты согласишься посмотреть на рукоятку переключения измеряемой величины (в середине), то увидишь восемь различных секторов (покрашенных в разный цвет). Связанные диапазоны всегда измеряют одну и ту же физическую величину (например, напряжение), но в разных диапазонах. Начиная сверху по направлению часовой стрелки можно увидеть:



мультиметр выключен («OFF»), переменное напряжение ( $V\sim$ ) в вольтах, *постоянный ток* ( $A\text{ ---}$ ) в миллиамперах, измерение частоты, проверка батарей, *проверка диодов*, *сопротивление* ( $\Omega$ ) в омах, килоомах и мегаомах и *постоянное напряжение* ( $V\text{ ---}$ ) в вольтах и милливольты. Курсивом отмечены функции, которыми мы будем заниматься в этой главе.

## Измерения постоянного тока

Сначала мы будем измерять напряжение постоянного тока (чаще говорят просто постоянное напряжение). Это делают, подключая щупы параллельно электрической цепи.

Прежде всего нужно установить наш мультиметр на 20 В постоянного напряжения (на изображенном выше мультиметре это наверху слева, серый диапазон с цифрой 20). Это значит, что мы можем измерять напряжение до 20 В. Кстати, желательно начинать в самом большом диапазоне напряжения, но у нашего мультиметра он слишком большой для Arduino.

Для измерения построим обычную схему – просто возьмем светодиод и соединим его с Arduino. Затем мы приложим оба вывода мультиметра к ножкам светодиода (на следующем рисунке это оранжевые провода).

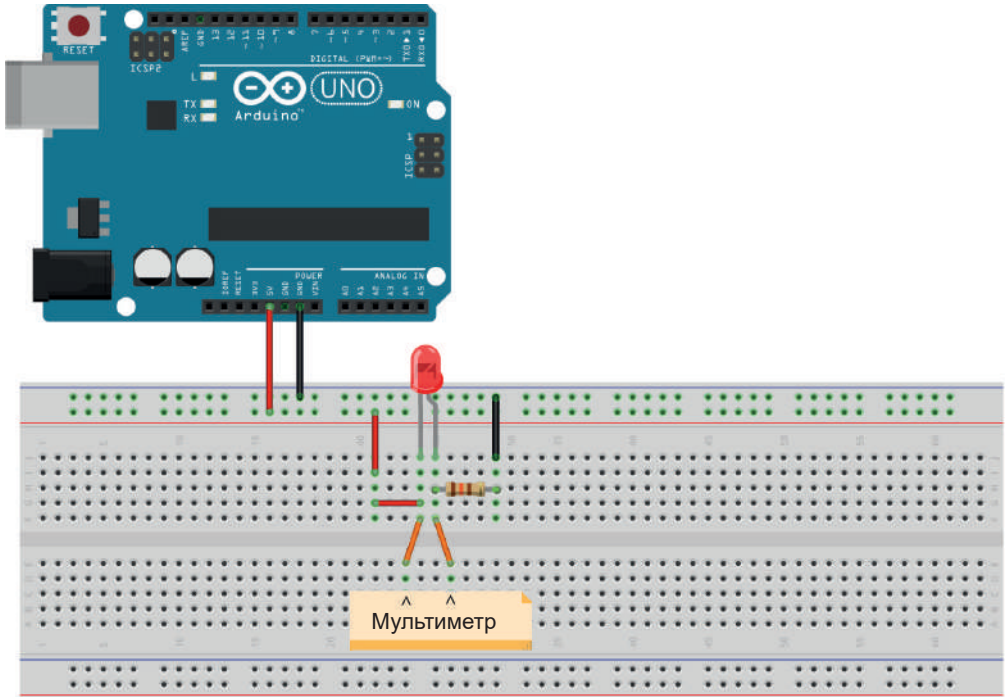
Если ты все правильно соединил, мультиметр должен показать в диапазоне 1,5–3 В, в зависимости от типа использованного светодиода. Если перед цифрой стоит «-» (минус), значит, ты неправильно подключил мультиметр (перепутал полярность щупов).

Единица вольт служит в физике для измерения напряжения, которое можно рассматривать как своего рода давление, возникающее, когда электроны перемещаются по проводу (сравни с потоком воды). Наш Arduino постоянно выдает 5 В (В = вольт) и некоторые другие напряжения, которыми мы здесь не пользуемся. Это питающее напряжение. Мультиметром мы в примере выше измеряем напряжение, которое из этих 5 вольт приходится на определенный компонент (как говорят электронщики, сколько «падает» вольт напряжения на этом компоненте).





7



fritzing

## Измерение силы тока

Сила тока показывает, сколько электронов проходит через определенное место проводника. Сила тока выражается в амперах. Мультиметром мы можем измерить, течет ли ток по проводнику и насколько он сильный.

Чтобы измерить силу тока, нам нужно настроить на мультиметре 200 мА (= миллиампер, тысячных долей ампера). Это на желтом диапазоне справа (в направлении «на 3 часа», как говорят военные). В отличие от напряжения, для измерения силы тока всегда необходимо последовательное подключение к сети, поскольку мы измеряем не «давление», а «течение» – количество прошедших частиц-электронов. Иными словами, для измерения силы тока цепь надо разорвать и в «разрыв» подключить мультиметр.

На следующем рисунке ты видишь две схемы. Мы измеряем силу тока между парами оранжевых проводов, помеченных буквами. Чтобы схема работала, все пары оранжевых проводов, кроме тех, к которым подключен мультиметр, должны быть соединены перемычками (схема не отличает перемычку от мультиметра в режиме измерения тока).

Остерегайся включать мультиметр, установленный в режиме измерения тока, параллельно компонентам в цепи и особенно подключать его к выводам источника питания – могут сгореть либо компоненты, либо предохранитель мультиметра, до которого очень трудно добраться. Если ты попытаешься сделать наоборот – подключить в разрыв цепи мультиметр в режиме измерения напряжения, то ничего страшного не произойдет, но схема работать не будет, а мультиметр покажет напряжение, близкое к напряжению источника питания (5 В).



Проверив сейчас ток через светодиод в обычной его схеме подключения (схема А, мультиметр включается между оранжевыми проводами, помеченными А1), мы обнаружим, что его величина находится в пределах нескольких миллиампер и, как и в случае напряжения, зависит от типа светодиода.

На второй схеме (В) все значительно интереснее: там к источнику подключены два светодиода параллельно. Измерив ток между точками В1 и В4, ты получишь одинаковые значения. Но если взять точки В2 и В3, будут получены два разных значения, так как ты используешь светодиоды двух разных цветов. Сумма этих двух значений будет равна силе тока в точках В1 и В4. Это можно объяснить следующим образом: в точке В1 ты измеряешь ток, который «поступает» в схему со стороны положительного вывода источника питания. В точке В4 – ток, который «покидает» схему, утекая к отрицательному выводу источника. Но это один и тот же ток, так как количество входящих в схему и выходящих из нее электронов всегда одинаково.

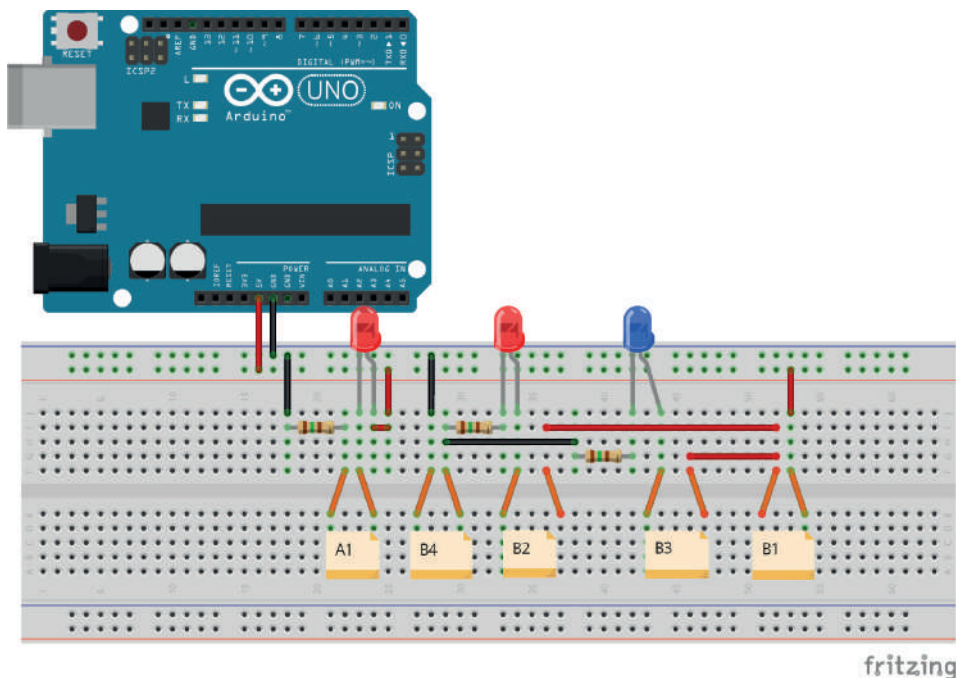
Интересно, что в точках В2 и В3 ток выбирает один из путей, чтобы достигнуть «выхода» (точки В4). Светодиодам разных цветов требуется разное количество электронов, чтобы загореться, вот поэтому в этих точках показатели силы тока отличаются. Но поскольку через схему течет одно и то же число электронов, сумма значений тока в точках В2 и В3 и дает показатель, который мы получаем при измерении на входе В1 или выходе В4.

Это можно сравнить с автогонками. Например, на старте зарегистрировались десять автомобилей, которые принимают участие в гонке. Если мы предположим, что ни с одним автомобилем не случилась авария, то все автомобили придут к цели, как и в случае с электронами. Теперь допустим, что гоночная трасса разделяется на две трассы разной ширины. Если, например, на пути В будет три машины, а на

## 7

пути А – семь, то на обоих путях будет вместе десять автомобилей. То же самое с нашими электронами в схеме.

Если ты сейчас согласишься на схему, то поймешь, что электроны должны проходить через мультиметр, чтобы электрическая цепь была неразрывна; то есть мультиметр должен быть подключен. Если ты согласишься на схему для измерения напряжения, ты заметишь, что там электроны проходят как через мультиметр, так и мимо него, независимо, подключен он или нет.



Итак, мы разобрали уже две из обсуждаемых в этой главе тем.

## Измерение сопротивления

Мы подошли к теме, изучение которой я бы хотел подкрепить небольшим практическим действием, как и при измерении тока или напряжения. Ты уже знаком с принципом действия потенциометра. Ты знаешь, что сопротивление меняется при повороте, но не знаешь, в каком диапазоне.

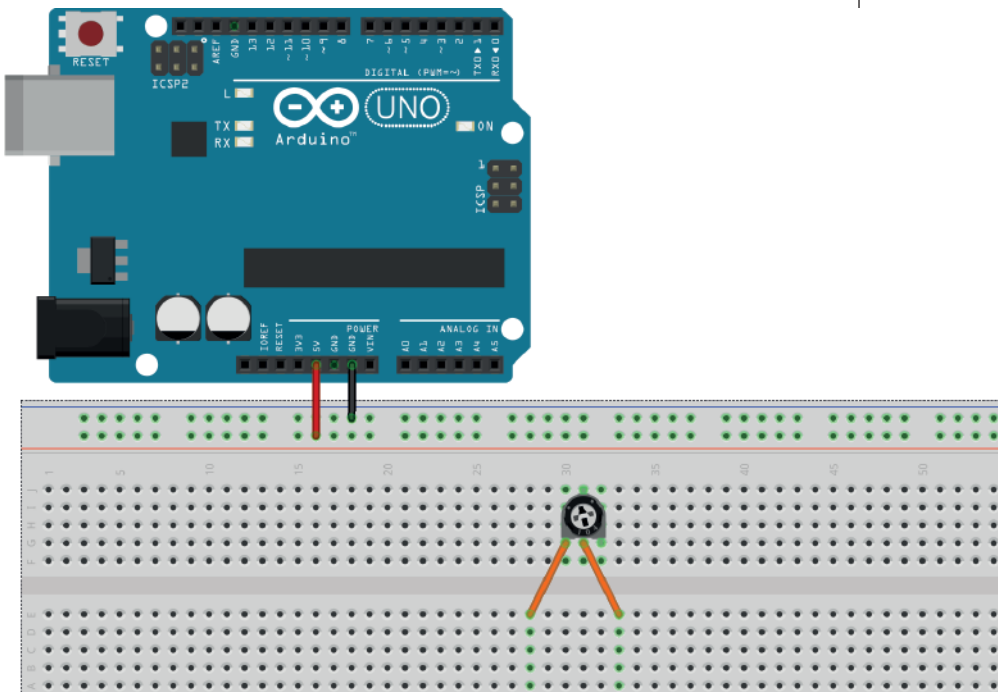
Сначала небольшое задание. Если ты запомнил цветные коды на резисторе, тем лучше. В противном случае не смотри на цветные кольца. Установи мультиметр на желтое поле внизу слева 2000 Ом или 20 кОм. Теперь возьми любой резистор и подключи к нему оба щупа мультиметра. Ты сра-

зу увидишь значение сопротивления на дисплее мультиметра (возможно, придется перейти на другие диапазоны, если сопротивление превышает установленный диапазон). Измеренное сопротивление немного отличается от номинального значения, обозначенного цветными кольцами, поскольку резисторы имеют *величину допуска* (то есть будут не совсем точно совпадать с заданным значением). В основном допуск составляет 5% (золотое кольцо), но может быть и 10% (серебряное) или, очень редко, 1% (коричневое).

Как рассчитать значение допуска? Например, у нас резистор на 130 Ом и с 5%-ным допуском (т. е. цветовой код: коричневый, оранжевый, коричневый, золотой). Расчет осуществляется так:  $130 \text{ Ом} \times 0,05 = 6,5 \text{ Ом}$ . Таким образом, значение резистора 130 Ом может отклоняться на 6,5 Ом, то есть составлять 123,5 Ом, или 136,5 Ом, или любое другое значение в этих пределах.

## Изучаем потенциометр

Вернемся к нашему эксперименту. Чтобы узнать немного больше о потенциометре, построй следующую схему (потенциометр, как и ранее, имеет номинал 1,5 кОм):



# 7

Теперь нужно подключить оба оранжевых кабеля к мультиметру. Установи переключатель режимов на диапазон «2000» (Ом) для измерения сопротивления. На дисплее мультиметра должно появиться значение, которое будет меняться при вращении ползунка потенциометра (обычно это делается с помощью отвертки или ручки, если потенциометр ее имеет). Это происходит потому, что в потенциометре находится дорожка из угля, которая оказывает сопротивление проходящему току, а ползунок позволяет установить контакт на разном расстоянии от ее концов.

