## 1.Введение.

В настоящее время исследования в области робототехники являются весьма актуальными. Одними из популярных конструкций роботов, являются робототехнические комплексы различного назначения.

В работе рассматривается примеры программирования на платформе Arduino.

Робототехника — новое интересное направление. Бум робототехники во многом связан с тем, что она позволяет ответить на вопрос: «Зачем нужно учиться программированию?».

Появление первых микроконтроллеров ознаменовало начало новой эры в развитии микропроцессорной техники. Наличие в одном корпусе большинства системных устройств сделало микроконтроллер подобным обычному компьютеру. В связи с этим появилось желание использовать микроконтроллеры как обычные компьютеры.

Сейчас, с появлением устройств, дающих возможность работать с микроконтроллерами без наличия серьезной материальной базы и знания многих предметов, все изменилось. Примером такого устройства может служить проект Arduino итальянских разработчиков. Arduino представляют собой наборы, состоящие из готового электронного блока и программного обеспечения. Электронный блок - это печатная плата с установленным микроконтроллером и минимумом элементов, необходимых для его работы. Фактически электронный блок Arduino является аналогом материнской платы современного компьютера. На нем имеются разъемы для подключения внешних устройств, а также разъем для связи с компьютером, по которому и осуществляется программирование микроконтроллера. Все, что нужно для создания нового электронного устройства, — это плата Arduino, кабель связи и компьютер. Второй частью проекта Arduino является программное обеспечение для создания управляющих программ. Оно объединило в себе простейшую среду разработки и язык программирования, представляющий собой вариант языка C/C++ для микроконтроллеров. В него добавлены элементы, позволяющие создавать программы без изучения аппаратной части. Создано для Arduino и множество библиотек, содержащих код, работающий с различными устройствами.

Впервые с Arduino и микроконтроллерами мы познакомились, просматривая видео в интернете. И задались целью научиться программировать на платформе Arduino.

Главной проблемой на начальном этапе был недостаток теоретических знаний по программированию.

Мы изучили книги, которые смогли найти: «Блокнот программиста ардуино», «Знакомство с ардуино». Затем решили применить полученные знания на практике, адаптировать известные проекты под свою платформу и создать свой проект «Движение детской машинки».

**Цель:** Получить базовые знания по программированию на платформе Arduino.

**Задачи:**

1. Научиться основам электротехники.
2. Описать преимущества микроконтроллера Arduino.
3. Показать применение микроконтроллера Arduino в робототехнике.
4. Получить знания по программированию на платформе Arduino
5. Создать несколько схем на основе полученных знаний.

**Актуальность темы:** В связи с развитием робототехники и программирования появилась возможность изучения азов программирования с помощью платформы Arduino для создания простых программируемых схем и даже полноценных роботов. Эта тема на данный момент актуальна ещё и тем, что в нашем обиходе появилась «разумная техника» способная выполнять, то, что может и чего не может человек.

**Проблема исследования** заключается в поиске точек соприкосновения компьютерной грамотности в области программирования роботов и физико-математического образования получаемого в школе.

**Объект исследования** – платформа Arduino.

**Предмет исследования** – Построение моделей на основе микроконтроллера Arduino.

**Гипотеза:** Построенные модели роботов будут более функциональны, если использовать в качестве основы для построении модели микроконтроллер Arduino, чем модели, построенные на основе других микроконтроллеров.

**Методы исследования**:

- исследование научных форумов;
- поиск информации о современных этапах робототехники, построение и изучении моделей роботов;
- анализ теоретических источников по теме исследования;
- наблюдение;
- анализ работы.

**Теоретическая значимость** работы заключается в представлении платформы Arduino, подбору материалов для первоначального знакомства.

**Практическая значимость:** представленный материал может быть использован в работе педагогов, использующим микроэлектронику в своей деятельности. Так же она будет полезна ученикам, изучающих курс программиро-

вания на языках Паскаль и Си и посещающих кружки и факультативы по робототехнике.

# 2. Аналитический обзор

## 2.1.Микроконтроллеры в робототехнике.

Микроконтроллеры можно встретить почти в любом современном электронном цифровом (и не только цифровом) устройстве: мобильных телефонах, фотокамерах, калькуляторах, часах, телевизорах, плеерах, компьютерах, в промышленной электронике, автомобильной электронике, военной технике и др. В основном микроконтроллеры применяться там, где приоритетным является уменьшение размеров, потребляемой мощности, увеличение устойчивости к внешним факторам, например в роботах. Быстродействие, значительно меньше чем у мощных процессоров, но его хватает для выполнения большинства требуемых от устройства функций. Технологии совершенствуется, и быстродействие микроконтроллеров возрастает. Новые поколения МК могут выполнять сложные расчеты за малое время. Но, хотя производители стремятся обеспечить работу своих изделий на высоких частотах, они, в то же время, предоставляют заказчикам выбор, выпуская модификации, рассчитанные на разные частоты и напряжения питания.

Теоретическая робототехника представляет собой пример впечатляющих результатов междисциплинарного взаимодействия. Создаваемые на этой основе технические системы способны исследовать не только поверхности удаленных планет и глубины океана, но и внутреннее строение человеческих органов, а также молекулярную структуру органических и неорганических веществ на нано-уровне. Робототехнические системы представляют собой устройства, позволяющие наиболее адекватно смоделировать характеристики и принципы управления, соответствующие некоторым функциям человеческого тела [8].

Исследования по робототехнике и мехатронике в основном начались в самом начале 70-х годов. Они были связаны с проблемой создания роботов и транспортных средств нового типа - шагающих аппаратов. С теоретической точки зрения, задачи управления движением таких механических объектов составили принципиально новый класс задач управления сложными системами со многими механическими степенями свободы [9]

В настоящее время эти исследования продолжаются. Широко используются современные средства работы на ЭВМ (например, графика и мультимедиа), применяется современная микрокомпьютерная техника, бортовые ЭВМ, интегральные датчиковые системы [10]. Создаются реальные прототипы будущих машин.

Разработаны эффективные методы удаленного управления роботами через Интернет и, более широко, в случаях управления роботами с задержками управляющего сигнала и при нагруженных каналах связи. Методы основаны на использовании "виртуального дублера" - трехмерных моделей робота и его рабочего пространства, функционирующих в масштабе реального времени.

## 2.2.Преимущество Arduino.

Плата Arduino позволяет непрофессионалам создавать и программировать простые устройства на микроконтроллерах. С её помощью можно изучать работу цифровых и аналоговых портов ввода и вывода, принимать сигналы от датчиков и управлять работой приводов и индикаторов.

