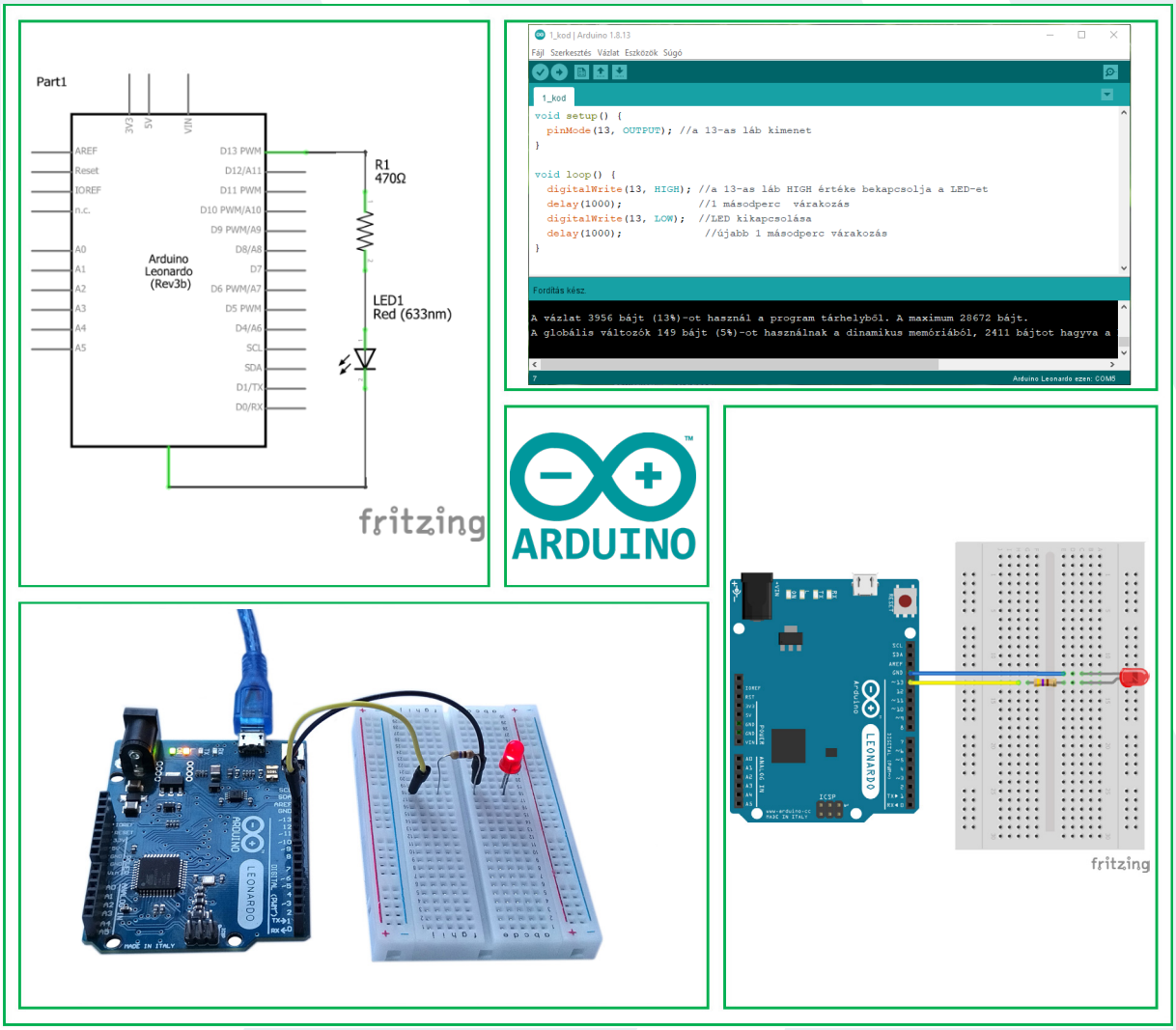
MAGYAR AGRÁR- ÉS ÉLETTUDOMÁNYI EGYETEM

Barna Róbert – Honfi Vid

KÜTYÜZNI JÓ!

Bevezetés az Arduino programozásába



2022

Kütyüzni jó!

Bevezetés az Arduino programozásába

Barna Róbert – Honfi Vid

Kütyüzni jó!

