ардуино
книжка
за програмиране

от Браян Евънс (Brian W. Evans)

Ардуино книжка за програмиране
От Браян Евънс (Brian W. Evans)

Ползвани са информация и вдъхновение от:
http://www.arduino.cc
http://www.wiring.org.co
http://www.arduino.cc/en/Booklet/HomePage
http://cslibrary.stanford.edu/101/

Включително и материали от:
Масимо Банци (Massimo Banzi)
Ернандо Бараган (Hernando Barragan)
Дейвид Куартиелес (David Cuartieles)
Том Игое (Tom Igoe)
Даниел Джолиф (Daniel Jolliffe)
Тод Курт (Todd Kurt)
Дейвид Мелис (David Mellis)
и др.

Първо издание, Август 2007

предговор

Тази книжка представлява удобен и лесен за употреба наръчник за програмната структура и основните команди за програмиране на микроконтролера Ардуино. Част от командите не са включени в тази книжка за да остане по-достъпна. Тази книжка е най-добре да се ползва като вторичен източник наред с уебсайтове, книги, семинари и занятия. Поради тази причина в книжката има лек превес към използването на Ардуино за самостоятелни проекти, и са изключени, например, по-сложни употреби на масиви или по-сложни форми за серийна комуникация.

Тази книжка включва основите на програмната структура на програмния език на Ардуино (базиран на C) и описва синтакса на най-често използваните команди, илюстрирайки тяхната употреба с примери и фрагменти програмен код. Включени са много от основните функции, последвани от приложение с прости схеми и програми за начинаещи. Където е възможно форматът на тази книжка допълва Physical Computing на О'Съливан (O'Sullivan) и Игое (Igoe).

За въведение в Ардуино и интерактивен дизайн се насочете към книжката на Банци – *Запознанство с Ардуино (Getting Started with Arduino),* също преведена и на български език. За малцината смелчаци, които искат да се задълбочат в програмирането на C, препоръчваме второто издание на *The C Programming Language* от Керниган (Kernighan) и Ричи (Ritchie), и *C in a Nutshell* от Принц (Prinz) и Кроуфорд (Crawford).

Написването на тази книжка не би било възможно без страхотното общество, и огромното количество помощни материали достъпни на сайта на Ардуино – http://www.arduino.cc

съдържание

структура (structure)

Основната структура на езика за програмиране на Ардуино е относително проста и се състои от поне две части. Тези две задължителни части (или функции) обгръщат блокове изявления (statements).

```
void setup ()
{
	изявления;
}

void loop ()
{
	изявления;
}
```

Тук setup() е подготовката (preparation), а loop() е изпълнението (execution). И двете функции са задължителни за да работи програмата.

Функцията setup() трябва да е след декларирането на променливи, което се прави в самото начало на програмата. Тя е първата функция, която се изпълнява, протича само веднъж и служи за да зададе pinMode (режим на пиновете) или да отвори серийната комуникация.

loop() функцията се изпълнява втора и съдържа код, който се изпълнява продължителен период от време – отчита показания от пиновете, пуска сигнали до пиновете и т.н. Тази функция е ядрото на всяка Ардуино програма и изпълнява повечето задачи.

setup()

Функцията setup() се повиква веднага щом програмата тръгне. Използва се при задаване ролята на пиновете или за да започне серийна комуникация. Тя трябва да бъде включена в програмата дори ако не съдържа никакви изявления.

```
void setup()
{
	pinMode(pin, OUTPUT);		// zadava OUTPUT rejim na pina
}
```

loop()

След като функцията setup() е повикана, loop() функцията прави точно това, което името и подсказва че ще прави – цикли постоянно и позволява на програмата да се променя, да реагира и да контролира Ардуино платката.

```
void loop()
{
	digitalWrite(pin, HIGH);	// vklyuchva pina
	delay(1000);			// izchakva edna sekunda
	digitalWrite(pin, LOW);		// izklyuchva pina
	delay(1000);			// izchakva edna sekunda
}
```

функции (functions)

Функцията е откъс код с име и блок от изявления (statements), които се изпълняват когато функцията бъде повикана. Функциите void loop() и void setup() вече бяха обяснени, а други функции ще покрием по-късно в тази книжка.

Индивидуализирани (custom) функции могат да се използват за повтарящи се задачи и за да бъде кодът по-подреден и разбираем. Функциите се декларират като първо се декларира вида на функцията. Това е вида стойност, например int за цяло число (integer), който функцията връща като резултат. Ако функцията не връща никакъв резултат тя е невалидна. След вида се декларира името, което даваме на функцията и в скоби се поставят различни параметри, които се подават на функцията.

```
вид имеНаФункцията(параметри)

{
        изявления;
}
```

Следната функция от вид „integer" izchakvaneStojnost() се използва за да се зададе изчакване в програма на базата на отчетена стойност от потенциометър. Първоначално тя декларира локална променлива v, задава й стойност от потенциометъра (число от 0 до 1023), после разделя числото на 4 за да се получи крайна стойност между 0 и 255, и на края връща тази стойност на програмата.

```
int izchakvaneStojnost ()
{
        int v;                          // suzdava lokalna promenliva v
        v = analogRead(pot);            // otchita stojnostta ot potenciometara
        v /= 4;                         // konvertira 0-1023 kam 0-255
        return v;                       // vrushta krainata stojnost
}
```

{} къдрави скоби (curly braces)

Къдравите скоби обозначават началото и края на блокове от функции или изявления, като например void loop() функцията и for и if изявленията (for and if statements).

```
вид функция()
{
        изявления;
}
```

Всяка отварящата къдрава скоба { трябва да е последвана от затваряща }. Това се нарича баланс на скобите. Небалансираните скоби могат да доведат до криптически, и неразгадаеми грешки при компилирането и понякога могат да бъдат трудни за откриване в по-голяма програма.

