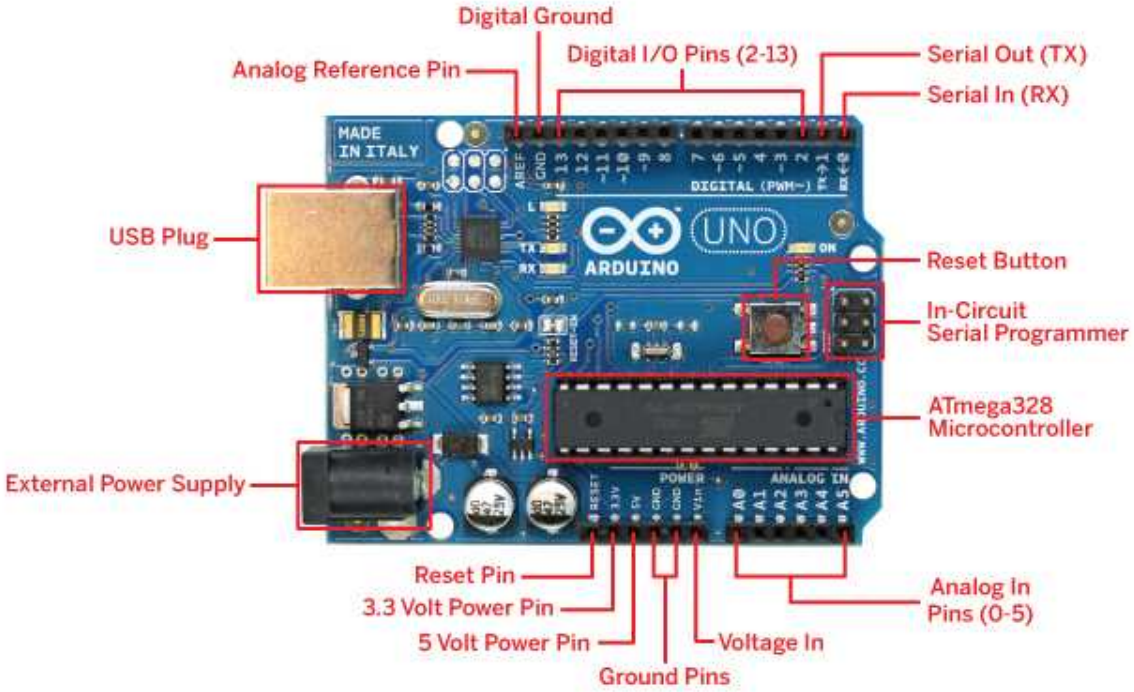
## *Лабораторная работа №1*
## *Введение в программирование в среде Arduino IDE*

В качестве основы лабораторных работ используется контроллер Arduino. Выбор обусловлен низкой стоимостью при достаточно неплохих характеристиках (от 100-150 рублей за самый дешевый Arduino Pro Mini до 1200 рублей за Arduino Due – флагман основной линейки Arduino), большим количеством библиотек и модулей, разработанных для работы с Arduino и расширяющих его возможности, например, датчики, драйверы двигателей, модули для коммуникации через Ethernet, Bluetooth, Wi-Fi и т.д.

Arduino имеет большое и дружественное сообщество, выкладывающее в открытый доступ в Интернете схемы, примеры программ и даже готовые проекты, которыми можно воспользоваться в обучении и реализации своих проектов

Внешний вид и разъемы контроллера Arduino Uno:




**Параметры контроллера Arduino Uno**:

| | |
|---|---|
| Микроконтроллер | ATmega328 |
| Питание | От USB компьютера (+5 В, USB Plug) или внешнего источника (+7…12 В, External Power Supply) |
| Digital I/O Pins<br>Цифровые входы/выходы | 14 штук – D0…D13, каждый из которых может выдавать уровень напряжения 0 В или 5 В или считывать их<br>6 из них (D3, D5, D6, D9, D10, и D11, обычно помечены на плате) могут использоваться как выходы регулируемого уровня напряжения в диапазоне 0…5 В |
| Analog In<br>Аналоговые входы | 6 штук – A0…A5. Измеряют значения напряжения на соответствующем пине Arduino в диапазоне 0…5 В.<br>Могут использоваться как цифровые входы/выходы (D14…D18) |
| Максимальный ток | 40 мА (достаточно, чтобы питать светодиод, но недостаточно, чтобы |

| | |
|---|---|
| через вход/выход | питать электромотор). При превышении тока контроллер может выйти из строя |
| Флеш-память | 32 Кб, при этом 2 Кб используются для загрузчика, а 30 – для хранения написанной программы для контроллера |
| ОЗУ | 2 Кб |
| Индикаторы на плате | Светодиод *ON*, загорающийся при подключении контроллера к питанию<br>Светодиоды *RX, TX*, мигающие в процессе прошивки контроллера, а также при передаче/приеме информации с компьютера<br>Светодиод L, соединенный с цифровым контактом D13 |

Хочется прояснить ситуацию с так называемым «языком программирования Arduino», который «основан на Wiring». Подобные сочетания слов часто встречаются на страничках, посвященных Arduino. На официальном сайте так и пишут: "...is programmed using the Arduino programming language (based on Wiring)". По факту нет никакого особого языка программирования, и фактически программы пишутся на C/C++, а компилируются и собираются с помощью широко известного avr-gcc.

