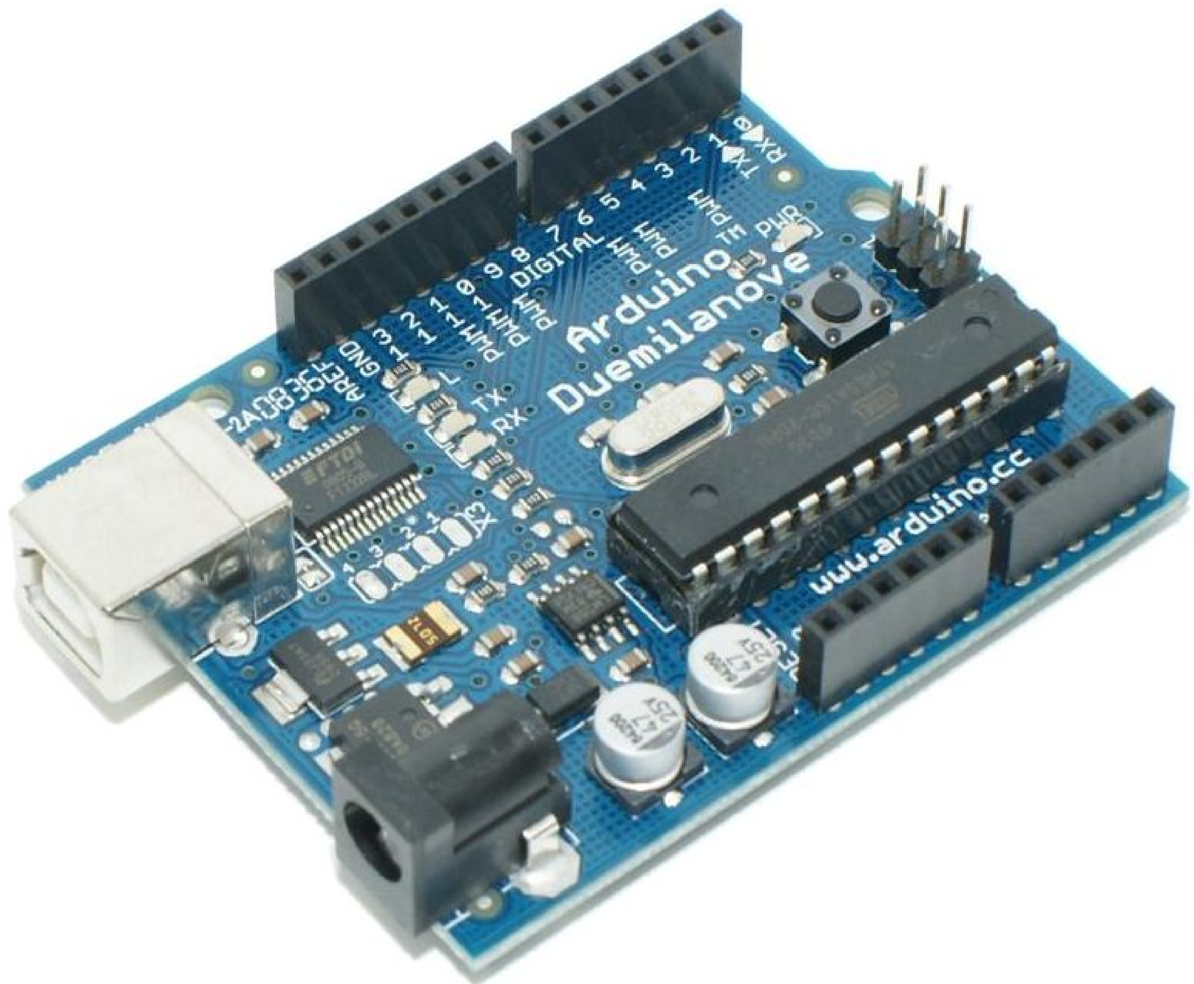
Arduino

Programozási kézikönyv



Budapest, 2011. © TavIR

Brian W. Evans írása alapján
fordította, kiegészítette és frissítette Cseh Róbert

Arduino programozási kézikönyv

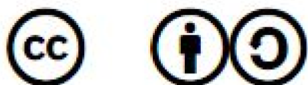
Az eredeti angol nyelvű kiadást írta és összeállította:
Brian W. Evans.

A magyar nyelvű kiadást fordította, kiegészítette és átdolgozta:
Cseh Róbert (csehrobert@tavir.hu, www.tavir.hu).

A felhasznált írások szerzői: Massimo Banzi, Hernando Barragan, David Cuartielles, Tom Igoe, Daniel Jolliffe, Todd Kurt, David Mellis, Cseh Róbert és sokan mások.

A felhasznált főbb forrásmunkák:
http://www.arduino.cc
http://www.wiring.org.co
http://www.arduino.cc/en/Booklet/HomePage
http://cslibrary.stanford.edu/101/
http://avr.tavir.hu

Angol nyelvű kiadás első változata 2007. augusztusában került publikálásra.
A magyar nyelvű fordítás *Brian W. Evans: Arduino Programming Notebook (2008) Second Edition* alapján 2008. októberében készült el, majd átdolgozásra került 2011. áprilisában.



Ez a munka a *Creative Commons Attribution-Share Alike 2.5 License* alapján készült. A magyar nyelvű kiadás szerzője betartja és el is várja az ezen licenszben foglaltakat.

A licensz részletes feltételei megtekinthetőek a http://creativecommons.org/licenses/by-sa/2.5/ oldalon, illetve kivonata a könyv mellékletében is elolvasható.

Tartalom

Tartalom....3
Előszó....5
Előszó a magyar kiadáshoz....7
Arduino....8
Szerkezet - struktúra....8
    setup()....9
    loop()....9
    Functions (funkcióhívások)....9
    {} Kapcsos zárójel....10
    ; Pontosvessző....11
    /*... */ Blokk-megjegyzés....11
    // Egysoros megjegyzés....12
Változók....13
    Változók deklarálása....14
    Változó terület....14
    Byte....15
    Int....15
    Long....15
    Float....15
    Tömb....16
    Aritmetika....17
    Komponens összevonás....17
    Összehasonlító operátorok....18
    Logikai műveletek....18
    Konstansok....18
    Igaz/Hamis (True/False)....19
    Magas/Alacsony (High/Low)....19
    Bemenet/Kimenet (Input/Output)....19
    If utasítás....19
Folyamat kontrol....21
    If... Else....21
    For....21

While..22
Do...While..23
Ki- és bemenetek kezelése...24
pinMode(pin, mode)..24
digitalRead(pin)...24
digitalWrite(pin, value)..25
analogRead(pin)..25
analogWrite(pin, value)..25
Egyéb utasítások...27
Delay(ms)..27
Millis()...27
Min(x,y)...27
Max(x, y)...27
randomSeed(seed)..27
random(max) random(min, max)...28
Serial.begin(rate)...28
Serial.println(data)...28
Mellékletek..30
Digitális kimenet..30
Digitális bemenet...30
Digitális kimenet..31
PWM kimenet..32
Potenciométer bemenet (Analóg bemenet)...32
Változtatható ellenállás a bemeneten..33
Szervomotor..34
AVR-Duino Uno kapcsolási rajz..35
AVR-Duino Duemilanove kapcsolási rajz..36
CC BY-SA 2.5 licensz...37

Előszó

A kézikönyv segítségével könnyen elsajátíthatjuk a programozási szerkezeteket, a programírás alapjait. A minták Arduino / AVR-Duino alaprendszerhez készültek, de a programozási alapok más nyelvek esetén is támpontot adhatnak.

A könyv megírásakor a legfontosabb szempont az volt, hogy egy kezdőknek szóló referenciagyűjtemény álljon ezzel rendelkezésre. Így ezt egyszerűen a felkeresett honlapok, olvasott szakkönyvek mellett lehessen használni amolyan kézikönyvként; oktatásban, szakmai műhelyekben is egyszerűen kezelhető "puskaként" álljon rendelkezésre. A könyv felépítésének eldöntésében befolyásolt, hogy csak egyszerű minták, feladatok kerüljenek leírásra és ezen programtöredékek ugyanakkor már önállóan is használhatóak legyenek. Így a későbbiekben, ezek felhasználásával és továbbfejlesztésével az Arduino-t komplexebb alkalmazásokban is könnyen lehessen használni.

Az Arduino alapvetően a C nyelv szerkezetét követi, abból származik. A kézikönyv leírja a C és Arduino nyelv legtöbb közös elemének a szintaxisát és példákkal, kódtöredékekkel illusztrálja a használatukat. A legtöbb Arduino funkció egy alap eljárásgyűjteményben került meghatározásra (ún. függvény-könyvtárban vagy eljárás-gyűjteményben). Az alapeljárások használatának ismertetése, kapcsolási rajzokkal és mintaprogramokkal kiegészítve a mellékletben található. A könyv felépítése, összeállítása során számos helyen köszönetet kell mondani *Dan O'Sulivan, Tom Igoe: Physical Computing: Sensing and Controlling the Physical World with Computers (2004)* című könyvének; a programok jelzései, belső hivatkozások e könyv alapján készültek.