Arduino — это электронный конструктор и удобная платформа быстрой разработки электронных устройств для новичков и профессионалов. Платформа пользуется огромной популярностью во всем мире благодаря удобству и простоте языка программирования, а также открытой архитектуре и программному коду. Устройство программируется через USB без использования программаторов.

Arduino позволяет компьютеру выйти за рамки виртуального мира в физический и взаимодействовать с ним. Устройства на базе Arduino могут получать информацию об окружающей среде посредством различных датчиков, а также могут управлять различными исполнительными устройствами.

Микроконтроллер на плате программируется при помощи языка Arduino и среды разработки Arduino. Проекты устройств, основанные на Arduino, могут работать самостоятельно, либо же взаимодействовать с программным обеспечением на компьютере (напр.: Flash, Processing, MaxMSP). Платы могут быть собраны пользователем самостоятельно или куплены в сборе. Программное обеспечение доступно для бесплатного скачивания. Исходные чертежи схем (файлы CAD) являются общедоступными, пользователи могут применять их по своему усмотрению.

Пользователь современного компьютера не задумывается о функционировании отдельных частей ПК. Он просто запускает нужные программы и работает с ними. Точно так же и Arduino позволяет пользователю сосредоточиться на разработке проектов, а не на изучении устройства и принципов функционирования отдельных элементов. Нет надобности и в создании законченных плат и модулей. Разработчик может использовать готовые платы расширения или просто напрямую подключить к Arduino необходимые элементы. Все остальные усилия будут направлены на разработку и отладку управляющей программы на языке высокого уровня. В итоге доступ к разработке микропроцессорных устройств получили не только профессионалы, но и просто любители что-то сделать своими руками. Наличие готовых модулей и библиотек программ позволяет непрофессионалам в электронике создавать готовые работающие устройства для решения своих задач. Варианты использования Arduino ограничены только возможностями микроконтроллера и имеющегося варианта платы и фантазией разработчика.

## 2.3.Основные термины

**Аппаратная часть**

Arduino — аппаратная вычислительная платформа, основными компонентам которой являются простая плата ввода/вывода и среда разработки.

Плата Arduino состоит из микроконтроллера и элементной обвязки для программирования и интеграции с другими схемами. На каждой плате обязательно присутствуют линейный стабилизатор напряжения 5 В и 16 МГц кварцевый генератор (в некоторых версиях керамический резонатор). В микроконтроллер предварительно прошит загрузчик, поэтому внешний программатор не нужен.

**А что такое контроллер?**

Контроллер это такое электронное устройство, которое что-нибудь контролирует - реагирует на изменения одних параметров изменением других. Кондиционер, mp3 плеер, велокомпьютер, сигнализация, мобильник, навигатор - всё это пример специализированных контроллеры. Компьютер настольный это контроллер универсальный, расширяемый, с его помощью всё вышеперечисленное можно реализовать. Нужны будут только соответствующие платы расширения и софт.

**Arduino**

Ардуино универсальный контроллер, который можно заточить под какую-нибудь задачу и превратить в законченное электронное устройство произвольного назначения, от часов с будильником до робота.

Так же подключая к Arduino различные устройства - шилды (shields) можно добавить различные функции - так можно управлять всякими двигателями, сервомашинками, сетевой нагрузкой (свет, обогреватель, чайник и т.п.). Можно подключить GPS или GSM модуль и получать координаты со спутника Можно вставить Ethernet-модуль и выпустить свой девайс в интернет, который сможет передавать данные на сайт, или писать всё на SD-карту вставленную в соответствующий шилд. Можно добавить каналы связи – ИК, радиоканал, Bluetooth и тому подобное.

**Ардуино имеет ряд преимуществ:**

- Не нужен программатор.
- Не нужны особо глубокие познания в программирования микроконтроллеров.
- Проект Ардуино полностью открытый.
- Платформа набирает популярность - много сайтов с библиотеками, схемами и проектами.
- Стандартизация расположения выводов - это делает её привлекательной для производителей – появляются всё новые шилды.

Arduino - настоящий конструктор, для создания прототипов и реализации идей.

**Sketch (скетч)** - креатив, то что должна будет делать плата. Пишется в IDE на языке Wiring.

Оригинальные платы.

Сами итальянцы выпускают плату в нескольких основных форм-факторах:

ArduinoUNO - стандартный размер, 20входо-выходов, полная совместимость со всеми шилдами.

ArdinoMega - увеличенный размер, 70входо-выходов, совместимость не со всеми шилдами.

ArdinoNano - уменьшеный размер, 22входо-выхода, не совместима с шилдами.

ArdinoMini — ещё меньший размер, 20входо-выхоов, не совместима с шилдами, не имеет USB.

**Клоны**

Такие платы («клоны») полностью повторяют Arduino и полностью совместимы с ней. То есть, разница между клоном и оригиналом — только в производителе (иногда в цвете) — соответственно различия могут быть только в качестве сборки, качестве компонентов, строгости выходного контроля.

**Питание**

Arduino Uno может питаться как от USB подключения, так и от внешнего источника: батарейки или обычной электрической сети. Источник определяется автоматически.

Платформа может работать при наличии напряжения от 6 до 20 В. Однако при напряжении менее 7 В работа может быть неустойчивой, а напряжение более 12 В может привести к перегреву и повреждению. Поэтому рекомендуемый диапазон: 7−12 В.

На Arduino доступны следующие контакты для доступа к питанию:

- Vin предоставляет тот же вольтаж, что используется для питания платформы. При подключении через USB будет равен 5 В.
- 5V предоставляет 5 В вне зависимости от входного напряжения. На этом напряжении работает процессор. Максимальный допустимый ток, получаемый с этого контакта — 800 мА.
- 3.3V предоставляет 3,3 В. Максимальный допустимый ток, получаемый с этого контакта — 50 мА.
- GND — земля.

**Память**

Платформа оснащена 32 кБ flash-памяти, 2 кБ из которых отведено под так называемый bootloader. Он позволяет прошивать Arduino с обычного компьютера через USB. Эта память постоянна и не предназначена для изменения по ходу работы устройства. Её предназначение — хранение программы и сопутствующих статичных ресурсов.

Также имеется 2 кБ SRAM-памяти, которые используются для хранения временных данных вроде переменных программы. По сути, это оперативная память платформы. SRAM-память очищается при обесточивании.

Ещё имеется 1 кБ EEPROM-памяти для долговременного хранения данных. По своему назначению это аналог жёсткого диска для Arduino.