Все особенности сводятся к тому, что имеется набор библиотек, включающий в себя некоторые функции (вроде pinMode) и объекты (вроде Serial), а при компиляции Вашей программы среда разработки создает временный .cpp файл, в который кроме Вашего кода включается еще несколько строчек, и полученный результат скармливается компилятору а затем линковщику с нужными параметрами.

Все особенности сводятся к тому, что имеется набор библиотек, включающий в себя некоторые функции (вроде pinMode) и объекты (вроде Serial), а при компиляции программы среда разработки создает временный .cpp файл, в который кроме исходного кода включается еще несколько строчек, и полученный результат скармливается компилятору а затем линковщику с нужными параметрами.

Язык программирования Arduino является стандартным C++ (используется компилятор AVR-GCC) с некоторыми особенностями, облегчающими новичкам написание первой работающей программы.

Программы, написанные программистом Arduino называются наброски (или иногда скетчи — калька от англ. sketch) и сохраняются в файлах с расширением ino. Эти файлы перед компиляцией обрабатываются препроцессором Ардуино. Также существует возможность создавать и подключать к проекту стандартные файлы C++.

Обязательную в C++ функцию main() препроцессор Arduino создает сам, вставляя туда необходимые «черновые» действия.

Программист должен написать две обязательные для Arduino функции setup() и loop(). Первая вызывается однократно при старте, вторая выполняется в бесконечном цикле.

В текст своей программы (скетча) программист не обязан вставлять заголовочные файлы используемых стандартных библиотек. Эти заголовочные файлы добавит препроцессор Arduino в соответствии с конфигурацией проекта. Однако пользовательские библиотеки нужно указывать.

Менеджер проекта Arduino IDE имеет нестандартный механизм добавления библиотек. Библиотеки в виде исходных текстов на стандартном C++ добавляются в специальную папку в рабочем каталоге IDE. При этом название библиотеки добавляется в список библиотек в меню IDE. Программист отмечает нужные библиотеки и они вносятся в список компиляции.

Arduino IDE не предлагает никаких настроек компилятора и минимизирует другие настройки, что упрощает начало работы для новичков и уменьшает риск возникновения проблем.

Простейшая Arduino-программа состоит из двух функций:

setup(): функция вызывается однократно при старте микроконтроллера.

loop(): функция вызывается после setup () в бесконечном цикле все время работы микроконтроллера.

Одна из простейших схем на Arduino — это подключение внешнего светодиода, управление которым происходит при помощи программы (скетча).

Так выглядит полный текст простейшей программы (скетча) мигания светодиодом, подключенного к 13 выводу Arduino, с периодом 2 секунды:

```
void setup () {
pinMode (13, OUTPUT); // Назначение 13 вывода Arduino выходом
}
void loop () {
  digitalWrite (13, HIGH); // Включение 13 вывода, параметр вызова функции
  delay (1000); // Цикл задержки на 1000 мс - 1 секунду
  digitalWrite (13, LOW); // Выключение 13 вывода, параметр вызова LOW
  delay (1000); // Цикл задержки на 1 секунду
}
```

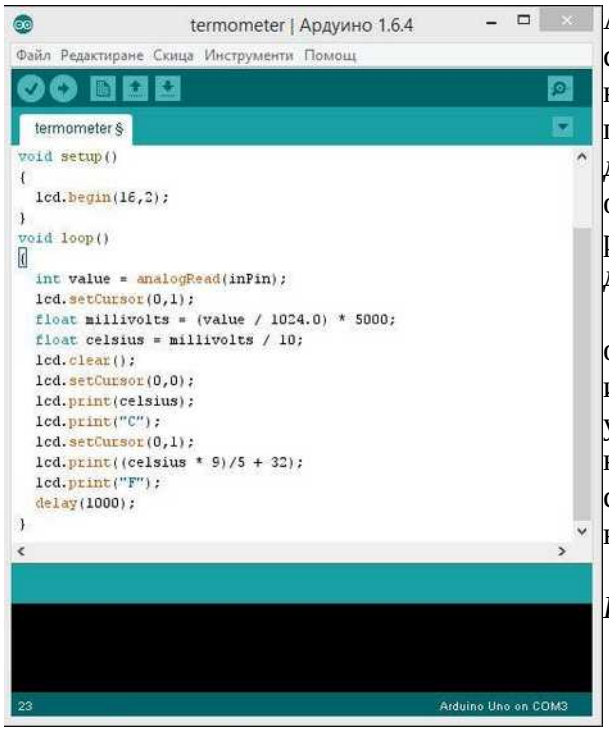
Все используемые в примере функции являются библиотечными. В комплекте Arduino IDE имеется множество примеров программ. Существует перевод документации по Arduino на русский язык.