Meg kell említeni *Massimo Banzi: Getting started with Arduino (2009)* című könyvét, mely más néven az "Arduino biblia". Ez a rövid könyvecske ennek előfutára. Felhasználásra került a C nyelvű programozók alapműveként számon tartott *Brian W. Kernighan, Dennis M. Ritchie: The C Programming Language (1988)* írásról, ahogyan

P. Prinz, T. Crawford: C in a Nutshell (2005) könyvekről, melyek betekintést nyújtanak az eredeti C nyelv szintaxisba.

Azonban nem szabad elfelejtkezni a kézikönyv létrejöttét inspiráló, a kollektív tudat és központi honlap közösségéről. Nélkülük ez az összefoglaló könyv soha nem jött volna létre. Az itteni mintákra épülve a közösség által bővülő *http://www.arduino.cc* honlap fóruma, twitter-csatornája és játszótere sem működhetne.

Brian W. Evans

Előszó a magyar kiadáshoz

A magyar nyelvű kiadást sok vívódás előzte meg, az első változat publikálása és kiadása csak szűk körben történt meg. A visszajelzések, a további kérések és igények egy komplex, többkötetes könyv kiadását vetítették előre. Azonban erre - a mindenki által jól ismert okok miatt - nem került sor. Ezek közül kiemelkedik a 24→48 óra időkibővítő készülék korlátozott működése, mely a 2010. március 28-i* kísérletben és a 2011. március 27-i próbán is csúfosan leszerepelt. A mérsékelt kudarcként megélhető 24→23 óraszűkítést eredményezte a gép működése. Így a további kísérleti projektet befejezve (megakadályozandó a 24→20 vagy még rosszabb eredményeket) - így az eddig elkészült könyv inkább kiadásra került.

Remélhetőleg hasznos segédeszköz lesz a mikrokontrolleres világgal ismerkedők részére ezen kiadás. Haszonnal forgatják a most kezdő-ismerkedőktől kezdve a profi, de 1-1 mintából ötletet merítő felhasználók is. Az egyedi minták, fejlesztések és önálló alkalmazásoknak a *http://www.tavir.hu* oldal mellett működő AVR fejlesztői közösség (*http://avr.tavir.hu*) is tágas teret biztosít a megjelenésre. Jelenleg ez Magyarországon, magyar nyelven a legnagyobb szakmai AVR-specifikus közösség.

Az angol nyelvű könyv magyar változata nem jöhetett volna létre a TavIR-AVR fórum tagjainak unszolásai illetve segítő javaslatai nélkül.

A könyv frissítései, kiegészítései és hasznos tippek érhetőek el a TavIR oldalain:

- http://avr.tavir.hu (A honlap maga)
- http://www.twitter.com/taviravr (Twitter csatorna)
- http://tavir.blog.hu (Blog, hírekkel)
- http://www.facebook.com/taviravr (A közösségi háló)

Ha a könyvben leírt mintákat szeretnéd kipróbálni, alkalmazni, továbbfejleszteni és élőben kipróbálni: a hozzávaló fejlesztőkészletek a http://avr.tavir.hu oldalon megvásárolhatóak. Vagy akár a tanfolyamon is szívesen látlak.

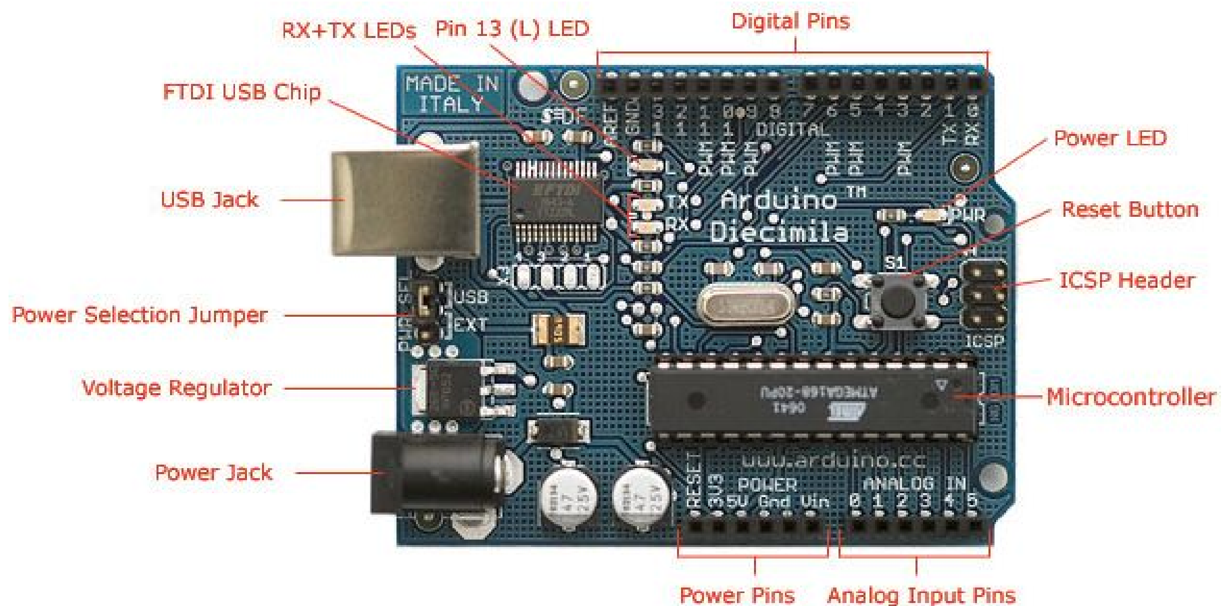
Cseh Róbert
(csehrobert@tavir.hu.hu)

* Téli-nyári óraállítás

ARDUINO

Az Arduino egy egyszerű elektronikus áramkörön és egy szoftverfejlesztő környezetén alapuló nyílt (physical computing) platform. Az Arduino programozási nyelve a C alapokon nyugvó C++ implementáció.

Az Arduino interaktív tárgyak készítésére használható, számtalan kapcsolót vagy szenzort bemenetként olvasva; lámpák, motorok és egyéb kimenetek kimeríthetetlen választékát képes vezérelni. Az Arduino projektek állhatnak önmagukban, vagy különböző számítógépes programokkal kommunikációval elkészítve.



SZERKEZET - STRUKTÚRA

Az Arduino programozási nyelv szerkezete igen egyszerű. A program két fő részből áll, ezek szintaktikája a következő:

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

```
   statements;
}
```

Ahol a *setup()* az előkészítést, alapbeállítások inicializálását írja le, míg a *loop()* a tényleges művelet-végrehajtó, futó rész. Mindkét funkcióra szükség van a programok működéséhez.

A *setup()* funkcionális rész a változók deklarálását tartalmazza, melyet a program legelején kell megtenni. Ez az első funkcióhívás, mely lefut a program elején, de csak egyetlen egyszer és csak ekkor. Itt kell meghatározni például az egyes chip lábak irányát (*pinMode* utasítás) vagy például a soros kommunikációt inicializálni.

A *loop()* funkció a következő, melyben a felsorolt utasítások folyamatosan ismétlődnek - bemenetek olvasása, kimenetek beállítása, stb. Ez a funkció az Arduino programmagja!

SETUP()

A *setup()* funkció a program indulásakor egyetlen egyszer fut le. Itt kerül a programba például a kivezetések inicializálása, a soros port kezelése. A programba mindenképp be kell illeszteni ezt a funkcióhívást, még ha semmit sem végzünk benne - akkor is!

```
void setup() {
   pinMode(pin, OUTPUT);     //sets the 'pin' as output
}
```

A programban a *pin* (chip láb) kimenetként kerül beállításra.