Bevezetés az Arduino programozásába

Magyar Agrár- és Élettudományi Egyetem Kaposvári Campus
Kaposvár, 2022

Szerzők

Dr. Barna Róbert, PhD (Magyar Agrár- és Élettudományi Egyetem)
Dr. Honfi Vid, PhD (Milton Friedman Egyetem)

Lektorok

Dr. Kovács Árpád Endre, PhD
Rumbus Anikó

Kiadja

Magyar Agrár- és Élettudományi Egyetem Kaposvári Campus
7400 Kaposvár, Guba Sándor u. 40.
Tel.: +36-82-505-800, +36-82-505-900
Fax: +36-82-505-896
e-mail: barna.robert@uni-mate.hu.hu

Felelős Kiadó
Vörös Péter, Campus-főigazgató

Korrektor
G. Szabó Sára

ISBN 978-615-5599-93-4 (pdf)

Tartalom

Bevezetés

Könyvünk címe saját tapasztalaton alapul: kütyüzni jó! Gyerekként különösen jólesik alkotni valamit, aminek az eredménye kézzel fogható. Ha jól sikerült, büszke rá az alkotó, a család, az iskola.

Ebben a könyvben az elektronika iránt érdeklődőknek szeretnénk bemutatni az Arduinoval készülő varázslatot, amelynek során az addig holt anyag egyszer csak működni kezd, üzenetet küld, villog, csattog. Lépésről lépésre vezetjük be az olvasót a hardver és „szoftverépítés" világába.

A könyv anyagát a *Digisuli*[1] nyári táborainak gyakorlatai alapján állítottuk össze. A táborban általános iskolai felsős fiatalok vettek részt. Az általunk bemutatott, kevés elemből álló kapcsolások egyszerűen megépíthetők próbapanelen. A hardver nem drága, beszerzése egyszerű. A kapcsolások nem igényelnek előzetes ismereteket, ahol mégis szükséges, ott röviden összefoglaltuk a tudnivalókat. A könyvben szereplő programok az egyszerűtől haladnak a bonyolultabbak felé, a kevés utasítástól a hosszabb kódok felé. A programozási környezet ingyenesen telepíthető, magyar nyelvű menüvel rendelkezik.

Az Arduino-t a hozzá elkészített fejlesztői környezetben a C nyelvre épülő kódok írásával lehet programozni. A későbbiekben tervezzük a könyvben található kódok blokk nyelven történő bemutatását is.

Az interneten rengeteg oldal foglalkozik az Arduino-val, főként angol nyelven, de a magyar oldalak száma is több tízezer. Az Arduino felhasználása nagyon sokrétű, a mérés-adatrögzítéstől a GPS-vevőn át a robotvezérlésig nagyon sok kapcsolás építhető vele. Számtalan lelkes építő osztja meg az interneten az általa készített projektek leírását, a kapcsolástól a programkódig. Az Arduino-hoz és a hozzá kapcsolódó eszközökhöz ingyenesen elérhető könyvtárakat készítettek. A bennük található függvények, példaprogramok segítik a felhasználókat az eszközök megismerésében, használatában.

Könyvünket ajánljuk a programozás bevezető lépéseinek oktatásához. A megírt kód működését a felépített hardveren mindjárt le lehet próbálni. Az azonnali sikerélmény miatt az ismeretek elsajátításának mélysége is nagyobb lesz.

Könyvünket ajánljuk továbbá mindenkinek, akit érdekel az elektronika, a programozás, de még soha nem mert belevágni. Családi programnak sem rossz, a hosszú téli estéken a szülők és gyerekek együtt próbálhatnak ki új kapcsolásokat, így az eredmény örömében is osztozhatnak. Sok esetben a próbálkozásokból hiánypótló megoldások születnek.

Szóval tessék belevágni, mert kütyüzni jó!