*Загрузка программы в микроконтроллер*

Закачка программы в микроконтроллер Arduino происходит через предварительно запрограммированный специальный загрузчик (все микроконтроллеры от Ардуино продаются с этим загрузчиком). Загрузчик создан на основе Atmel AVR Application Note AN109. Загрузчик может работать через интерфейсы RS-232, USB или Ethernet в зависимости от состава периферии конкретной процессорной платы. В некоторых вариантах, таких как Arduino Mini или неофициальной Boarduino, для программирования требуется отдельный переходник.

Пользователь может самостоятельно запрограммировать загрузчик в чистый микроконтроллер. Для этого в IDE интегрирована поддержка программатора на основе проекта AVRDude. Поддерживается несколько типов популярных дешёвых программаторов.

*Arduino IDE*




Для начала работы необходимо установить среду программирования Arduino IDE. Arduino IDE является бесплатной программой, скачать которую может любой желающий. На нашем сайте вы можете скачать любую подходящую для вас версию среды. Также доступ к скачиванию IDE предоставлен на официальном сайте компании, а при желании, разработчиков можно отблагодарить, сделав денежный перевод.

Среда IDE поддерживается такими операционными системами, как Windows, MacOs и Linux. На официальном сайте компании указанно, что данный язык программирования написан на Wiring, но на самом деле его не существует и для написания используется C++ с небольшими изменениями.

*Как настроить Ардуино на компьютере?*

Делается это просто. Необходимо выполнить следующие действия:

- необходимо подключить собранное вами изделие к компьютеру посредством USB кабеля;
- в диспетчере устройств необходимо проверить, к какому порту подключен ваш микроконтроллер. Если он не отображается или написано, что устройство не опознано – значит, вы не правильно установили драйвер или ваша плата нуждается в диагностике;
- следующим шагом будет запуск нашего языка программирования Arduino IDE. В меню необходимо выбрать вкладку инструменты. При ее нажатии откроется список, в котором необходимо выбрать пункт – порт. Там надо выбрать порт, указанный в диспетчере устройств;
- конечным пунктом является выбор платы, которую мы будем использовать для загрузки скетчей.

**Вывод в монитор порта через Serial print, println, write**

Serial.print() и Serial.println() – это основные функции Arduino для передачи информации от платы ардуино к компьютеру через последовательный порт. На самых популярных платах Arduino Uno, Mega, Nano нет встроенного дисплея, поэтому взаимодействие с помощью монитора порта Arduino IDE является самым простым и доступным способом получения информации во время работы скетча. В этой статье мы научимся использовать методы класса Serial для отладки программ и взаимодействия с другими устройствами через UART и последовательный порт.

*Функция print() библиотеки Serial*

Функция print является одной из самых часто встречающихся в скетчах ардуино. Она принимает на вход текст или цифры и затем передает их через открытый последовательный порт в формате ASCII. Примеры:

***Serial.print(“Hello, Arduino”)*** – на экране монитор порта мы увидим надпись Hello, Arduino
***Serial.print(12)*** – число будет автоматически преобразовано и выведено как текст: 12
***Serial.print(2.9)*** – это число тоже будет преобразовано в текст: 12.9
***Serial.print(65)*** – в данном случае 65 передается как int и будет сконвертировано в строку “65”, которая и будет отправлена в последовательный порт.

Т.к. функция является членом класса Serial, мы должны использовать название класса и символ точки “.” в ее синтаксисе . Класс Serial объявляется в библиотеке Serial, но нам не нужно подключать эту библиотеку отдельно, она является внутренней библиотекой ардуино и подключается автоматически. Функция не возвращает значение, поэтому ее нельзя использовать для форматирования и последующего использования в скетче. Для этих целей больше подойдут методы класса String.

Перед использованием функции необходимо предварительно открыть работу с последовательным портом, используя функцию begin().

Serial begin – крайне важная инструкция Arduino, она позволяет установить контроллеру соединение с внешними устройствами. Чаще всего таким «внешним устройством» оказывается компьютер, к которому мы подключаем Arduino. Поэтому Serial begin интенсивней всего используется в скетчах, выводящих какую-то информацию на экран монитора порта, например, для отладки программы. С помощью последовательного порта плата ардуино соединяется с Bluetooth, GSM, GPS и многими другими модулями, поэтому и Serial begin – частый гость в скетчах ардуинщиков.