LOOP()

A *setup()* funkcióhívás után a *loop()* következik a programban. Ez a programrész az Arduino áramkört működteti, válaszol a külső kérésekre, interakciót hajt végre.

```
void loop(){
   digitalWrite(pin, HIGH);      // turns 'pin' on
   delay(1000);                  // pauses for one second
   digitalWrite(pin, LOW);       // turns 'pin' off
   delay(1000);                  // pauses for one second
}
```

A főprogram a *pin* (kiválasztott chipkivezetés) magas állapotba kerül, majd a program vár 1000 msec időt. Ezután a chip kivezetése alacsony szintbe vált, majd újra 1000 msec várakozás következik. És utána ez ismétlődik újra meg újra meg újra…

FUNCTIONS (FUNKCIÓHÍVÁSOK)

A funkcióhívás vagy funkcióblokk nem más, mint egy összetartozó programrész, melyre akkor kerül a vezérlés, mikor a blokkot a főprogramból meghívjuk. A funkció-program elnevezése és szerkezete:

```
void funkcionev()
```

A *setup()* illetve a *loop()* dedikált funkció szerepét az előbbiekben már körbejártuk, a többi beépített funkció a későbbiekben kerül bemutatásra.
Saját funkció-rutint is írhatunk, ennek például ismétlődő programrészek esetén van jelentősége. (Így átláthatóbb lesz a program, könnyebb a javításokat kézben tartani és csökken a kész program mérete is.) A funkció-blokkokat használat előtt deklarálni kell. Itt meg kell mondani, hogy a funkció lefutása után valamilyen eredményt adjon-e vissza a rutin (pl. *int* típusú eredményt, mely egy 1...65535 közti szám lehet). Ha ilyen érték visszaadásra nincsen szükség - mert például csak egy interakció kezelésre és nem számolásra szolgál a funkcióblokk - ez is megoldható: ekkor a *void* kulcsszót kell használni. Ha a funkcióhívásnak valamiféle paramétert kell átadni, akkor azt zárójel segítségével jelölve tehetjük meg.

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

Itt a *type* a visszaadott érték jellemzője, a *parameters* az átadott paraméterek, a *statements* pedig a funkció belső programjának utasításai.

A következő egész (integer vagy int) típusú függvény, mely a *delayVar()* névre hallgat - és késleltetésre használatos. A mintában egy potenciométer állását olvassuk be és ennek megfelelő várakozás történik a program futása során. Ebben az esetben először létrehozunk egy helyileg használt változót (*v*-vel jelölve), majd a potenciométer állásának értékét ebbe a változóba beolvassuk. A potméter-állás értéke 0…1023 közti érték lehet, melyet azután néggyel osztunk. A végeredményként így 0…255 közti számot kapunk - ezt adjuk vissza végül a főprogramnak.

```
int delayVal(){
   int v;                     // create temporary variable 'v'
```

```
    v = analogRead(pot);       // read potentiometer value
    v /= 4;                    // converts 0-1023 to 0-255
    return v;                  // return final value
}
```

{} Kapcsos zárójel

A kapcsos zárójel definiálja a funkcióblokk kezdetét illetve végét. A zárójelek közé írjuk például az utasításokat, más funkcióhívásokat illetve egyéb utasításokat.

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

A nyitó zárójelet { mindenképpen valahol követnie kell egy záró } zárójelnek. Pár nélkül nem maradhatnak! Ezek gyakran vezetnek misztikus hibákhoz, és nagyon nehéz sok esetben megtalálni, hogy hol helyezkedik el a nyitó és hol a záró zárójel. Párba rendezni őket hosszabb program esetén igen nehéz, így a hiba nyomára bukkanni is bonyolult lehet.
Az Arduino fejlesztőkörnyezet a zárójelek párjának kijelölésében nagy segítséget nyújt, ugyanis a nyitó zárójelre kattintva a legvalószínűbb párját kiemeli.

; Pontosvessző

A pontosvesszőt kell használnunk az utasítások után, illetve a programrészek elválasztásához. Ezt használjuk például a *loop()*-on belül is. Például:

```
int x = 13;       // declares variable 'x' as the integer 13
```

Az „x” változó értéke: 13.

Fontos! Lefelejtve a sorok végéről a pontosvesszőt, a fordító hibaüzenetet fog adni. Az üzenet általában nyilvánvaló, a fordító a hiba helyét is általában megnevezi. Ha netán átláthatatlan vagy logikátlan hibaüzenetet kapunk, akkor az első dolog, amire gyanakodjunk: lefelejtettünk valahol egy pontosvesszőt. Ezt a hiányzó pontosvessző-helyet a legegyszerűbben a fordító által jelzett sor közelében találhatjuk meg.

/*… */ Blokk-megjegyzés

A blokk-megjegyzések vagy a több soros megjegyzések olyan szöveges részek, melyet a program fordítása során figyelmen kívül kell hagyni. Ezek közt van lehetőség magyarázatokat, megjegyzéseket beszúrni a programba. Segítik megérteni a program

működését, ha például egy-két év múlva elővesszük. Írásmódban a legegyszerűbb, ha /* jellel kezdjük, és */ jellel fejezzük be. Egy példán keresztül:

```
/* this is an enclosed block comment
don't forget the closing comment -
they have to be balanced! */
```

Ez egy zárt blokk-megjegyzés nyitó és záró jelekkel

Érdemes megjegyzéseket használni, hisz a lefordított programkódba nem kerülnek bele, így helyet sem foglalják. A hibakeresést viszont nagymértékben megkönnyítik, főleg ha több hibát is hibát keresni kell…

Fontos! Amíg lehetőség van rá érdemes egysoros megjegyzéseket írni. Figyeljünk arra, hogy megjegyzésen belül másik megjegyzést írni nem túlságosan szerencsés.

// Egysoros megjegyzés

Egysoros megjegyzés a // jellel kezdődik és automatikusan a sor végén soremeléssel ér véget. Hasonlóan a blokk-megjegyzéshez viselkednek, azaz a fordítás során mintha ott sem lennének.

```
// this is a single line comment
```

Ez egy egysoros megjegyzés

Az egysoros megjegyzéseket gyakran változók után használjuk, a magyarázatához, hogy milyen célt szolgál. Így később könnyebb átlátni a program működését. Nem is kell emiatt külön sorba beírni, az utasítás utáni területre is lehetőség van.

Változók

A változók számértékek tárolására szolgálnak, melyeket a program futása során használhatunk fel. Nevük is mutatja funkciójukat: a bennük tárolt értékek változnak. Hasonló célt szolgálnak az állandó (szám)értékre szolgáló elnevezések, ezeket azonban konstansnak hívjuk (az értékük sosem változik meg). A változókat használat előtt mindenképp deklarálni kell és ha ismert/fontos a kiindulási értéke, akkor azzal fel is kell tölteni. A következő mintában deklarálunk egy *inputVariable* változót, és a 2-es analóg lábon beolvasható értéket rendeljük hozzá.

```
int inputVariable = 0;           // declares a variable and
                                 // assigns value of 0
inputVariable = analogRead(2);   // set variable to value of
                                 // analog pin 2
```

Az *inputVariable* maga a változó. Az első sorban deklaráljuk, típusának egészértéket mondunk (integer) és kezdeti értékkel is feltöltjük. A második sorban értékének a 2. Analóg bemeneten mérhető értéket adjuk. Utána ezt bárhol másutt a kódban már felhasználhatjuk.

Ha egyszer a változónak a programban adtunk már értéket, akkor ellenőrizhetjük is ezt, hogy valamely feltételnek megfelel - vagy akár számolhatunk is vele. A mintaprogramban három hasznos műveletet is megmutatok. Ha az *inputValue* értéke kisebb, mint 100, akkor az értékét 100-ra változtatom. Majd ezután egy késleltetést használok, ami az *inputValue* msec időtartamot jelenti. Ez az előző feltétel alapján legalább 100 msec.

```
if (inputVariable < 100) {  // tests variable if less than 100
   inputVariable = 100;     // if true assigns value of 100 }
   delay(inputVariable);    // uses variable as delay
}
```

Fontos! A változóknak érdemes olyan nevet adni, ami utal a funkciójára. Ezáltal a kód áttekinthető és olvashatóbb lesz. A változónév lehet például a *tiltSensor* vagy a *pushButton* - ezek segítenek a programozónak vagy bárkinek, aki a programmal foglalkozik, hogy megértse melyik változó mit jelent. A változó neve lehet megnevezés vagy érték és a kód így válik ember által olvashatóvá. A fordítóprogramnak szinte

mindegy, hogy ez értelmes vagy véletlen betűhalmaz. Azonban a változók névadására nem használhatóak utasításnevek, funkciónevek, kulcsszavak.

Változók deklarálása

Minden változót deklarálni kell, mielőtt használatba vennénk. A változók deklarálása azt jelenti, hogy meg kell adnunk a típusát (pl. int, long, float...), a nevét és opcionálisan a kezdőértékét. Ezt - természetesen - elegendő egyszer megtenni a programban, a használat során a változó értékét - a meghatározott típus specifikációin belül - szabadon változtathatjuk.
A következő mintában az *inputVariable* változót int (vagyis egész) típusúként deklaráljuk, melynek kezdő értéke nulla. Ez egyszerűen leírható:

```
int inputVariable = 0;
```

Az, hogy egy változót hol deklarálunk: befolyásolja azt, hogy a változó hol használhatjuk.

