Министерство образования и науки Российской Федерации

Калужский филиал федерального государственного бюджетного образовательного учреждения высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»

(КФ МГТУ им. Н.Э. Баумана)

РАЗРАБОТКА ПРОГРАММ ДЛЯ МИКРОКОНТРОЛЛЕРОВ НА C++, ПРОГРАММИРОВАНИЕ В UECIDE

Методические указания по выполнению практических занятий по курсам «Основы САПР», «Системотехника электронных средств, комплексы и сети»

Калуга, 2018

структура

Базовая структура программы для Arduino довольно проста и состоит, по меньшей мере, 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.

setup()

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
}
```

loop()

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.

функции

Функция — это блок кода, имеющего имя, которое указывает на исполняемый код, который выполняется при вызове функции. Функции void setup() и void loop() уже обсуждались, а другие встроенные функции будут рассмотрены позже.

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
}
```

{} 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.

; 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 стоку 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.

/* ... */ 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.

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.

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.

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
```

byte

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
```

int

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.

long

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
```

float

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

массивы

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

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
  }
}
```

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 (нзвание 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.

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.

операторы сравнения

Сравнения одной переменной или константы с другой используются в выражении для if, чтобы проверить истинность заданного условия. В примерах на следующих страницах ?? используется для обозначения любого из следующих условий:

```
x == y      // x равно y
x != y      // x не равно y
x <  y      // x      меньше, чем   y
x >  y      // x      больше, чем   y
x <= y      // x      меньше, чем или равно  y
x >= y      // x      больше, чем или равно  y
```

логические операторы

Логические операторы, чаще всего, это способ сравнить два выражения и вернуть ИСТИНА или ЛОЖЬ, в зависимости от оператора. Есть три логических оператора: AND, OR и NOT, часто используемые в конструкциях if:

```
Logical AND:
if (x > 0 && x < 5)      // true, только если оба
                         // выражения true

Logical OR:
if (x > 0 || y > 0)      // true, если любое из
                         // выражений true

Logical NOT:
if (!x > 0)              // true, если только
                         // выражение false
```

константы

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

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;
}
```

high/low

Эти константы определяют уровень выводов как HIGH или LOW и используются при чтении 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);
```

управление программой

if

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: 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. Запомните «=» - равно, а «==» - равно ли?

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.

for

Конструкция for используется для повторения блока выражений, заключённых в фигурные скобки заданное число раз. Наращиваемый счётчик часто используется для увеличения и прекращения цикла. Есть три части, разделённые точкой с запятой, в заголовке цикла for:

```
for ( инициализация; условие; выражение )
{
  doSomething;
}
```

«Инициализация» локальной переменной, или счётчика, имеет место в самом начале и происходит только один раз. При каждом проходе цикла проверяется «условие». Если условие остаётся истинным, то следующее выражение и блок выполняются, а условие проверяется вновь. Когда условие становится ложным, цикл завершается.

Следующий пример начинается с целого i равного 0, проверяет, остаётся ли i ещё меньше 20, и, если так, увеличивает i на 1 и выполняет блок в фигурных скобках:

```
for (int i=0; i<20; i++)  // декларирует i, проверяет меньше ли
{                         // чем 20, увеличивает i на 1
  digitalWrite(13, HIGH); // устанавливает вывод 13 в ON
  delay(250);             // пауза в 1/4 секунды
  digitalWrite(13, LOW);  // сбрасывает вывод 13 в OFF
  delay(250);             // пауза в 1/4 секунды
}
```

Примечание: В Си цикл for более гибок, чем это можно обнаружить в других языках программирования, включая Basic. Любые или все три элемента заголовка могут быть опущены, хотя точка с запятой требуется. Также выражения для инициализации, условия и выражения могут быть любыми правильными выражениями Си с несвязанными переменными. Такие необычные типы выражений могут помочь в решении некоторых редких программных проблем.