Функция begin() является методом класса Serial. Для работы с этим классом не нужно подключать внешних библиотек, т.к. он встроен в среду разработки ардуино. Использовать Serial.begin() целесообразно один раз, в самом начале работы скетча, в функции void setup(). В качестве аргументов нужно указать скорость обмена данными через последовательный порт. Самым распространенным значением является 9600 – именно такая скорость обмена

данными в мониторе порта Arduino IDE стоит по умолчанию. После выполнения функции ардуино будет способна как отправлять данные на внешние устройства, так и получать их. Пример использования функции:

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

Другие популярные методы класса Serial:

**print()**
**println()**
**read()**
**available()**
**parseInt()**

На платах с несколькими «железными» последовательными портами, например, Arduino Mega, для каждого из них создается свой объект Serial, поэтому вместе с Serial могут встречаться вызовы объектов Serial1, Serial2, Serial3.

Команда **Serial.begin ()** состоит из двух частей. Первая часть, Serial, является названием класса библиотеки для работы с монитором порта. Вторая часть, begin() обозначает название метода, которому мы должны передать один аргумент – скорость обмена данными. Функция не возвращает значений. Синтаксис функции, два варианта:

**Serial.begin(<скорость>)**
**Serial.begin(<скорость>, <config> )**

Параметры:

скорость – скорость взаимодействия (бит/сек или бод).

config – настройки режима работы последовательного порта. По умолчанию используется наиболее распространенный вариант, SERIAL_8N1.

*Скорость для последовательного порта в Serial.begin()*

Стандартное значение скорости в аргументе функции begin() – 9600. Эта цифра означает, что плата ардуино будет посылать по последовательному порту данные со скоростью 9600 бит в секунду (нужно учитывать, что для каждых 8 бит еще высылаются служебный стоповый бит). Естественно, приемно-передающее устройства на другом конце провода тоже должно быть настроено на эту же скорость, иначе оно будет пропускать некоторые биты или излишне торопиться, что приведет к рассинхронизации – информация будет как бы рваться на кусочки. Поэтому без лишней надобности не меняйте параметр по умолчанию – 9600.

Кроме стандартного значения можно устанавливать и другие: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200. Чем выше цифра, тем выше скорость обмена, но нужно следить, чтобы эту скорость поддерживало и внешнее устройство. В некоторых ситуациях слишком высокая скорость обмена может привести к ошибкам, нужно учитывать и это. Так, например, значение 115200 устанавливаются обычно специальными высокоскоростными устройствами, например, полетными контроллерами. Значения ниже 9600 используются крайне редко.

*Второй параметр для последовательного порта*

Параметр config в функции begin обычно не указывается, т.к. крайне редко надо менять значения по умолчанию. Но на всякий случай приведем здесь возможные варианты параметров, которые можно найти в HardwareSerial.h:

```
#define SERIAL_5N1 0x00
#define SERIAL_6N1 0x02
#define SERIAL_7N1 0x04
#define SERIAL_8N1 0x06
#define SERIAL_5N2 0x08
```

#define SERIAL_6N2 0x0A
#define SERIAL_7N2 0x0C
#define SERIAL_8N2 0x0E
#define SERIAL_5E1 0x20
#define SERIAL_6E1 0x22
#define SERIAL_7E1 0x24
#define SERIAL_8E1 0x26
#define SERIAL_5E2 0x28
#define SERIAL_6E2 0x2A
#define SERIAL_7E2 0x2C
#define SERIAL_8E2 0x2E
#define SERIAL_5O1 0x30
#define SERIAL_6O1 0x32
#define SERIAL_7O1 0x34
#define SERIAL_8O1 0x36
#define SERIAL_5O2 0x38
#define SERIAL_6O2 0x3A
#define SERIAL_7O2 0x3C
#define SERIAL_8O2 0x3E

В этих константах обозначаются разные варианты структуры пакета данных. Например, SERIAL_8N2 означает, что по последовательному порту ардуино передаст пакет длиной 8 бит без бита контроля четности (указано N) и с одним стоповым битом, который будет добавлен после байта данных.

Естественно, мы можем передавать в качестве параметров для функции print() не константы, а переменные определенных типов. Например, так:

***String stringForPrinting = "Hello from Arduino";***
***Serial.print (stringForPrinting);***

Текст, передаваемый с помощью print() обязательно завершается символом конца строки "\0", что является стандартом для C++ и Ардуино.

*Функция println и отличия от print*

Если вы попробовали использовать функцию print(), то уже обратили внимание, что вся информация в мониторе порта выводится в одной строке. Если же мы хотим вывести текст в новых строках, то должны использовать близкого родственника функции – println() .