Változó terület

A változót deklarálhatjuk a program elején (a *void setup()* sor előtt), a funkciókon belül vagy valahol a program működésén belül, ahol ciklikus programvégrehajtás van (pl. *for* utasítás esetén). A változó deklarációja meghatározza a változó felhasználhatóságát, azaz hogy a program mely részén hivatkozhatunk rá.
A globálisan deklarált változót a programon belül bárhol használhatjuk. Ehhez a változót a *setup()* funkció előtt kell deklarálni.
Egy lokális változót - melyet funkcióblokkban definiálunk - csak ezen belül használhatunk. Ez azt jelenti, hogy lehetőségünk van két vagy több esetben azonos változónevet használni a programon belül - más-más funkcióblokkban. Az, hogy csak egy funkció férjen hozzá a változóhoz, nagyban csökkenti a programozás során elkövetett hibákat.
A következő minta megmutatja, hogy hogyan deklarálhatóak a különböző típusú változók, és ezek honnan érhetőek el:

```
int value;                      // 'value' is visible
                                // to any function
void setup() {
                                // no setup needed
}
```

```
void loop() {
  for (int i=0; i<20;)      // 'i' is only visible
       {                    // inside the for-loop
       i++;
       }
  float f;                  // 'f is only visible
}                           // inside loop
```

Byte

Byte típusban 8-bites egész számértéket tárolhatunk, azaz 0-255 közötti értéket vehet fel.

```
byte someVariable = 180;    // declares 'someVariable'
                            // as a byte type
```

Int

Egész vagy más néven integer típusú változóban egész számot tárolhatunk 16 biten, melynek értéke -32768 és +32767 közé esik.

```
int someVariable = 1500;    // declares 'someVariable'
                            // as an integer type
```

Fontos! Egész típusú változó esetén ún. túlcsordulással is számolnunk kell. Ekkor a legnagyobb érték után a legalacsonyabbat kapjuk! Illetve csökkentéskor a legkisebb után a legnagyobbat. Például, ha x = 32767 és ehhez hozzáadunk egyet (programban: x=x+1 vagy x++), akkor az eredmény: -32768!

Long

Bővített méretű adattípus az ún. long (long integer – kb. hosszú egész). Az adattípus egész számok tárolására alkalmas 4 byte (32 bit) lefoglalásával. A tárolható számérték 2,147,483,647 és -2,147,483,648 közé esik.

```
long someVariable = 90000;  // declares 'someVariable'
                            // as a long type
```

FLOAT

A lebegőpontos (float) adattípus tizedes törtek tárolására is alkalmas. Sokkal nagyobb számtartomány lefedésére megfelelő, mint az int (egész) típus. A float típus értéke -3.4028235E+38-tól -3.4028235E+38-ig terjed.

```
float someVariable = 3.14;  // declares 'someVariable'
                            // as a floating-point type
```

Fontos! A lebegőpontos számok nem mindig pontosak. A tizedes törtek tört része (is) 2-es számrendszerben kerül ábrázolásra! A lebegőpontos számokkal való matematikai műveletek is lassabbak, mintha az egész (int) típussal végeznénk. Erre - főleg sok számolást tartalmazó program esetén - figyelni kell!

TÖMB

A tömb nem más, mint értékek/számok gyűjteménye, melyeket az ún. index számán (kb. sorban elfoglalt helye) érhető el. A tömb bármely elemét meg lehet így címezni. A tömb indexelése a 0. azonosítóval kezdődik. A tömböt - a többi változóhoz hasonlóan - használat előtt deklarálni kell és lehetőség szerint a kezdőértékkel feltölteni.

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

Látható, hogy a tömb deklarálásakor meg kell adni a típusát, méretét és opcionálisan az elemeit:

```
int myArray[5];   // declares integer array w/ 6 positions
myArray[3] = 10;  // assigns the 4th index the value 10
```

Az első lépésben definiálunk egy 6 elemű tömböt, majd ennek 4. eleme a 10-es értéket kapja.

Kiolvasni a tömb adott elemét és ezt egy változó értékének átadni az indexszámmal azonosítva lehetséges:

```
x = myArray[3];   // x now equals 10
```

A tömböket gyakran használjuk ciklusokban, ahol egy számláló növekszik és ez "véletlenül" a tömb index-azonosítója is egyben. A következő példában egy tömb segítségével LED fényét vibráltatjuk. A ciklusban a számláló 0-ról indul és a tömb 0. elemét kiolvassuk (180) és ezt a PWM kimenetre (10) írjuk. 200 msec szünet után a tömb következő elemével folytatjuk.

```
int ledPin = 10;
                                  // LED on pin 10
byte flicker[] = {180, 30, 255, 200, 10, 90, 150, 60};
                                  // above array of 8
void setup()                      // different values
{
       pinMode(ledPin, OUTPUT);  // sets OUTPUT pin
}
void loop()
{
  for(int i=0; i<7; i++)          // loop equals number
       {                          // of values in array
            analogWrite(ledPin, flicker[i]);
                                  // write index value
            delay(200);           // pause 200ms
       }
}
```

Aritmetika

Számtani (aritmetikai) műveletek - például az összeadás, kivonás, szorzás, osztás - szintén alkalmazhatóak a programozás során. Ezek írásmódja a következő:

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

A művelet során fontos, hogy az adattípusokban eltérés - lehetőleg - ne legyen. A megfelelő adattípus kiválasztása nagyon fontos, mert például a 9/4 eredménye 2, nem pedig 2.25. Ennek oka, hogy a változókat *int* (egész) típusként definiáltuk. Ez azt is jelenti, hogy ún. túlcsordulás (overflow) léphet fel, ha az eredmény nagyobb, mint az eredmény változótípusában letárolható.

Ha a változók - melyeken műveletet végzünk - eltérő típusúak, akkor a megengedőbb (nagyobb tartományt lefedő) határozza meg a végeredmény pontosságát. Például, ha az egyik szám *float* (lebegőpontos) típusú, a másik *int* (egész), akkor a *float* (lebegőpontos) típusúként keletkezik az eredmény.

Az eredmény-változó típusának legalább azt a típust kell választani, amibe az eredmény belefér. Tudni kell ehhez, hogy a változó túlcsordulása okoz/okozhat-e hibás eredményt. Matematikai, pontos műveletkor a *float* típus definiálása javasolt. Viszont ennek ára a nagyobb programkód és a hosszabb időt igénybevevő számolás!

Fontos! Változótípusok közötti konverzió/adatátadás lehetséges, például ha *float* típusból *int*-be másolunk: *i (int) = 3.6 (float)* eredményeképpen: *i = 3*!

Komponens összevonás

Komponens összevonás (Compound assignments) nem más, mint hogy az aritmetikai műveletet és a változót írjuk le összevontan. Ezeket legtöbbször ún. hurkokban vagy ciklikus utasítássorokban használjuk. A leggyakoribb változataik:

```
x ++                    // ugyanaz:  x = x + 1
x --                    // ugyanaz:  x = x -  1
x += y                  // ugyanaz:  x = x + y
x -= y                  // ugyanaz:  x = x -  y
x *= y                  // ugyanaz:  x = x * y
x /= y                  // ugyanaz:  x = x / y
```

Fontos: Például x *= 3 nem jelent mást, minthogy az x értékét szorozza hárommal.

Összehasonlító operátorok

Egy változó és egy konstans összehasonlítása gyakran merül fel a programok futása során. A feltétel igaz vagy hamis volta esetén a programfutás más-más ágon folytatódik tovább. A feltétel többféle lehet, ebből néhány példa:

```
x == y                  // x egyenlő y
x != y                  // x nemegyenlő y
x < y                   // x kisebb, mint y
x > y                   // x nagyobb, mint y
x <= y                  // x kisebb vagy egyenlő, mint y
x >= y                  // x nagyobb vagy egyenlő, mint y
```

Logikai műveletek

Logikai műveletek segítségével általában két kifejezést hasonlíthatunk össze, melynek eredménye igaz (*true*) vagy hamis (*false*) lehet. Háromféle logikai műveletet használunk gyakran, ezek az *AND* (és), az *OR* (vagy) illetve a *NOT* (logikai nem). Ezeket gyakran az *if* utasítással használjuk.

AND (logikai és):

```
if (x > 0 && x < 5)     // igaz, ha mindkét feltétel igaz
```

OR (logikai vagy):

```
if (x > 0 || y > 0)     // igaz, ha valamelyik
                        // kifejezés igaz
```

NOT (logikai nem):

```
if (!x > 0)             // igaz, ha a kifejezés hamis
```

Konstansok

Az Arduino nyelv számos előre definiált jellemzőt tartalmaz, melyek konstans típusú "változók". Ezek a program írása során szabadon felhasználhatóak, gyárilag ún. csoportba vannak rendezve.