Средата за програмиране на Ардуино предлага удобна възможност за проверка баланса на къдравите скоби. Просто маркирайте някоя къдрава скоба, или празното разстояние веднага след нея, и логическият и спътник ще бъде посочен.

; точка и запетая (semicolon)

В края на всяко изявление, както и в края на всеки отделен елемент на програмата, трябва да се поставя точка и запетая. Точка и запетая се използват и за отделяне на елементите във for циклите (for loops).

```
int x = 13;                    // deklarira promenlivata x kato integer-a 13
```

Забележка: Забравянето на точка и запетая в края на реда ще даде грешка при компилирането. Дефиницията на грешката може да е ясна, и да казва за липсващи точка и запетая, или пък да не е. Ако получите неразгадаема или привидно нелогична грешка при компилирането, едно от първите неща за които трябва да проверите е за липсваща точка и запетая близо до реда от който компилаторът се е оплакал.

/* ... */ блок коментар (block comments)

Блок коментарите, или коментарите от по няколко реда, са парчета текст които програмата не взима предвид и се използват за по-дълги текстови описания. Те помагат на другите да разберат по-лесно какви роли изпълняват частите на програмата. Блок коментарите започват с /* и завършват с */ и могат да се простират на няколко реда.

```
/*      това е блок коментар
        не забравяйте да затворите коментара
        той също трябва да е балансиран
*/
```

Тъй като коментарите се игнорират от програмата и не заемат никакво място в паметта те не трябва да се пестят. Коментарите могат да се използват и за изключването на парчета код при отстраняването на бъгове (debugging).

Забележка: Въпреки че е възможно да обозначавате коментари от по един ред като блок коментари, не е позволено коментарът да съдържа втори коментар в себе си.

// коментар на реда (line comment)

Коментари от по един ред започват с // и завършват със следващия ред код. Също като блок коментарите те се игнорират от програмата и не заемат място в паметта.

```
//  това е коментар от един ред
```

Коментарите на реда често се използват след изявления за да дадат информация за това какво постига изявлението или да послужат като подсказка при преработване на кода.

променливи (variables)

Променливата е начин да се именува и да се запази цифрова стойност която да се използва по-късно в програмата. Както предполага името им, променливите могат постоянно да се променят за разлика от константите, чиито стойности никога не се променят. Променливата трябва задължително да се декларира, а задаването на стойността която да се запази е по избор. Кодът долу декларира променлива наречена inputPromenliva и после й задава стойността получена от аналоговия входен пин 2:

```
int inputPromenliva = 0;                // deklarira promenliva i
                                        // j zadava stojnost 0
inputPromenliva = analogRead(2);        // zadava na promenlivata stojnostta ot
                                        // analogoviya pin 2
```

‘inputPromenliva’ е самата променлива. Първият ред декларира, че тя ще приема стойности от вида integer (int). Вторият ред задава на променливата стойността от аналоговия пин 2. По този начин стойността от пин 2 е достъпна и в други части на кода.

След като на променливата е зададена или повторно зададена стойност, може да се провери дали тази стойност отговаря на дадено условие или да се използва направо. Като пример ще покажем три полезни операции с променливи. Следният код проверява дали inputPromenliva е по-малка от 100, и ако условието е вярно задава на променливата стойност 100 и след това задава изчакване (delay) равно на стойността на променливата, която вече е равна поне на 100:

```
if (inputPromenliva < 100)// proveryava dali stojnostta e po-malka ot 100
{
inputPromenliva = 100;     // ako gornoto uslovie e vyarno zadava stojnost 100
}
Delay(inputPromenliva);   // izpolzva promenlivata za izchakvane (delay)
```

Забележка: На променливите трябва да се задават описателни имена за да бъде кодът по-лесно разбираем. Имена на променливи като ‘senzorNaklon’ или ‘pushButon’ помагат на програмиста и всеки който чете кода му да разбере каква е променливата. От друга страна, имена като ‘promenliva’ (‘variable’) изобщо не помагат да се разбере по-лесно кода и тук се използват само като примери. Променливата може да приеме всяко име стига то да не е команда или някоя от ключовите думи в езика за програмиране на Ардуино.

деклариране на променливите (variable declaration)

Всяка променлива трябва да бъде декларирана преди да може да се използва. Декларирането на променливата означава да се дефинира нейния вид (например integer, long, float, и др.), да й се даде име, и по избор може да и се зададе първоначална стойност. Достатъчно е променливата да се декларира веднъж в програмата, а нейната стойност може да бъде променена по всяко време чрез аритметични функции или други методи на задаване.

Този пример декларира променливата inputPromenliva като integer (int) и й задава първоначална стойност 0.

```
int inputPromenliva = 0;
```

Променливата може да бъде декларирана на различни места в програмата и в зависимост от това къде е дефинирана се определя кои части на програмата могат да я ползват.

обхват на променливата (variable scope)

променливата може да бъде декларирана в началото на програмата преди void setup(), локално във функцията и понякога в блок изявления като for циклите. Мястото където е декларирана променливата определя нейния обхват (или възможността на определени части от програмата да я използват).

Глобалната променлива е тази, която може да бъде „видяна" и използвана от всяко изявление или функция на програмата. Такава променлива се дефинира в началото на програмата, преди setup() функцията.

Локалната променлива се дефинира вътре в самата функция или като част от for цикъл. Тя може да се използва само от функцията в която е декларирана. Поради тази причина е възможно да има две или повече променливи с еднакви имена в различни части на програмата и тези променливи да имат различни стойности. За да е по-разбираема програмата и за да се избегне възможността за грешки е добре само една функция да има достъп до стойностите на променливата.