Метод println () класса Serial выполняет ту же функцию, что и print() – он выводит в последовательный порт ASCII-текст. Аргументы у методов тоже совпадают – мы передаем текст или число с возможным вторым аргументом, определяющим формат. Отличие же println заключается в принудительном добавлении в конце передающейся строки символа новой строки "\r" (ASCII код 13). Суффикс ln обозначает сокращенное слово line (строка). Используя println, мы можем быть уверены, что следующая (но не текущая) строка будет выведена с новой строки.

Например, следующие команды выведут три строки текста, каждое предложение в новой строке.

**Serial.println("Line number 1");**
**Serial.println("Line number 2");**
**Serial.println("Line number 3");**

При формировании строки мы также можем использовать следующие специальные символы: "\0", "\r", "\t" (символ табуляции). Табулирование позволяет печатать на экране что-то типа таблицы значений:

**Serial.print("Column1\t\t");**
**Serial.println("Column2");**

**Serial.print("Cell 11\t\t");**
**Serial.println("Cel l2");**
**Serial.print("Cell 21\t\t");**
**Serial.println("Cel 22");**

Форматирование и конвертация строк

Если мы передаем в последовательный порт число и хотим, чтобы оно интерпретировалось как число определенной системы счисления и выводилось в соответствующем формате, то можем использовать второй аргумент функции.

Варианты значений констант для форматирования:

DEC – обычное число в десятичной системе исчисления

BIN – преобразует в двоичный код и выведет строку, содержащую только символы 0 и 1

OCT – преобразует в восьмеричную систему исчисления

HEX – преобразует в шестнадцатеричную систему

Цифра от 0 до 9 – используется, если первый аргумент – вещественное число с плавающей запятой. Форма указывает количество знаков после запятой, которые останутся при выводе. Само число при этом будет округлено. Примеры:

**Serial.println(65, DEC); // выведет “65”**

**Serial.println(65, BIN); // выведет “1000001”**

**Serial.println(65, OCT); // выведет 101, т.к. 65 в 8-ной системе исчисления равно 101**

**Serial.println(65, HEX); // выведет 41, т.к. 65 в 16-ной системе исчисления равно 41**

**Serial.println(9.876, 2); // выведет два символа после запятой, предварительно округлив 9.88**

**Serial.println(9.876, 0); // выведет число 10**

В старых версиях ардуино можно было использовать еще один параметр BYTE. Начиная с версии 1.0 эта константа не поддерживается и компилятор выдаст вам ошибку «Ключевое слово ‘BYTE’ больше не поддерживается». Для вывода ASCII символа по его коду нужно использовать метод write().

*Функция write() библиотеки Serial*

Если мы хотим передать в монитор порта не ASCII код числа, а само число, то у нас есть возможность это сделать с помощью функции write().

**Serial.write(65);** – В данном случае на экране мы увидим символ A, так как 65 – это символ A в ASCII. Используя функцию write, мы отправляем число 65 монитору порта в виде… числа 65, а не текста “65”, как в случае функции print(). При выводе на экран уже сам монитор порта будет конвертировать его в соответствие с таблице ASCII. В данном случае, таким символом является латинская буква A.

**Serial.print(65);** – А теперь мы увидим на экране 65, т.к. в монитор придет уже строка, а не цифра. Каждая цифра исходного числа 65 будет предварительно сконвертирована в ASCII код и монитор порта увидит сроку «65», а не число. Этим и отличаются друг от друга функции print и write.

*Проблемы с несколькими аргументами в Serial.print*

Проблема при объединении текста и чисел и выводе в print в одной строке

Часто для отладки программы требуется вывести несколько значений, снабдив их каким-то комментарием. Например, такой текст: «Sensor’s value is: 15». Если вы просто используете такой код:

**Serial.print(“Sensor’s value is: 15”);**

,то все отобразится правильно. Но если вы попытаетесь вместо подстроки «15» вставить реальное показание датчика, объединив строку и числовое значение, то увидите, что строка выводится некорректно.

**Serial.print(“Sensor’s value is:  ” + analogRead (A0));**

Этот код даст непредсказуемый результат, в мониторе порта вы не увидите или пустоту или случайный набор символов. Причина ошибки в механизме конвертации типов данных. При объединении строк  «на лету», как в нашем примере, ардуино не знает, как интерпретировать типы данных для аргументов при операторе суммирования. В процессе такой непредсказуемой конвертации и результат может быть непредсказуемым.

Для решения этой проблемы вы можете использовать два способа:

Первый вариант. Объявить предварительно переменную типа String, инициализировать ее константой-строкой и использовать в аргументе print. Такой вот пример будет работать:

**String str = “Sensor’s value is:  “;**

**Serial.println(str + analogRead(A0));**

Второй, более предпочтительный и удобный вариант: использование функции print несколько раз:

**Serial.print(“Sensor’s value is:  “);**

**Serial.println(analogRead(A0));**

Более подробно прочитать об особенностях  работы со строками при выводе информации в монитор порта можно на официальной странице Ардуино.