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.

digitalRead (pin)

Считывает значение заданного цифрового вывода (pin) и возвращает результат HIGH или LOW. Вывод должен быть задан либо как переменная, либо как константа (0-13).

```
value = digitalRead(Pin);    // задаёт 'value' равным
                             // входному выводу 'Pin'
```

digitalWrite (pin, value)

Выводит либо логический уровень HIGH, либо LOW (включает или выключает) на заданном цифровом выводе pin. Вывод может быть задан либо как переменная, либо как константа (0-13).

```
digitalWrite(pin, HIGH);   // sets 'pin' to high
```

Следующий пример читает состояние кнопки, соединённой с цифровым входом, и включает LED (светодиод), подключённый к цифровому выходу, когда кнопка нажата:

```
int led   = 13;    // соединяем LED с выводом 13
int pin   =  7;    // соединяем кнопку с выводом 7
int value =  0;// переменная для хранения прочитанного значения

void setup()
{
  pinMode(led, OUTPUT);     // задаём вывод 13 как выход
  pinMode(pin, INPUT);      // задаём вывод 7 как вход
}

void loop()
{
  value = digitalRead(pin);  // задаём 'value' равной
                             // входному выводу
  digitalWrite(led, value);  // устанавливаем 'led' в
}                            // в значение кнопки
```

analogRead (pin)

Считывает значение из заданного аналогового входа (pin) с 10-битовым разрешением. Эта функция работает только на аналоговых портах (0-5). Результирующее целое значение находится в диапазоне от 0 до 1023.

```
value = analogRead(pin);  // задаёт 'value' равным 'pin'
```

Примечание: Аналоговые выводы не похожи на цифровые, и нет необходимости предварительно объявлять их как INPUT или OUTPUT (если только вы не планируете использовать их в качестве цифровых портов 14-18).

analogWtite (pin, value)

Записывает псевдо-аналоговое значение, используя схему с широтно-импульсной модуляцией (PWM), на выходной вывод, помеченный как PWM. На новом модуле Arduino с ATmega168 (328), эта функция работает на выводах 3, 5, 6, 9, 10 и 11. Старый модуль Arduino с ATmega8 поддерживает только выводы 9, 10 и 11. Значение может быть задано как переменная или константа в диапазоне 0-255.

```
analogWrite(pin, value);  // записывает 'value' в аналоговый 'pin'
```

Значение 0 генерирует устойчивое напряжение 0 вольт на выходе заданного вывода; значение 255 генерирует 5 вольт на выходе заданного вывода. Для значений между 0 и 255 вывод быстро переходит от 0 к 5 вольтам — чем больше значение, тем чаще вывод в состоянии HIGH (5 вольт). Например, при значении 64 вывод будет в 0 три четверти времени, а в состоянии 5 вольт одну четверть; при значении 128 половину времени будет вывод будет в 0, а половину в 5 вольт; при значении 192 четверть времени вывод будет в 0 и три четверти в 5 вольт.

Поскольку эта функция схемная (встроенного модуля), вывод будет генерировать устойчивый сигнал после вызова analogWrite в фоновом режиме, пока не будет следующего вызова analogWrite (или вызова digitalRead или digitalWrite на тот же вывод).

Примечание: Аналоговые выводы, не такие как цифровые, и не требуют предварительной декларации их как INPUT или OUTPUT.

Следующий пример читает аналоговое значение с входного аналогового вывода, конвертирует значение делением на 4 и выводит PWM сигнал на PWM вывод:

```
int led = 10;     // LED с резистором на выводе 10
int pin =  0;     // потенциометр на аналоговом выводе 0
int value;        // значение для чтения

void setup(){}    // setup не нужен

void loop()
{
  value =  analogRead(pin);  // задаёт 'value' равной 'pin'
  value /= 4;                // конвертирует 0-1023 в 0-255
  analogWrite(led, value);   // выводит PWM сигнал на LED
}
```

время и математика

delay (ms)

Приостанавливает вашу программу на заданное время (в миллисекундах), где 1000 равно 1 секунде.

```
delay(1000);    // ждём одну секунду
```

millis()

Возвращает число миллисекунд, как unsigned long, с момента старта программы в модуле Arduino.

```
value = millis();  // задаёт 'value' равной millis ()
```

Примечание: Это число будет переполняться (сбрасываться в ноль), после, примерно, 9 часов.

min (x, y)