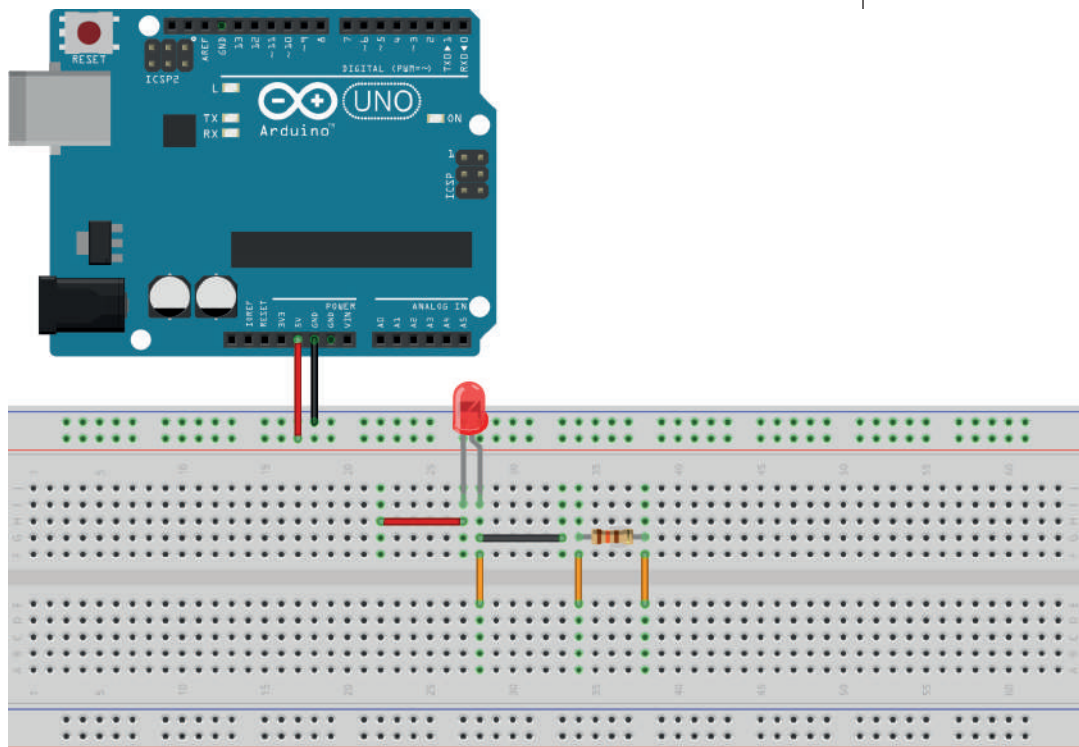
## Измерение сопротивления

А теперь давай узнаем сопротивление светочувствительного резистора. Для этого просто подключим резистор к мультиметру и измерим сопротивление. Конечно, чтобы было интереснее, нужно держать резистор на свету. Согласно моим измерениям, показатель колеблется между 3 и 18–19 кОм, в зависимости от освещенности. Чем светлее, тем ниже сопротивление.

Кстати, на некоторых мультиметрах можно с помощью специальной кнопки поставить отображаемый результат на паузу.

## Метод прозвонки

С помощью метода прозвонки можно проверить, есть ли соединение между двумя точками схемы. На моем мультиметре есть функция, чтобы проверить диоды, но мы можем также проверить детали на протекание тока. Для этого схема не должна находиться под напряжением, то есть нам нужно отсоединить Arduino от кабеля USB. Построй следующую схему:



fritzing

Здесь есть небольшая ошибка, на которую ты сразу должен обратить внимание. Резистор не соединен с питанием. Чтобы это проверить, мы используем функцию проверки диодов (внизу, помечена красным значком диода  $\rightarrow$ ). Мультиметр должен показать 1, если щупы ни к чему не подключать. Если соединить оба измерительных щупа, должна отобразиться цифра 3 (по крайней мере, на моем мультиметре, в других может показывать ноль или какие-то еще цифры). То есть если соединение есть, устройство покажет любую другую цифру, кроме единицы. Так мы можем доказать наличие ошибки в схеме (даже если мы ее знаем). Многие мультиметры имеют специальный режим «прозвонки» и издают звуковой сигнал при наличии контакта (отсюда и название этого приема).



Обнаружить соединение можно также в режиме омметра: при наличии соединения отобразится более низкое значение или ноль. Если соединение обнаружено там, где его быть не должно, или, наоборот, не обнаружено там, где оно должно быть, то проверь схему визуально.



# 7

## Заключение

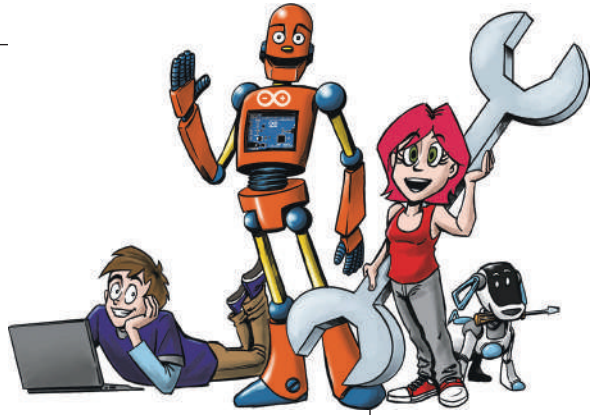
В этой главе ты познакомился с использованием мультиметра. Этот инструмент имеет гораздо больше функций, но для нас достаточно тех, что я показал.

## Несколько вопросов...

1. С помощью чего определяется наличие или отсутствие контакта? Что нужно для этого настроить?
2. Сопротивление можно измерять прямо в работающей схеме или при выключенном питании?
3. При параллельном подключении измеряются амперы/сила тока или вольты/напряжение?

## ...и несколько заданий

1. Найди в интернете информацию о различиях между осциллографом и мультиметром, обрати внимание также на цену.
2. Измерь все имеющиеся у тебя резисторы и распредели их по соответствующим категориям.
3. В интернете или в других книгах найди информацию о значении остальных функций мультиметра.
4. Построй небольшое устройство на основе мультиметра для измерения напряжения батарей; при этом тебе нужно построить крепление для мультиметра и его щупов, а также оборудовать гнездо для подключения батареи. Это требует определенных технических навыков.
5. Измерь несколько значений напряжения и тока на других схемах из книги, заведи журнал измерений. Так ты потренируешься в использовании мультиметра и его единиц измерения.



# 8

## Arduino online

В этой главе мы будем заниматься платой Arduino Ethernet Shield. С помощью этой платы можно установить соединение с сетью. Для этой главы тебе понадобится Arduino Uno, так как плата Shield совместима только с ним.

Ты узнаешь:

- ⦿ что такое Shield и как ее использовать;
- ⦿ как использовать язык разметки текста HTML;
- ⦿ как с помощью Arduino создать небольшой веб-сервис.

В начале этой главы ты изучишь основы так называемого «языка разметки» HTML. Если ты его уже знаешь, можешь пропустить эту главу.

Также можешь сразу перейти к следующей главе, если тебе не интересна эта тема.

## HTML – ворота в интернет

HTML (Hypertext Markup Language, язык разметки гипертекста) – это специальный язык, который делает из обычного текста форматированный и пригодный для интернета. При этом отдельные команды напрямую превращаются в текст и потом интерпретируются браузером (например, IE, Firefox, Chrome и т. д.). Следующий код является примером HTML:



## 8

```
<html>
<body>
Обычный текст превращается в текст с <b>полужирным</b>,
<i>курсивным</i> или <u>подчеркнутым</u> написанием.
</body>
</html>
```

Для написания можно использовать любой текстовый редактор, код потом сохраняется в файле с окончанием .htm или .html (все равно, какое больше нравится). Результат в браузере выглядит так:

Обычный текст превращается в текст с **полужирным**, *курсивным* или подчеркнутым написанием.

Ты мог заметить, что начальный тег (так называется код в угловых скобках) должен всегда закрываться вторым, чтобы прекратить действие команды (<b></b>). Буквы – английского алфавита. Следующие теги ты вначале будешь использовать чаще всего:

Тег	Значение (расшифровка)	Объяснение
<html></html>	HTML-документ	Вставляет исходный HTML-код в один файл
<title></title>	Title	Определяет название страницы в заголовке браузера
<b></b>	Bold	Полужирный шрифт
<u></u>	Underline	Подчеркивание текста
<i></i>	Italic	Курсив
<a href="www.wikipedia.org">Отображаемый текст</a>	Ссылка	Открывает сайт <a href="http://www.wikipedia.org">www.wikipedia.org</a> , если кликнуть на эту ссылку (обычно выделяется синим цветом с подчеркиванием)
<body></body>	Body (= тело)	Выделяет главную часть страницы с содержательным текстом
 	break;	Переводит на новую строку (второй – закрывающий – тег здесь не нужен)

Чтобы последний тег таблицы (<br>) был тебе более понятен, вот еще один небольшой код:

```
<html>
<body>
HTML совсем не сложен,<br>
поэтому я пишу сейчас код<br> и
создаю ссылку на статью в Википедии о HTML:
<a href="http://en.wikipedia.org/wiki/HTML">HTML в англ.
```

```
Википедии. </а>  
</body>  
</html>
```

Тогда все выглядит так:

HTML совсем не сложен, поэтому я пишу сейчас код и создаю ссылку на статью в Википедии о HTML: [HTML в англ. Википедии](#)<sup>1</sup>.

(Обрати внимание, что в оригинале третья строка начинается после предлога «и», но мы указали тег `<br>` перед ним, и браузер последовал нашей команде, а не изначальному форматированию строк, которое может быть каким угодно.)

HTML предлагает значительно больше команд форматирования, но в таблице пока те, которые мы будем использовать, так как мы хотим сосредоточиться на подключении к сети с помощью Arduino, а не на создании хорошего сайта (это отдельная «наука»).



## «Сеть, нам нужна сеть»

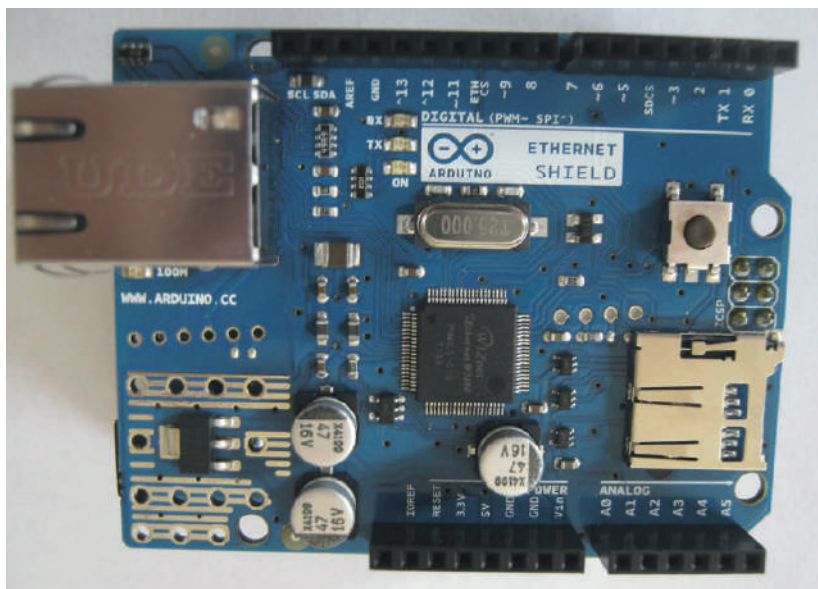
Однако прежде, чем мы создадим подключение к сети, нам нужно достать нужную плату (Shield) для Arduino.

Но что же такое Shield (в русском интернете их часто так и называют – «шилды»)? Shield – это плата-расширение, которая устанавливается на Arduino Uno и добавляет ему функций. В данном случае нам нужны *сетевые функции*, которые и реализует Arduino Ethernet Shield.

Для установки Ethernet Shield необходимо, чтобы четырехугольный Ethernet-разъем на ней смотрел в том же направлении, что и USB-порт Arduino.

Чтобы вставить Shield, тебе нужно немного надавить на нее сверху, чтобы плата вошла в гнезда Arduino. После продолжительного использования будет значительно легче соединять эти два «компонента».

<sup>1</sup> <http://en.wikipedia.org/wiki/HTML>.



Затем с помощью Ethernet-кабеля (разъем RJ-45) соедини Ethernet Shield с роутером и подключи USB-кабель к Arduino, как обычно. Код для достижения первого результата относительно простой. В нем требуется установить конкретный IP-адрес, по которому Arduino будет откликаться на запросы сети. Адрес, в принципе, можно указать любой, но для надежности лучше выбрать соответственно твоей домашней сети (это можно настроить в роутере). Большинство роутеров для домашних сетей по умолчанию предоставляет адреса из диапазона 192.168.0.0–192.168.0.255. Например, сам роутер в моей сети можно найти по IP 192.168.0.1, а Arduino тогда обнаружится по указанному в коде адресу 192.168.0.99 (адрес не должен совпадать с адресами других устройств в домашней сети). Кроме IP-адреса, необходим так называемый MAC-адрес Ethernet-контроллера, он может быть написан прямо на плате, или можно указать любой другой (например, тот, что приведен в этом примере).

```
#include <SPI.h> //подключение библиотек
#include <Ethernet.h>
byte mac[] = {0x90, 0xA2, 0xDA, 0x0D, 0xB4, 0x8C}; //MAC-адрес
IPAddress ip(192,168,0,99); //IP-адрес
EthernetServer server(80); //номер порта

void setup() {
  Ethernet.begin(mac, ip);
```

```
server.begin();
}

void loop() {
  EthernetClient client = server.available();
  if (client) {
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        //Подождать, готов ли запрос браузера
        char c = client.read();
        if (c == '\n' && currentLineIsBlank) {
          //Ответ браузера
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println();
          //Начни с ответа пользователю
          //--Ответ--\
          client.print("Привет из Ethernet");
          client.println("<br />");
          //--Конец ответа--\
          break;
        }
        if (c == '\n') {
          //Запрос браузера закончен
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          //Браузер продолжает отправлять запросы
          currentLineIsBlank = false;
        }
      }
    }
    delay(1);
    client.stop();
  }
}
```

Этот код действует таким образом, что при вводе установленного в коде IP-адреса в адресную строку браузера ты видишь в браузере небольшое приветствие «Привет из Ethernet».