[1] *URL: https://www.programoznifogok.hu/*

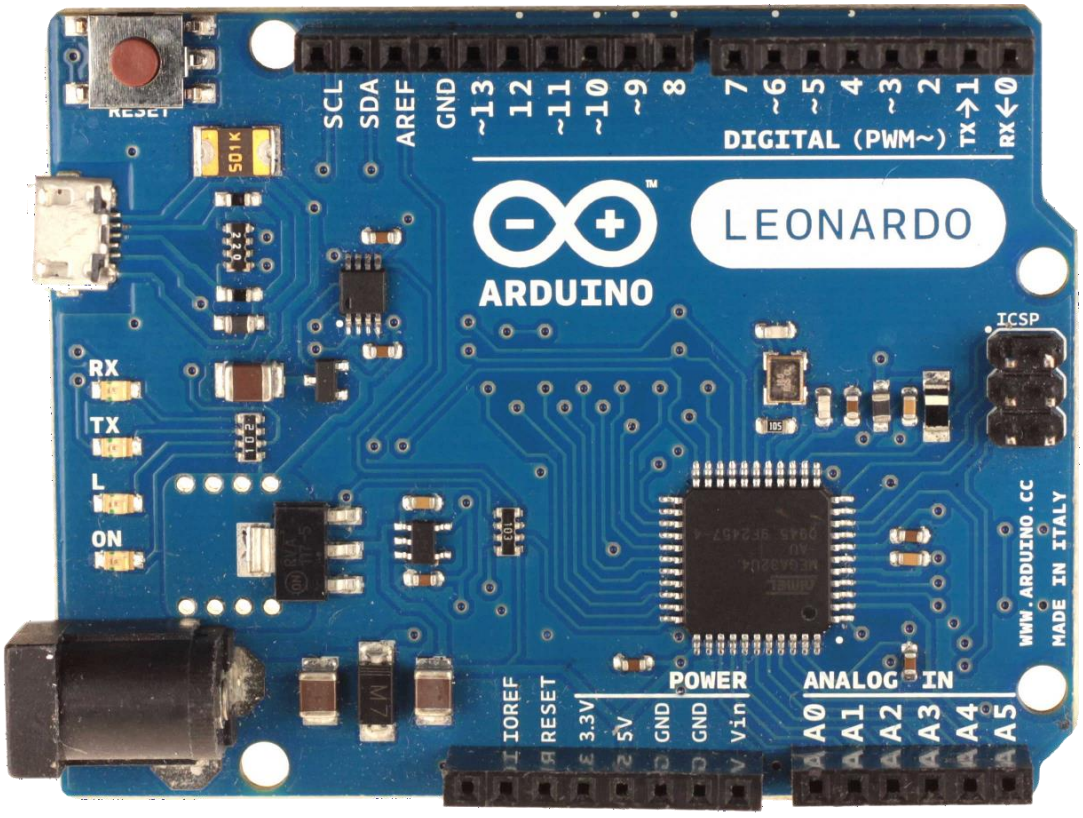
Mi is az Arduino?

Az *Arduino* egy *mikrovezérlő* (mikrokontroller) áramkör, egy MCU (*microcontroller unit*). A mikrovezérlőnek bemenetei és kimenetei vannak. A bemenetek a külvilágból érkező jeleket is fogadják, az Arduino-tól a környezet felé küldött jelek pedig a kimenetek. Az embernek is vannak ki- és bemenetei, a szemünk és a fülünk lehet például bemenet, amelyek érzékelik a környezetünk változásait, valamit a hangunk lehet kimenet, amivel hatunk a környezetre.

A mikrovezérlő – a neve ellenére – nagyon sokféle érzékelőt (*szenzort*) képes használni, és nagyon sokféle eszközt lehet vele vezérelni. A szenzorok például hőmérsékletet, fényerőt, távolságot képesek érzékelni. A szenzorok által mért értéket az Arduino-val beolvashatjuk, majd megjeleníthetjük kijelzőn, a beolvasott értéktől függően hangjelzést adhatunk, motort kapcsolhatunk be stb.

Több fajta Arduino-áramkör létezik, a könyvben található kapcsolások az Arduino *Leonardo-áramkörre* készültek (*1. ábra*). Az áramkört *alaplapnak, lapkának* is nevezik. A Leonardo-ra írt programok legtöbbje átalakítás nélkül más Arduino-családba tartozó lapkán (Uno, Mega stb.) is működik.

1. ábra: Arduino Leonardo panel



Forrás: https://caxtool.hu/custom/mysticnails/image/cache/w1000h1000wt1/product/alkatrészek1/AA151.jpg