Igaz/Hamis (True/False)

Ezek a logikai konstansok logikai szinteket (is) jelentenek. Hamis (*False*) meghatározása egyezik a 0-val, míg az Igaz (*True*) gyakran 1-ként definiálódik. Azonban a logikai igaz bármely értéket felvehet a 0-t kivéve! Így a logikai konstans (Boolean) -1, 2 és akár a 200 mind *True*-nak definiálódik…

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

Magas/Alacsony (High/Low)

Ezek a konstansok a chip kivezetések (angolul: pin) állapotát jellemzik. Magas (*High*) a lábon közel a tápfeszültség mérhető, míg alacsony (*Low*) esetén GND. Ha a kivezetés kimeneti állapotát írjuk, akkor is a *High/Low* értéket használjuk, de amikor visszaolvassuk, hogy milyen állapota van, akkor is *High* illetve *Low* értéket kapunk vissza. A *High* logikai 1-ként definiálódik, a *Low* pedig logikai 0-ként.

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

Bemenet/Kimenet (Input/Output)

Konstansokat használva a *pinMode()* funkcióval definiálhatjuk, hogy egy chip kivezetés kimenet (*output*) vagy bemenet (*input*) legyen.

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

If utasítás

Az *If* utasítás segítségével ellenőrizhetünk 1-1 feltételt, azaz hogy a feltétel teljesül-e; például egy analóg bemenőjel értéke meghalad-e egy szintet. Ha a feltétel teljesül, akkor a feltételt követő kapcsos-zárójelben levő utasítások végrehajtódnak; ha pedig nem, akkor kihagyásra kerülnek. Az utasítás a következőképpen néz ki:

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

A fenti mintában az összehasonlításban leírt *someVariable*-t hasonlítjuk össze egy másik értékkel, mely lehet változó vagy akár konstans is.

Fontos! Óvakodjunk a feltételben a '=' használatától (pl. *if x=10 …*), amely az értékadást reprezentálja és így a feltétel mindig igaz lesz! A helyes írásmód a '==' használatát jelenti.

Folyamat kontrol

If... Else

Az *if...else* utasítás a feltételes elágazást testesíti meg. Az *else* utáni rész a feltétel nemteljesülése esetén hajtódik végre. Például: ha egy bemenet állapotát vizsgáljuk, akkor magas állapot esetén történik valami, de ugyanakkor alacsony állapotban végrehajtandó utasítást is megadhatjuk így. Írásmódjában a következőképpen néz ki:

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

Az *else* rész segítségével többszörös feltételt is lehet vizsgálni, egyetlen utasítássorban. Lehetőség van az utasítássorok végrehajtására, de ne felejtkezzünk el, hogy a nyitott zárójelet minden esetben be kell zárni a következő feltétel vizsgálata előtt! A halmozott feltétel-vizsgálat a következőképp írható le:

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

For

A *for* utasítás segítségével a kapcsos zárójelen belüli utasításokat egy számláló segítségével meghatározott számban, ismétlődően hajtjuk végre. A legtöbbször a ciklusszámláló növekszik és egy határt elérve lép ki a ciklusból. A *for* utasításnak három paramétere van, pontosvesszővel elválasztva:

```
for (initialization; condition; expression) {
   doSomething;
   }
```

Az *initialization*-két jelzett részben egy lokális változót definiálunk, mely értéke ciklus lefutásonként változik. Mindannyiszor, ahányszor a ciklus újra lefutna, a feltétel (itt *condition*-ként írva) ellenőrzésre kerül. Ha a kifejezés igaz, akkor ciklusmag (kapcsos-zárójelben levő utasításhalmaz) végrehajtódik, majd az ellenőrzés kezdődik újra. Amikor a feltétel hamissá válik, a körforgás véget ér.
A következő példában az egészként definiált ciklusszámláló (*i*) kiindulási értéke 0; a feltétel, hogy értéke kisebb, mint 20; és ha ez igaz, akkor az értéke nő 1-gyel és a ciklusmag újra végrehajtódik:

```
for (int i=0; i<20; i++)          // declares i, tests if less
   {                              // than 20, increments i by 1
        digitalWrite(13, HIGH);   // turns pin 13 on
        delay(250);               // pauses for 1/4 second
        digitalWrite(13, LOW);    // turns pin 13 off
        delay(250);               // pauses for 1/4 second
   }
```

Fontos! A C nyelvben a *for* ciklus igen flexibilis a többi nyelvhez képest, ideértve a Basic-ot is. A három paraméterből tetszőlegesen el lehet hagyni, de a pontosvesszőt ez esetben is fel kell tűntetni. A feltétel, a változó deklarálása, illetve ciklusváltozó számláltatása során bármely szabványos C kifejezést lehet használni. Ez a fokú rugalmasság - főleg ha eltérünk a megszokott szabványtól - azonban számos programhiba forrása lehet!

WHILE

A while-ciklus folyamatosan hajtódik végre, míg a zárójelben levő kifejezés hamissá nem válik. A vizsgált jellemzőnek változnia kell a ciklusmag futása alatt, különben soha nem lép tovább a program. Ez lehet akár egy növekvő változó, vagy külső feltétel - például egy érzékelő jele is.

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

A következő példa során a *'someVariable'* értéke, ha kisebb, mint 200 akkor a ciklusmagban levő utasítások végrehajtódnak, és ez ismétlődik amíg a *'someVariables'* értéke meg nem haladja a 200-at.

```
While (someVariable < 200)  // tests if less than 200
```

```
    {
        doSomething;          // executes enclosed statements
        someVariable++;       // increments variable by 1
    }
```

Do…While

A *do-while* ciklus hasonlít az előzőekben leírt *while* típushoz. A két megoldás közt a különbség, hogy a ciklusvége-feltétel ebben az esetben a ciklusmag után van, azaz a ciklusmag legalább egyszer mindenképp lefut!

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

A következő példa során a *readSensors()* visszaadott értékét az *x* változóba írjuk, majd 50 msec eltelte után megvizsgáljuk, hogy az *x* értéke kisebb-e, mint 100. Ekkor a ciklus újra lefut.

```
do {
        x = readSensors(); // assigns the value of
                           // readSensors() to x
        delay(50);         // pauses 50 milliseconds
   } while (x < 100);      // loops if x is less than 100
```

Ki- és bemenetek kezelése

pinMode(pin, mode)

A *void setup()* rutinban használjuk a *pinMode* utasítást, hogy a chip adott kivezetése be-/kimenet legyen.

```
pinMode(pin,   OUTPUT);     // sets 'pin' to output
```

A mikrokontroller digitális kivezetései alapesetben bemenetként vannak konfigurálva, nem szükséges külön ezt megtenni.

A bemeneteken chipen belül egy-egy kb. 20..80 kohm felhúzó-ellenállás van beépítve, melyek ki/bekapcsolhatóak. Ha a lábak bemenetre vannak állítva, akkor a lábra való kiírással kapcsolhatóak be a felhúzó-ellenállások:

```
pinMode(pin, INPUT);        // set 'pin' to input
digitalWrite(pin, HIGH);    // turn on pullup resistors
```

A felhúzó-ellenállások ki/bekapcsolása ezesetben úgy viselkedik, mintha egy kapcsolót be-/kikapcsolnánk. Fontos megjegyezni, hogy a kiírás nem változtatja meg a kivezetés adatirányát, csak a felhúzó-ellenállást kapcsolja ki/be!

Ha a kivezetés kimenetként (output) lett beállítva, akkor az alacsony impedanciás kivezetésként viselkedik. Ez azt jelenti, hogy kb. 40 mA árammal képes meghajtani az ide csatlakoztatott áramkört! Ez elegendő LED meghajtására (az áramkorlátozó soros ellenállásról ne feledkezzünk meg!), de kevés a legtöbb relé, motor vagy szolenoid meghajtására!

A rövidzár, túl nagy áram túlterheli és tönkreteszi az áramkör ezen részét, vagy akár az egész chipet is! A legtöbb esetben a kivezetések megóvása érdekében soros 220 ohm...1 kohm ellenállás beépítése javasolt.

digitalRead(pin)

Az utasítás segítségével a kivezetés logikai alacsony vagy magas állapotát olvashatjuk be. Az eredmény *High* vagy *Low* lehet. A kivezetés azonosítója 0...13 (Arduino Mega esetén: 0..53).

```
value = digitalRead(Pin);   // sets 'value' equal to
                            // the input pin
```

digitalWrite(pin, value)

A kimenetek logikai magas (*High*) vagy logikai alacsony (*Low*) állapotban lehetnek (azaz a kivezetést ki-/bekapcsolhatjuk). A kivezetés azonosítója 0..13 (Arduino Mega esetén: 0..53).