Сначала браузер отправляет так называемый заголовок HTTP. Мы считываем его с помощью переменных. Как только браузер готов, мы тоже отвечаем заголовком:

## 8

```
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: text/html");
client.println();
```

Только после этого появляется текст, который также адресован «обычному пользователю»:

```
client.print("Привет из Ethernet");
client.println("<br />");
```

Сначала давай немного поэкспериментируем. Мы расширим этот код так, чтобы он правильно отображал пример выше. Вот еще раз HTML-код:

```
<html>
<body>
HTML совсем не сложен,<br>
поэтому я пишу сейчас код!
</body>
</html>
```

Для этого нам нужно доработать скетч в области «ответ» до «конец ответа». Код в этой области будет выглядеть так:

```
//-Ответ-\\
client.print("<html><body>");
client.print("HTML совсем не сложен,<br>");
client.print("поэтому я пишу сейчас");
client.print(" код!<br>");
client.println("</body></html>");
//-Конец ответа-\\
```

На основе этой системы ты можешь теперь выводить целые веб-сайты.

Даже во время нашего небольшого экскурса в веб-программирование ты наверняка заметил, что работать с Ethernet Shield сложно.

## Заключение

В этой главе мы немного разобрались, что такое веб-программирование и тексты в HTML. Мы даже разработали свой небольшой веб-сервер – то есть то, что посылает информацию. Другим примером мог бы быть клиент (то, что информацию принимает и отображает для пользователя),

но для этого у нас есть браузер в компьютере. С помощью Ethernet Shield можно сделать гораздо больше: например, выводить значения с датчиков (см. измерение яркости, глава 4) или управлять бытовыми приборами через смартфон.

Ты изучил:

- форматирование веб-сайтов с помощью HTML;
- отображение веб-сайтов с помощью Arduino и Ethernet Shield;
- а также слегка познакомился с серверным программированием.

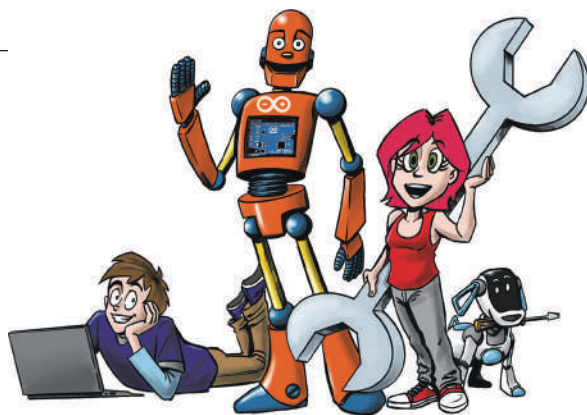
## Несколько вопросов...

1. Что такое Shield?
2. Какую плату Shield мы использовали?
3. С помощью каких команд можно создавать ссылки на HTML-документы?
4. Какой заголовок мы отправляем браузеру?
5. Текст для чтения находится в какой-то отдельной области?

## ...и несколько заданий

1. Попробуй сделать так, чтобы в браузере отображались прочитанные переменные (например, значение на цифровом выводе).
2. Дополни программу так, чтобы на HTML-странице отобразилась краткая автобиография.
3. Найди в других источниках больше информации о HTML, а также о CSS, чтобы создать еще более интересный сайт.
4. Если тебя заинтересовало веб-программирование, можешь попробовать также язык программирования PHP.
5. И наконец: разработай веб-сайт, который в виде краткой таблицы будет отображать значения на всех выводах Arduino. При этом для всех аналоговых входов должны быть указаны аналоговые значения.





# 9

## Клавиатура с Arduino Leonardo

Через несколько лет после того, как вышел Arduino Uno, был разработан *Arduino Leonardo*. У него есть небольшие отличия в составе компонентов на плате и, следовательно, некоторые дополнительные функции. В этой главе я хотел бы поработать с самыми интересными функциями Arduino Leonardo. В частности, мы разработаем основу для клавиатуры.

В этой главе мы:

- ⊙ найдем различия между Uno и Leonardo с точки зрения состава компонентов;
- ⊙ протестируем Leonardo;
- ⊙ разработаем клавиатуру для компьютера;
- ⊙ запрограммируем физический (аппаратный) ключ-брелок для обеспечения секретности, в котором будут храниться пароли.

Лично я считаю последнюю тему очень интересной, а именно разработку секретного ключа с паролями. На английском языке подобные компактные устройства, предназначенные для удостоверения личности владельца, называют «токенами». Но токен – довольно многозначный термин (токены употребляют в программировании, в финансах, в настольных играх, в науках филологии и семиотике и т. д.), поэтому мы здесь не будем им пользоваться.

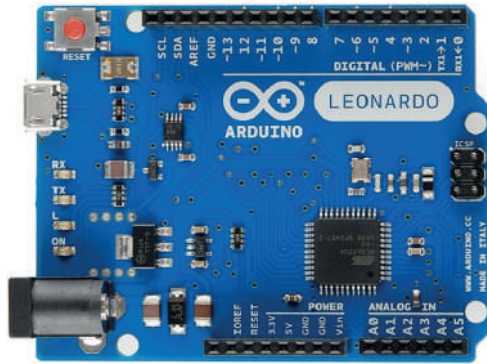


## 9

## Первые шаги с Leonardo

На первый взгляд, Arduino Leonardo выглядит так же, как Arduino Uno, но на самом деле есть несколько существенных различий. Во-первых, отличается расположение выводов, в результате чего нельзя использовать некоторые платы Shield, в том числе Ethernet Shield. Во-вторых, здесь установлен разъем микро-USB, а не обычный квадратный, как у UNO, так для соединения с ПК требуется кабель другой конфигурации. Если ты внимательно посмотришь на плату, то заметишь, что на Leonardo установлен миниатюрный SMD-процессор (в отличие от процессора у Arduino Uno, его нельзя снять и заменить).

А самое главное – здесь отсутствует отдельная микросхема контроллера, обеспечивающего соединение с компьютером. Эту задачу теперь выполняет основной микроконтроллер Leonardo. В результате можно напрямую соединяться с компьютером – отправлять команды мыши и клавиатуры, как будто это настоящие мышь и клавиатура. Но поскольку процессор при программировании должен автоматически сбрасываться, при сбросе ненадолго прерывается соединение с ПК.



### Включение светодиода

Давай протестируем это сами. Нам нужно включить светодиод.

```
void setup() {
    pinMode(13,OUTPUT);
}
```

```
void loop() {  
  digitalWrite(13,HIGH);  
  delay(1000);  
  digitalWrite(13,LOW);  
  delay(1000);  
}
```

Если ты сейчас попробуешь загрузить код, то получишь сообщение об ошибке. До сих пор мы всегда работали с Arduino Uno. Для Leonardo скетч нужно компилировать по-другому. Поэтому выбери в меню **Инструменты** | **Плата** Arduino Leonardo. Если теперь загрузить код, светодиод начнет гореть.

Ты можешь проверить следующее: подключи Leonardo к USB-кабелю, открой в Windows Диспетчер устройств и найди в группе **Порты COM и LPT** Leonardo. Затем в IDE Arduino кликни на кнопку **Загрузить**. Leonardo ненадолго исчезнет в окне и потом снова появится на том же месте. Можно поступить и иначе (это будет работать в Linux и MacOS тоже): нужно обратиться только к IDE Arduino, кликнуть на кнопку **Загрузить** и искать Leonardo в списке **Инструменты** | **Последовательные порты**. Если не получается быстро кликнуть, открой список снова и нажми на плате Leonardo кнопку перезагрузки. Здесь порт тоже должен ненадолго исчезнуть и снова появиться.

## Эмуляция клавиатуры

А теперь давай займемся эмуляцией клавиатуры (*эмулировать* у нас здесь означает «подделывать»). Для этого мы используем следующий код:

```
#include <Keyboard.h>  
  
void setup() {  
  Keyboard.begin();  
  Serial.begin(9600);  
}  
  
void loop() {  
  if (Serial.available() > 0) {  
    char input = Serial.read();  
    Keyboard.print(input);  
  }  
}
```

## 9

Этот код делает следующее: сначала мы запускаем эмуляцию клавиатуры и последовательный порт. Если сейчас мы получим через последовательный порт букву, с помощью `Keyboard.print(input)` она будет послана обратно в компьютер так, как будто вы нажали ее на своей клавиатуре. Само по себе это не очень интересно. Поэтому давай попробуем следующий код:

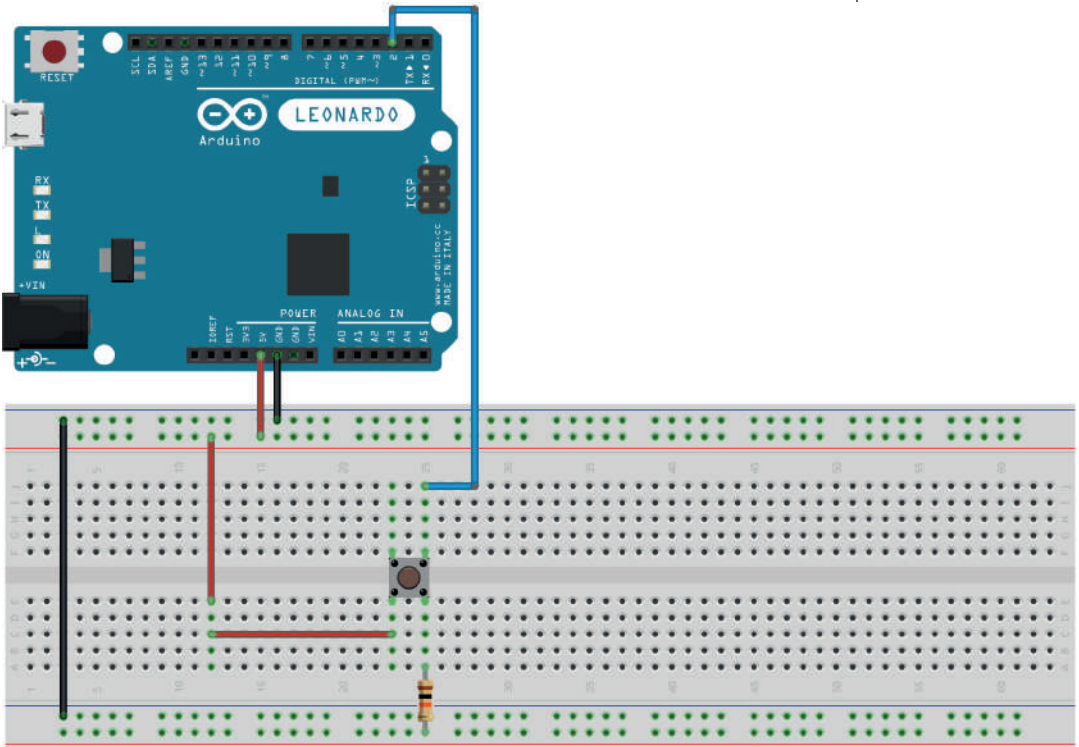
```
#include <Keyboard.h>

void setup() {
  Keyboard.begin();
  Keyboard.print("First text from the keyboard");
  //"Первый текст с клавиатуры"
}
void loop() {}
```

Этот код при старте выводит текст «First text from the keyboard» (что переводится как «первый текст с клавиатуры» – к сожалению, как и в случае ЖК-дисплея, библиотека `Keyboard` без дополнительных усилий может передавать только буквы английского алфавита). Чтобы увидеть это, открой текстовый редактор Блокнот, кликни на текстовое поле и сбрось Arduino Leonardo кнопкой на его плате. Теперь текст должен появиться в текстовом поле.

## Первая маленькая клавиатура

Чтобы не приходилось для набора текста каждый раз сбрасывать Leonardo, мы установим кнопку, чтобы с ее помощью выводить текст. Построй следующую схему:



fritzing

Как и в главе 1, здесь мы подключили кнопку к выводу 2. Она должна стать основой для минимальной клавиатуры. Теперь нам нужно сделать так, чтобы при нажатии кнопки отображалась буква «Е». Для этого мы возьмем следующий код (попробуй самостоятельно разработать решение и только потом посмотри на мой скетч):

```
#include <Keyboard.h>

int inp = 2; //inp->Ввод пин/кнопка
void setup() {
  Keyboard.begin();
}

void loop() {
  if(digitalRead(inp)==1) {
    Keyboard.println("E") //Английская буква!!
  }
}
```

## 9

Этот простой код всего лишь спрашивает, нажата ли кнопка. Если да, на клавиатуре выводится знак. Протестировав этот код, ты заметишь, что чем дольше ты нажимаешь на кнопку, тем больше букв «Е» выводится. Давай перепишем код так, чтобы на каждое нажатие выводилась только одна «Е»:

```
#include <Keyboard.h>

int inp = 2; //inp->Ввод пин/кнопка
void setup() {
    Keyboard.begin();
}

void loop() {
    if(digitalRead(inp)==1) {
        Keyboard.print("E") //английская буква!!
        while(digitalRead(inp)==1) {
        }
    }
}
```

Этот код делает следующее: он спрашивает, нажата ли кнопка. Если да, он отправляет «Е». Если кнопка все еще нажата, запускается пустой цикл while. Этот цикл закончится, когда ты снова отпустишь кнопку.