*Проблема объединения нескольких строк в аргументе функции print*

Попробуйте загрузить скетч с таким фрагментом:

*Serial.print(“Sensor’s value is:  ” + analogRead (A0) + “15 cm”);*

В данном случае ошибка возникнет уже на стадии компиляции программы: оператор «+» не может использовать больше, чем  два аргумента. Вам придется разбить одно действие на несколько составляющих, создав предварительно готовые «заготовки» строк.

*Пример скетча для print и println*

В завершении давайте рассмотрим реальный скетч, в котором мы используем монитор порта Arduino IDE для вывода отладочной информации с платы контроллера.

```
void setup() {
  // Объявляем работу с последоватлеьным портом в самом начале
  Serial.begin(9600);
  // Теперь мы можем писать сообщения
  Serial.println ("Hello, Arduino Master");
}

void loop() {
  // Выводим таблицу с информацией о текущих значениях портов
  Serial.print("Port #\t\t");
  Serial.println("Value");
  Serial.print("A0\t\t");
  Serial.println(analogRead(A0));
  Serial.print("A1\t\t");
  Serial.println(analogRead(A1));
  Serial.println("--------");
  delay(1000);
}
```

*Приём данных*

Выводить данные в порт просто. А вот принимать данные с компьютера и других источников сложнее. При отправлении данных, они складываются в буфер, ожидая, когда плата их прочитает. Объём буфера составляет 64 байта. Чтобы постоянно не читать пустой буфер, есть специальная функция проверки буфера **Serial.available().** Она возвращает число байт, которые лежат в буфере. Обычно в коде создают условие проверки - если в буфере больше 0 байт, то выполняем какие-то команды.

Для демонстрации создадим странный пример - создадим переменную, присвоим ей данные через Serial.read() и попросим её прислать полученные данные через Serial.print(). Получится круговорот данных или эхо.

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  if (Serial.available() > 0) {
    int data = Serial.read();
    Serial.println(data);
  }
}
```

Проверяем на числах. Отправляем число 9, а получаем 57. Если вы получаете две строки с числами 57 и 10, то в нижней части окна выберите настройку No line ending вместо Newline.

Попробуем также отправить букву. Опять вместо t возвращается 116. Ерунда какая-то. Всё просто, функция read() работает с символьными значениями и мы видим код символа из стандартной таблицы символов ASCII.

Чтобы решить проблему, нужно изменить тип данных на char.

```
char data = Serial.read();
```

Вроде проблема решена. Мы можем принимать отдельные цифры и буквы. Но буквы только английские, а числа только однозначные.

Если мы планируем работать только с однозначными числами, то можно написать такой код.