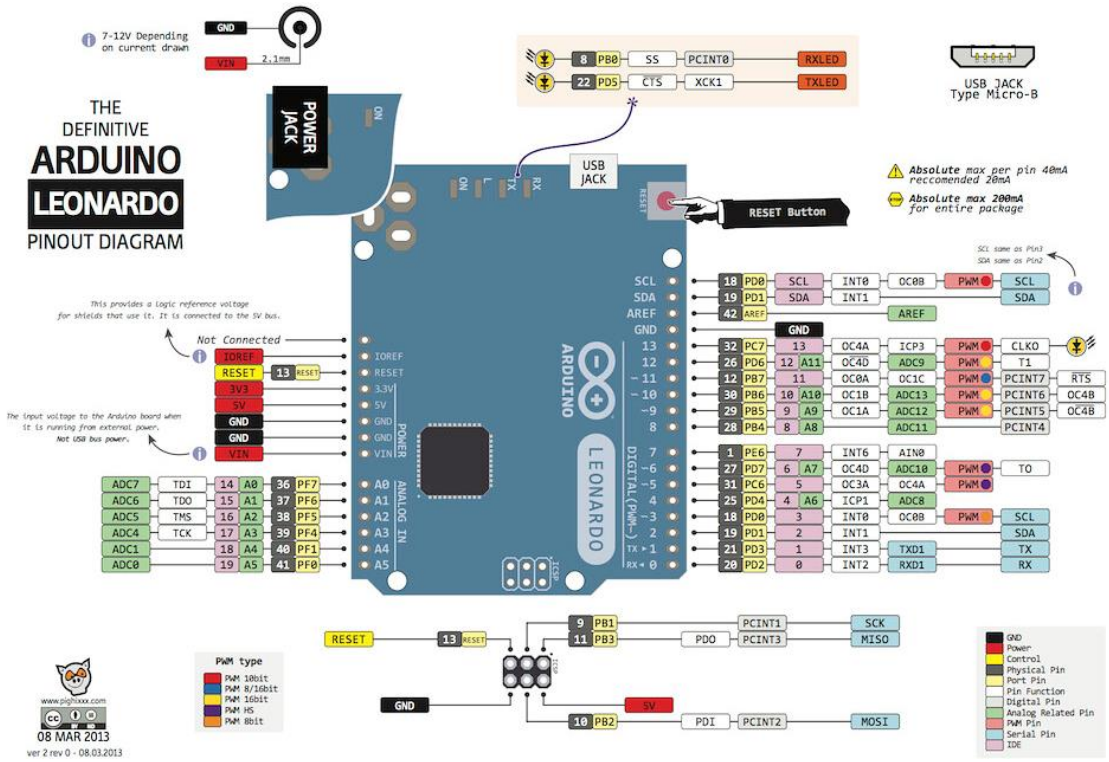
Az Arduino-t USB kábelen keresztül tudjuk számítógéphez kapcsolni. Ez a kapcsolat egyrészt biztosítja az alaplap tápfeszültség-ellátását, másrészt a soros vonali kommunikációt, amivel például a megfelelő működési programot lehet áttölteni (*2. ábra*).

2. ábra: Arduino–áramkör kapcsolása számítógéphez

A *3. ábrán* látható, hogy az áramkör az USB és a tápegység-csatlakozón kívül további kivezetésekkel (lábbal) rendelkezik, ezeket angolul pin-nek (tűnek) nevezik. A kivezetések négy csoportba oszthatók: analóg (A0-A5), digitális (0-13), tápfeszültség (3,3 V, 5V, GND, VIN) és a „maradék": a vezérlő pinek. Az egyes lábak tulajdonságait a használatkor részletezzük. A bekapcsolást a lapkán ON felirattal jelölt zöld színű LED jelzi. Az alaplapon ezenkívül még három sárga fényű LED található, kettő a soros vonali kommunikációt jelző (RX, TX) LED és egy L betűvel jelölt beépített LED, amiről később bővebben szó lesz.

Figyeljük meg, hogy a kivezetések számozása nem 1-gyel, hanem 0-val kezdődik! Ez a programjainkban is így lesz.

3. ábra: Az Arduino Leonardo kivezetései

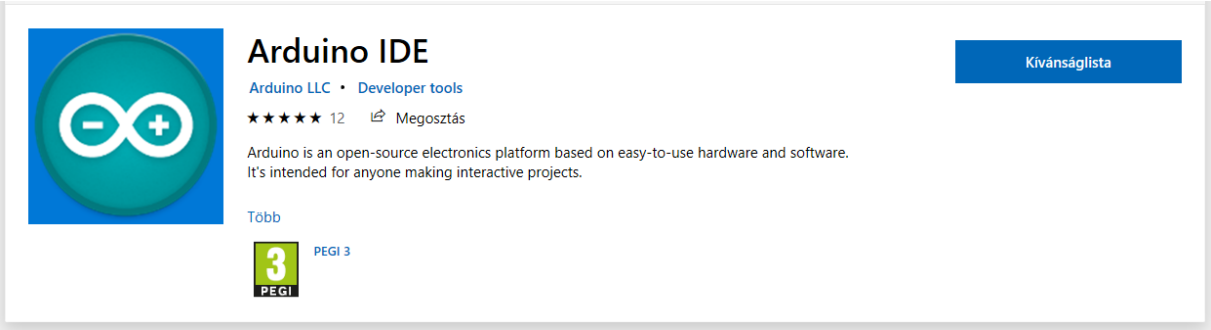


Forrás: *https://duino4projects.com/wp-content/uploads/2013/04/Ardunio_leonardo_pinout.jpg*