Но для полноценной клавиатуры нам не хватает еще двух деталей: написание заглавными буквами и остальные кнопки. Вот новый скетч, а далее электрическая схема:

```
#include <Keyboard.h>

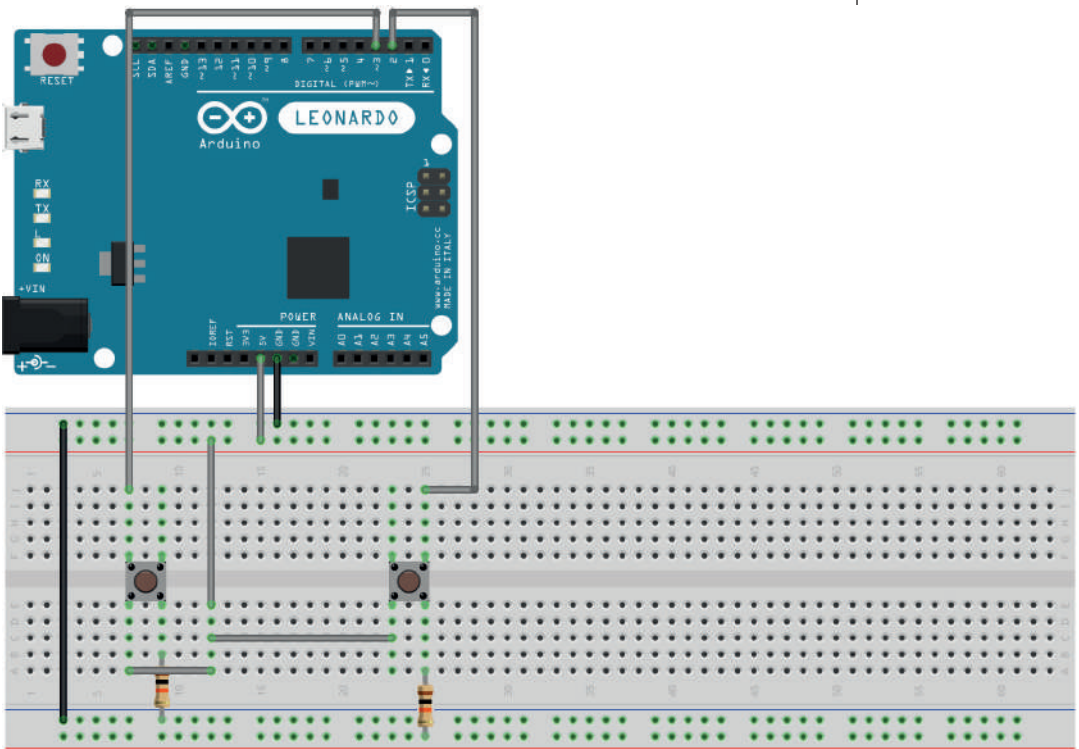
int inp_e = 2; //inp_e->Ввод (клавиша E)
int cap = 3; //cap -> Capitel (Заглавные буквы)

void setup() {
    Keyboard.begin();
}

void loop() {
    if(digitalRead(inp_e)==1) {
        if(digitalRead(cap==1)) {
            Keyboard.write("E");
        }
        else {
            Keyboard.write("e");
        }
    }
}
```

```
    }  
    while(digitalRead(i_np_e)==1) {  
    }  
  }  
}
```

В этом коде мы сначала проверяем, нажата ли кнопка «Е». Если она нажата, мы проверяем, нажата ли также «Сар». Если да, мы выводим «Е», если нет – «е». Затем в обоих случаях следует цикл while, если клавиша (кнопка) еще нажата. Вот схема с обеими кнопками. Верхняя – кнопка «Е», нижняя – кнопка «Сар»:



fritzing

**Совет:** если ты хочешь написать заглавную «Е», должны быть нажаты обе кнопки одновременно. Если так не всегда получается, можешь сначала нажать «Сар».



Это основа для клавиатуры. При сборке целой клавиатуры нужно, по сути, много раз повторить вышеприведенную

## 9

схему (возможно, без кнопки «Cap») и размножить код. С Arduino Mega, имеющим много выводов, это была бы не проблема. Но Mega не может делать эмуляцию клавиатур и мышей, подобно Leonardo. Поэтому нам нужно найти какое-то другое решение, ведь у Leonardo не хватит выводов, чтобы сделать кнопки под каждую букву.

## Как устроена настоящая клавиатура

Поскольку это довольно трудоемко, мы не будем больше строить клавиатуру, но я расскажу об устройстве настоящей клавиатуры. Попробуй найти старую клавиатуру, которую тебе можно разобрать. Для этого нужно снять корпус, потом резиновую вкладку и пленку с проводящими дорожками. После этого останется большая зеленая печатная плата. На ней есть несколько рядов металлических контактов. Их можно использовать, чтобы отображать знаки на клавиатуре. Для этого нужно посредством кабеля подключить плату к компьютеру. Если взять перемычку и соединить два разных контакта, в текстовом редакторе на компьютере появятся символы.

## Ключ обеспечения секретности

Мы подошли к последней теме этой главы, секретному ключу для хранения и ввода паролей. Речь идет о небольшом устройстве, которое экономит время на ввод пароля и при этом не сохраняет его на компьютере. В некоторых браузерах (например, Chrome или Firefox) можно прочесть все пароли, которые сохранил пользователь. А мы сохраним пароли на Arduino, где их будет немного сложнее прочесть. Первый вариант сможет хранить только один пароль. Мы можем запрограммировать также цель использования отдельных паролей. Этот ключ нужно беречь так же, как ключи от дома, потому что злоумышленникам его легко украсть. Используй следующий код и следующую схему:

```
#include <Keyboard.h>

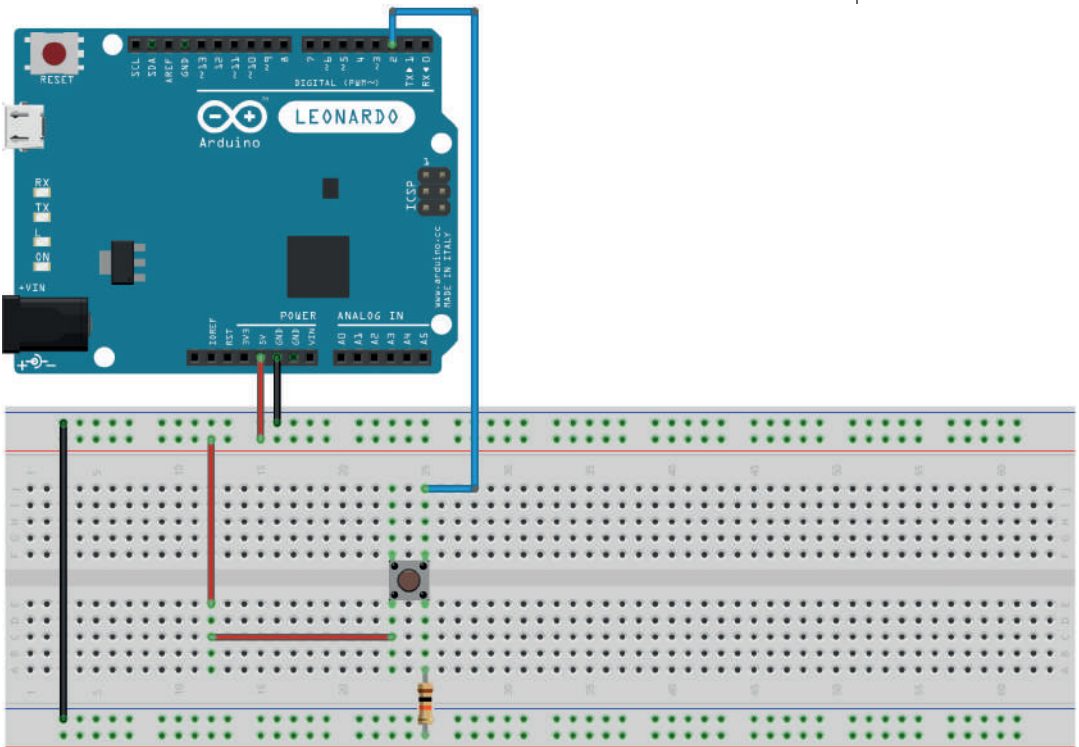
int buttonPin = 2;

void setup() {
  pinMode(buttonPin, INPUT);
  Keyboard.begin();
}
```



```
void loop() {  
  int buttonState = digitalRead(buttonPin);  
  if (buttonState == HIGH) {  
    Keyboard.print("31415"); //Мой секретный PWD  
                                //PWD -> Password = пароль  
    delay(1000); //Подожди, чтобы компьютер успел интерпретировать  
                //нажатия клавиш  
  }  
}
```

По сути, это не что иное, как исходный код клавиатуры, который мы совсем недавно использовали:



fritzing

Но чтобы его доработать, давай добавим код клавиши ввода **Enter** (перевода строки KEY\_RETURN):

```
#include <Keyboard.h>  
  
int buttonPin = 2;
```



## 9

```

int enterPin = 3;

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(enterPin, INPUT);
  Keyboard.begin();
}
void loop() {
  int buttonState = digitalRead(buttonPin);
  int enterState = digitalRead(enterPin);
  if (buttonState == HIGH) {
    Keyboard.print("31415"); //Мой секретный PWD
                               //PWD -> Password = пароль
    delay(1000); //Подождать, чтобы интерпретировать
                 //нажатие клавиши
  }
  if(enterState == HIGH) {
    Keyboard.press(KEY_RETURN); //Отправить код
                                //ввода Enter = конец строки
                                //Return-keycode

    delay(1000);
  }
}

```

Так мы можем вводить пароль из ключа и одновременно заполнять поля веб-формы с обычной клавиатуры. Конечно, плата Leonardo слишком велика, чтобы сделать законченное устройство, которым удобно пользоваться. Знай, что на этот случай есть ее миниатюрный аналог под названием Arduino Micro.

## Заключение

В этой главе мы довольно подробно изучили функции Arduino Leonardo. Лично я нахожу Leonardo интересным именно из-за его способности эмулировать клавиатуру и мышь. Ты увидел, как легко собрать свою собственную клавиатуру.

## Несколько вопросов...

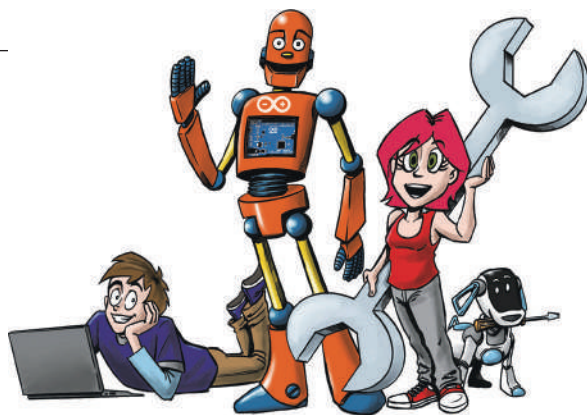
1. Микросхема у Leonardo больше или меньше, чем у Uno?
2. Можно ли исполнить приведенный выше код с Arduino Uno?

3. Сколько клавиш потребуется, чтобы реализовать функцию переключения регистра (прописных и строчных букв)?
4. Можно ли с помощью Arduino Leonardo собрать целую клавиатуру?
5. Что такое физический секретный ключ?

## ...и несколько заданий

1. Найди в интернете информацию о том, что такое виртуальный (программный) секретный ключ (вот тут вполне уместен термин «токен»).
2. Собери клавиатуру так, чтобы иметь возможность набрать свое имя (не забывай, что библиотека Keyboard не обучена русскому языку!). Повезло, если у тебя короткое имя.
3. Купи или найди где-нибудь (возможно, на чердаке, в подвале и т. д.) старую клавиатуру и разбери ее. Постарайся понять, как работают настоящие клавиатуры. Будет полезно подключить к плате Leonardo, соединить его с компьютером и переключать («закорачивать») два контакта переключкой из провода. Если сделать это правильно, можно «писать проводом».
4. Последнее задание этой главы: найди в интернете, как через Arduino Leonardo можно присвоить символу комбинацию **Ctrl+буква**. Попробуй собрать на макетной плате схему, с помощью которой ты сможешь копировать (**Ctrl+C**), вставлять (**Ctrl+V**) и вырезать (**Ctrl+X**) текст. Для этого используй либо три клавиши, которые реализуют сочетание **Ctrl+буква**, либо сделай клавишу управления и три клавиши для букв и используй пример для нашей клавиши переключения регистра **Caps**.





# 10

## Взгляд за пределы IDE

В этой главе мы будем заниматься приемами программирования за пределами IDE, потому что Arduino – это не весь мир и большинство программистов со временем меняют Arduino на профессиональные инструменты. IDE автоматически работает только со встроенными инструментами, но ведь мы можем все сделать и вручную.

Мы познакомимся с такими вопросами:

- ⊙ собственный язык Arduino, C++;
- ⊙ структура программ C/C++;
- ⊙ создание и перенос программы в IDE;
- ⊙ ручное создание скетча, компиляция и загрузка в Arduino.

Эта глава основана на использовании терминала (но не терминала последовательного интерфейса). Приемы, которые мы используем в данной главе, применяются также при разработке обычных компьютерных программ, если отказаться от любых IDE.

### C++, сердце Arduino

Возможно, ты уже немного занимался программированием на компьютере, помимо изучения Arduino. Познакомившись, например, с Java-программами, ты заметишь,

## 10

что их *синтаксис* (то есть грамматика и «правописание» языка программирования) во многом такой же, как у скетчей Arduino. Это потому, что Java ориентирован на синтаксис языка программирования C. C++ является расширением языка C. Arduino использует язык C++ в качестве основы для программирования. Сейчас ты узнаешь немного о C++ и синтаксисе C++ с диалектом Arduino. Диалект – это вариант языка, то есть Arduino – это вариант C++; диалект Arduino здесь для простоты называется Arduino C++.

Так же, как и у Arduino C++, у C++ есть функция `setup()`, но с другим названием. Функции `loop()` не существует, ее нужно создавать самостоятельно. Возьмем обычную пустую программу Arduino:

```
void setup() {
}
void loop() {
}
```

Из этого скетча мы можем создать C++-код. Получится примерно следующее:

```
#include "Arduino.h"

int main() {
  init();
  #if defined(USBCON)
    USB.attach();
  #endif setup();
  for (;;) {
    loop();
    if (serialEventRun) serialEventRun();
  }

  return 0;
}
```

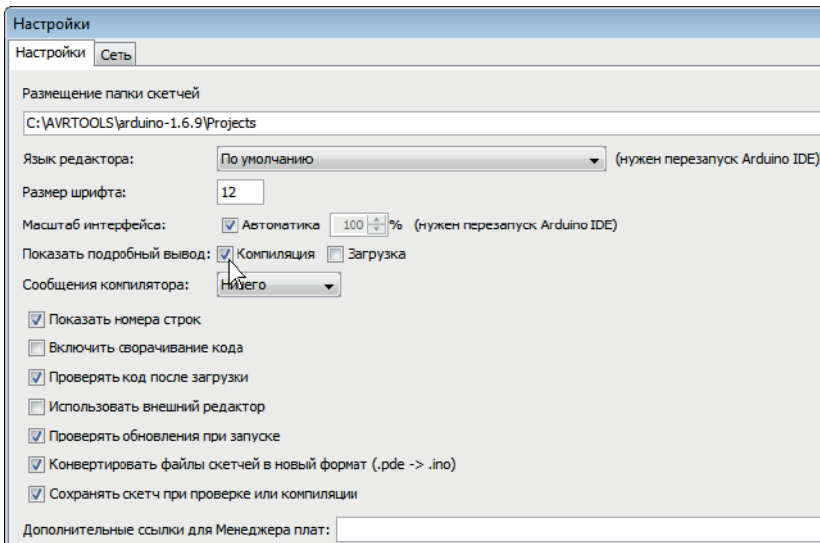
Как видишь, функции `setup()` и `loop()`, которые мы определяем в IDE, здесь вызываются специально. Функция `int main()` – это C++-эквивалент для `setup()` и `loop()`. Код внутри `setup()` выполняется однократно, но с помощью бесконечного цикла `for(;;)` конец файла никогда не достигается, и таким образом остаток соответствует циклу `loop()`. Чтобы код правильно исполнялся, нужно также определить функции `loop()` и `setup()`, но это делается за пределами данного исходного кода.