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

A következő példában egy nyomógomb állapotát olvashatjuk be, mely függvényében az egyik digitális kivezetésre kötött LED gyullad ki vagy alszik el.

```
int led = 13;          // connect LED to pin 13
int pin = 7;           // connect pushbutton to pin 7
int value = 0;         // variable to store the read value
void setup()
{
       pinMode(led, OUTPUT); // sets pin 13 as output
       pinMode(pin, INPUT);  // sets pin 7 as input
}
void loop() {
       value = digitalRead(pin); // sets 'value' equal to
                                 // the input pin
       digitalWrite(led, value); // sets 'led' to the
   }                             // button's value
```

analogRead(pin)

Az analóg kivezetéseken levő feszültséget 10-bites (0...1023 tartományt lefedő) felbontásban olvashatjuk be az *analogRead(pin)* utasítással. Ez a funkció csak az analóg kivezetéseken működik (0..5).

```
value = analogRead(pin);   // sets 'value' equal to 'pin'
```

Fontos! Az analóg kivezetések különböznek a digitális kivezetésektől, így - az Arduino projektben - nem használhatóak digitális kimenetként (más nyelveken pl. C, Bascom-AVR stb. van csak erre lehetőség).

analogWrite(pin, value)

Pszeudo-analóg jelet ún. PWM (impulzusszélesség moduláció) segítségével az Arduino egyes digitális kivezetéseire lehetséges kiküldeni. Az újabb chippel (ATMega168) felszerelt panelen ez a funkció a 3, 5, 6, 9, 10 és 11 jelű digitális kivezetéseken érhető el. Az ATMega8 chippel szerelt Arduino esetén ez csak a 9, 10

és 11 jelű digitális lábakon érhető el. A PWM értékmegadása történhet konstans vagy változó segítségével is, ezek értéke 0..255 közt lehet.

```
analogWrite(pin, value); // writes 'value' to analog 'pin'
```

Ha a PWM éréke 0, akkor a kimeneten 0 V feszültség mérhető, míg 255 esetén 5V. A szélső értékek közötti érték és a 0-5V feszültség közt egyenes arányosság van. A nagyobb érték esetén azonos idő alatt hosszabb ideig magas a kivezetés szintje. Például 64-es érték esetén az időszelet 1/4-ében alacsony, 3/4-ében magas értéket mérhetünk (átlagolt feszültségszint kb. 1.25V), míg 128 esetén az idő felében alacsony, másik felében magas szintet mérhetünk (az átlag feszültségszint 2.5V).
Ezt a funkciót a chip hardveresen biztosítja, így a kimeneten a négyszögjelet automatikusan generálja. Az *analogWrite* funkció hatása a kikapcsolásig illetve a következő *analogWrite()* utasításig megmarad.

A következő mintaprogramban beolvassuk az analóg lábon levő jelet és a kapott érték 1/4-vel hajtjuk meg a PWM lábat:

```
int led = 10;       // LED with 220 resistor on pin 10
int pin = 0;        // potentiometer on analog pin 0
int value;          // value for reading
void setup(){}      // no setup needed
void loop()
{
  value = analogRead(pin); // sets 'value' equal to 'pin'
  value /= 4;              // converts 0-1023 to 0-255
  analogWrite(led, value); // outputs PWM signal to led
}
```

Egyéb utasítások

Delay(ms)

A *delay()* utasítás segítségével a programban megadott *ms* milisec-nyi várakozást lehet beiktatni. Ha 1000 ms-t írunk, az 1 sec várakozást jelent.

```
delay(1000);  // waits for one second
```

Millis()

A *milis()* utasítással kérdezhetjük le a bekapcsolás óta eltelt időt msec egységben.

```
value = millis();    // sets 'value' equal to millis()
```

Fontos! Az eltelt idő számlálója kb. 9 óránként túlcsordul és nullázódik!

Min(x,y)

Két számérték közül a kisebb értéket adja vissza (független az adattípustól).

```
value = min(value, 100); // sets 'value' to the smaller of
                         // 'value' or 100, ensuring that
                         // it never gets above 100.
```

Max(x, y)

Két számérték közül a nagyobbat adja eredményül. Működése hasonló a *Min()* függvényhez.

```
value = max(value, 100); // sets 'value' to the larger of
                         // 'value' or 100, ensuring that
                         // it is at least 100.
```

randomSeed(seed)

Kezdőérték beállítása, illetve kezdőpont a *random()* függvény működéséhez.

```
randomSeed(value);  // sets 'value' as the random seed
```

Az Arduino nyelvben (és a legtöbb mikrokontrolleres alkalmazásban) nem lehetséges valódi véletlen számot létrehozni. A kezdőérték segít még véletlenebbé beállítani a

véletlen szám generálást. Erre lehetőség van a *milis()* számérték vagy az analóg bemeneten levő érték az *analogRead()* funkcióval való beolvasásával.

RANDOM(MAX) RANDOM(MIN, MAX)

A *random()* funkcióhívás segítségével egy véletlenszerű értéket kapunk vissza a megadott min és max értékek között.

```
value = random(100, 200);  // sets 'value' to a  random
                           // number between 100-200
```

Fontos! Ezt a funkciót a *randomSeed()* utasítás után használhatjuk csak.

A következő mintaprogram véletlen számot hoz létre 0...255 közt, és ezt a PWM kimenetre küldi.

```
int  randNumber;    // variable to store the  random value
int led = 10;       // LED with 220  resistor on pin  10
void setup()  {}    // no setup needed
void loop()
{
  randomSeed(millis());          // sets millis() as seed
  randNumber = random(255);      // random number from 0-255
  analogWrite(led, randNumber); // outputs PWM signal
  delay(500);                    // pauses for half a second
}
```

SERIAL.BEGIN(RATE)

A soros kommunikáció megkezdésekor a kommunikációs sebességet deklarálni kell. A legelterjedtebb a 9600 bps, de a soros kommunikáció 150...115.200 bps (sorosport illetve RS-485) illetve 300..960.000 bps (USB sorosport) lehet.

```
void setup() {
  Serial.begin(9600); // opens serial port
}                     // sets data rate to 9600 bps
```

Fontos! Amikor a soros kommunikáció megkezdődik, a 0 (Rx) illetve 1 (Tx) digitális kivezetések ki-/bemenetként nem, csak soros kommunikációs csatornaként használhatóak!

Serial.println(data)

A *Serial.println(data)* utasítás segítségévek a *data* adatsort a sorosporton kiküldjük. Az adatsor végét soremelés+kocsivissza jel (kb. *Enter*) zárja. Az utasítás működése megegyezik a serial.print() utasítással, csak terminálprogram segítségével az előbbi könyebben olvasható.

```
Serial.println(analogValue); // sends the value of
                             // 'analogValue'
```

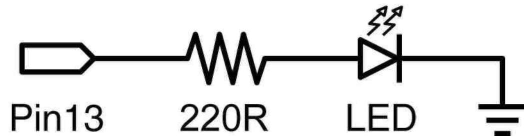
Fontos! A *serial.print()* és a *serial.println()* utasításnak számos paramétere létezik még, mely ismertetése meghaladja a könyv kereteit. Ezt részletesen a *http://www.arduino.cc* honlapon lehet megtalálni.

A következő egyszerű példa az egyik analóg bemenet értékeit olvassa és másodpercenként küldi a sorosporton át a PC felé.

```
void setup() {
  Serial.begin(9600);            // sets serial to 9600bps
}
void loop()
{
Serial.println(analogRead(0));   // sends analog value
delay(1000);                     // pauses for 1 second
}
```

Mellékletek

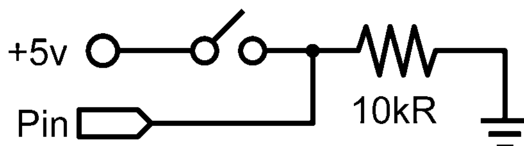
Digitális kimenet



A mikroelektronikában az, ami bármely programozási nyelvben a "Hello world!" program - az itt egy kivezetésre kötött LED kigyújtása. A legegyszerűbb az Arduino 13-as digitális kivezetésére kötött LED kihasználása. A mintaprogramban ennek egy másodperces időközű felgyújtását valósítjuk meg. Az ellenállást mindenképp építsük be, ha LED-et kötünk be (A 13-as digitális kivezetésre gyárilag bekötött LED előtt az ellenállás beépítésre került). Az ellenállás értéke 5V tápfeszültség esetén 220..680 ohm közé essen.

```
int ledPin = 13;                  // LED on digital pin 13
void setup()                      // run once
{
       pinMode(ledPin, OUTPUT);   // sets pin 13 as output
}
void loop()                       // run over and over again
{
  digitalWrite(ledPin,HIGH);      // turns the LED on
  delay(1000);                    // pauses for 1 second
  digitalWrite(ledPin,LOW);       // turns the LED off
  delay(1000);                    // pauses for 1 second
}
```