Az Arduino fejlesztési környezete

Az Arduino-hoz tartozik egy programozási környezet is, ahol a mikrovezérlő (*hardware*) működtetéséhez szükséges programot (*software*) lehet megírni. Az Arduino IDE egy nyílt forráskódú integrált fejlesztői környezet (IDE: *Integrated Development Environment*), ami díjmentesen letölthető a fejlesztői weboldalról (https://www.arduino.cc/) vagy például a Microsoft Store-ból (*4. ábra*) is, így akár okoseszközzel is programozhatjuk. Az Áruházban ez olvasható: *„Az Arduino egy könnyen használható hardver és szoftver alapú nyílt forráskódú elektronikai platform. Olyan felhasználók számára készült, akik interaktív projekteket készítenek.”*

4. ábra: Az Arduino IDE programozói alkalmazás letöltése

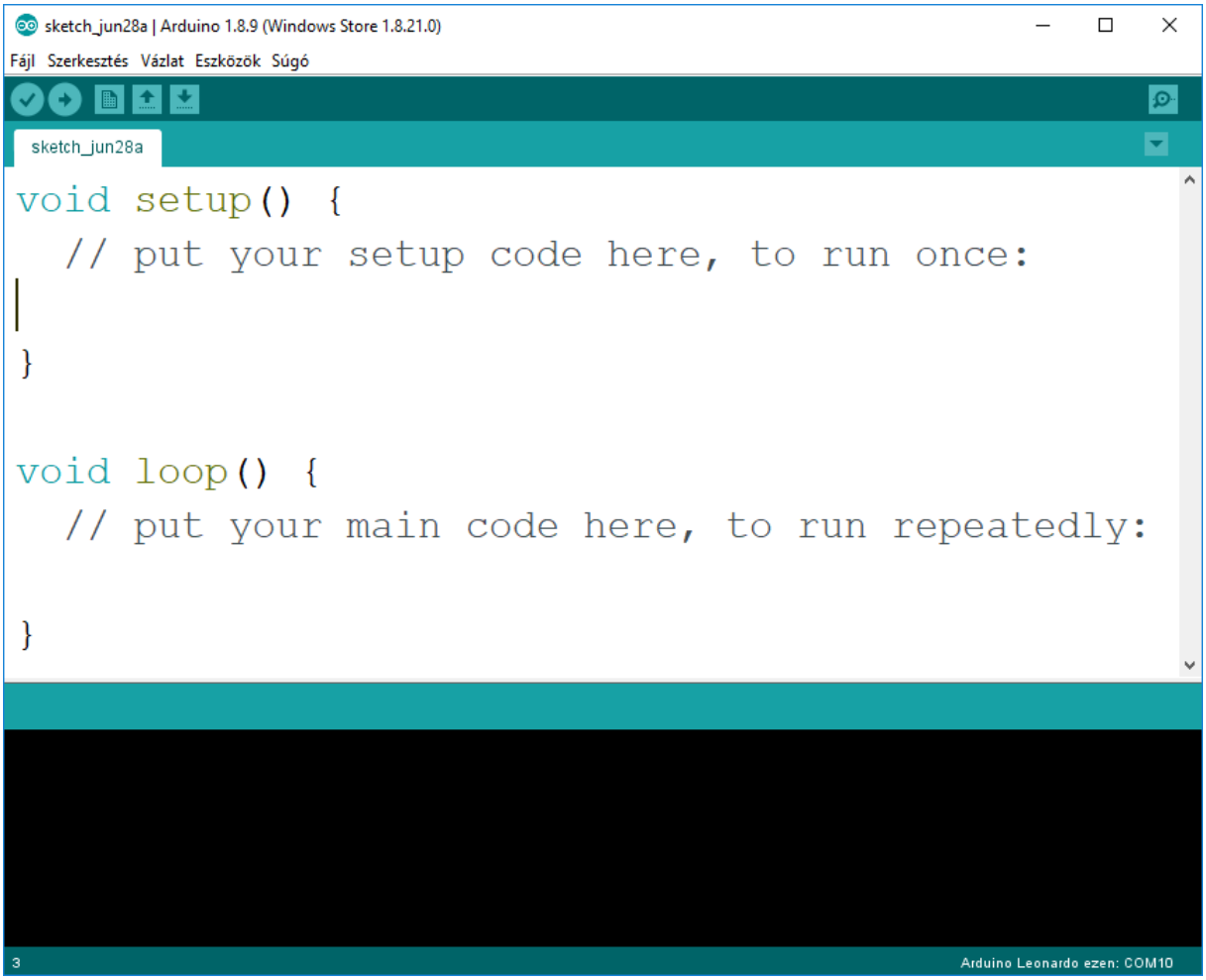


Az Arduino-hoz használható blokk nyelv is, az ArduBlock. Az ArduBlock is díjmentesen *letölthető a* https://sourceforge.net/projects/ardublock/files/latest/download oldalról.

Hasznos kiegészítő a *Fritzing szoftver*, amivel megtervezhetők az Arduino-áramkörök. A könyvben előforduló kapcsolási rajzokat a 0.9.3b verziójú programmal rajzoltuk. A Fritzing bárki számára elérhető, nyílt forráskódú hardvertervező szoftver, amely támogatja az Arduino-eszközöket és a hozzá csatlakoztatható szenzorokat. Annak ellenére, hogy ez a szoftver is ingyenesen használható sokrétű funkciókat biztosít: lehetővé teszi a dokumentációt, megosztást másokkal, elektronikaoktatást és professzionális nyomtatott áramkör tervezését. Bővebb leírás a http://fritzing.org/home/ oldalon található, a program letöltése a http://fritzing.org/download/ oldalról történhet. A Fritzing használata nem témája ennek a könyvnek.