## Изучение фона IDE

Если мы сейчас возьмем чистую C++-программу, в таком виде, как она выполняется на компьютерах, то она будет выглядеть так:

```
int main() { //Соответствует setup();
return 0; //Возвращает 0, но для этой книги не важна
}
```

Эта программа выглядит как еще один вариант наших предыдущих скетчей. А теперь давай посмотрим на фоновые процессы в IDE во время работы. Для этого зайти в IDE в меню **Файл | Настройки** и в открывшемся окне поставь галочку перед пунктом **Компиляция**. После этого ты получишь подробное отображение всего происходящего при компиляции.



Закрой настройки и разверни в IDE черное окно внизу до максимального размера (просто потяни наверх). Теперь набери минимальный скетч (`void setup(){} void loop(){}`) и нажми кнопку для тестирования (кнопка-ссылка с галочкой).

## 10

The screenshot shows the Arduino IDE window titled "sketch\_oct31a | Arduino 1.6.9". The menu bar includes "Файл", "Правка", "Скетч", "Инструменты", and "Помощь". The sketch editor displays the following code:

```

1
2 void setup() {
3   // put your setup code here, to run once:
4
5 }
6
7 void loop() {
8   // put your main code here, to run repeatedly:
9
10 }

```

Below the editor, the "Компиляция завершена" (Compilation finished) message is displayed. The terminal window shows the following compilation commands:

```

"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-ar" rcs "C:\Users\YURYI
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-ar" rcs "C:\Users\YURYI
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-ar" rcs "C:\Users\YURYI
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-ar" rcs "C:\Users\YURYI
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-ar" rcs "C:\Users\YURYI
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-ar" rcs "C:\Users\YURYI
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-ar" rcs "C:\Users\YURYI
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-ar" rcs "C:\Users\YURYI
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-ar" rcs "C:\Users\YURYI
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-ar" rcs "C:\Users\YURYI
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-ar" rcs "C:\Users\YURYI
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-ar" rcs "C:\Users\YURYI
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-ar" rcs "C:\Users\YURYI
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-gcc" -w -Os -WL,--gc-seg
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-objcopy" -O ihex -j .eep
"C:\AVRTOOLS\arduino-1.6.9\hardware\ttools\avr\bin\avr-objcopy" -O ihex -R .eep

```

At the bottom of the terminal, the following memory usage information is shown:

```

Скетч использует 4 054 байт (14%) памяти устройства. Всего доступно 28 672 байт
Глобальные переменные используют 149 байт (5%) динамической памяти, оставляя

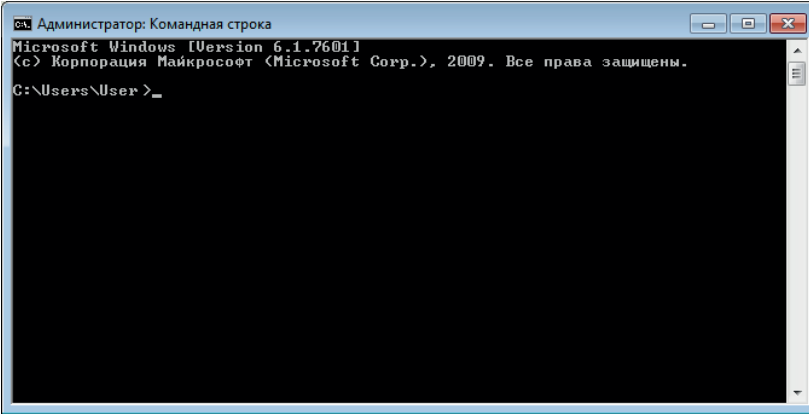
```

The status bar at the bottom indicates "Arduino Leonardo на COM12".

## Что такое терминал?

О, чудо, в нижней строке теперь отображается каждый шаг, который IDE сделал при трансляции твоей программы. Но теперь нам необходимо все внимательно проанализировать. Для этого обязательно нужно разобраться с управлением через консоль. Консоль – это программа, с помощью которой можно управлять компьютером только посредством клавиатуры, без какой-либо графической операционной среды. Обычно консоль уже предустановлена на компьютеры, использующие Linux, Mac OS X или Windows. В Windows консоль называется «командная строка» или просто «Cmd». В Linux и Mac OS она называется «оболочка», по-английски *shell*. Часто она также называется «командная строка» или «терминал» (который, кстати, не имеет ничего общего с терминалом из главы 2 под названием Монитор порта).

В Windows консоль вызывается через **Пуск | Все программы | Стандартные | Командная строка**. После запуска там должен отобразиться текущий путь доступа на жестком диске. В Windows, например, это может выглядеть так:



В других системах это выглядит примерно так же. Слева ты видишь приглашение Windows с указанием пути доступа (здесь `C:\Users\User`, где `User` – имя пользователя), в Linux также с именем пользователя.

## Навигация в командной строке

Теперь познакомься со следующими командами для навигации через консоль.

Команда	Функция	Пример
<code>cd</code>	Меняет директорию	<code>cd testordner</code>
<code>cd ..</code>	Меняет на один уровень выше, в предыдущем примере это было бы <code>C:\Users</code>	<code>cd ..</code>
<code>dir</code>	Отображает файлы в текущей директории	<code>dir</code>
<code>progname - flags</code>	Исполняет команду (программу) с заданными параметрами (flags)	<code>gcc main.c -v -o main</code>

В Linux в последнем пункте нужно ставить `./` перед именем программы. Смысл команд выше ты со временем поймешь самостоятельно, а пример для последней команды я сейчас разъясню. Давай разработаем программу для первого теста, которая ничего не делает, только содержит пустой цикл `loop()`. Но при этом мы не будем пользоваться IDE Arduino. Простая программа выглядит так:



## 10

```
int main() {
  for(;;){}
  return 0;
}
```

Туда специально вставлен *бесконечный цикл*, так как Arduino может исполнять только одну-единственную программу. Теперь подумай, что происходит, когда мы заканчиваем программу, например посредством `return`. Процессор находился бы в неопределенном состоянии. Чтобы избежать этого, в программу для Arduino всегда вставляется бесконечный цикл.

С помощью Блокнота создай файл с именем `main.cpp` и скопируй в него содержание приведенного выше исходного кода. Сохрани его по адресу:

Папка `Arduino\hardware\tools\avr\bin`

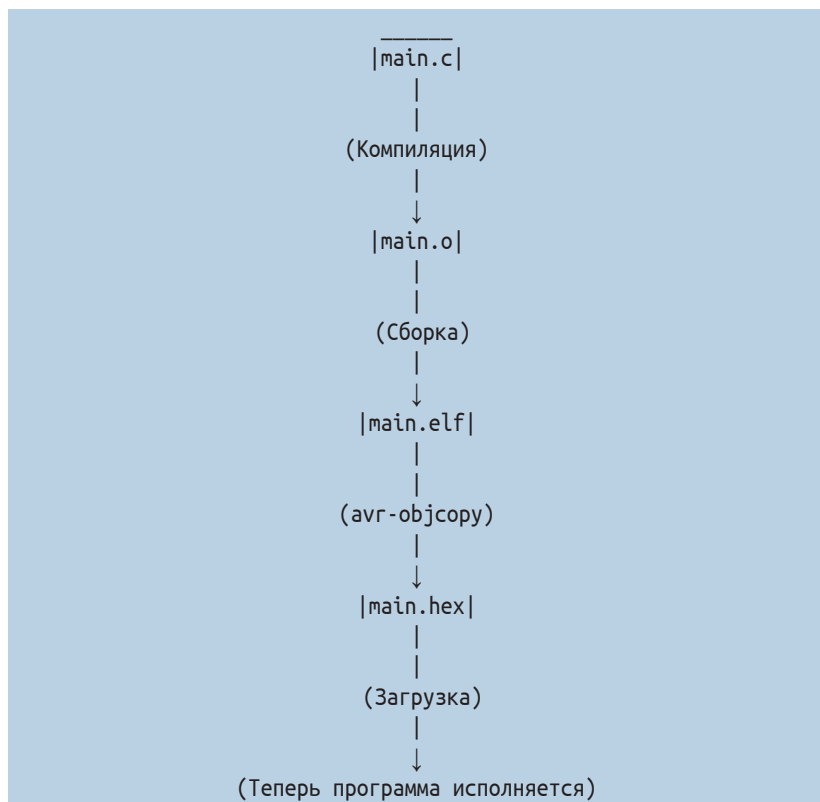
Затем тебе нужно с помощью команд `cd` открыть ту же папку в консоли. В Windows все может выглядеть, например, так (папка Arduino находится по адресу `C:\AVRTOOLS` и называется `arduino-1.6.9`):

```
Администратор: Командная строка
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

C:\Users\User > cd C:\AVRTOOLS\arduino-1.6.9\hardware\tools\avr\bin
C:\AVRTOOLS\arduino-1.6.9\hardware\tools\avr\bin>
```

## Перенос программы на Arduino

Здесь начинается довольно сложная процедура. Тебе нужно вызвать несколько программ, чтобы загрузить программу на Arduino. При этом вызови сначала компилятор, который переведет твой исходный код в машинный код. Посмотри на следующую блок-схему, на ней представлено, как программа приходит на Arduino:



Файл `main.hex` содержит данные, которые должен исполнить контроллер, – машинный код. После этого другая программа загружает `*.hex` на нашу плату.

## Компиляция кода

Для компилирования кода мы вызываем программу `avr-gcc`. Это осуществляется только по имени в командной строке. Так как `avr-gcc` не знает, где находится исходный код, как должен называться выходной файл и какие требуются опции, нужно еще передать ей параметры. Вызов параметров может, например, выглядеть так:

```
avr-gcc main.cpp -o main.o
```

Первый аргумент `main.cpp` – это исходный файл. Посредством параметра `-o` мы сообщаем компилятору, что хотим использовать опцию `o`, которая означает вывод (output), т. е. имя выходного файла должно быть такое, как указано далее в конце. Вызов компиляции программы из командной строки в общем виде выглядит так:

## 10

```
prg quellname(n) -flagname paramater -flagname ...
```

Не забудьте, что написание строчными и прописными буквами различается, т. е. «o» и «O» – это два разных параметра. Возможно, ты обратил внимание, что в примере выше я назвал выходным файлом так называемый объектный файл `main.o`, а не `main.hex`. Это потому, что нам нужно еще обработать скомпилированный файл, прежде чем \*.hex-файл можно будет загрузить на Arduino.

В результате команда для компиляции выглядит следующим образом:

```
avr-gcc main.c -c -o main.o -Os -mmcu=atmega328p
```

Введи ее в командную строку – и сразу получишь *объектный файл* \*.o, который пока не подходит для загрузки. Но сначала давай разберемся с назначением параметров командной строки.

Флаг	Цель
-c	Только переводит код
-o	Указывает имя выходного файла
-Os	Оптимизирует размер кода
-mmcu=xy	Указывается тип контроллера xy (=atmega328p)

## Сборщик

Продолжаем преобразование в исполняемый файл. Так называемый «сборщик» (программисты часто говорят «линковщик» – от английского `linking`, что значит «связывающий», «соединяющий») добавляет к коду библиотеки и другие необходимые файлы, то есть делает его пригодным для исполнения:

```
avr-gcc main.o -o main.elf --mmcu=atmega328p
```

Теперь, наконец, можно из .elf-файла сделать .hex-файл, который непосредственно загружается в Arduino. Для этого нам понадобится вторая программа, `avr-objcopy`:

```
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
```

## Загрузка

Вот мы и подошли к самой интересной части, загрузке на Arduino. Для этого нам нужна третья программа, `avrdude`.

Avrdude – это программа для загрузки твоего скетча на Arduino. Ранее это выполняла Arduino IDE.

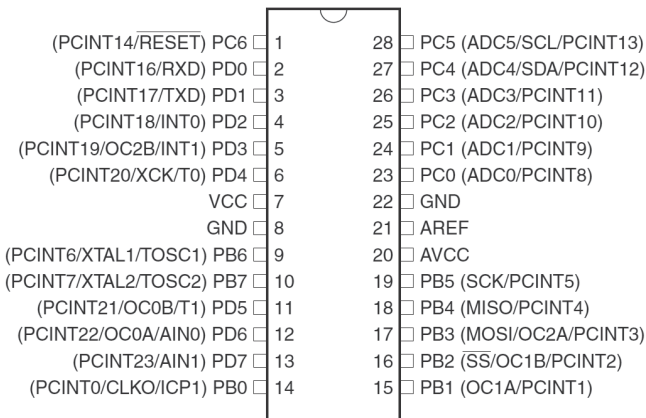
Если ты используешь контроллер с готовым загрузчиком Arduino (например, такой устанавливается в Uno), то можешь применить следующую команду:

```
avrdude -c arduino -p m328p -P usb -U flash:w:main.hex
```

## Программирование AVR

Если все получилось и нет сообщения об ошибке, мы можем приступить к программированию светодиода, но сначала немного теории о программировании AVR-контроллеров на языке C. Программирование на языке C очень похоже на программирование на C++, однако мы используем здесь совершенно другую библиотеку, нежели в проекте Arduino. Одним из основополагающих различий является то, что выводы нужно программировать не как отдельные пины, а как составную часть порта, объединяющего несколько выводов.

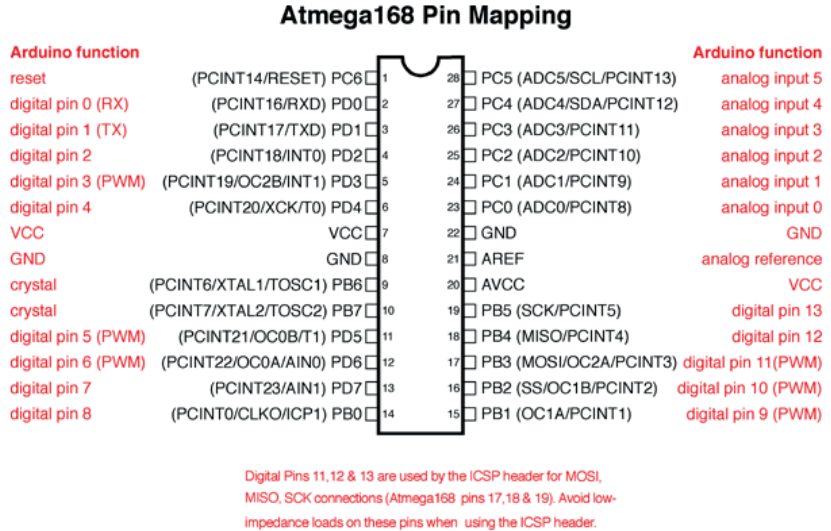
Поэтому программирование на C будет чуть более трудоемким, но полученная программа будет работать значительно эффективнее и иметь меньший объем. Функции при этом должны быть привязаны к другим названиям выводов, соответствующим «голой» микросхеме, а не готовой плате Arduino. Следующая схема от фирмы Atmel (изготовителя AVR-микроконтроллеров) демонстрирует эти названия:



Здесь ты видишь название и номер вывода (например, внизу справа PB1, он соответствует выводу микросхемы

## 10

номер 15). На следующем изображении приведено соответствие этих названий именам выводов Arduino, чтобы можно было работать напрямую с Arduino Uno:



На рисунке изображен Atmega168, но выводы на нем точно соответствуют Atmega328p, который установлен на нашем Arduino.