**Ввод / вывод**

На платформе расположены 14 контактов (pins), которые могут быть использованы для цифрового ввода и вывода. Какую роль исполняет каждый контакт, зависит от вашей программы. Все они работают с напряжением 5 В, и рассчитаны на ток до 40 мА. Также каждый контакт имеет встроенный, но отключённый по умолчанию резистор на 20 - 50 кОм. Некоторые контакты обладают дополнительными ролями:

- Serial: 0-й и 1-й. Используются для приёма и передачи данных по USB.
- Внешнее прерывание: 2-й и 3-й. Эти контакты могут быть настроены так, что они будут провоцировать вызов заданной функции при изменении входного сигнала.
- PWM: 3-й, 5-й, 6-й, 9-й, 10-й и 11-й. Могут являться выходами с широтно-импульсной модуляцией с 256 градациями.
- LED: 13-й. К этому контакту подключен встроенный в плату светодиод. Если на контакт выводится 5 В, светодиод зажигается; при нуле — светодиод гаснет.

Помимо контактов цифрового ввода/вывода на Arduino имеется 6 контактов аналогового ввода, каждый из которых предоставляет разрешение в 1024 градации. По умолчанию значение меряется между землёй и 5 В, однако возможно изменить верхнюю границу, подав напряжение требуемой величины на специальный контакт AREF.

Кроме этого на плате имеется входной контакт Reset. Его установка в логический ноль приводит к сбросу процессора. Это аналог кнопки Reset обычного компьютера.

**Взаимодействие**

Arduino Uno обладает несколькими способами общения с другими Arduino, микроконтроллерами и обычными компьютерами. Платформа позволяет установить последовательное соединение через контакты 0 (RX) и 1 (TX). Установленный на платформе чип транслирует это соединение через USB: на компьютере становится доступен виртуальный COM-порт. Программная часть Arduino включает утилиту, которая позволяет обмениваться текстовыми сообщениями по этому каналу.

Встроенные в плату светодиоды RX и TX светятся, когда идёт передача данных между чипом и USB компьютера.

Отдельная библиотека позволяет организовать последовательное соединение с использованием любых других контактов, не ограничиваясь штатными 0-м и 1-м.

С помощью отдельных плат расширения становится возможной организация других способов взаимодействия, таких как ethernet-сеть, радиоканал, Wi-Fi.

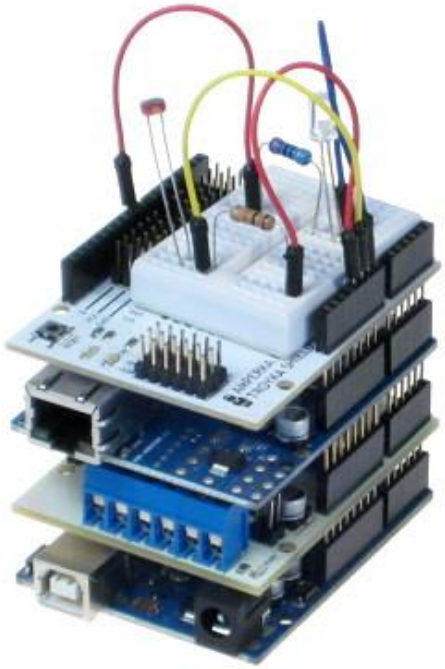
**Защита USB**

Arduino Uno обладает предохранителем, защищающим USB-порты вашего компьютера от перенапряжения и коротких замыканий. Хотя большинство компьютеров обладают собственными средствами защиты, предохранитель да-

ёт дополнительную уверенность. Он разрывает соединение, если на USB-порт подаётся более 500 мА, и восстанавливает его после нормализации ситуации.

**Габариты**

Размер платы составляет 6,9 × 5,3 см. Гнёзда для внешнего питания и USB выступают на пару миллиметров за обозначенные границы. На плате предусмотрены места для крепления на шурупы или винты. Расстояние между контактами составляет 0,1, но в случае 7-го и 8-го контакта — расстояние: 0,16″.



**Принцип бутерброда**

Ещё одной отличительной особенностью Arduino является наличие плат расширения, так называемых shields или просто «шилдов». Это дополнительные платы, которые ставятся подобно слоям бутерброда поверх Arduino, чтобы дать ему новые возможности. Так например, существуют платы расширения для подключения к локальной сети и интернету (Ethernet Shield), для управления мощными моторами (Motor Shield), для получения координат и времени со спутников GPS (модуль GPS) и многие другие.

## 3. Основы программирование на Arduino

### 1.структура

Базовая структура программы для Arduino довольно проста и состоит из двух частей. В этих двух обязательных частях, или функциях, заключён выполняемый код.

```
void setup()
{
  statements;
}

void loop()
{
  statements;
}
```

Где setup() — это подготовка, а loop() — выполнение. Обе функции требуются для работы программы. Перед функцией setup - в самом начале программы, обычно, идёт, объявление всех переменных. setup - это первая функция, выполняемая программой, и выполняемая только один раз, поэтому она используется для установки режима работы портов (pinMode()) или инициализации последовательного соединения

Следующая функция loop содержит код, который выполняется постоянно - читаются входы, переключаются выходы и т.д. Эта функция — ядро всех программ Arduino и выполняет основную работу.

#### *1.1setup()*

Функция setup() вызывается один раз, когда программа стартует. Используйте её для установки режима выводов или инициализации последовательного соединения. Она должна быть включена в программу, даже если в ней нет никакого содержания.

```
void setup()
{
  pinMode(pin, OUTPUT);       //  устанавливает 'pin' как выход
}
```

#### *1.2loop()*

После вызова функции setup() – управление переходит к функции loop() , которая делает в точности то, что означает её имя — непрерывно выполняется, позволяя

```
void loop()
{
  digitalWrite(pin, HIGH);    // включает 'pin'
  delay(1000);                // секундная пауза
  digitalWrite(pin, LOW);     // выключает 'pin'
  delay(1000);                // секундная пауза
}
```

программе что-то изменять, отвечать и управлять платой Arduino.