Този пример показва как да се декларират различните видове променливи и техния обхват:

```
int stojnost;                    // 'stojnost' e vidima za
                                 // vsyaka funkciya
void setup()
{
      //nyama nujda ot setup
}

void loop()
{
for (int i=0; i<20;)             // 'i' e vidima samo
      {                          // za for cikyla
      i++;
      }
      float f;                   // 'f' e vidima samo
}                                // vytre v cikula
```

байт (byte)
Байтът съхранява 8-битова цифрова стойност без десетични запетаи. Използва се за числа от 0 до 255.

```
byte nyakakvaPromenliva = 180;        //deklarira ‘nyakakvaPromenliva’
                                      //kato vid byte
```

цяло число (int)
Целите числа (integer) са основния вид данни за съхранение на числа без десетични запетаи и съхраняват 16-битова стойност за числа от 32,767 до -32,768.

```
int nyakakvaPromenliva = 1500;        // deklarira ‘nyakakvaPromenliva’
                                      // kato vid integer
```

Забележка: Променливите от вид integer ще се ‘превъртят’ ако се преминат минимално или максимално допустимите стойности. Например ако x = 32767 и ако някое изявление добави 1 към x (x = x + 1) тогава x ще се превърти и ще приеме стойност равна на -32,768.

лонг (long)
Това е разширен вид данни за по-дълги integer-и, без десетични запетаи, съхранявани като 32-битови стойности с обхват от 2,147,483,647 до -2,147,483,648.

```
long nyakakvaPromenliva = 90000;      //deklarira ‘nyakakvaPromenliva’
                                      //kato vid long
```

флоут (float)
Вид данни за числа с десетична запетая. Данни от този вид са с по-голяма резолюция от integer, запазват се като 32-битови стойности с обхват от 3.4028235E+38 до -3.4028235E+38.

```
float nyakakvaPromenliva = 3.14;      // deklarira ‘nyakakvaPromenliva’
                                      // kato vid float
```

Забележка: Данните от вид float не са точни и може да дават странни резултати когато бъдат сравнявани. Изчисленията с такива данни протичат много по-бавно отколкото изчисления с числа от вида integer и затова трябва да се избягват винаги когато е възможно.

масиви (arrays)

Масивът е поредица от стойности, достъпни чрез поредните им номера. Всяка стойност в масива може да бъде повикана чрез името на масива и поредния й номер. Масивите са ‘zero indexed’, което означава, че първата стойност в масива има пореден номер 0. За да може да се използва, масивът трябва да бъде деклариран и по-избор може да му се зададат стойности.

```
int moyatMasiv [] = {stojnost0, stojnost1, stojnost2…}
```

Също така е възможно първо да се декларира вида и броя позиции на масива и по-късно да се зададат стойности на съответните позиции:

```
int moyatMasiv [5];              // deklarira masiv ot vid integer s 6 pozicii
moyatMasiv [3] = 10;             // zadava 10 na chetvyrtata stojnost v masiva
```

За да използвате стойност от масива, задайте на някоя променлива името на масива и поредния номер на стойността:

```
x = moyatMasiv [3];                    // sega x e raven na 10
```

Масивите често се използват във for цикли, в които „брояч” се използва за да вика последователно стойностите от масива. Този пример използва масив за да контролира примигването на светодиод. С помощта на for цикъл, броячът започва от 0, задава стойността на позиция 0 в масива ‘primigvane’, в случая 180, на пин 10, изчаква 200ms, и преминава към следващата стойност от масива.

```
int svetodiodPin = 10;                 // svetodiod na pin 10
byte primigvane[] = {180, 30, 255, 200, 10, 90, 150, 60};
                                       // masiv s 8 stojnosti
void setup()
{
      pinMode(svetodiodPin, OUTPUT);           // zadava output pin
}

void loop()
{
      for(int i = 0; i < 7; i++)               // cikylyt syotvetstva na
      {                                        // broya pozicii v masiva
      analogWrite(svetodiodPin, primigvane[i]);// zadava nomer na poziciyata
      delay(200)                               // izchakva 200ms
      }
}
```

аритметика (arithmetic)

Операторите за аритметика включват събиране, изваждане, умножение и деление. Те връщат сбора, разликата, произведението, или частното на две величини.

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

Операцията се извършва като се използва вида данни на величините които участват в операцията. Така например 9 / 4 ще е равно на 2 вместо на 2.25 тъй като 9 и 4 са от вида integer и не могат да приемат десетични запетаи. Това също значи и че операцията може да „превърти" стойностите ако надвишава възможните стойности за дадения вид данни.

Ако величините са с различен вид данни за операцията се използва по-големия вид данни. Например ако една от величините е от вид float, а другата е от вид integer, за калкулацията ще се използва вида float.

Подбирайте вида данни така че да е достатъчно голям и за най-големите възможни резултати от калкулациите. Помнете при какви стойности става превъртането и какъв ще е резултатът ако числото се превърти. За изчисления които изискват да се смята с дроби използвайте float, но имайте предвид и недостатъците – по-голяма програма и по-бавни калкулации.

Забележка: Използвайте каст оператор (cast operator), примерно (int)moyatFloat за да смените вида данни за величината в движение. Например `i = (int)3.6` ще зададе на i стойност равна на 3.

compound assignments

Compound assignments комбинират операторите за аритметика с variable assignment. Те често се срещат във for цикли, както ще покажем по-късно в книжката. Най-често употребяваните compound assignments включват:

```
x++         // syshtoto kato x = x + 1, ili uvelichava x s + 1
x- -        // syshtoto kato x = x - 1, ili namalyava x s - 1
x += y      // syshtoto kato x = x + y, ili uvelichava x s + y
x -= y      // syshtoto kato x = x - y, ili namalyava x s - y
x *= y      // syshtoto kato x = x * y, ili umnojava x s y
x /= y      // syshtoto kato x = x / y, ili deli x s y
```

Забележка: Например x *= 3 ще утрои стойността на x и ще зададе получената стойност на x.