Вычисляется минимум двух чисел любого типа данных и возвращает меньшее число.

```
value = min(value, 100); // устанавливает 'value' в наименьшее из
                         // value и 100, обеспечивая, что
                         // оно никогда не превысит 100
```

max (x, y)

Вычисляется максимум двух чисел любого типа данных и возвращает большее число.

```
value = max(value, 100); // устанавливает 'value' в наибольшее из
                         // value и 100, обеспечивая, что
                         // оно всегда не меньше 100
```

случайные числа

randomSeed (seed)

Устанавливает значение, или начальное число, в качестве начальной точки функции random().

```
randomSeed(value);     // задаёт 'value' как начальное значение random
```

Поскольку Arduino не может создавать действительно случайных чисел, randomSeed позволяет вам поместить переменную, константу или другую функцию в функцию random, что помогает генерировать более случайные «random» числа. Есть множество разных начальных чисел, или функций, которые могут быть использованы в этой функции, включая millis(), или даже analogRead() для чтения электрических шумов через аналоговый вывод.

random (max)
random (min, max)

Функция random позволяет вам вернуть псевдослучайное число в диапазоне, заданном значениями min и max.

```
value = random(100, 200);   // задаёт 'value' случайным
                            // числом между 100 и 200
```

Примечание: Используйте это после использования функции randomSeed().

Следующий пример создаёт случайное число между 0 и 255 и выводит PWM сигнал на PWM вывод, равный случайному значению:

```
int randNumber;  // переменная для храниения случайного значения
int led = 10;    // LED с резистором на выводе 10

void setup() {}  // setup не нужен

void loop()
{
  randomSeed(millis());          // задаёт millis() начальным числом
  randNumber = random(255);      // случайное число из 0-255
  analogWrite(led, randNumber); // вывод PWM сигнала
  delay(500);                    // пауза в полсекунды
}
```

последовательный обмен

Serial.begin (rate)

Открывает последовательный порт и задаёт скорость для последовательной передачи данных. Типичная скорость обмена для компьютерной коммуникации — 9600, хотя поддерживаются и другие скорости.

```
void setup()
{
  Serial.begin(9600);    // открывается последовательный порт
}                        // задаётся скорость обмена 9600
```

Примечание: При использовании последовательного обмена, выводы 0 (RX) и 1 (TX) не могут использоваться одновременно как цифровые.

Serial.println (data)

Передаёт данные в последовательный порт, сопровождая автоматическим возвратом каретки и переходом на новую строку. Команда такая же, что и Serial.print(), но легче для последующего чтения на данных в терминале.

```
Serial.println(analogValue);  // отправляет значение
                              // 'analogValue'
```

Примечание: За дальнейшей информацией о различных изменениях Serial.println() и Serial.print() обратитесь на сайт Arduino.

Следующий простой пример читает аналоговый вывод 0 и отсылает эти данные на компьютер каждую секунду.

```
void setup()
{
  Serial.begin(9600);              // задаём скорость 9600 bps
}

void loop()
{
  Serial.println(analogRead(0)); // шлём аналоговое значение
  delay(1000);                   // пауза 1 секунда
}
```

приложение

цифровой выход

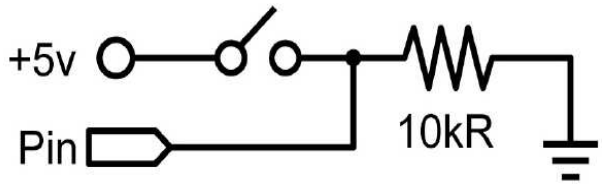
Это базовая программа «hello world», используемая для включения и выключения чего-нибудь. В этом примере светодиод подключён к выводу 13 и мигает каждую секунду. Резистор в данном случае может быть опущен, поскольку на 13-м порту Arduino уже есть встроенный резистор.

```
int ledPin = 13;                // LED на цифровом выводе 13

void setup()                    // запускается один раз
{
  pinMode(ledPin, OUTPUT);      // устанавливае вывод 13 на выход
}

void loop()                     // запускаем вновь и вновь
{
  digitalWrite(ledPin, HIGH);   // включаем LED
  delay(1000);                  // пауза 1 секунда
  digitalWrite(ledPin, LOW);    // выключаем LED
  delay(1000);                  // пауза 1 секунда
}
```