### *2.функции*

Функция — это блок кода, имеющего имя, которое указывает на исполняемый код, который выполняется при вызове функции. Функции void setup() и void loop(). Могут быть написаны различные пользовательские функции, для

выполнения повторяющихся задач и уменьшения беспорядка в программе. При создании функции, первым делом, указывается тип функции. Это тип значения, возвращаемого функцией, такой как 'int' для целого (integer) типа функции. Если функция не возвращает значения, её тип должен быть void. За типом функции следует её имя, а в скобках параметры, передаваемые в функцию.

```
type functionName(parameters)
{
  statements;
}
```

Следующая функция целого типа delayVal() используется для задания значения паузы в программе чтением значения с потенциометра. Вначале объявляется локальная переменная v, затем v устанавливается в значение потенциометра, определяемое числом между 0 — 1023, затем это значение делится на 4, чтобы результирующее значение было между 0 и 255, а затем это значение возвращается в основную программу.

```
int delayVal()
{
  int v;                    // создаём временную переменную 'v'
  v  = analogRead(pot);     // считываем значение с потенциометра
  v /= 4;                   // конвертируем 0 - 1023 в 0 - 255
  return v;                 // возвращаем конечное значение
}
```

***3.{} фигурные скобки***

Фигурные скобки (также упоминаются как просто «скобки») определяют начало и конец блока функции или блока выражений, таких как функция void loop() или выражений (statements) типа for и if.

```
type function()
{
  statements;
}
```

За открывающейся фигурной скобкой { всегда должна следовать закрывающаяся фигурная скобка }. Об этом часто упоминают, как о том, что скобки должны быть «сбалансированы». Несбалансированные скобки могут приводить к критическим, неясным ошибкам компиляции, вдобавок иногда и трудно выявляемым в больших программах.

Среда разработки Arduino, включает возможность удобной проверки баланса фигурных скобок. Достаточно выделить скобку, или даже щёлкнуть по точке вставки сразу за скобкой, чтобы её пара была подсвечена.

***3.; точка с запятой***

Точка с запятой должна использоваться в конце выражения и разделять элементы программы. Также точка с запятой используется для разделения элементов цикла for.

```
int x = 13;    // объявляет переменную 'x' как целое 13
```

**Примечание:** Если забыть завершить строку точкой с запятой, то это приведёт к возникновению ошибки компиляции. Текст ошибки может быть очевиден и указывать на пропущенную точку с запятой, но может быть и не таким очевидным.

Если появляется маловразумительная или нелогичная ошибка компилятора, первое, что следует проверить — не пропущена ли точка с запятой вблизи строки, где компилятор выразил своё недовольство.

***4. /* ... */ блок комментария***

Блок комментария или однострочный комментарий — это область текста, которая игнорируется программой и используется для добавления текста с описанием кода или примечаний. Комментарий помогают другим понять эту часть программы. Он начинается с /* и заканчивается */ и может содержать множество строк.

/* это «огороженный» блок комментария, и не забудьте «закрыть» комментарий - он должен быть сбалансирован! */

Поскольку комментарии игнорируются программой, а, следовательно, не занимают места в памяти, они могут быть достаточно ёмкими, но кроме того, они могут использоваться для «пометки» блоков кода с отладочной целью.

**Примечание:** Хотя допускается вставка однострочного комментария в блоке комментария, второй блок комментария не допускается.

***// однострочный комментарий***

Однострочный комментарий начинается с // и заканчивается (внутренним) кодом перехода на другую строку. Как и блок комментария, он игнорируется программой и не занимает места в памяти. // вот так выглядит однострочный комментарий

Однострочный комментарий часто используется после действенного выражения, чтобы дать больше информации о том, что выражение выполняет или в качестве напоминания на будущее.

**5. переменные**

Переменные — это способ именовать и хранить числовые значения для последующего использования программой. Само название - переменные, говорит о том, что переменные - это числа, которые могут последовательно меняться, в отличие от констант, чьё значение никогда не меняется. Переменные нужно декларировать (объявлять), и, что очень важно - им можно присваивать значения, которые нужно сохранить. Следующий код объявляет переменную inputVariable, а затем присваивает ей значение, полученное от 2-го аналогового порта:

```
int inputVariable = 0;          // объявляется переменная и
                                // ей присваивается значение 0
inputVariable = analogRead(2); //  переменная получает значение
                                // аналогового вывода 2
```

'inputVariable' — это наша переменная. Первая строка декларирует, что она будет содержать int, короткое целое. Вторая строка присваивает ей значение аналогового вывода 2. Это делает значение на выводе 2 доступным в любом месте программы.

Когда переменной присвоено значение, или пере-присвоено, вы можете проверить это значение, если оно встречается в некотором условии, или использовать его непосредственно. Рассмотрим пример, иллюстрирующий три

операции с переменными. Следующий код проверяет, не меньше ли 100 значение переменной, а если так, переменной inputVariable присваивается значение 100, а затем задаётся пауза, определяемая переменной inputVariable, которая теперь, как минимум, равна 100:

```
if (inputVariable < 100) // проверяем, не меньше ли 100 переменная
{
  inputVariable = 100;   // если так, присваиваем ей значение 100
}
delay(inputVariable);    // используем переменную как паузу
```

**Примечание:** Переменные должны иметь наглядные имена, чтобы код был удобочитаемый. Имена переменных как tiltSensor или pushButton помогают программисту при последующем чтении кода понять, что содержит эта переменная. Имена переменных как var или value, с другой стороны, мало делают для понимания кода, и здесь используются только в качестве примера.

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

***6.объявление переменных***

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

Следующий пример показывает, что объявленная переменная inputVariable имеет тип int, и её начальное значение равно нулю. Это называется простым присваиванием. int inputVariable = 0; Переменная может быть объявлена в разных местах программы, и то, где это сделано, определяет, какие части программы могут использовать переменную.

***7. границы переменных***

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

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

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

Следующий пример показывает, как декларировать несколько разных типов переменных, и демонстрирует видимость каждой переменной:

```
int value;                  // 'value' видима
                            // для любой функции
void setup()
{
  // no setup needed
}

void loop()
{
  for (int i=0; i<20;)   // 'i' видима только
  {                      // внутри цикла for
    i++;
  }
  float f;               // 'f' видима только
}                        // внутри loop
```

***7.1byte***

Байт хранит 8-битовое числовое значение без десятичной точки. Он имеет диапазон от 0 до 255.

```
byte someVariable = 180;    // объявление 'someVariable'
                            // как имеющей тип byte
```

***7.2int***

Целое — это первый тип данных для хранения чисел без десятичной точки, и хранит 16-битовое значение в диапазоне от 32767 до -32768.

```
int someVariable = 1500;    // объявляет 'someVariable'
                            // как переменную целого типа
```

**Примечание:** Целые переменные будут переполняться, если форсировать их переход через максимум или минимум при присваивании или сравнении.

Например, если x = 32767 и следующее выражение добавляет 1 к x, x = x +1 или x++, в этом случае x переполняется и будет равен -32678.