сравнителни оператори (comparison operators)

Сравнението на една променлива или константа с друга често се използва в if заявления за да провери дали дадено условие е истина. В примерите от следващите страници, ?? се използва за кое да е от следните условия:

```
x = = y        // x e raven na y
x ! = y        // x ne e raven na y
x < y          // x e po-malyk ot y
x >  y         // x e po-golyam ot y
x <= y         // x e po-malyk ili raven na y
x >= y         // x e po-golyam ili raven na y
```

логически оператори (logical operators)

Логическите оператори се използват за да сравнят два израза и като резултат да посочат ‘истина’ или ‘неистина’ (TRUE or FALSE) в зависимост от оператора. Има три логически оператора AND, OR и NOT, които често се използват в if изявления:

Логически AND:

```
if (x > 0 && x < 5)        // ‘istina’ samo ako i dvata
                           // izraza sa verni
```

Логически OR:

```
if (x > 0 | | y > 0)       // ‘istina’ ako koj da e ot
                           // dvata izraza e veren
```

Логически NOT:

```
If (! x > 0)               // ‘istina’ samo ako
                           // izrazyt ne e veren
```

константи (constants)
Езикът на Ардуино има няколко предварително заложени стойности, които се наричат константи. Те се използват за да направят разчитането на програмите по-лесно. Константите са класифицирани в няколко групи.

истина/неистина (true/false)
Тези са Boolean константи и дефинират ниво на логиката. Неистина (FALSE) лесно се дефинира като 0 (нула) докато истина (TRUE) често се дефинира като 1, но може и да е всяко друго число освен нула. Така например за Boolean логиката -1, 2, и -200 се дефинират като истина (TRUE).

```
if (b = = true);
{
        napraviNeshto;
}
```

включено/изключено (high/low)
Тези константи дефинират нивото на пиновете като HIGH или LOW и се използват когато се пише или чете от цифровите пинове. HIGH се дефинира като логическо ниво 1 (включено, или напрежение от 5 волта) докато LOW е с логическо ниво 0 (изключено, или 0 волта).

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

вход/изход (input/output)
Константи използвани с pinMode() функцията за да дефинират режима на цифров пин като вход (INPUT) или изход (OUTPUT).

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

if

if изявленията проверяват дали дадено условие е изпълнено, като например дали аналоговите стойности са над определена граница, и задават изпълнението на командите в къдравите скоби ако условието е изпълнено. Ако условието не е изпълнено програмата прескача тези команди. Форматът на if изявленията е следния:

```
if (nyakakvaPromenliva ?? stojnost)
{
        izpylniKomanda;
}
```

В горния пример nyakakvaPromenliva се сравнява с друга стойност, която на свой ред може да е променлива или константа. Ако сравнението, или условието, в скобите е вярно се изпълняват командите в къдравите скоби. Ако условието не е изпълнено програмата прескача командите в къдравите скоби.

Забележка: Внимавайте по грешка да не използвате '=', например if(x = 10) защото въпреки че това условие е технически правилно и Ардуино няма да го отхвърли като грешка, то задава стойност 10 на променливата x и така условието винаги ще е изпълнено. Използвайте '= =', като при if(x = = 10), което проверява дали x е равно на 10 или не.

if… else

if… else позволява да се взимат ‘или-или’ решения. Например, ако искате да проверите показанията на цифров вход и да зададете една команда ако показанието е HIGH или да зададете друга команда ако показанието е LOW можете да го направите по следния начин:

```
if (inputPin = = HIGH)
{
        napraviA;
}
else
 {
        napraviB;
}
```

else може да предхожда друг if тест така че множество, взаимно изключващи се условия да се проверяват едновременно. Възможно е и да имате неограничен брой else разклонения. Запомнете обаче, че само един от сетовете команди ще бъде изпълнен в зависимост от зададените условия:

```
if (inputPin < 500)
{
        napraviA;
}
else if (inputPin >= 1000)
{
        napraviB;
}
else
{
        napraviC;
}
```

Забележка: if изявленията просто проверяват дали условието в скобите е истина или неистина. Условието може да е всяко валидно изявление на езика C.

for

for изявленията се използват за да се изпълни даден блок от команди определен брой пъти. Често се използва като брояч който да следи колко пъти са изпълнени командите и да сложи край на цикъла. При обявяване на for цикъла се задават три параметъра отделени с точка и запетая (;), ето така:

```
for (inicializaciya; uslovie; izraz)
{
        izpulniKomanda;
}
```

Инициализацията (initialization) е локална променлива, изпълняваща ролята на брояч, която се задава в началото и се изпълнява само веднъж. При всяко преминаване през цикъла се проверява *условието* (condition) и ако то продължава да е вярно се изпълняват командите и условието на *израза* (expression), след което отново се проверява дали условието още е вярно. Когато условието престане да бъде истина цикълът приключва.