Az Arduino IDE letöltése és telepítése után elindítva megjelenik az *5. ábrán* látható kezelőfelület. Az applikációban írt programokat skicceknek (*sketch*) hívjuk. Egy új skicc alapértelmezett neve: `sketch_hónapnév-nap` és még egy betű, ami azt jelzi, hogy az adott napon hányadik skiccet készítjük. Az elsőt a betű jelöli, majd sorban az angol abc betűi következnek. A **`Fájl`** menüben az **`Új`** menüpontra kattintva létrejön az előbbi szabály szerint elnevezett mappa és abban egy ugyanilyen nevű fájl, aminek a kiterjesztése „`.ino`”. A skicc nevét természetesen módosíthatjuk. A menüpontokat most nem részletezzük, amikor szükséges a használatuk, akkor ismertetjük majd azokat.

5. ábra: Az Arduino API kezelőfelülete



Amint, arról már szó volt, az Arduino-alaplapot USB kábelen tudjuk a számítógéphez kapcsolni, és ott valamelyik soros porton keresztül tudunk vele kommunikálni. A *3. ábra* jobb alsó részén látszik, hogy az Arduino Leonardo vezérlő a COM10 portra csatlakozik.

Az Arduino programozása

Az Arduino programozási nyelve a C nyelvre épül, egyfajta *C++ megvalósítás*. A tanulás során ezáltal a C++ nyelv alapszintű elsajátítása szükséges lesz a példák megoldásához. A programozáshoz a *Blokk programozási nyelv* szintén használható lenne. Azért választottuk itt a hagyományos (parancsokból álló) programírást, mert ez megalapozhatja a későbbi komolyabb rendszerek programozását is. A megírt kódot az applikáció a használt Arduino-lapkának megfelelő gépi kódra fordítja le, és soros vonalon tölti fel azt az áramkörre.

Minden új skiccben megtalálható az *5. ábrán* látható néhány előre beírt sor. A `setup()` és a `loop()` az Arduino-áramkörök működésének alapfunkciói.

A `setup()` (beállítás) a program indulásakor beállítja, inicializálja az áramkör kivezetéseinek működési módját, a soros port tulajdonságait. A `setup()` a bekapcsolás után egyszer fut le.

A `loop()` (hurok) funkció a beállítást követően fut le, többször egymás után addig, amíg az áramkört ki nem kapcsoljuk. Ez a végrehajtó rész, érzékel és beavatkozik, ha szükséges.