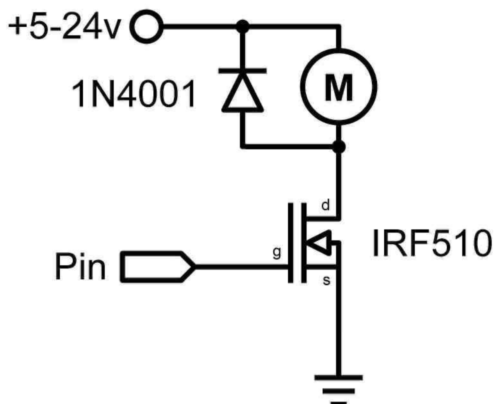
Digitális bemenet



Ez a legegyszerűbb formája a digitális bemeneten levő jel érzékelésének. A kapcsoló nyitott illetve zárt állását a segítségével beolvashatjuk. A példában egyszerű kapcsoló vagy nyomógomb kerül a 2-es kivezetésre. Ha a kapcsoló zárt állású, akkor a kimeneten (13-as kivezetés) levő LED kigyullad.

```
int ledPin = 13;                      // output pin for the LED
int inPin = 2;                        // input pin (for a switch)
void setup()
{
  pinMode(ledPin, OUTPUT);            // declare LED as output
  pinMode(inPin, INPUT);              // declare switch as input
}
void loop() {
  if (digitalRead(inPin) == HIGH) // check if input is HIGH
  {
  digitalWrite(ledPin, HIGH);          // turns the LED on
  delay(1000);                         // pause for 1 second
  digitalWrite(ledPin, LOW);           // turns the LED off
  delay(1000);                         // pause for 1 second
  }
}
```

Digitális kimenet



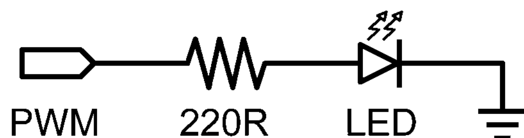
Néhány esetben szükség lehet a kivezetés által biztosított 40 mA-t meghaladó áram biztosítására. Ekkor a legegyszerűbb tranzisztorral, vagy MOSFET segítségével kapcsolni a nagyobb áramot. A következő mintában a MOSFET segítségével 5 sec időnként be- majd kikapcsolunk egy motort.

Fontos! Az ábrán a motorral párhuzamosan egy védődióda található. Ám ha a terhelés nem induktív (például izzólámpa), akkor elhagyható.

```
int outPin = 5;                    // output pin for the MOSFET
void setup() {
pinMode(outPin, OUTPUT);           // sets pin5 as output
}
void loop() {
```

```
  for (int i=0; i<=5; i++)      // loops 5 times
   {
    digitalWrite(outPin, HIGH);  // turns MOSFET on
    delay(250);                  // pauses 1/4 second
    digitalWrite(outPin, LOW);   // turns MOSFET off
    delay(250);                  // pauses 1/4 second
   }
  delay(1000);                   // pauses 1 second
}
```

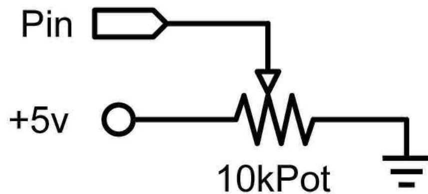
PWM KIMENET



Az *I*mpulzus *S*zélesség *M*oduláció (*P*ulse*W*idth *M*odulation (PWM)) az egyik (legegyszerűbb) útja egy ál-analóg jel előállításának. Ekkor a kimeneten pulzáló egyenfeszültség jelenik meg, melyet az ezt követő elektronika folyamatos analóg jelként érzékel. Ezt használhatjuk például LED fényerejének szabályozására, vagy később szervomotor vezérlésére is. A következő mintaalkalmazás során a LED fénye lassan növekszik, majd csökken. Majd ez ismétlődik folyamatosan.

```
int ledPin = 9;                 // PWM pin for the LED
void setup(){}                  // no setup needed
void loop() {
  for (int i=0; i<=255; i++)    // ascending value for i
   {
     analogWrite(ledPin, i);    // sets brightess level to i
     delay(100);                // pauses for 100ms
   }
for (int i=255; i>=0; i--)      // descending value for i
   {
     analogWrite(ledPin, i);    // sets brightess level to i
     delay(100);                // pauses for 100ms
   }
 }
```

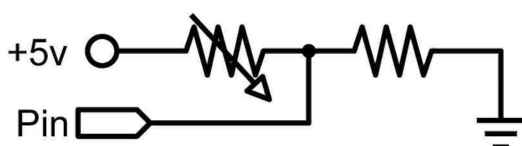
Potenciométer bemenet (Analóg bemenet)



Potenciométert használva az Arduino rendszerben - ez gyakorlatilag az ATMega chip analog-digital bemenetére csatlakozik - a változtatható ellenállás által leosztott feszültséget kapjuk meg a bemeneten. A beolvasott érték 0..1023 közé esik. A következő mintában a potméterrel a LED villogási frekvenciáját szabályozhatjuk.

```
int potPin = 0;      // input pin for the potentiometer int
ledPin = 13;                     // output pin for the LED
void setup() {
    pinMode(ledPin, OUTPUT);     // declare ledPin as OUTPUT
 }
void loop()
{
    digitalWrite(ledPin, HIGH);  // turns ledPin on
    delay(analogRead(potPin));   // pause program
    digitalWrite(ledPin, LOW);   // turns ledPin off
    delay(analogRead(potPin));   // pause program
}
```

Változtatható ellenállás a bemeneten



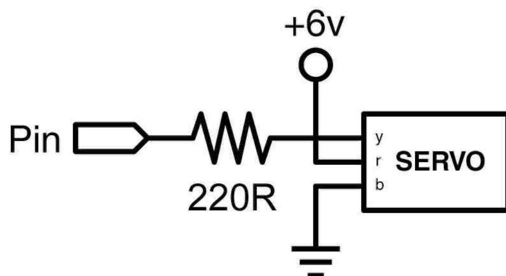
Változtatható ellenállás - például a CdS alapú fotoellenállás, termisztor, nyúlásmérő bélyeg, stb. - értéke megmérhető az analóg bemenet segítségével. A példában az analóg értéket beolvasva várakozási időt szabályozunk vele. Ez a LED kivilágosodása és elsötétedése közötti időt befolyásolja.

```
int ledPin   = 9;         // PWM pin for the LED
int analogPin = 0;        // variable resistor on analog pin 0
void setup(){}            // no setup needed
void loop() {
  for (int i=0; i<=255; i++)      // ascending value for i
    {
```

```
        analogWrite(ledPin, i);     // sets brightess level to i
        delay(delayVaK));           // gets time value and pause
      }
    for (int i=255; i>=0; i--)      // descending value for i
      {
        analogWrite(ledPin, i);     // sets brightess level to i
        delay(delayVaK));           // gets time value and pause
      }
    }

  int (delayVaK)
    {
      int v;                        // create temporary variable
      v = analogRead(analogPin);    // read analog value
      v /= 8;                       // convert 0-1024 to 0-128
      return v;                     // returns final value
  }
```