Примерът долу задава integer i със стойност 0, проверява дали i има стойност по-малка от 20, и ако условието е истина прибавя 1 към i и изпълнява командите в къдравите скоби:

```
for (int i = 0; i < 20; i++)            // deklarira I; proveryava dali i e
{                                       // po-malko ot 20; pribavya 1 kym i
        digitalWrite (13, HIGH);        // vklyuchva pin 13
        delay(250);                     // izchakva ¼ sekunda
        digitalWrite(13, LOW);          // izklyuchva pin 13
        delay(250);                     // izchakva ¼ sekunda
}
```

Забележка: for цикълът при C е много по-гъвкав отколкото при някои други компютърни езици, включително и BASIC. Всеки (или дори всички) от трите елемента при задаването на цикъла могат да се пропуснат, въпреки че точка и запетаите са задължителни. Също така изявленията за инициализация, условие и израз могат да са представени от всяко валидно изявление на C. Такива необичайни изявления могат да се окажат полезни за разрешаването на някои по-сложни проблеми.

while

while циклите се изпълняват продължително и непрестанно докато изразът в скобите престане да е истина. Проверяваната променлива трябва да се промени или цикълът никога няма да свърши. Променливата може да се променя чрез задание в кода (зависимост от друга променлива), или външно условие (показанията от сензор).

```
while (nyakakvaPromenliva ?? stojnost)
{
        izpulniKomanda;
}
```

Следният пример проверява дали ‘nyakakvaPromenliva’ е със стойност по-малка от 200 и ако условието е вярно командите в къдрави скоби се изпълняват докато ‘nyakakvaPromenliva’ придобие стойност не по-малка от 200.

```
while (nyakakvaPromenliva < 200) // proveryava dali stoijnostta e po-malka ot 200
{
        izpulniKomanda;             // komandite se izpulnyavat
        nyakakvaPromenliva++;       // pribavya 1 kym nyakakvaPromenliva
}
```

do… while

do цикълът работи по същия начин като while цикъла, като единствената разлика е, че условието се проверява в края на цикъла, така че цикълът се изпълнява поне веднъж.

```
do
{
        izpulniKomanda;
} while (nyakakvaPromenliva ?? stojnost);
```

Следният пример прикача readSensors() към променливата ‘x’, изчаква 50 милисекунди, и след това изпълнява цикъла докато ‘x’ вече не е по-малко от 100:

```
do
{
        x = readSensors();   // zadava na x stojnostta ot readSensors()
        delay(50);           // izchakva 50 milisekundi
} while (x < 100);           // izpulnyava cikula otnovo ako x e po-malko ot 100
```

pinMode(pin, rejim)

Използва се във void setup() за да конфигурира определен пин като вход (INPUT) или изход (OUTPUT).

```
pinMode(pin, OUTPUT);           // opredelya ‘pin’ kato izhod (OUTPUT)
```

По подразбиране пиновете на Ардуино са входове (INPUT) и не е необходимо да се задават като входове чрез pinMode(). Пинове конфигурирани като INPUT са в състояние на висок импеданс (Анг. high-impedance state).

Atmega чипът разполага с удобни 20К ома резистори, които могат да бъдат активирани чрез софтуерни команди. Тези вградени резистори се командват от софтуера по следния начин:

```
pinMode (pin, INPUT);           // opredelya ‘pin’ kato vhod (INPUT)
digitalWrite (pin, HIGH);       // vklyuchva vgradenite rezistori
```

Тези резистори обикновено се използват за свързване на различни входни устройства като например бутони. Забележете как в горния пример не обръщаме пина в изход (OUTPUT), а просто прилагаме метод за да активираме вградените резистори.

Пинове конфигурирани като изходи (OUTPUT) са в състояние на нисък импеданс (Анг. low-impedance state) и могат да доставят до 40 mA (милиампера) напрежение на външни устройства или ел. вериги. Това може да запали ярко светодиод (като не забравяте да поставите резистор), но е недостатъчно за повечето релета, соленоиди или електромоторчета.

Къси съединения в OUTPUT пиновете на Ардуино или много висока сила на тока могат да повредят или унищожат пина и дори Atmega чипа. Поради тази причина е добре винаги да свързвате OUTPUT пиновете към външни устройства във верига с резистори от 470 ома или 1К ома.

digitalRead(pin)

Отчита показанието от определен цифров пин и връща като резултат HIGH или LOW. Пинът може да се определи като променлива или константа.

```
stojnost = digitalRead(pin);        // zadava ‘stojnost’ ravna na pokazaniyata ot
                                    // INPUT pina
```

digitalWrite(pin, stojnost)

Задава логически HIGH или LOW (включва или изключва) определен пин. Пинът може да се определи като променлива или константа.

```
digitalWrite(pin, HIGH);    // zadava HIGH na ‘pin’
```

Примерът по-долу отчита показанието от бутон свързан към цифров входен (INPUT) пин и включва светодиод свързан към цифров изходен (OUTPUT) пин когато бутонът е натиснат:

```
int svetodiod = 13          // svyrzva svetodiod kym pin 13
int pin = 7                 // svyrzva buton kym pin 7
int stojnost = 0            // promenliva za syhranenie na otchetenata stojnost

void setup()
{
      pinMode (svetodiod, OUTPUT);   // opredelya pin 13 kato OUTPUT
      pinMode (pin, INPUT);          // opredelya pin 7 kato INPUT
}

void loop()
{
      stojnost = digitalRead (pin);            // zadava ‘stojnost’ ravna na INPUT
                                               // pina
      digitalWrite (svetodiod, stojnost);      // zadava na ‘svetodiod’ stojnostta
                                               // ot butona
}
```

analogRead(pin)

Отчита стойността от определен аналогов пин с резолюция от 10 бита. Тази функция работи само на аналоговите пинове на Ардуино (0 – 5). Числото, което се връща като резултат има стойности от 0 до 1023.

```
stojnost = analogRead(pin);        // zadava ‘stojnost’ ravna na pokazanieto ot ‘pin’
```

Забележка: За разлика от цифровите пинове, аналоговите не е нужно да се декларират нито като INPUT, нито като OUTPUT.

analogWrite(pin, stojnost)

Задава псевдо-аналогова стойност използвайки хардуер способен на широчинно импулсна модулация, или ШИМ (анг. pulse width modulation (PWM)) на изходен пин маркиран ‘PWM’. На по-новите платки Ардуино с чипове ATmega168 това са пинове 3, 5, 6, 9, 10 и 11. По-стари платки с чипове ATmega8 подържат ШИМ само на пинове 9, 10 и 11. Стойността се задава като променлива или константа от 0 до 255.

```
analogWrite (pin, stojnost);            // zadava ‘stojnost’ na analogoviya ‘pin’
```

Стойност 0 постоянно ще задава 0 волта на определения пин; стойност 255 генерира постоянни 5 волта на определения пин. За стойности между 0 и 255 пинът бързо ще сменя 0 и 5 волта и колкото по-голямо е числото толкова по-често на пина ще се задават 5 волта (или HIGH). Например стойността 64 ще задава 0 волта през три четвърти от времето и 5 волта през една четвърт от времето; 128 задава 0 волта половината от времето и 5 волта през останалата половина; а 192 ще бъде 0 волта през една четвърт от времето и 5 волта през три четвърти от времето.