Vegyünk egy példát: egy locsoló automatát! Először be kell állítani a `setup()` funkcióval, hogy melyik lábon érzékeli az Arduino a talajnedvességet, és melyik lábon kapcsolja be a locsolót. Ezután fut a `loop()` funkció, amelyik megméri a nedvességet, és ha kell, locsol. A mérés, locsolás addig folytatódik, amíg az áramkör bekapcsolt állapotban van.

A `setup()` és a `loop()` elnevezése C környezetben: függvény (function). A `void` (üres) kulcsszó a függvények neve előtt azt jelenti, hogy azok nem szolgáltatnak, nem adnak vissza eredményt. A későbbiekben erről még lesz szó.

Ezt a két függvényt tehát minden Arduino-programnak tartalmaznia kell.

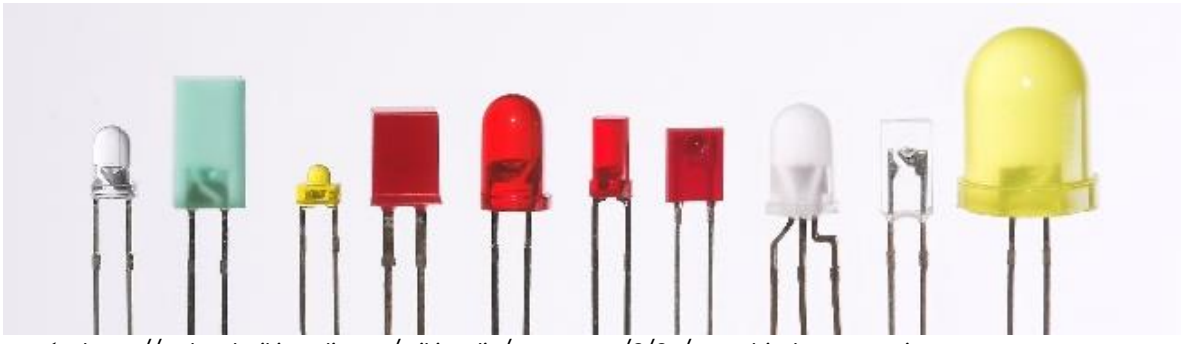
A függvények neve után, amint láttuk, zárójeleket `'()'` kell írnunk. Ezek után egy nyitó kapcsos zárójellel `'{'` kezdődik a függvény, és csukó kapcsos zárójellel `'}'` fejeződik be. A kapcsos zárójelek közé írhatók a programsorok. Az *5. ábrán* látható program nem csinál semmit, a benne látható sorok csak megjegyzések, amelyek az előbb elmondottakat tartalmazzák röviden. A megjegyzéseket kettős perjel `'//'` után írhatjuk, ezeket a fordító figyelmen kívül hagyja. Jelentőségük abban van, hogy egy program olvashatóságát lehet velük javítani. Amikor egy programot írunk, tudjuk, hogy melyik sort miért írtuk úgy, ahogy. Amikor azonban azt később újra elő kell venni, már nem biztos, hogy emlékszünk minderre. Ilyenkor jönnek jól ezek a beszédes megjegyzések.

Az első program

Minden programnyelv első kódja hagyományosan a beköszönés, a „Hello World” (Helló világ) kiírása. Az Arduino képernyő hiányában a beépített LED bekapcsolásával tudja ezt megtenni. A beépített LED a *3. ábrán* `'L'` betűvel jelölt LED, ami az áramkör 13-as lábára van bekötve.

A LED (Light-Emitting Diode – fényt kibocsátó dióda) egy kis fogyasztású, mára hétköznapivá vált félvezető. Számtalan eszközben használatos kis fogyasztása és nagy fényereje miatt. A fejlesztéseknek köszönhetően ma már világításra is alkalmas akár lakásokban, akár járművekben. A LED különböző megjelenési formáit a *6. ábra* mutatja.

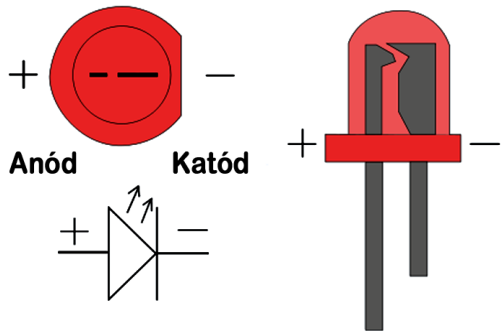
6. ábra: A LED különböző megjelenési formái



Forrás: https://upload.wikimedia.org/wikipedia/commons/9/9e/Verschiedene_LEDs.jpg

LED használatakor ügyelni kell a polaritásra. A LED kialakítása olyan, hogy az anód (+) és a katód (-) megkülönböztethető legyen (*7. ábra*). A katód kivezetése rövidebb, és a műanyag „házon” egy kis síkfelület is található. Ha jól megnézzük a LED-et, látható, hogy a műanyag tokban a katód „zászlócskája” nagyobb. A LED anód lába hosszabb, mint a katód. A LED áramköri jele annyiban különbözik a dióda jelétől, hogy két kis nyilacska szemlélteti a fénykibocsájtást.