```
int data = Serial.read() - '0';
```

Решение какое-то половинчатое. А как быть с большими числами или словами?

Если отправить двузначное число 23, то ответ разбивается на части - 2 и 3. Получается, что переменная получит последнее число 3 (промежуточные значения перезаписываются). Чтобы обработать всё число, нужно использовать метод parseInt().

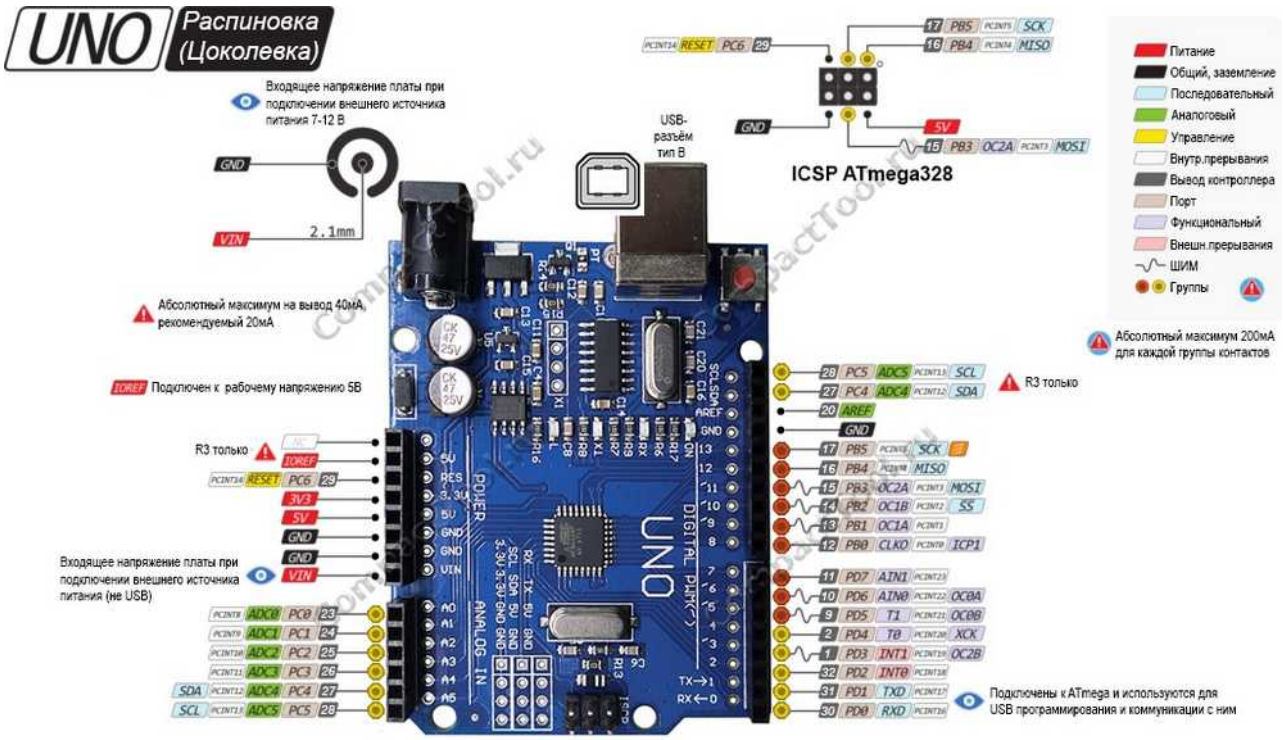
```
int data = Serial.parseInt();
```

Теперь вы можете вводить любые числа. Но, наверное, вы заметите теперь небольшую задержку в ответах. Метод внутри себя перемалывает данные. Кстати, вы можете использовать и обычные символы. Если набор символов состоит только из букв, то вернётся 0. Если будут попадаться и цифры, то будут возвращаться цифры. Попробуйте комбинировать различные сочетания цифр и букв, чтобы понять, как будут обрабатываться данные.

**Приложения:**

# ASCII Table

| Dec | Hex | Oct | Char | Dec | Hex | Oct | Char | Dec | Hex | Oct | Char | Dec | Hex | Oct | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | | 32 | 20 | 40 | [space] | 64 | 40 | 100 | @ | 96 | 60 | 140 | ` |
| 1 | 1 | 1 | | 33 | 21 | 41 | ! | 65 | 41 | 101 | A | 97 | 61 | 141 | a |
| 2 | 2 | 2 | | 34 | 22 | 42 | " | 66 | 42 | 102 | B | 98 | 62 | 142 | b |
| 3 | 3 | 3 | | 35 | 23 | 43 | # | 67 | 43 | 103 | C | 99 | 63 | 143 | c |
| 4 | 4 | 4 | | 36 | 24 | 44 | $ | 68 | 44 | 104 | D | 100 | 64 | 144 | d |
| 5 | 5 | 5 | | 37 | 25 | 45 | % | 69 | 45 | 105 | E | 101 | 65 | 145 | e |
| 6 | 6 | 6 | | 38 | 26 | 46 | & | 70 | 46 | 106 | F | 102 | 66 | 146 | f |
| 7 | 7 | 7 | | 39 | 27 | 47 | ' | 71 | 47 | 107 | G | 103 | 67 | 147 | g |
| 8 | 8 | 10 | | 40 | 28 | 50 | ( | 72 | 48 | 110 | H | 104 | 68 | 150 | h |
| 9 | 9 | 11 | | 41 | 29 | 51 | ) | 73 | 49 | 111 | I | 105 | 69 | 151 | i |
| 10 | A | 12 | | 42 | 2A | 52 | * | 74 | 4A | 112 | J | 106 | 6A | 152 | j |
| 11 | B | 13 | | 43 | 2B | 53 | + | 75 | 4B | 113 | K | 107 | 6B | 153 | k |
| 12 | C | 14 | | 44 | 2C | 54 | , | 76 | 4C | 114 | L | 108 | 6C | 154 | l |
| 13 | D | 15 | | 45 | 2D | 55 | - | 77 | 4D | 115 | M | 109 | 6D | 155 | m |
| 14 | E | 16 | | 46 | 2E | 56 | . | 78 | 4E | 116 | N | 110 | 6E | 156 | n |
| 15 | F | 17 | | 47 | 2F | 57 | / | 79 | 4F | 117 | O | 111 | 6F | 157 | o |
| 16 | 10 | 20 | | 48 | 30 | 60 | 0 | 80 | 50 | 120 | P | 112 | 70 | 160 | p |
| 17 | 11 | 21 | | 49 | 31 | 61 | 1 | 81 | 51 | 121 | Q | 113 | 71 | 161 | q |
| 18 | 12 | 22 | | 50 | 32 | 62 | 2 | 82 | 52 | 122 | R | 114 | 72 | 162 | r |
| 19 | 13 | 23 | | 51 | 33 | 63 | 3 | 83 | 53 | 123 | S | 115 | 73 | 163 | s |
| 20 | 14 | 24 | | 52 | 34 | 64 | 4 | 84 | 54 | 124 | T | 116 | 74 | 164 | t |
| 21 | 15 | 25 | | 53 | 35 | 65 | 5 | 85 | 55 | 125 | U | 117 | 75 | 165 | u |
| 22 | 16 | 26 | | 54 | 36 | 66 | 6 | 86 | 56 | 126 | V | 118 | 76 | 166 | v |
| 23 | 17 | 27 | | 55 | 37 | 67 | 7 | 87 | 57 | 127 | W | 119 | 77 | 167 | w |
| 24 | 18 | 30 | | 56 | 38 | 70 | 8 | 88 | 58 | 130 | X | 120 | 78 | 170 | x |
| 25 | 19 | 31 | | 57 | 39 | 71 | 9 | 89 | 59 | 131 | Y | 121 | 79 | 171 | y |
| 26 | 1A | 32 | | 58 | 3A | 72 | : | 90 | 5A | 132 | Z | 122 | 7A | 172 | z |