Szervomotor

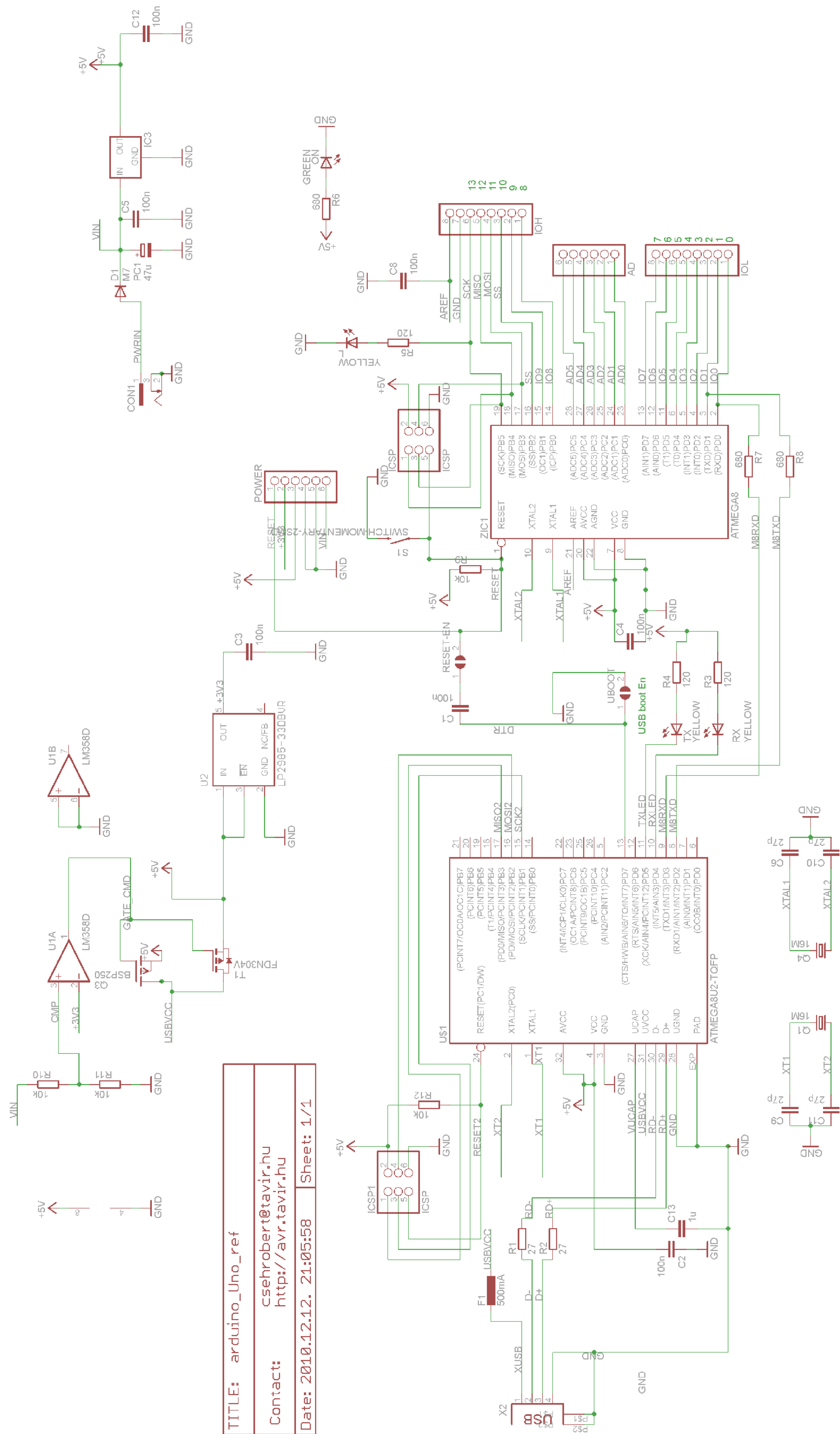


A hobbi szervomotorok önálló, beépített meghajtó-egységgel rendelkeznek, mely segítségével 180 fokos elfordulásban lehet a kitérést szabályozni. Ehhez 20 msec-enkénti impuzus kiadása szükséges. A mintaprogramban 10…170 fok elfordulást valósítunk meg, majd ugyanennek ellenkező irányát is végigjárja a motor.

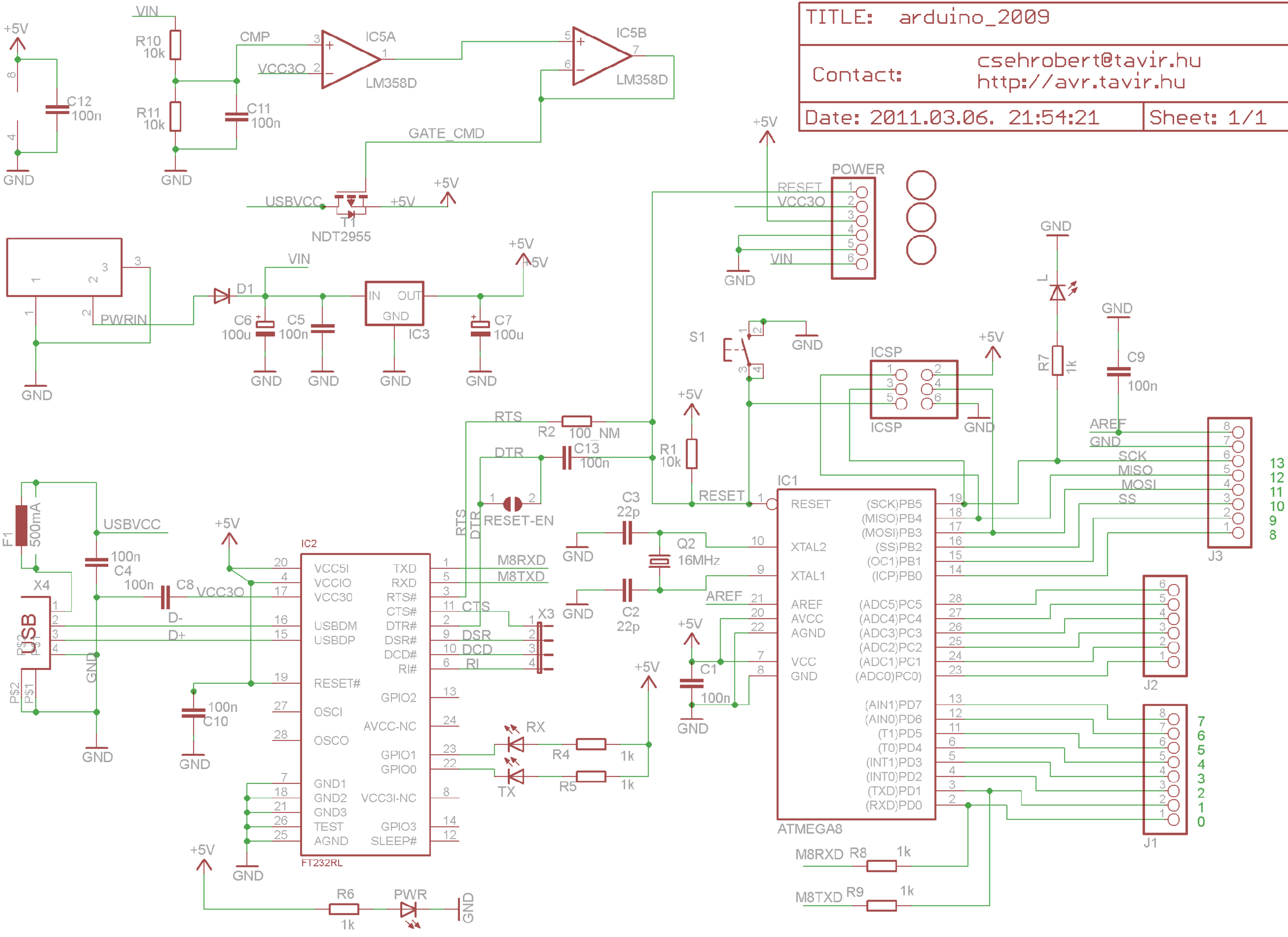
```
  int servoPin =  2; // servo connected to digital pin 2
  int myAngle; // angle of the servo roughly 0-180
  int pulseWidth;   // servoPulse function variable
  void setup() {
     pinMode(servoPin, OUTPUT);          // sets pin 2 as output
   }
  void servoPulse(int servoPin, int myAngle) {
     pulseWidth = (myAngle * 10) + 600; //determines delay
     digitalWrite(servoPin, HIGH);      //set servo high
     delayMicroseconds(pulseWidth);     //microsecond pause
     digitalWrite(servoPin, LOW);       //set servo low
   }
  void loop() {
         // servo starts at 10 deg and rotates to 170 deg
      for (myAngle=10; myAngle<=170; myAngle++)
        {
```

```
        servoPulse(servoPin, myAngle); // send pin and angle
        delay(20);                     // refresh cycle
       }
        // servo starts at 170 deg and rotates to 10 deg
    for (myAngle=170; myAngle>=10; myAngle--)
       {
         servoPulse(servoPin, myAngle);// send pin and angle
         delay(20);                    // refresh cycle
   }
}
```

AVR-Duino Uno kapcsolási rajz

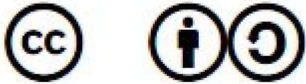


AVR-Duino Duemilanove kapcsolási rajz



CC BY-SA 2.5 LICENSZ

Nevezd meg! - Így add tovább! 2.5 Általános (CC BY-SA 2.5) licensz kivonat



A következőket teheted a művel:

Szabadon másolhatod, terjesztheted, bemutathatod és előadhatod a művet

Származékos műveket (feldolgozásokat) hozhatsz létre

Az alábbi feltételekkel:

Nevezd meg! - A szerző, fordító vagy egyéb rendelkező jogosult által meghatározott módon fel kell tüntetned a műhöz kapcsolódó információkat (pl. a szerző, fordító, átdolgozó, illusztrátor nevét vagy álnevét, a Mű címét).

Így add tovább! - Ha megváltoztatod, átalakítod, feldolgozod ezt a művet, az így létrejött alkotást csak a jelenlegivel megegyező licenc alatt terjesztheted.

Az alábbiak figyelembevételével:

Elengedés: A szerzői jogok tulajdonosának írásbeli előzetes engedélyével bármelyik fenti feltételtől eltérhetsz.

Közkincs: Ha a munkában bármely elemére a public domain jogkör vonatkozik, ezt a státuszt - a public domain jogállású részre - a jelen jogi konstrukció nem bírálja felül.

Más jogok: A következő, és egyéb jogokat a licenc semmiben nem befolyásolja:

A szerző személyhez fűződő jogai

Más személyeknek a művet vagy a mű használatát érintő jogai, mint például a személyiségi jogok vagy az adatvédelmi jogok.

Jelzés: Bármilyen felhasználás vagy terjesztés esetén egyértelműen jelezned kell mások felé ezen mű licencfeltételeit.

A licensz részletes feltételei megtekinthetőek:

http://creativecommons.org/licenses/by-sa/2.5/ oldalon.