цифровой ввод



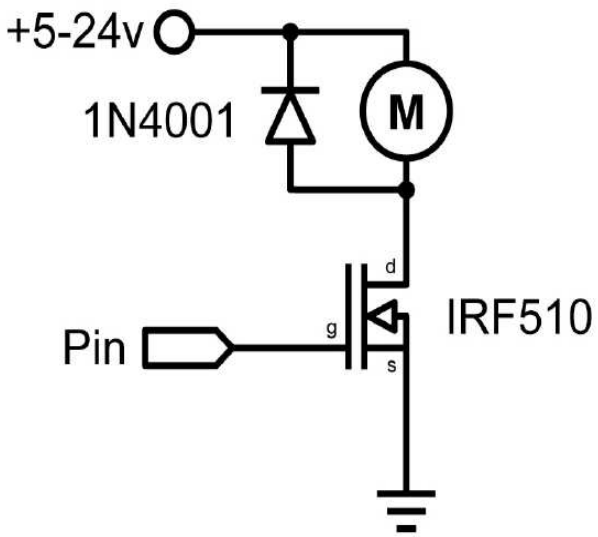
Это простейшая форма ввода с двумя возможными состояниями: включено или выключено. В примере считывается простой переключатель или кнопка, подключённая к выводу 2. Когда выключатель замкнут, входной вывод читается как HIGH и включает светодиод.

```
int ledPin = 13;                // выходной вывод для LED
int inPin = 2;                  // входной вывод (для switch)

void setup()
{
  pinMode(ledPin, OUTPUT);  // объявляем LED как выход
  pinMode(inPin, INPUT);    // объявляем switch как вход
}

void loop()
{
  if (digitalRead(inPin) == HIGH) // проверяем вход, HIGH?
  {
    digitalWrite(ledPin, HIGH);   // включаем LED
    delay(1000);                  // пауза 1 секунда
    digitalWrite(ledPin, LOW);    // выключаем LED
    delay(1000);                  // пауза 1 секунда
  }
}
```

сильноточный выход



Иногда возникает необходимость в управлении более, чем 40 мА от Arduino. В этом случае может использоваться транзистор MOSFET для коммутации сильноточной нагрузки. В следующем примере MOSFET быстро включается и выключается 5 раз в секунду.

Примечание: Схема показывает мотор и диод защиты, но другие, не индуктивные, нагрузки могут включаться без диода.

```
int outPin  =  5;         // выходной вывод для MOSFET

void setup()
{
  pinMode(outPin, OUTPUT);       // задаём вывод 5 как выход
}

void loop()
{
  for (int i=0; i<=5; i++)       // цикл 5 раз
  {
    digitalWrite(outPin, HIGH); // включаем MOSFET
    delay(250);                 // пауза в 1/4 секунды
    digitalWrite(outPin, LOW);  // выключаем MOSFET
    delay(250);                 // пауза в 1/4 секунды
  }
  delay(1000);                  // пауза 1 секунда
}
```

pwm выход

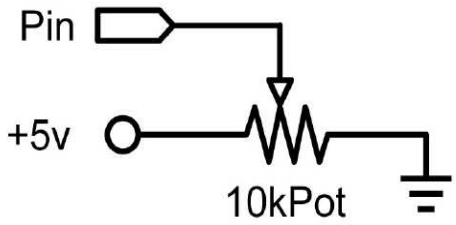
Широтно-импульсная модуляция (PWM) — это способ имитировать аналоговый выход с помощью импульсного сигнала. Это можно использовать для гашения и увеличения яркости светодиода или позже для управления сервомотором. Следующий пример медленно увеличивает яркость и гасит LED, используя цикл for.

```
int ledPin  =  9;      // PWM вывод для LED

void setup(){}         //  setup не нужен

void loop()
{
  for (int i=0; i<=255; i++)  // растущее значение для i
  {
    analogWrite(ledPin, i);   // устанавливаем уровень яркости для i
    delay(100);               //  пауза 100 мС
  }
  for (int i=255; i>=0; i--)  // спадающее значение для i
  {
    analogWrite(ledPin, i);   // устанавливаем уровень яркости для i
    delay(100);               //  пауза 100 мС
  }
}
```