## Адаптация исходного кода Arduino к AVR

Теперь нам нужно посмотреть, как адаптировать исходный код Arduino к исходному коду на языке C. В качестве отправной точки мы возьмем следующую программу Arduino:

```
void setup() {
  pinMode(9,OUTPUT); //Pin 9 = PB1
}

void loop() {
  digitalWrite(9,HIGH);
}
```

Вот соответствующая C-программа:

```
#include <avr/io.h>
int main() {
  DDRB = 0b11111111; for(;;) {
```

```
PORTB = 0b00000010
}
return 0;
}
```

Сначала верхняя строка в заголовке: ты включаешь файл `io.h`, в котором определены переменные для программирования выводов AVR. После этого мы назначаем переменной `DDRB` значение 255, что в двоичной форме значит восемь единиц подряд:

```
DDRB = 0b11111111;
```

Посредством `0b...` вводится число в двоичной форме (число с нулями и единицами). *DDRB* означает *Data Direction Register B*, то есть регистр направления передачи данных для порта B. С его помощью указывается настройка порта B (знакомое нам `INPUT` или `OUTPUT`). С помощью единицы в нужном разряде здесь маркируется вывод как выход, то есть этой командой мы устанавливаем все выходы порта B на выход, от `PB0` до `PB7`. Один порт может включать в себя максимум восемь выводов (кстати, у нашего контроллера три порта: порт B, порт C и порт D).

Переходим к следующей строке. С помощью регистра `PORTB` устанавливается определенное значение выходов порта B (`HIGH` или `LOW`). Здесь у нас в бесконечном цикле `for(;;)` в регистр `PORTB` загружается двоичное число, которое устанавливает вывод `PB1` в 1 (`HIGH`):

```
for(;;) {
    PORTB = 0b00000010
}
```

Но всегда вручную писать этот длинный ряд битов (нулей и единиц) для установки одного вывода довольно трудоемко. Поэтому существует альтернативный способ записи. Этот способ выглядит так:

```
#include <avr/io.h>
int main() {
    DDRB = 255; //== 0b11111111
    for(;;) {
        PORTB = (1<<PB1);
    }
    return 0;
}
```

## 10

## Булева алгебра

Чтобы понять смысл записи  $(1 \ll PB1)$ , нам нужно сначала немного позаниматься математикой: обращением с битами, а также командами, которые работают именно с битами: это так называемая булева алгебра.

Самое важное для нас – битовый сдвиг. Это когда в последовательности битов сдвигается один отдельный бит. Посмотри на следующий код:

```
int bitfolge = 0b00000000;
    bitfolge = (1<<3);
//bitfolge = 0b00000100
```

Команда  $1 \ll x$  сдвигает 1 на  $x$  пунктов влево. В результате приведенная выше команда относительно `PORTB` также становится понятной, поскольку `PB1` означает позицию, на которой находится бит, управляющий выводом `PB1` (вывод микросхемы № 15, см. схему выводов выше). Сложнее будет, если мы захотим одновременно включить, например, выходы `PB1` и `PB3`. Команда для этого выглядит так:

```
PORTB = (1<<PB1) | (1<<PB3);
```

Между указаниями здесь стоит прямая черта. Она читается как «ИЛИ». Здесь «ИЛИ» означает арифметическую операцию в булевой алгебре, называемую «логическим сложением»:

```
0b00000010
|0b00001000
-----
0b00001010
```

Это значит, что из двух чисел формируется одно, причем в результат переносятся единицы из обоих чисел.

Но если ты хочешь установить оба пункта в порту В в 0 (LOW), то код будет выглядеть так:

```
#include <avr/io.h>
int main() {
    DDRB = 0b11111111;
    for(;;) {
        PORTB &= ~( (1<<PB1) | (1<<PB3) );
    }
}
```

```
return 0;  
}
```

Здесь ты устанавливаешь в 0 (LOW) уже существующую последовательность битов, не меняя уже установленные биты. Если ты захочешь установить в 1 (HIGH) отдельные биты, не меняя существующие, то это будет выглядеть подобно примеру выше, но с другими операторами:

```
PORTB |= (1<<PB1) | (1<<PB3);
```

Операции, обозначенные значками `&=` и `|=`, - просто замный способ записи булевых операций логического умножения «И» ( $x \&= y$  - то же самое, что  $x = x \& y$ ) и логического сложения «ИЛИ» ( $x \mid= y$  - то же самое, что  $x = x \mid y$ ).

На этом мы закончили изучение теории, необходимой для продвинутого программирования без Arduino-IDE.

Эти знания позволят тебе в дальнейшем работать с более сложными IDE, так как они не поддерживают синтаксис языка Arduino C++, а также с другими микроконтроллерами AVR. Сама по себе Arduino IDE не может работать ни с какими контроллерами, за исключением старых моделей ATmega8, ATmega 168, а также ATmega328, который применяется в настоящее время.

Но Arduino IDE можно все-таки заставить работать с любыми AVR-контроллерами, если использовать все выше-приведенные команды в IDE. Нужно только включить в заголовок вызов библиотеки `<avr/io.h>`. Как мы уже знаем, код, равносильный командам `pinMode(9,OUTPUT)`; и `digitalWrite(9,HIGH)`; может выглядеть так (вывод 9 Arduino == вывод PB1 контроллера == вывод 15 микросхемы ATmega 328, см. схемы выводов выше):

```
#include <avr/io.h>  
  
void setup() {  
  DDRB = 0b11111111;  
}  
  
void loop() {  
  PORTB = (1<<PB1);  
}
```

Так мы можем применять эти команды для любых контроллеров в пределах IDE. Кстати, установка битов без `digitalWrite`



## 10

talWrite() принесет тебе преимущество в виде более быстрого исполнения кода.

Но недостаток этого метода состоит в том, что порты могут отличаться в зависимости от типа микроконтроллера, и придется все время справляться с технической документацией (на английском языке!), в то время как на Arduino ты работаешь в «стандартизированной» обстановке со всеми пояснениями по-русски.

## Заключение

В этой главе мы занимались теоретическими вопросами работы за пределами IDE. Теперь ты знаешь, как IDE компилирует код и переносит его на Arduino, но можешь сделать это и сам «вручную», компилируя его с помощью avr-gcc и перенося через avrdude.

## Несколько вопросов...

1. В какой библиотеке определяются переменные DDRB, PORTB и PB1?
2. Для чего служит логическое «ИЛИ» («|») при назначении битов?
3. Что такое бит?
4. Сколько выводов может объединять в себе порт?

## ...и несколько заданий

1. Напиши еще раз нашу программу для мигания светодиодов в Arduino-IDE, но используй для этого непосредственное манипулирование портами:

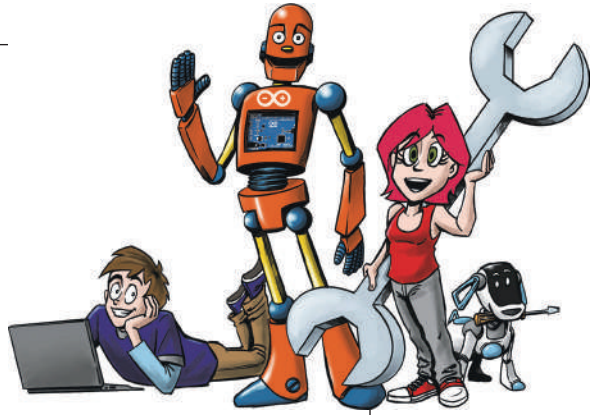
```
void setup() {
  pinMode(13,OUTPUT);
}

void loop() {
  digitalWrite(13,HIGH);
  delay(1000);
  digitalWrite(13,LOW);
  delay(1000);
}
```

Найди для этого соответствующий порт и название вывода 13 на схематических изображениях контроллеров в этой главе.

2. Напиши программу, которая использует только манипулирование портом, чтобы создать гирлянду с мигающими огоньками. Идея состоит в том, чтобы сначала поочередно включить все лампочки одного порта, и только когда все они загорятся, снова все их выключить одну за другой.
3. Если хочешь узнать больше по этой теме, набери в поисковике «манипулирование битами», а также «битовые операции» – и получишь больше информации по математическому аспекту этой главы.





# 11

## Не забудь меня – использование EEPROM

В этой главе мы будем заниматься установленной в Arduino долговременной памятью EEPROM. Как ее использовать и на какие преимущества и недостатки нужно обратить внимание, а также действительно ли эта память не стирается при выключении питания.

Ты узнаешь:

- ⊙ преимущества и недостатки EEPROM;
- ⊙ чтение EEPROM;
- ⊙ запись EEPROM.

Также ты узнаешь немного о черном ящике самолета и о том, как построить его с помощью Arduino!

### Общая информация о EEPROM

Наверняка у тебя уже были неприятные ситуации, когда после перезапуска Arduino исчезало содержание переменной. Этого можно избежать, если использовать EEPROM. EEPROM – это еще одна память в твоём Arduino (она прочно интегрирована в микроконтроллер), которая может сохранять данные даже без питания.

## 11

К сожалению, у нее есть и недостатки: ограниченное количество циклов записи-стирания. Это значит, что читать ее можно неограниченное количество раз, а записывать только примерно 100 000 раз. Казалось бы, очень много, но на практике этого оказывается мало. Например, если сохраняешь один результат температурного датчика в секунду, то за 24 часа ты сменишь около 86 000 результатов, что уже близко к предельной величине. Потому текущие часто меняющиеся значения в EEPROM сохранять не рекомендуется. Даже при записи одного измерения в минуту за 24 часа будет 1140 результатов для сохранения, и ресурса EEPROM хватит всего на 100 суток (примерно три месяца).

У контроллера, установленного в Arduino, 1024 ячейки памяти EEPROM объемом 1 байт каждая. Проблем с объемом при записи констант и результатов с датчиков возникнуть не должно.



Возможно, ты заметил, что число 1024 соответствует 1 Кбайт (килобайту). У Arduino память EEPROM на 1 Кбайт (= 1024 байта).

Чтобы использовать EEPROM, нам нужно сначала подключить библиотеку EEPROM.h (она по умолчанию устанавливается вместе с IDE). После этого первая программа выглядит, например, так:

```
#include <EEPROM.h>
int address = 0;
byte value; //Целое число в диапазоне 0-255

void setup() {
  Serial.begin(9600);
}

void loop() {
  value = EEPROM.read(address);
  Serial.print(address);
  Serial.print("\t");
  Serial.print(value, DEC);
  Serial.println();
  address = address += 1; //инкремент address++
  if(address == 512) { address = 0;
  }
  delay(500);
}
```

До функции `EEPROM.read` скетч не содержит ничего нового. Это очень простой код. Выражение `if` здесь должно помешать Arduino прочесть ячейку памяти, которой не существует. Таким образом, если была прочитана ячейка памяти номер 511, все началось бы заново. Если у тебя свой собственный Arduino Uno, можешь заменить число 512 на 1024. В вызове `EEPROM.Read()`, в котором отображается содержимое переменной `value`, ты найдешь новый параметр `DEC`. `DEC` – это сокращение от «десятичный» (англ. decimal) и означает, что показатель выводится в десятичной системе, то есть в обычном виде. Ты можешь также написать `HEX` для шестнадцатеричной системы. Наша десятичная система состоит из цифр от 0 до 9, а шестнадцатеричная система знает числа от 0 до F (1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Существуют и другие системы счисления, которые для нас сейчас не имеют значения (например, восьмеричная, двоичная и др.). Если ты еще не программировал Arduino на запись в память EEPROM, там везде должно стоять значение 255.

## Что можно запрограммировать в EEPROM?

Сейчас ты наверняка спросишь, какие данные теперь можно сохранить. Как насчет данных считывателя температуры? Сможем ли снова изобрести наше устройство регистрации данных с Ethernet Shield, не используя Shield? Трудно найти привлекательную цель использования EEPROM (распространенной целью была бы, например, нестираемая память для настроек типа яркости светодиодов). Хотя мне пришла в голову одна хорошая идея: мы построим черный ящик (также он называется бортовым самописцем) на Arduino.

Ты, наверно, уже слышал термин «*черный ящик*» в сообщениях об авиакатастрофах. Черные ящики встраиваются в самолеты и записывают информацию на борту, например разговоры в каюте пилота, высоту полета и т. д. При этом черный ящик особенно хорошо защищен и может выдержать крушение самолета.