***7.3long***

Тип данных увеличенного размера для больших целых, без десятичной точки, сохраняемый в 32-битовом значении с диапазоном от 2147383647 до -2147383648.

```
long someVariable = 90000; // декларирует 'someVariable'
                           // типа long
```

***7.4float***

Тип данных для чисел с плавающей точкой или чисел, имеющих десятичную точку. Числа с плавающей точкой имеют большее разрешение, чем целые и сохраняются как 32-битовые значения в диапазоне от 3.4028235E+38 до -3.4028235E+38.

```
float someVariable = 3.14; // объявление 'someVariable'
                           //   как  floating-point тип
```

**Примечание:** Числа с плавающей точкой не точные, и могут выдавать странные результаты при сравнении. Вычисления с плавающей точкой медлен-

нее, чем вычисления целых при выполнении расчётов, так что, без нужды, их следует избегать.

```
int myArray[] = {value0, value1, value2...}
```

***массивы***

Массив — это набор значений, к которым есть доступ через значение индекса.

Любое значение в массиве может быть вызвано через вызов имени массива и индекса значения. Индексы в массиве начинаются с нуля с первым значением, имеющим индекс 0. Массив нуждается в объявлении, а дополнительно может заполняться значениями до того, как будет использоваться.

Схожим образом можно объявлять массив, указав его тип и размер, а позже присваивать значения по позиции индекса:

```
int myArray[5];     // объявляет массив целых длиной в 6 позиций
myArray[3] = 10;    // присваивает по 4у индексу значение 10
```

Чтобы извлечь значение из массива, присвоим переменной значение по индексу массива:

```
x = myArray[3];     // x теперь равно 10
```

Массивы часто используются в цикле for, где увеличивающийся счётчик применяется для индексации позиции каждого значения. Следующий пример использует массив для мерцания светодиода. Используемый цикл for со счётчиком, начинающимся с 0, записывает значение из позиции с индексом 0 массива flicker[], в данном случае 180, на PWM-вывод (широтно-импульсная модуляция) 10; затем пауза в 200 ms, а затем переход к следующей позиции индекса.

```
int ledPin = 10;                        // LED на выводе 10
byte flicker[] = {180, 30, 255, 200, 10, 90, 150, 60};
                                        // выше массив из 8
void setup()                            // разных значений
{
  pinMode(ledPin, OUTPUT);              //задаём OUTPUT вывод
}

void loop()
{
  for(int i=0; i<7; i++)                // цикл равен числу
  {                                     // значений в массиве
    analogWrite(ledPin, flicker[i]);    //пишем значение по индексу
    delay(200);                         // пауза 200 мС
  }
}
```

**8.арифметика**

Арифметические операции включают сложение, вычитание, умножение и деление. Они возвращают сумму, разность, произведение или частное (соответственно) двух операндов.

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

Операция управляется используемым типом данных операндов, так что, например, 9/4 даёт 2 вместо 2.25, поскольку 9 и 4 имеют тип int и не могут использовать десятичную точку. Это также означает, что операция может вызвать переполнение, если результат больше, чем может храниться в данном типе.

Если используются операнды разного типа, то для расчётов используется больший тип. Например, если одно из чисел (операндов) типа float, а второе целое, то для вычислений используется тип с плавающей точкой.

Выбирайте типы переменных достаточные для хранения результатов ваших вычислений. Прикиньте, в какой точке ваша переменная переполнится, а также, что случится в другом направлении, то есть, (0-1) или (0- -32768). Для вычислений, требующих дробей, используйте переменные типа float, но остерегайтесь их недостатков: большой размер и маленькая скорость вычислений.

**Примечание:** Используйте оператор приведения типа (название типа) для округления, то есть, (int)myFloat - для преобразования переменной одного типа в другой «на лету». Например, i = (int) 3.6 - поместит в i значение 3.

***9.смешанное присваивание***

Смешанное присваивание сочетает арифметические операции с операциями присваивания. Чаще всего встречается в цикле for, который описан ниже.

Наиболее общее смешанное присваивание включает:

```
x ++       // то же, что x = x + 1, или увеличение   x на +1
x --       // то же, что x = x - 1, или уменьшение   x на -1
x += y     // то же, что x = x + y, или увеличение   x на +y
x -= y     // то же, что x = x - y, или уменьшение   x на -y
x *= y     // то же, что x = x * y, или умножение    x на y
x /= y     // то же, что x = x / y, или деление   x на y
```

**Примечание:** Например, x *= 3 утроит старое значение x и присвоит полученный результат x.

**10 константы**

Язык Arduino имеет несколько предопределённых величин, называемых константами. Они используются, чтобы сделать программу удобной для чтения. Константы собраны в группы.

***10.1 true/false***

Это Булевы константы, определяющие логические уровни. FALSE легко определяется как 0 (ноль), а TRUE, как 1, но может быть и чем-то другим, отличным от нуля. Так что в Булевом смысле -1, 2 и 200 — это всё тоже определяется как TRUE.

```
if (b == TRUE);
{
  doSomething;
}
```

***10.2 high/low***

Эти константы определяют уровень выводов как HIGH или LOW и используются причтении или записи на логические выводы. HIGH определяется как

логический уровень 1, ON или 5 вольт(3-5), тогда как LOW — 0, OFF или 0 вольт(0-2).

```
digitalWrite(13, HIGH);
```

***input/output***

Константы используются с функцией pinMode() для задания режима работы цифровых выводов: либо как INPUT (вход), либо как OUTPUT (выход).

```
pinMode(13, OUTPUT);
```

**11 управление программой**

***11.1if***

Конструкция if проверяет, будет ли выполнено некое условие, такое, как, например, будет ли аналоговое значение больше заданного числа, и выполняет какое-то выражение в скобках, если это условие true (истинно). Если нет, то выражение в скобках будет пропущено. Формат для if следующий:

```
if (someVariable ?? value)
{
  doSomething;
}
```

Пример выше сравнивает someVariable со значением (value), которое может быть и переменной, и константой. Если выражение или условие в скобках истинно, выполняется выражение в фигурных скобках. Если нет, выражение в фигурных скобках пропускается, и программа выполняется с оператора, следующего за скобками.

**Примечание:** Нужно остерегайтесь случайного использования «=», как в if (x = 10), что технически правильно, определяя x равным 10, но результат этого всегда true. Вместо этого используйте «==», как в if (x == 10), что осуществляет проверку значения x — равно ли оно 10 или нет. Нужно запомнить «=» - равно, а «==» - равно ли?