Тъй като това е хардуерна функция пинът ще генерира постоянна вълна до следваща команда analogWrite (или команда digitalRead или digitalWrite на същия пин.)

Забележка: За разлика от цифровите пинове, аналоговите не е нужно да се декларират нито като INPUT, нито като OUTPUT.

Следният пример отчита аналогова стойност от аналогов входен пин, конвертира стойността като я разделя на 4, и я изпраща като ШИМ сигнал към ШИМ пин:

```
int svetodiod = 10;  // svetodiod s 220 oma rezistor na pin 10
int pin = 0;         // potenciometyr na analogov pin 0
int stojnost;        // promenliva za otchitane

void setup () {}     // nyama nujda ot setup

void loop ()
{
      stojnost = analogRead (pin);            // zadava ‘stojnost’ ravna na ‘pin’
      stojnost /= 4;                          // konvertira 0-1023 kym 0-255
      analogWrite (svetodiod, stojnost);      // izprashta PWM signal kym
                                              // svetodiod
}
```

изчакване(милисекунди) delay(ms)

Спира четенето на програмата за зададения в милисекунди период от време, като 1000 се равнява на 1 секунда.

```
delay(1000);                    // izchakva edna sekunda
```

millis()

Връща число във формат long с броя на милисекундните откакто Ардуино платката е започнала изпълнението на програмата.

```
stojnost = millis();  // zadava ‘stojnost’ ravna na millis()
```

Забележка: След приблизително девет часа това число ще се превърти (ще започне да брои отново от нула).

min(x, y)

Изчислява и връща по-малкото от две числа от кои да е вид данни.

```
stojnost = min (stojnost, 100);    // zadava stojnost ravna na po-malkoto
                                   // ot ‘stojnost’ i 100, garantirajki che
                                   // ‘stojnost’ nikoga nyama da e nad 100
```

max(x, y)

Изчислява и връща по-голямото от две числа от кои да е вид данни.

```
stojnost = max (stojnost, 100);    // zadava stojnost ravna na po-golyamoto
                                   // ot ‘stojnost’ i 100, garantirajki che
                                   // ‘stojnost’ vinagi shte e pone 100
```

randomSeed(seed)

Задава стойност (или seed) като отправна точка за random() функцията (от анг. случайна стойност).

```
randomSeed (stojnost);    // zadava 'stojnost' kato otpravna tochka
```

Понеже Ардуино не може да създаде истински случайно число randomSeed позволява да заложим променлива, константа или функция във функцията за генериране на случайни стойности и така да генерираме по-случайно „случайно" число. Има много различни seed-ове които могат да се използват в тази функция, включително millis() и дори analogRead() който да отчита електрическия „шум" в аналогов пин.

random(max)
random(min, max)

Random функцията позволява да се генерират псевдо-случайни числа в определена рамка зададена чрез минимално и максимално възможни стойности.

```
stojnost = random (100, 200);     // zadava na 'stojnost' sluchajno
                                  // chislo sys stojnost mejdu 100 i 200
```

Забележка: Използвайте тази функция след като сте използвали randomSeed() функцията.

Примерът по-долу генерира случайно число между 0 и 255 и изпраща ШИМ сигнал равен на случайно генерираното число към ШИМ пина:

```
int sluchajnoChislo;        // promenliva za suhranenie na sluchajnata stojnost
int svetodiod = 10;         // svetodiod s 220 oma rezistor na pin 10

void setup () {}            // nyama nujda ot setup

void loop ()
{
      randomSeed (millis());                    // zalaga millis() kato otpravna
                                                // tochka
      sluchajnoChislo = random (255);           // sluchajno chislo ot 0 do 255
      analogWrite (svetodiod, sluchajnoChislo);     // izprashta PWM signal
      delay (500);                              // izchakva polovin sekunda
}
```

Serial.begin(skorost)

Отваря серийния порт и задава скоростта за обмен на данни. Обичайната скорост за обмен на данни с компютъра е 9600 бита в секунда (bps) въпреки че Ардуино подържа и други скорости.

```
void setup()
{
        Serial.begin (9600);        // otvarya serijniya port i zadava
}                                   // skorost za obmen na danni ot 9600 bps
```

Забележка: Когато използвате серийна комуникация, цифровите пинове 0 (RX) и 1 (TX) не могат да бъдат използвани едновременно.

Serial.printIn(danni)

Принтира данни в серийния порт, последвани от автоматичен „carriage return" и "line feed". Тази команда изпълнява същата роля като Serial.print(), но е по-лесна за отчитане на данните на "Serial Monitor".

```
Serial.println (analogovaStojnost);       // izprashta stojnostta na analogovaStojnost
```

Забележка: Допълнителна информация за различните пермутации на Serial.printIn() и Serial.print() функциите може да намерите на уебсайта на Ардуино (www.arduino.cc).