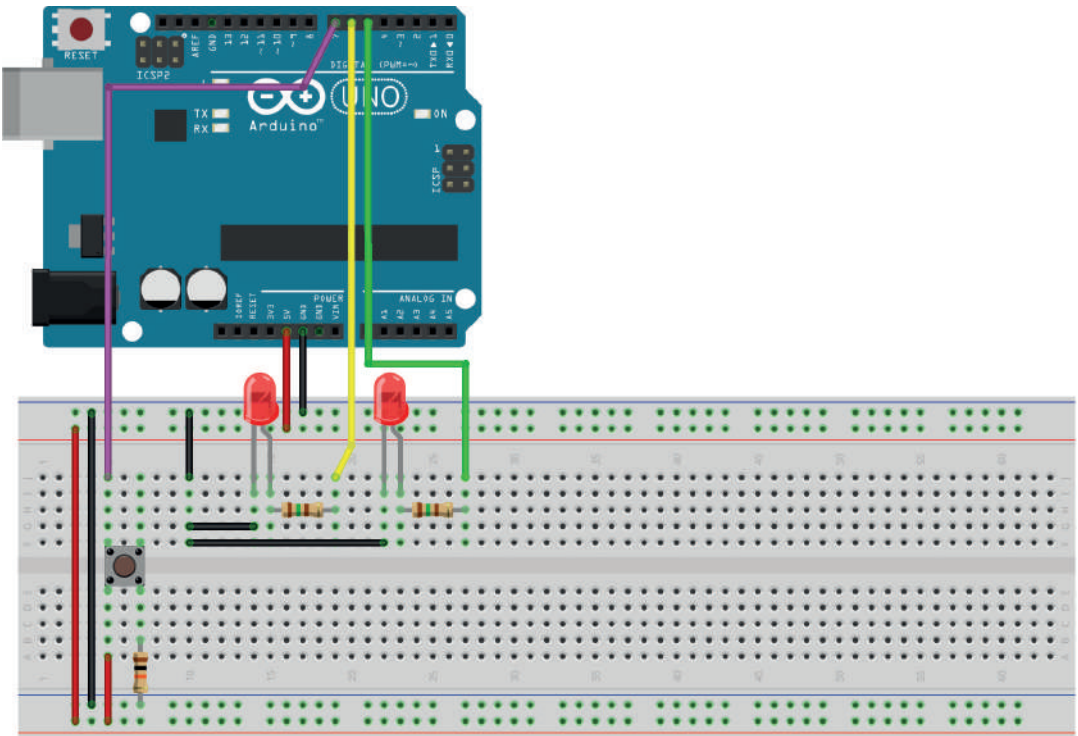
## Проект: черный ящик

Так как у нас, конечно же, нет самолета для получения данных, мы будем записывать включение светодиодов, нажатия кнопок и внутренние команды программы. Протоколы

## 11

будут сохраняться в EEPROM и просматриваться при сбросе Arduino. При этом Arduino управляется через последовательный интерфейс. Поскольку EEPROM слишком мала для сохранения текстов, мы в нее записываем только цифры от 0 до 255. И нам нужно закодировать показатели (то есть назначить каждому действию определенное число). Тогда цифра 1, например, будет означать, что был включен светодиод 1. Я предлагаю следующую кодировку:

Название по-русски	Название по-английски (для вывода)	Кодировка
Светодиод включен	LED On	1
Светодиод выключен	LED Off	2
Кнопка нажата	Button is pressed	3



fritzing

Кнопка на вывод 7, светодиод 1 на вывод 6, а светодиод 2 на вывод 5. Включение всех компонентов ты уже знаешь, поэтому обратимся сейчас к описанию EEPROM: для функции записи нужны два аргумента: адрес и записываемая величина в пределах одного байта (то есть от 0 до 255). Чтобы

можно было записать в EEPROM автоматически, мы заменим функцию `digitalWrite()` на `digitalWrite_p()`. Функция будет работать как оригинальная, то есть устанавливать уровень на указанном выводе, но при этом также записывать код в EEPROM. Сначала функция, которая находит первую свободную ячейку памяти:

```
#include <EEPROM.h>
int address = 0;
byte value;
bool start = true;
int led_1 = 6;
int led_2 = 5;

void setup() {
  Serial.begin(9600);
  pinMode(led_1,OUTPUT);
  pinMode(led_2,OUTPUT);
}
void loop() { if (start) {
  value = EEPROM.read(address);
  while(value != 255) {
  value = EEPROM.read(address);
  Serial.println(address); address++;
  if (address = 512) {
  Serial.println("Memory full, reset ");
  //"Память заполнена, сбросить "
  }
  }
  start = false; //"Адрес обнаружен или вывести сообщение об ошибке!
}
}
```

Этот скетч просто находит ячейку памяти в диапазоне адресов 0–511, в которой стоит значение 255 (это стандартное значение, если Arduino новый и память в нем ни разу не записывалась). Если свободная ячейка не найдена, Arduino посылает сообщение через последовательный порт. Затем запускается бесконечный цикл ожидания.

В противном случае у нас есть адрес, куда записывать новые данные. Конструкция `address++`, как ты помнишь, просто сокращение от `address = address + 1`, то есть переход к следующему по порядку адресу в EEPROM.

Теперь распишем функцию `digitalWrite_p`, которая управляет светодиодами. Она записывает показатель в EEPROM, а потом вызывает соответствующую функцию `digitalWrite`:



## 11

```
void digitalWrite_p(int pin, int state) {
  if(state==1) {
    EEPROM.write(address,1);
    digitalWrite(pin,HIGH);//HIGH = 1, таким образом, на
    address++;
  }
  if(state==0) {
    EEPROM.write(address,2);
    digitalWrite(pin,LOW);
    address++;
  }
}
```

Добавь этот код перед `setup()` и протестируй его. А теперь напиши функцию чтения по нажатию кнопки, я думаю, ты легко с этим справишься самостоятельно. Если не получилось, функция может выглядеть так:

```
void digitalRead_p(int pin) {
  int state = digitalRead(pin);
  if(state == HIGH) {
    EEPROM.write(address,3);
  }
}
```

Теперь есть два варианта: либо мы встроим функцию чтения в имеющийся скетч, либо разработаем вторую программу, которая читает данные по желанию. Я предлагаю второй вариант, чтобы данные не были доступны посторонним через последовательный порт и можно было их прочесть только с помощью программы, которая будет лишь у тебя и которая может расшифровать числовой код. Так все будут видеть EEPROM, но не будут знать, что там записано. За основу возьмем скетч для чтения EEPROM и затем проверим, какая величина записана в каждой ячейке памяти. После этого определим текст согласно таблице выше (например, 1 соответствует «светодиод включен» и так далее) и выведем его через последовательный порт. Это может выглядеть так:

```
#include <EEPROM.h>
int address = 0;
byte value;
void setup() {
  Serial.begin(9600);
}
void loop() {
```

```
value = EEPROM.read(address);
Serial.print(address);
Serial.print("\t");
if (value == 1) {
  Serial.print("LED was On"); //"Светодиод был включен "
}
if (value == 2) {
  Serial.print("LED was Off "); //"Светодиод был выключен"
}
if (value == 3) {
  Serial.print("Button was pressed"); //"Кнопка была нажата"
}
if(value == 255) {
  Serial.print("Empty memory location"); //"Пустая ячейка памяти"
}
Serial.println();
address++;
if(address == 512) {
  address = 0;
}
delay(500);
}
```

Сейчас ты наверняка спросишь, почему я использую только 512 ячеек памяти, а не все, ведь почти у всех Arduino Uno с контроллером ATmega328 их вдвое больше? Последние ячейки можно использовать для других функций, к которым мы сейчас перейдем.

Если у тебя нет Arduino Uno, можешь взять и другие разновидности плат Arduino. При этом обрати внимание, что для первого скетча ты используешь половину EEPROM, а вторая половина тебе понадобится для следующих модификаций.



Теперь нам нужно записать функции, которые начинают запись/чтение с адреса 512 и заканчивают на 1023. Чтобы мы могли так же запроотолировать работу устройства, вот расширенная таблица:

Название по-русски	Название по-английски (для вывода)	Кодировка
Светодиод включен	LED On	1
Светодиод выключен	LED Off	2
Кнопка нажата	Button is pressed	3
analogRead_p()	---	4

## 11

Ты уже знаешь, что `analogRead()` возвращает показатель от 0 до 1023. Его нам нужно разделить на 4 и получить показатель в пределах 0–255, который можно записать в ячейку памяти. Функция при этом выглядит так:

```
void analogRead_p(int pin) {
  int value = analogRead(pin);
  value = value / 4; //1023 / 4 = 255
  EEPROM.write(address,4);
  EEPROM.write(analog address,value);
}
```

Ты наверняка спросишь, откуда нам взять `analog_address`. Для этого нужно немного переписать в скетче функцию для нахождения свободного адреса:

```
void loop() {
  if (start) {
    value = EEPROM.read(address);
    while(value != 255){
      value = EEPROM.read(address);
      Serial.print(address);
      Serial.print("\t");
      Serial.print(value,DEC);
      Serial.println();
      ++address;
      if (address == 512) {
        Serial.println("Memory full, reset ");
        //"Память заполнена, сбросить "
        address = 0;
      }
    }
    /*Здесь начинается новый цикл!*/
    while(value != 255){
      value = EEPROM.read(address);
      Serial.print(analog_address);
      Serial.print("\t");
      Serial.print(value,DEC);
      Serial.println();
      ++_analog_address;
      if (analog_address == 1024) {
        Serial.println("Memory full, reset ");
        //"Память заполнена, сбросить "
        address = 512;
      }
    }
  }
}
```

```
/*Конец нового цикла*/
address--;
Serial.print("adresse: ");
Serial.print(address);
Serial.println();
Serial.println("Start trying"); //"Начни попытку"
delay(1000);
digitalWrite_p(led_1,1); delay(1000);
digitalWrite_p(led_2,1); delay(1000);
digitalWrite_p(led_2,0); delay(1000);
digitalWrite_p(led_1,0);
start = false; //адрес обнаружен
}
}
```

Цикл работает как первый, только он читает в диапазоне адресов 512–1023. Но чтобы узнать, что было запротоколировано, нам нужно еще исправить скетч для чтения. Я запишу его здесь:

```
#include <EEPROM.h>
int address = 0;
int analog_address = 512;
byte value;

void setup() {
  Serial.begin(9600);
  while(address != 512) {
    value = EEPROM.read(address);
    Serial.print(address);
    Serial.print("\t");
    if(value == 1) {
      Serial.print("LED On"); //"Светодиод выключен"
      Serial.println();
    }
    if(value == 2) {
      Serial.print("LED Off"); //"Светодиод выключен"
      Serial.println();
    }
    if (value == 3) {
      Serial.print("Button was pressed");//"Кнопка была нажата"
    }
    if(value == 255) {
      Serial.print("Empty memory location");//"Пустая ячейка памяти"
      Serial.println();
    }
  }
}
```

## 11

```

address = address + 1;
}
while(analog_address != 1024) {
value = EEPROM.read(address);
value = value * 4; //Чтобы у нас было прочитанное
//оригинальное значение
value = value / 200; //Чтобы у нас был показатель в вольтах
Serial.print(address);
Serial.print(„\t");
Serial.print(„Volt");
Serial.println();
analog_address++;
}
}

void loop(){
//Пустой цикл!
}

```

Я перенес код в Setup(), чтобы все работало. Код очень понятный. Можешь проверить,  $1023 / 2$  будет 511,5, но переменные типа integer в любом случае округлят это значение до целого.

Для полноты картины – вот еще скетч для записи:

```

#include <EEPROM.h>
int address = 0;
int analog_address = 512;
byte value;
bool start = true;
int led_1 = 6;
int led_2 = 5;
int button = 7;
int buttonstate = 0;

void digitalWrite_p(int pin, int state) {
    if(state == 1) {
        EEPROM.write(address,1);
        digitalWrite(pin,HIGH);
        ++address;
    }
    if(state == 0) {
        EEPROM.write(address,2);
        digitalWrite(pin, LOW);
        ++address;
    }
}

```

```
}

void digitalRead_p(int pin) {
int state = digitalRead(pin);
if(state == HIGH) {
    EEPROM.write(address,3);
}
}

void setup() {
Serial.begin(9600);
pinMode(led_1,OUTPUT);
pinMode(led_2,OUTPUT);
pinMode(button,INPUT);
}

void loop() {
if (start) {
    value = EEPROM.read(address);
    while(value != 255){
        value = EEPROM.read(address);
        Serial.print(address);
        Serial.print("\t");
        Serial.print(value,DEC);
        Serial.println();
        ++address;
        if (address == 512) {
            Serial.println("Memory full, reset ");
            //"Память заполнена, сбросить "
            address = 0;
        }
    }
    while(value != 255){
        value = EEPROM.read(address);
        Serial.print(analog_address);
        Serial.print("\t");
        Serial.print(value,DEC);
        Serial.println();
        ++analog_address;
        if (analog_address == 1000) {
            Serial.println("Memory full, reset ");
            //"Память заполнена, сбросить "
            address = 513;
        }
    }
}
address - - ;
```

## 11

```

Serial.print("Address: ");
Serial.print(address);
Serial.println();
Serial.println("Start trying"); //"Начни попытку"
delay(1000);
digitalWrite_p(led_1,1);
delay(1000);
digitalWrite_p(led_2,1);
delay(1000);
digitalWrite_p(led_2,0);
delay(1000);
digitalWrite_p(led_1,0);
start = false; //адрес обнаружен
}
}

```

Эту программу можно расширить по своему желанию. Можешь дополнительно сохранять еще несколько сенсоров с их данными – и тогда получишь исчерпывающий черный ящик для своей комнаты.

## Заключение

Теперь ты знаешь, как...

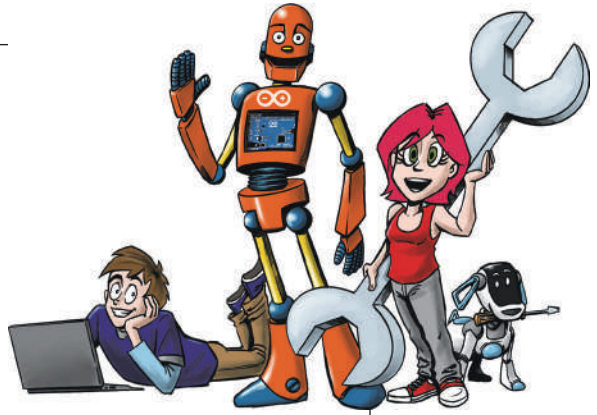
- прочесть EEPROM;
- записать в EEPROM;
- спланировать и разработать проект;
- использовать тип переменных byte (0–255) и...
- что такое черный ящик.

## Несколько вопросов...

1. У Arduino 512 или 1024 ячейки памяти EEPROM?
2. Где чаще всего используется черный ящик?
3. Какие операторы для вычисления знает Arduino?

## ...и несколько заданий

На этот раз заданий не будет.



# A

## Установка IDE

IDE – это один из твоих самых важных инструментов (после Arduino, конечно), без которого ты не сможешь работать и программировать. Поэтому вот инструкция по установке:

- ⦿ скачать IDE на компьютер;
- ⦿ распаковать на рабочем столе;
- ⦿ Windows: установить драйвер;
- ⦿ выполнить настройки в IDE.

Далее обзор моментов, которые нужно учитывать при установке.

## Установка

Сначала нужно скачать программное обеспечение здесь: <http://arduino.cc/en/Main/Software>. На момент написания этой книги актуальная версия 1.8.5.

Для тех, кто пользуется Windows, на сайте имеется выбор: самоустанавливающийся EXE-архив или обычный архив в формате ZIP. Мы остановимся на втором варианте. Скачанный архив распакуй с помощью любого архиватора (например, Winrar или 7Zip, можно использовать и собственные возможности Windows). Содержащуюся там папку под названием **arduino** (маленькими буквами с добавлением



## А

номера версии) можно сохранить, например, на рабочем столе. В папке находится так называемая IDE, программа для создания исходного кода и его преобразования в «машинный язык». По сути, IDE представляет собой обычную программу для работы с текстом (текстовый редактор), которая различным цветом помечает команды программирования.