| 27 | 1B | 33 | | 59 | 3B | 73 | ; | 91 | 5B | 133 | [ | 123 | 7B | 173 | { |
| 28 | 1C | 34 | | 60 | 3C | 74 | < | 92 | 5C | 134 | \ | 124 | 7C | 174 | \| |
| 29 | 1D | 35 | | 61 | 3D | 75 | = | 93 | 5D | 135 | ] | 125 | 7D | 175 | } |
| 30 | 1E | 36 | | 62 | 3E | 76 | > | 94 | 5E | 136 | ^ | 126 | 7E | 176 | ~ |
| 31 | 1F | 37 | | 63 | 3F | 77 | ? | 95 | 5F | 137 | _ | 127 | 7F | 177 | |




# Эмуляторы Ардуино:

**1. VirtualBreadBoard**
Официальный сайт разработчика VirtualBreadBoard – http://www.virtualbreadboard.com/
Полезные ссылки
https://voltiq.ru/virtual-bread-board-arduino-simulator/
https://studfile.net/preview/396729/page:7/
https://habr.com/ru/sandbox/34607/
http://nauchebe.net/2012/01/otladka-programmy-na-virtualnoj-plate/

**2. Fritizing**
Сайт https://fritzing.org/
Полезные ссылки:
https://voltiq.ru/fritzing-review/
http://blog.amperka.ru/%D1%83%D1%80%D0%BE%D0%BA%D0%B8-fritzing
http://microsin.net/adminstuff/others/fritzing-overview.html
http://robotosha.ru/arduino/own-part-fritzing-part1.html
http://robotosha.ru/arduino/own-part-fritzing-part-2.html
http://fritzing.org/learning/tutorials/creating-custom-parts/providing-part-graphics/

**3. UnoArduSim**
Сайт https://www.sites.google.com/site/unoardusim/services
Полезные ссылки:
http://digitrode.ru/computing-devices/mcu_cpu/1717-emulyator-arduino-unoardusim-pozvolyaet-testirovat-kod-bez-apparatnyh-sredstv.html
https://kolotushkin.com/article.php?id=2
https://www.youtube.com/watch?v=qJUYlKSfdr8

**Задания к лабораторной работе №1 (в зависимости от режима обучения)**
**В лаборатории (очно):**
1. Программа hello world — вывод в консоль
2. Вывод случайного числа в консоль в десятичном, шестнадцатеричном и двоичном виде
3. Изменение скорости порта до 115200, с выводом этого значения в консоль
4. Ввод символьного значения (чтение введенного числа от 0 до 9)
5. Создание простейшего командного интерпретатора. Должны поддерживаться команды:
help — экран помощи
command1 — отображение в консоли введенной команды command1
command2 - отображение в консоли введенной команды command2
setInt — ввод целого числа от -100 до +200 с выводом ошибки или введенного числа
setStr — ввод строки до 8 символов с отображение введенной строки (только латинские буквы и цифры), вывод ошибки
info — вывод скорости порта

**Удаленно (необходимо сделать одно это задание):**
Создание простейшего командного интерпретатора. Должны поддерживаться команды:
help — экран помощи
command1 — отображение в консоли введенной команды command1
time - отображение в консоли результата вызова функции millis()
setInt — ввод целого числа от -100 до +200 с выводом ошибки или введенного числа
setStr — ввод строки до 8 символов с отображение введенной строки (только латинские буквы и цифры), вывод ошибки.
info — версии интерпретатора, автора разработки (задаются константами в коде)

*Для продвинутых*, реализовать интерпретатор без использования класса String и реализация дополнительной команды setFloat — ввод дробного числа от -30 до +50 с точностью 2 десятичных разряда и выводом ошибки или введенного числа.