**11.2 if...else**

Конструкция if...else позволяет сделать выбор «либо, либо». Например, если вы хотите проверить цифровой вход и выполнить что-то, если он HIGH, или выполнить что-то другое, если он был LOW, вы должны записать следующее:

```
if (inputPin == HIGH)
{
  doThingA;
}
else
{
  doThingB;
}
```

else может также предшествовать другой проверке if так, что эти множественные, взаимоисключающие проверки могут запускаться одновременно. И возможно даже неограниченное количество подобных else переходов. Хотя следует помнить, что только один набор выражений будет выполнен в зависимости от результата проверки:

```
if (inputPin < 500)
{
  doThingA;
}
else if (inputPin >= 1000)
{
  doThingB;
}
else
{
  doThingC;
}
```

Примечание: Конструкция if просто проверяет, будет ли выражение в круглых скобках истинно или ложно. Это выражение может быть любым правильным, относительно языка Си, выражением, как в первом примере if (inputPin == HIGH).

В этом примере if проверяет только то, что означенный вход в состоянии высокого логического уровня или действительно ли напряжение на нём 5 вольт.

***while***

Цикл while продолжается, и может продолжаться бесконечно, пока выражение в скобках не станет false (ложно). Что-то должно менять проверяемую переменную, иначе из цикла никогда не выйти. И это должно быть в вашем коде, как, скажем, увеличение переменной, или внешнее условие, как, например, проверяемый сенсор.

```
while (someVariable ?? value)
{
  doSomething;
}
```

Следующий пример проверяет, будет ли someVariable меньше 200, и если да, то выполняются выражения в фигурных скобках, и цикл продолжается, пока someVariable остаётся меньше 200.

```
While (someVariable < 200) // проверяет, меньше ли 200
{
  doSomething;             //  выполняет выражение в скобках
  someVariable++;          //  увеличивает переменную на 1
}
```

***do...while***

Цикл do управляемый «снизу» цикл, работающий на манер цикла while, с тем отличием, что условие проверки расположено в конце цикла, таким образом, цикл выполнится хотя бы один раз.

```
do
{
  doSomething;
} while (someVariable ?? value);
```

Следующий пример присваивает readSensor переменной x, делает паузу на 50 миллисекунд, затем цикл выполняется, пока x меньше, чем 100.

```
do
{
  x = readSensors();      // присваиваем значение
                          // readSensors() переменной x
  delay(50);              // пауза 50 миллисекунд
} while (x < 100);        //продолжение цикла, если x меньше 100
```

**цифровой ввод/вывод**

***pinMode (pin, mode)***

Используется в void setup () для конфигурации заданного вывода, чтобы он работал на вход (INPUT) или на выход (OUTPUT).

```
pinMode(pin, OUTPUT);      // устанавливает 'pin' на выход
```

Цифровые выводы в Arduino предустановлены на вход, так что их нет нужды явно объявлять как INPUT с помощью pinMode (). Выводы, сконфигурированные как INPUT, подразумеваются в состоянии с высоким импедансом (сопротивлением).

В микроконтроллере Atmega, есть также удобные, программно доступные подтягивающие резисторы 20 кОм. Эти встроенные подтягивающие резисторы

доступны следующим образом:

```
pinMode(pin, INPUT);          // настраиваем 'pin' на вход
digitalWrite(pin, HIGH);  //включаем подтягивающие резисторы
```

Подтягивающие резисторы, как правило, используются при соединении входов с переключателями. Заметьте, что в примере выше нет преобразования pin на выход, это просто метод активизации встроенных подтягивающих резисторов.

Выводы, сконфигурированные как OUTPUT, находятся в низкоимпедансном состоянии и могут отдавать 40 мА в нагрузку (цепь, другое устройство). Это достаточный ток для яркого включения светодиода (не забудьте последовательный токоограничительный резистор!), но не достаточный для включения реле, соленоидов или моторов.

Короткое замыкание выводов Arduino или слишком большой ток могут повредить выходы или даже всю микросхему Atmega. Порой, не плохая идея — соединять OUTPUT вывод через последовательно включённый резистор в 470 Ом или 1 кОм.

**Особое внимание следует уделять технике безопасности.**

При работе с платформой Arduino необходимо научиться собирать безопасные электронные схемы. Безопасные не столько для разработчиков и пользователей, но и для самой платформы и тех деталей электронного конструктора, из которых будем собирать модели. Электронные устройства в неумелых руках могут перегореть.

## 4. Практическая часть

Когда у нас возникло желание научиться программировать Arduino, мы столкнулся с несколькими проблемами:

- Выбор модели из списка доступных
- Попытки понять, что понадобится кроме самой платформы
- Установка и настройка среды разработки
- Поиск и разбор тестовых примеров
- «Разборки» с экраном
- «Разборки» с процессором

Для решения этих проблем нами были просмотрены и прочитаны довольно много разных источников.

На первом этапе мы выполнили все проекты, представленные на сайте http://wiki.amperka.ru/конспект-arduino при этом адаптировали предложенные схемы под свою платформу. Мы проанализировали работу светодиодов, кнопок, регистров. Поработали со звуком и датчиком температуры, выполни работу со световым табло. В результате выполненных работ мы не только получили устойчивые результаты, но и смогли заинтересовать своей работой одноклассников. В приложении представлены скриншоты программ.

На втором этапе мы решил создать модель робота-машинки.

Для начала использовав, детский металлический конструктор, мы собрали модель машинки. Подключили два моторчика к заднему мосту. Собрали схему. Как только схема была собрана, приступили к программированию самого микроконтроллера Ардуино. Выполнили загрузку скетча, выполнили отладку программы. Процесс и результат выполнения проекта мы записали в видеофайл.

Затем мы модифицировали модель, заменив рулевой моторчик на более мощный так, как повороты осуществлялись только в движении, и для наилучшего прохождения препятствий сразу же лучше поставить дифференциал на оба колеса и на каждый моторчик редуктор.

Для поворотов мы использовал шаговый моторчик на 12 V. Его преимущества: мощный, его тяжело остановить, а так же повернуть, если подано на него напряжение. Из недостатков: нужен драйвер двигатель, малая скорость поворота и сложный процесс программирования. Но для первого запуска машинки нам хватило. Затем мы поставить сервопривод из преимуществ: высокая скорость поворота, программирование происходит по градусам поворота, а не по шагам, как это было с шаговым двигателем. Недостатки по сравнению с шаговым моторчиком: слабее при некотором усилии колёса могут повернуть сами, но это не помешало нашей модели. Таким образом, у нас получилась модель машинки, которая ездит по запрограммированной траектории.