Този пример отчита стойността от аналогов пин 0 и изпраща данните към компютъра на всяка секунда:

```
void setup ()
{
        Serial.begin (9600);                // otvarya serijniya port i zadava 9600 bps
}

void loop ()
{
        Serial.println (analogRead(0));     // izprashta analogovata stojnost
        Delay (1000);                       // izchakva edna sekunda
}
```

приложение

цифров изход (digital output)

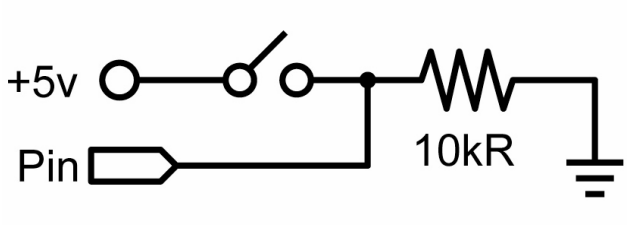
Това е основната „hello world!” програма с която се включват и изключват различни неща. В този пример светодиод свързан към пин 13 примигва на всяка секунда. Резисторът не е задължителен тъй като Ардуино си има вграден.

```
int svetodiod = 13;                     // svetodiod na pin 13

void setup ()                           // izpulni vednuj
{
      pinMode(svetodiod, OUTPUT);       // zalaga pin 13 kato izhod
}

void loop ()                            // izpulnyava se neprekusnato
{
      digitalWrite (svetodiod, HIGH);   // vklyuchva svetodioda
      delay (1000);                     // izchakva 1 sekunda
      digitalWrite (svetodiod, LOW);    // izklyuchva svetodioda
      delay (1000);                     // izchakva 1 sekunda
}
```

цифров вход (digital input)



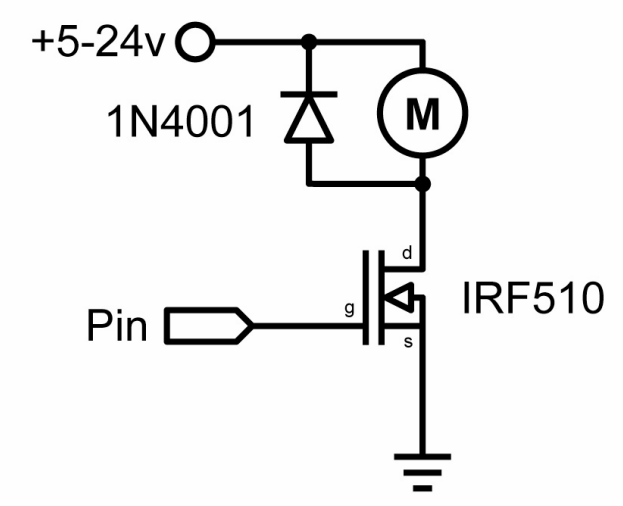
Това е най-елементарната форма на вход със само две възможности – включено или изключено. Този пример отчита прост бутон свързан към пин 2. Когато бутонът е натиснат (или затворен), пинът ще отчете HIGH и ще включи светодиода.

```
int svetodiod = 13;                 // izhoden pin za svetodioda
int butonPin = 2;                   // vhoden pin (za butona)

void setup ()
{
      pinMode (svetodiod, OUTPUT);  // deklarira svetodioda kato izhod
      pinMode (butonPin, INPUT);    // deklarira butona kato vhod
}

void loop ()
{
      if (digitalRead (butonPin) = = HIGH)    // proveryava dali butonyt e natisnat
                                              // (HIGH)
      {
      digitalWrite (svetodiod, HIGH);         // vklyuchva svetodioda
      delay (1000);                           // izchakva edna sekunda
      digitalWrite (svetodiod, LOW);          // izklyuchva svetodioda
      delay (1000);                           // izchakva edna sekunda
      }
}
```

изход с висока сила на тока (high current output)



Понякога е необходимо чрез Ардуино да се контролира сила на ток по-висока от 40 mA (милиампера). В такива случай се използва МОСФЕТ или транзистор за да превключва към по-висока сила на тока. Следния пример бързо включва и изключва МОСФЕТ-а по пет пъти в секунда.

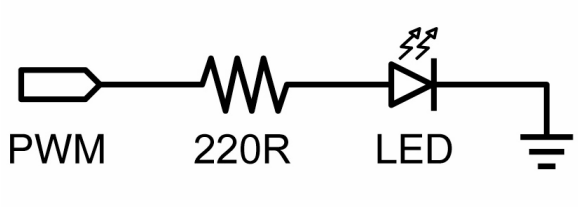
Забележка: Схемата по-горе показва електоромоторче и защитен диод, но някои не-индуктивни уреди (които не връщат ток обратно по веригата) може да се използват и без диода.

```
int izhodenPin = 5;                    // izhoden pin za MOSFET-a

void setup ()
{
        pinMode (izhodenPin, OUTPUT); // zadava pin 5 kato izhod
}

void loop ()
{
        for (int i = 0; i < 5; i++)            // izpulnyava se 5 puti
        {
        digitalWrite (izhodenPin, HIGH); // vklyuchva MOSFET-a
        delay (250);                           // izchakva ¼ sekunda
        digitalWrite (izhodenPin, LOW);  // izklyuchva MOSFET-a
        delay (250);                           // izchakva ¼ sekunda
        }
        delay (1000);                          // izchakva edna sekunda
}
```

шим изход (pwm output)