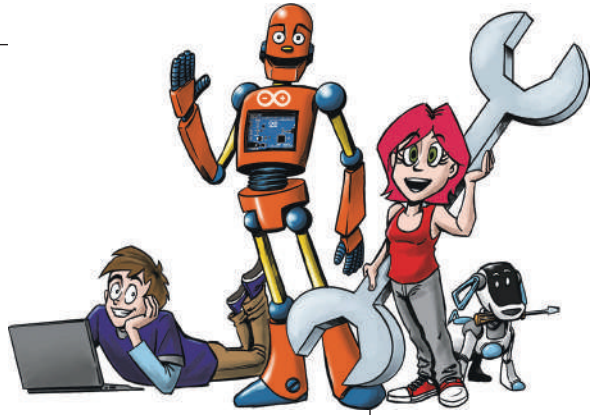
Теперь перейдем к самому сложному этапу – установке драйвера. Для операционной системы Windows нужно установить драйвер, который позволит опознавать подключенную плату Arduino и ее программировать. Подключи к USB-кабелю Arduino Uno и подожди. Через некоторое время должно появиться окошко, в котором предложат найти или обновить драйверы. Пройгнорируй его и открой панель управления. Затем нажми на раздел **Система и безопасность**. Там выбери пункт меню **Система | Диспетчер устройств**. В открывшемся окне нужно выбрать пункт **Порты (COM и LPT)**. Теперь там должно отображаться название платы Arduino.

Нажатием правой кнопки мыши на этом названии выбери **Обновить драйвер**, а в открывшемся окне – **Выполнить поиск драйверов на этом компьютере**. Теперь тебе нужно найти папку на рабочем столе, в которую ты распаковал IDE (в дальнейшем мы будем называть ее просто **Arduino**, хотя обычно она еще дополняется номером версии IDE). В подпапке **Drivers (... \Arduino \Drivers)** ты найдешь файл с названием *Arduino.inf*. Выбери его, и установка драйвера завершена.

(Но честно, быстрее можно установить операционную систему Linux, чем драйвер для Windows.)

На этом установка окончена. Теперь ты можешь через USB-кабель подключить Arduino к компьютеру и приступить к программированию. Но сначала тебе нужно настроить в IDE соответствующий Arduino. Если ты используешь рекомендуемый Arduino Uno, выбери Arduino Uno в меню **Инструменты | Плата**, в противном случае выбери тот Arduino, что ты купил.

Затем необходимо определить правильный порт: извлеки Arduino Uno (если ты его уже подключил), посмотри порты в расположении **Инструменты | Последовательный порт** и запиши их. Затем вставь Arduino Uno. Выбери порт, который появился заново. Теперь можно начать программировать.



# Б

## Ответы

### Глава 1

#### Вопросы

1. Как в IDE Arduino настроить используемую плату?
2. Для чего нужны переменные `bool`?
3. С какими типами циклов ты познакомился?
4. Какой результат у функции `millis()`?

#### Ответы

1. **Инструменты | Плата.**
2. Для сохранения логических значений (`true/false`).
3. Цикл `while`.
4. Измерение времени с момента запуска Arduino.

### Глава 2

#### Вопрос

1. Где можно получить данные по установленной скорости передачи последовательного порта?

#### Ответы

2. Внизу справа в окне Монитора порта.

# Б

## Глава 3

### Вопросы

1. Как называется первый датчик из этой главы?
2. Назови выводы транзистора.
3. Какая цепь в транзисторе усиливает другую?

### Ответы

1. Фоторезистор.
2. Коллектор, эмиттер, база.
3. Цепь эмиттер – база усиливает цепь эмиттер – коллектор.

## Глава 4

### Вопросы

1. Каким двигателем управляют заданием угла поворота в градусах?
2. На сколько градусов вращается стандартный сервопривод?
3. Можно ли управлять двигателем постоянного тока?
4. Что такое секундомер?

### Ответы

1. Сервоприводом.
2. 180°.
3. Да, но для этого потребуются специальные схемы.
4. Часы, которые считают секунды.

## Глава 5

### Вопросы

1. Для чего нужна документация?
2. Может ли документация быть очень краткой?
3. Какова функция последнего исходного кода этой главы?

### Ответы

1. Чтобы в дальнейшем было понятно, как работает программа (автору и другим людям).
2. Если программа очень короткая, документация тоже может быть очень краткой.
3. Программа с функцией для создания журнала данных (протокола).

## Глава 6

### Вопросы

1. Что значит сокращение ЖК-дисплей?
2. Какая библиотека нужна, чтобы управлять ЖК-дисплеем?
3. Обязательно ли нужен потенциометр, чтобы регулировать контраст?

### Ответы

1. Жидкокристаллический.
2. LiquidCrystal.
3. При заданном напряжении питания можно использовать постоянный делитель из резисторов подобранной величины.

## Глава 7

### Вопросы

1. С помощью чего определяется наличие или отсутствие контакта? Что нужно для этого настроить?
2. Сопротивление можно измерять прямо в работающей схеме или при выключенном питании?
3. При параллельном подключении измеряются амперы/сила тока или вольты/напряжение?

### Ответы

1. Например, с помощью функции проверки диодов.
2. Это нужно делать при выключенном питании.
3. Напряжение.

## Глава 8

### Вопросы

1. Что такое Shield?
2. Какую плату Shield мы использовали?
3. С помощью каких команд можно создавать ссылки на HTML-документы?
4. Какой заголовок мы отправляем браузеру?
5. Текст для чтения находится в какой-то отдельной области?

### Ответы

1. Плата расширения для Arduino, которое можно на него установить.

# Б

2. Мы использовали Arduino Ethernet Shield.
3. `<a href=""></a>`.
4. Статус «HTTP/1.1 200 OK», а также указание на формат документов «Content-Type: text/html».
5. Читаемый текст располагается между тегами `<body>` `</body>`.

## Глава 9

### Вопросы

1. Микросхема у Leonardo больше или меньше, чем у Uno?
2. Можно ли исполнить приведенный выше код с Arduino Uno?
3. Сколько клавиш потребуется, чтобы реализовать функцию переключения регистра (прописных и строчных букв)?
4. Можно ли с помощью Arduino Leonardo собрать целую клавиатуру?
5. Что такое физический секретный ключ?

### Ответы

1. Меньше, и она не заменяемая.
2. Нет, это особенность Arduino Leonardo.
3. Одна дополнительная клавиша.
4. В принципе, да.
5. Защита паролем, которая выполнена в виде физического устройства, например брелка для ключей.

## Глава 10

### Вопросы

1. В какой библиотеке определяются переменные DDRB, PORTB и PB1?
2. Для чего служит логическое «ИЛИ» («|») при назначении битов?
3. Что такое бит?
4. Сколько выводов может объединять в себе порт?

### Ответы

1. `io.h`.
2. Чтобы одновременно установить два бита.
3. Двоичное число, которое принимает только значения 1 или 0.
4. В Arduino порты могут содержать до 8 выводов.

## Глава 11

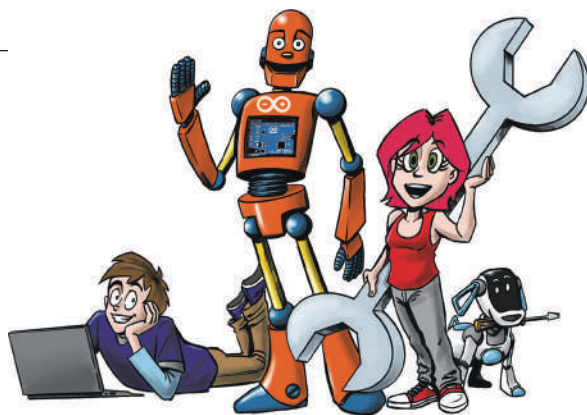
### Вопросы

1. У Arduino 512 или 1024 ячейки памяти EEPROM?
2. Где чаще всего используется черный ящик?
3. Какие операторы для вычисления знает Arduino?

### Ответы

1. Это зависит от конкретного Arduino. На более старых моделях, как правило, 512 ячеек памяти, на современных, как Arduino Uno, 1024 ячейки по одному байту.
2. Черный ящик используется на самолетах. Он сохраняет всю значимую информацию о работе систем самолета.
3. Прибавление, вычитание, деление, умножение, а также булевы операторы (например, «И» и «ИЛИ»).





# В

## Список материалов

В первом издании список материалов находился в конце введения. Поскольку многие читатели его не замечали, он теперь выведен в отдельное приложение.

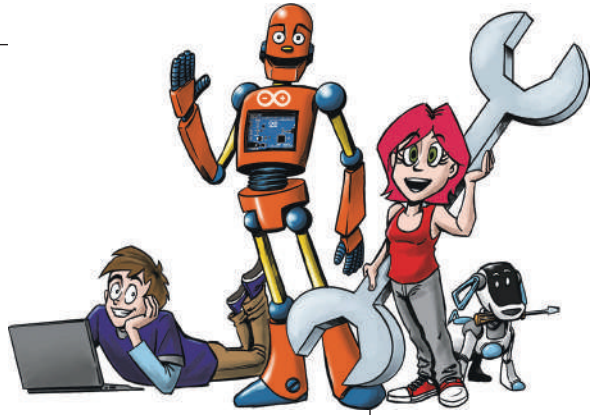
### Материалы:

- Arduino Uno, Arduino Leonardo (только для главы 9);
- Ethernet Shield (только для главы 8);
- минимум одна макетная плата;
- светодиоды разных цветов (например, готовые наборы);
- примерно 3 многоцветных светодиода (например, те, что могут гореть красным, зеленым, синим и имеют 4 вывода);
- различные резисторы (в продаже есть наборы);
- различные переключки (тоже продаются в наборе, разной длины);
- несколько кнопок (например, 10 штук);
- примерно 3 фоторезистора;
- пьезоэлектрический звукоизлучатель (его можно снять с очень старого ПК);
- два двигателя постоянного тока на напряжение 6 В небольшой мощности (1–2 Вт);



**В**

- маломощный сервопривод (1–2 кг·см);
- несколько маломощных транзисторов (типа *n-p-n*);
- драйвер для управления двигателем L293D;
- жидкокристаллический дисплей конфигурации 1602 (16 символов в 2 строки; должен иметь контроллер hd44780 или его аналог);
- мультиметр;
- паяльник с оловом для пайки;
- деревянная доска;
- кусок проволоки;
- клеевой пистолет
- и для полноты картины обычный компьютер для запуска IDE.



# Предметный указатель

## Символы

30-секундный таймер, 84  
#define, 97

## A

analogRead(), 64  
analogWrite(), 66  
Arduino Leonardo, 117  
Arduino Uno, 11  
avrdude, 136  
avr-gcc, 135

## B

bool, 26  
break, 45

## C

case, 45  
char, 57  
COM-порт, 51

## D

DDRB (Data Direction Register B), 139  
DEC, 147  
default, 45

## E

EEPROM, 145

EEPROM.h, библиотека, 146  
else, 33

## H

HTML (Hypertext Markup Language), 109

## I

INPUT, 23, 31  
integer, 25  
int main(), 130

## L

long, 44  
loop(), 22

## O

OUTPUT, 23

## S

setup(), 22  
Shield, 109  
switch, 45  
switch(), 39

## W

while, 47

## A

Аналоговый вход, 63

**Б**

Бесконечный цикл, 22, 134  
 Блок-схема, 33  
 Бод (baud), 53  
 Булева алгебра, 140

**В**

Величина допуска, 105  
 Вывод, 23

**Г**

Гирлянда со светодиодами, 27

**Д**

Двигатель постоянного тока, 76  
 Декремент, 58  
 Диалект, 10  
 Диммирование, 66  
 Динамик, 36  
 Документация, 87  
 Драйвер двигателя, 79

**Ж**

Жидкокристаллический дисплей, 94

**З**

Запрос if-else, 32

**И**

Игра «Горячий провод», 39  
 Измерение  
   напряжения, 101  
   силы тока, 102  
   сопротивления, 104  
 Инициализация порта на частоте, 52  
 Инкремент, 58

**К**

Ключ Морзе, 36  
 Ключ обеспечения секретности, 124  
 Командная строка, 132  
 Комментарий, 23  
 Константа, 25

**М**

Макетная плата, 11  
 Метод прозвонки, 106  
 Микроконтроллер, 9  
 Мультиметр, 100

**О**

Область действия, 30  
 Оболочка, 132  
 Объектный файл, 136  
 Отладчик, 53

**П**

Параметры, 23  
 Переменные, 25  
 Пины, 23  
 Полярность неправильная, 71  
 Порт, 22  
 Последовательный интерфейс, 51  
 Потенциометр, 77  
 Программирование AVR, 137  
 Программное обеспечение, установка, 19  
 Прототип функции, 41  
 Псевдокод, 32

**Р**

Резистор, 13

**С**

Сборщик, 136  
 Светодиод, 11  
   мигающий, 25  
 Светочувствительный резистор, 62  
 Сенсор, 61  
 Сервопривод, 81  
 Сетевые функции, 111  
 Синтаксис, 130  
 Синтаксические ошибки, 54  
 Скetch, 22  
 Строковый тип данных, 53  
 Структура дерева, 42  
 Структурное программирование, 29  
 Схема Дарлингтона, 71

**Т**

Тег, 110

Терминал, 52

Ток, усиление, 71

Токен, 117

Транзистор, 69

Транслятор, 24

**Ф**

Фоторезистор, 62

Функция, 23, 28

**Ц**

Целочисленная переменная, 25

Цифровой пин, 64

**Ч**

Черный ящик, 147

Чтение и вывод знаков, 56

**Ш**

Широтно-импульсная  
модуляция, 66

**Э**

Эмуляция, 119

**Я**

Язык

программирования С, 130

Яркость фона, 95

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «Планета Альянс» наложенным платежом, выслав открытку или письмо по почтовому адресу:  
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.  
При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя.  
Желательно также указать свой телефон и электронный адрес.  
Эти книги вы можете заказать и в интернет-магазине: [www.aliants-kniga.ru](http://www.aliants-kniga.ru).  
Оптовые закупки: тел. (499) 782-38-89.  
Электронный адрес: [books@aliants-kniga.ru](mailto:books@aliants-kniga.ru).

Эрик Шернич

## **Arduino для детей**

Главный редактор *Мовчан Д. А.*  
[dmkpress@gmail.com](mailto:dmkpress@gmail.com)  
Редактор *Ревич Ю. В.*  
Перевод *Степаненкова М. М.*  
Корректор *Синяева Г. И.*  
Верстка *Чаннова А. А.*  
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.

Гарнитура «PT Serif». Печать офсетная.  
Усл. печ. л. 13,81. Тираж 200 экз.

Веб-сайт издательства: [www.dmkpress.com](http://www.dmkpress.com)