Планируем добавление разных датчиков, например, дальномер, но из-за нехватки свободного времени и нужных приспособлений еще не удалось сделать.

## 5.Заключение

Данная тема актуальна. Работа над проектом позволила нам расширить знания не только в области информатики, но и физики, что, несомненно, пригодится в будущей профессиональной деятельности.

Мы сделали вывод, что для овладения наукой робототехника, нужно быть специалистом во многих областях: программирование, механика, физика, электроника и т.д. Требование времени и общества к информационной компетентности учащихся постоянно возрастают. Ученики должен быть мобильным, современным, готовым к разработке и внедрению инноваций в жизнь.

В ходе работы над проектом мы изучил множество моделей, а так же построил несколько своих опытных образцов, основанные на микроконтроллере Arduino. При создании моделей мы:

1. Научился программировать на Си-подобном языке.
2. Создали действующую модель машинки.
3. Доказали, что микроконтроллер Arduino многофункционален и может использоваться в робототехнике любым энтузиастом.
4. Заинтересовали программированием своих одноклассников.

Как *результаты* исследования считаем следующее: приобретенный опыт в программировании и конструировании, возможность применения полученных знаний на уроках информатики, технологии и физики, знакомство с аналогичными работами в литературе и Интернете.

Выполняя эту работу, мы получил базовые знания о платформе Arduino, научился основам электротехники, программированию на языке Arduino и создал несколько схем на основе полученных знаний. Считаем, что поставленные нами цели и задачи были достигнуты. Поставленная гипотеза доказана и подтверждена построенными моделями. Подобные модели не являются вершиной робототехники, но они показывают, на сколько обширен спектр применения микроконтроллера Arduino.

Данную тему мы собираемся развивать далее в рамках проекта «Умный дом» и полноценно заняться робототехникой и программированием.

## 6.Список литературы

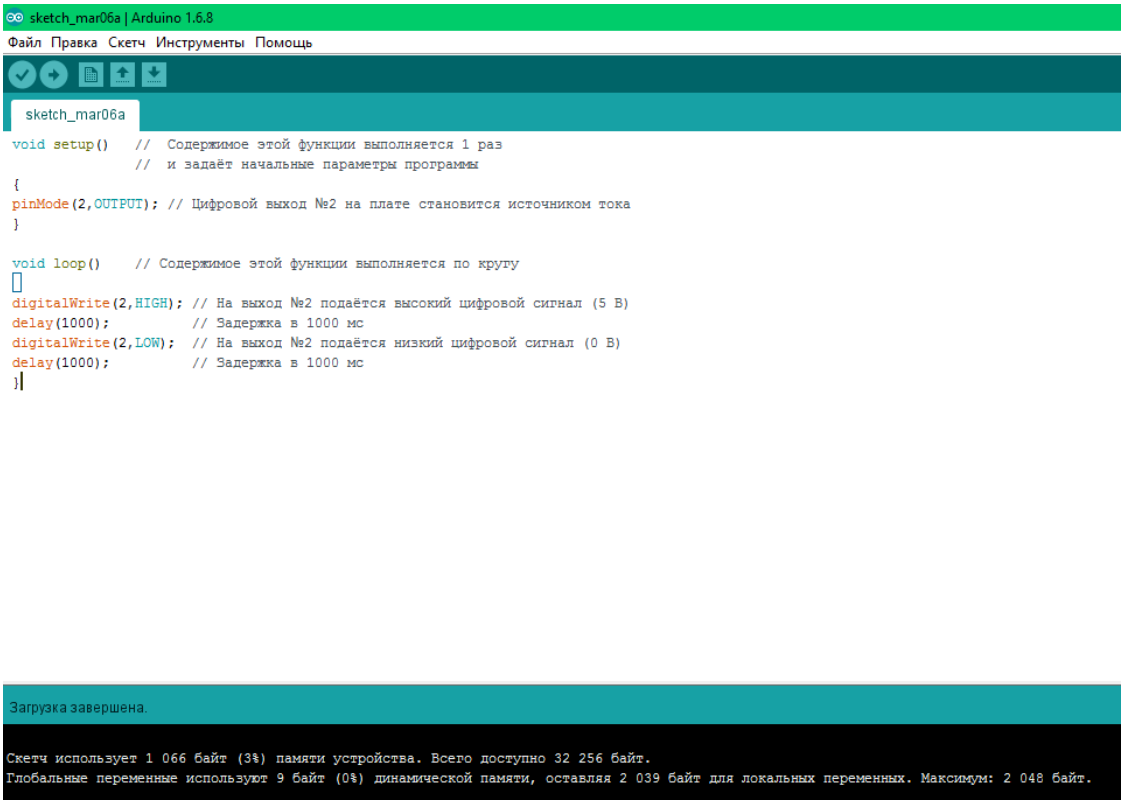

1. MaxKit. Быстрый старт. Набор конструктор начинающего изобретателя. Первые шаги по освоению Arduino.- 2015 г.
2. Бейктал Дж . Конструируем роботов на Arduino. -Первые шаги. Издательство: Лаборатория знаний. -2015
3. Блум Джереми. Изучаем Arduino. Инструменты и методы технического волшебства. -2015
4. Гололобов В.Н. О проекте Arduino для школьников (и не только). -2011
5. ИгоТ. Arduino, датчики и сети для связи устройств: Пер. с англ.-2-е изд.-СПб.: БХВ-Петербург, 2015. - 544 с.
6. Улли Соммер. Программирование микроконтроллерных плат ArduinoFreeduino.-2012
7. Учебник для образовательного набора «Амперка». Основы программирования микроконтроллеров. Москва. -2013
8. Интернет - Робототехника и мехатроника http://www.keldysh.ru/pages/anniver/achievment/21_robot.htm
9. Интернет - Анализ ситуации с технической точки зрения http://btvt.narod.ru/4/kg.htm
10. Мокров Е.А. Интегральные датчики. Состояние разработок и производства. Направления развития, объемы рынка // Датчики и системы.-2000.-№1.-С. 28…30.
11. Интернет – Заточка режущего инструмента http://library.novosel.ru/show.php?c=17&id=1647

Специализированные сайты:

http://arduino.ru/
http://arduino-projects.ru/
http://arduino-e.ru/
http://cxem.net/arduino/arduino.php
http://arduino.ua/ru/about/
http://radiohata.ru/arduino/
http://www.pvsm.ru/cat/