вход с потенциометра



Использование потенциометра и одного из аналоговых портов Arduino (аналого-цифрового преобразователя (ADC)) позволяет читать аналоговые значения в диапазоне 0-1023. Следующий пример показывает использование потенциометра для управления временем мигания светодиода LED.

```
int potPin = 0;     // входной вывод для потенциометра
int ledPin = 13;    // выходной вывод для LED

void setup()
{
  pinMode(ledPin, OUTPUT);  // объявляем ledPin как OUTPUT
}

void loop()
{
  digitalWrite(ledPin, HIGH);     // включаем ledPin
  delay(analogRead(potPin));      // пауза
  digitalWrite(ledPin, LOW);      // выключаем ledPin
  delay(analogRead(potPin));      // пауза
}
```

вход от переменного резистора

Переменные резисторы включают фотоприёмники, термисторы, тензодатчики и т.д. Данный пример использует функцию чтения аналогового значения и задаёт время паузы. Этим управляется скорость, с которой меняется яркость светодиода LED.

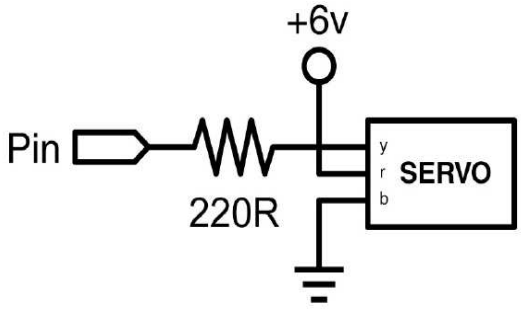
```
int ledPin    =  9;    // PWM вывод для LED
int analogPin =  0;    // переменный резистор на аналоговом выводе 0

void setup(){}        //   setup не нужен

void loop()
{
  for (int i=0; i<=255; i++)  // увеличивающееся значение для i
  {
    analogWrite(ledPin, i);   // устанавливаем уровень яркости по i
    delay(delayVal());        // берём значение времени и паузы
  }
  for (int i=255; i>=0; i--)  // уменьшающееся значение для i
  {
    analogWrite(ledPin, i);   // устанавливаем уровень яркости по i
    delay(delayVal());        // берём значение времени и паузы
  }
}

int delayVal()
{
  int v;                      // создаём временную переменную
  v  = analogRead(analogPin); // читаем аналоговое значение
  v /= 8;                     // конвертируем 0-1024 в 0-128
  return v;                   // возвращаем окончательное значение
}
```

серво вывод



Любительские сервомашинки — это разновидность полу-автономного мотор-редуктора, который может поворачиваться на 180^0. Всё, что нужно — это отправлять импульсы каждые 20 мС. В данном примере используется функция servoPulse для поворота мотора от 10^0 до 170^0 и обратно.

```
int servoPin = 2;   // привод соединён с цифровым выводом 2
int myAngle;        // угол привода грубо 0-180
int pulseWidth;     // servoPulse функции переменная

void setup()
{
  pinMode(servoPin, OUTPUT);   // устанавливаем вывод 2 на выход
}

void servoPulse(int servoPin, int myAngle)
{
  pulseWidth = (myAngle * 10) + 600;  // определяем паузу
  digitalWrite(servoPin, HIGH);       // устанавливаем привод в high
  delayMicroseconds(pulseWidth);      // микросекундная пауза
  digitalWrite(servoPin, LOW);        // устанавливаем привод в low
}

void loop()
{
  // привод стартует с 10 градусов и поворачивается до 170
  for (myAngle=10; myAngle<=170; myAngle++)
  {
    servoPulse(servoPin, myAngle);    // отправляем вывод и угол
    delay(20);                        // обновляем цикл
  }
  //  привод стартует со 170 и поворачивается до 10 градусов
  for (myAngle=170; myAngle>=10; myAngle--)
  {
    servoPulse(servoPin, myAngle);    //  отправляем вывод и угол
    delay(20);                        //  обновляем цикл
  }
}
```