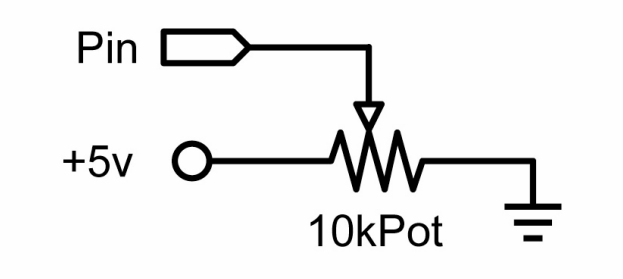
Широчинно импулсната модулация (ШИМ) е начин да се симулира аналогов изходен сигнал чрез пулсиране на изходния сигнал. Тази техника може да се използва за да се намали или увеличи яркостта на светодиод, а също така и за да се контролират серво машинки. Този пример бавно увеличава и намалява яркостта на светодиод с помощта на for цикли.

```
int svetodiod = 9;              // PWM pin za svetodioda

void setup () {}                // nyama nujda ot setup

void loop ()
{
        for (int i = 0; i <= 255; i++)        // pokachvashti se stojnosti za i
        {
        analogWrite (svetodiod, i);           // zadava yarkost ravna na i
        delay (100);                          // izchakva 100 milisekundi
        }
        for (int i = 255; i >= 0; i--)        // namalyavashti stojnosti za i
        {
        analogWrite (svetodiod, i);           // zadava yarkost ravna na i
        delay (100);                          // izchakva 100 milisekundi
        }
}
```

данни от потенциометър (potentiometer input)



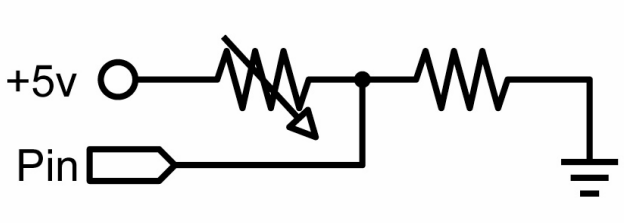
Използвайки потенциометър и един от Ардуино пиновете конвертиращи аналогов към цифров формат (analog-to-digital conversion) може да се отчитат стойности от 0 до 1024. Този пример използва потенциометър за да контролира скоростта с която мига светодиод.

```
int potPin = 0;       // vhoden pin za potenciometara
int svetodiod = 13;  // izhoden pin za svetodioda

void setup ()
{
      pinMode (svetodiod, OUTPUT);   // deklarira svetodioda kato izhod
}

void loop ()
{
      digitalWrite (svetodiod, HIGH);   // vklyuchva svetodioda
      delay (analogRead(potPin));       // izchakava
      digitalWrite (svetodiod, LOW);    // izklyuchva svetodioda
      delay (analogRead(potPin));       // izchakava
}
```

данни от променлив резистор (variable resistor input)



Променливите резистори включват фоторезистори, терморезистори, сензори за степен на изкривяване и други. Този пример използва функция за да отчете аналогови показания и на тяхна база да определи времето за изчакване (delay). Този код контролира скоростта с която се увеличава и намалява яркостта на светодиода.

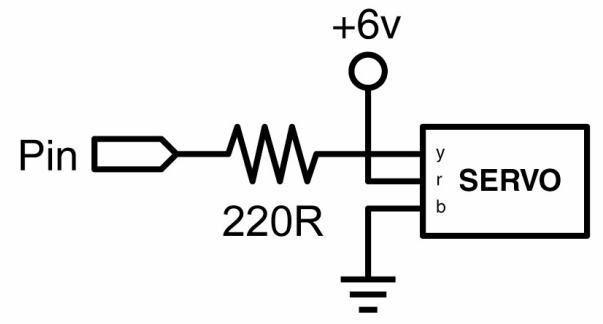
```
int svetodiod = 9;          // PWM pin za svetodioda
int analogPin = 0;          // analogow pin za promenliviya rezistor

void setup () {}            // nyama nujda ot setup

void loop ()
{
for (int i = 0; i <= 255; i++)                  // pokachvashti se stojnosti za i
        {
        analogWrite (svetodiod, i);             // zadava yarkost ravna na i
        delay (delayStojnost());                // otchita vreme i izchakva
        }
        for (int i = 255; i >= 0; i--)          // namalyavashti stojnosti za i
        {
        analogWrite (svetodiod, i);             // zadava yarkost ravna na i
        delay (delayStojnost());                // otchita vreme i izchakva
        }
}

int delayStojnost()
{
        int v;                          // syzdava vremenna promenliva
        v = analogRead (analogPin);     // otchita analogovata stojnost
        v /= 8                          // konvertira 0-1024 kym 0-128
        return v                        // dava krajna stojnost
}
```

задвижване на серво машинки (servo output)



Серво машинките са моторчета, които могат да се движат по ос от 180 градуса. Те се нуждаят само от един импулс на всеки 20 милисекунди. Този пример използва servoPulse функцията за да завърти серво машинката от 10 до 170 градуса и обратно.

```
int servoPin = 2;              // servo svyrzano kym cifrov pin 2
int moyatYgyl;                 // ygylyt na servoto – grubo 0 do 180
int pulseWidth;                // promenliva za servoPulse finkciyata

void setup ()
{
        pinMode (servoPin, OUTPUT);     // opredelya pin 2 kato izhod
}

void servoPulse (int servoPin, int moyatYgyl)
{
        pulseWidth = (moyatYgyl * 10) + 600;    // opredelya zabavyaneto
        digitalWrite (servoPin, HIGH);          // vklyuchva servoto
        delayMicroseconds (pulseWidth);         // izchakva mikrosekundi
        digitalWrite (servoPin, LOW);           // izklyuchva servoto
}

void loop ()
{
        // servoto trygva ot 10 gradusa i se zavyrta do 170
        for (moyatYgyl = 10; moyatYgyl <= 170; moyatYgyl++)
        {
        servoPulse (servoPin, moyatYgyl);       // izprashta pin i ygyl
        delay (20);                             // opresnyava cikyla
        }
        // servoto trygva ot 170 gradusa i se zavyrta do 10
        for (moyatYgyl = 170; moyatYgyl >= 10; moyatYgyl--)
        {
        servoPulse (servoPin, moyatYgyl);       // izprashta pin i ygyl
        delay (20);                             // opresnyava cikyla
        }
}
```