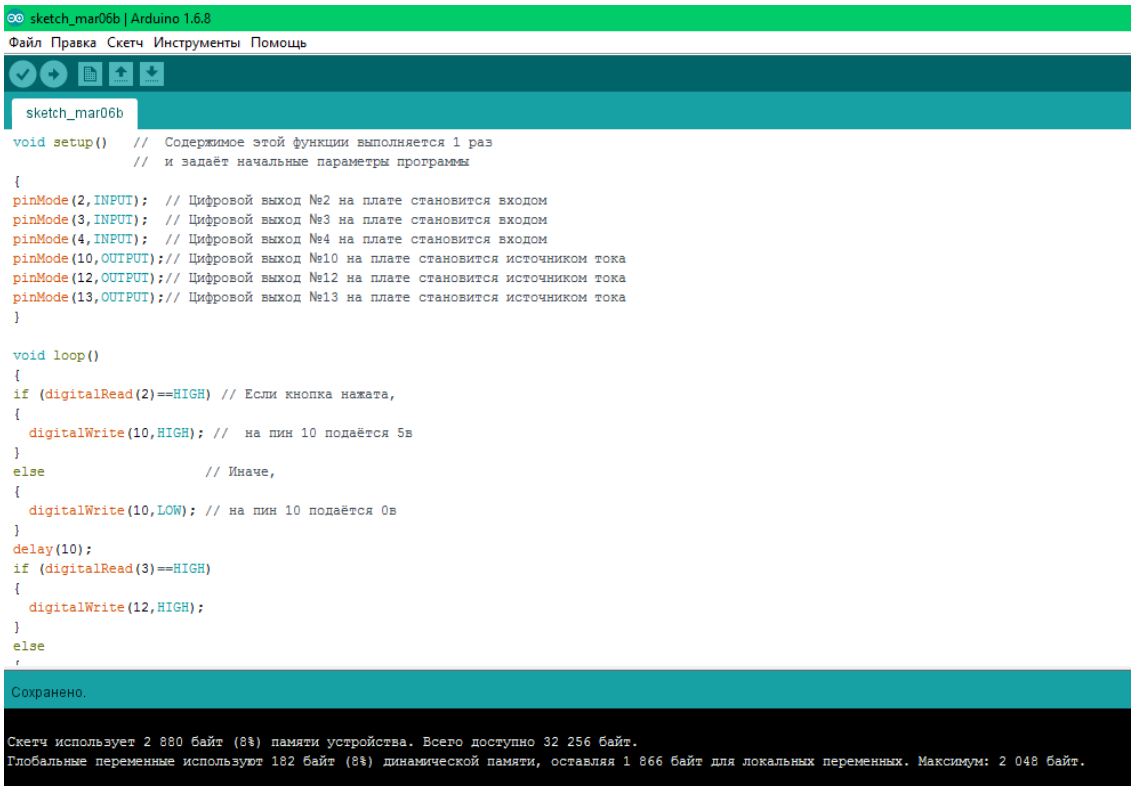
# 7.Приложение

## Скриншот программы «Один светодиод»



```
sketch_mar06a | Arduino 1.6.8
Файл Правка Скетч Инструменты Помощь
sketch_mar06a

void setup()    //  Содержимое этой функции выполняется 1 раз
                //  и задаёт начальные параметры программы
{
pinMode(2,OUTPUT); // Цифровой выход №2 на плате становится источником тока
}

void loop()     // Содержимое этой функции выполняется по кругу

digitalWrite(2,HIGH); // На выход №2 подаётся высокий цифровой сигнал (5 В)
delay(1000);          // Задержка в 1000 мс
digitalWrite(2,LOW);  // На выход №2 подаётся низкий цифровой сигнал (0 В)
delay(1000);          // Задержка в 1000 мс
}

Загрузка завершена.

Скетч использует 1 066 байт (3%) памяти устройства. Всего доступно 32 256 байт.
Глобальные переменные используют 9 байт (0%) динамической памяти, оставляя 2 039 байт для локальных переменных. Максимум: 2 048 байт.
```

## Скриншот программы «Кнопочный светофор»



```
sketch_mar06b | Arduino 1.6.8
Файл Правка Скетч Инструменты Помощь
sketch_mar06b

void setup()    //  Содержимое этой функции выполняется 1 раз
                //  и задаёт начальные параметры программы
{
pinMode(2,INPUT);  // Цифровой выход №2 на плате становится входом
pinMode(3,INPUT);  // Цифровой выход №3 на плате становится входом
pinMode(4,INPUT);  // Цифровой выход №4 на плате становится входом
pinMode(10,OUTPUT);// Цифровой выход №10 на плате становится источником тока
pinMode(12,OUTPUT);// Цифровой выход №12 на плате становится источником тока
pinMode(13,OUTPUT);// Цифровой выход №13 на плате становится источником тока
}

void loop()
{
if (digitalRead(2)==HIGH) // Если кнопка нажата,
{
  digitalWrite(10,HIGH); //  на пин 10 подаётся 5в
}
else                    // Иначе,
{
  digitalWrite(10,LOW); // на пин 10 подаётся 0в
}
delay(10);
if (digitalRead(3)==HIGH)
{
  digitalWrite(12,HIGH);
}
else

Сохранено.

Скетч использует 2 880 байт (8%) памяти устройства. Всего доступно 32 256 байт.
Глобальные переменные используют 182 байт (8%) динамической памяти, оставляя 1 866 байт для локальных переменных. Максимум: 2 048 байт.
```

# Скриншот программы «Машинка»

sketch_mar06c | Arduino 1.6.8

Файл Правка Скетч Инструменты Помощь

sketch_mar06c

```
#include <Stepper.h>

// указываем количество шагов нашего мотора - в нашем случае 64
#define STEPS 64

// Создаем экземпляр класса Stepper, и указываем
// количество шагов и пины подключния
Stepper stepper1(STEPS, 8, 9, 10, 11);
Stepper stepper2(STEPS, 7, 4, 6, 5);

void setup() {
  // Устанавливаем скорость вращения в об/мин
  stepper1.setSpeed(150);
  stepper2.setSpeed(150);
}

void loop() {
  // Прокрутить на 10 шагов
  stepper1.step(10);
  stepper2.step(10);

}
```

```
Загрузка завершена.

Скетч использует 2 208 байт (6%) памяти устройства. Всего доступно 32 256 байт.
Глобальные переменные используют 61 байт (2%) динамической памяти, оставляя 1 987 байт для локальных переменных. Максимум: 2 048 байт.
```

**Внешний вид платформы Arduino**