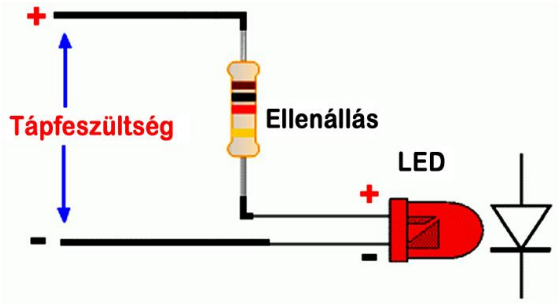
7. ábra: A LED áramköri jele és polaritása



Forrás: https://cdn.instructables.com/FPB/EG4N/FK8FVU92/FPBEG4NFK8FVU92.LARGE.jpg alapján

Fontos, hogy szükséges egy úgynevezett *előtét-ellenállás* bekötése is. Ennek értéke az 5V-os tápfeszültséggel működő Arduino-lapkánál 220-680 ohm közötti lehet. A bekötési rajzon az anód elé került az ellenállás, de az a katód elé is beköthető.

8. ábra: A LED bekötése



Forrás: https://320volt.com/en/wp-content/uploads/2014/06/tek-led-hesaplama-seri-led-direnc-hesap-lamasi.gif alapján

A lapkára szerelt LED természetesen rendelkezik előtét-ellenállással és polaritáshelyesen lett bekötve, így az első programunkhoz csak egy PC egy Arduino-áramkör és egy USB kábel szükséges. A kód írásához indítsuk el a fejlesztői felületet és írjuk be az alábbi kódot. A *5. ábrán* látott két megjegyzést magyarra lefordítva tartalmazza az alábbi kód, de azok természetesen elhagyhatóak. A továbbiakban már nem szerepeltetjük ezeket a megjegyzéseket.

1. kód

```
void setup() {
  // írd ide a beállító kódot, egyszer fut le:
  pinMode(13,OUTPUT);  //a 13-as láb kimenet
}

void loop() {
  // írd ide a fő programot, ismétlődően lefut:

  digitalWrite(13,HIGH);      //a 13-as láb HIGH értéke bekapcsolja a LED-et
  delay(1000);                //1 másodperc várakozás
  digitalWrite(13,LOW);       //LED kikapcsolása
  delay(1000);                //újabb 1 másodperc várakozás
}
```

Figyeljük meg, hogy az egyes sorokat mindig pontosvesszővel `';'` kell lezárni! Mindegy, hogy hol kezdjük a gépelést, akár több sorba is írhatunk egy utasítást, függvényt, a lezáró `';'` írásjel (karakter) tájékoztatja a fordítót az utasítás végéről.

A `setup()` függvényben meghívjuk a `pinMode(pin, mode)` függvényt. A függvények nevénél ügyeljünk a kis- és nagybetűk használatára! Ha helyesen gépeltünk, akkor a függvénynév barna színűvé változik, így jelezve, hogy az Arduino ismeri azt. A függvény hívásakor a zárójelek között adhatjuk meg a függvény paramétereit, először a láb (`pin`) számát, ezután a működési módját (`mode`), a kettőt vessző választja el egymástól. A láb most kimenetként, `OUTPUT`-ként működik, vagyis az értékét a program határozza meg. A függvényekben a paraméterátadás sorrendje kötött, azt nem lehet felcserélni!

Ezután a `loop()` függvény következik. Ebben először a `digitalWrite(pin, value)` függvényt hívjuk, ami szintén két paramétert használ. Elsőként a lábszámot kell megadni, majd a kimenet állapotát, ami digitális kimeneten magas (`HIGH`) vagy alacsony (`LOW`) lehet. (Ezek helyett írhatnánk `1` és `0` értéket is. A szöveges megadás talán könnyebben értelmezhető.) Amikor a LED anódjára magas feszültséget kapcsolunk, akkor az világítani kezd. A `delay(ms)`

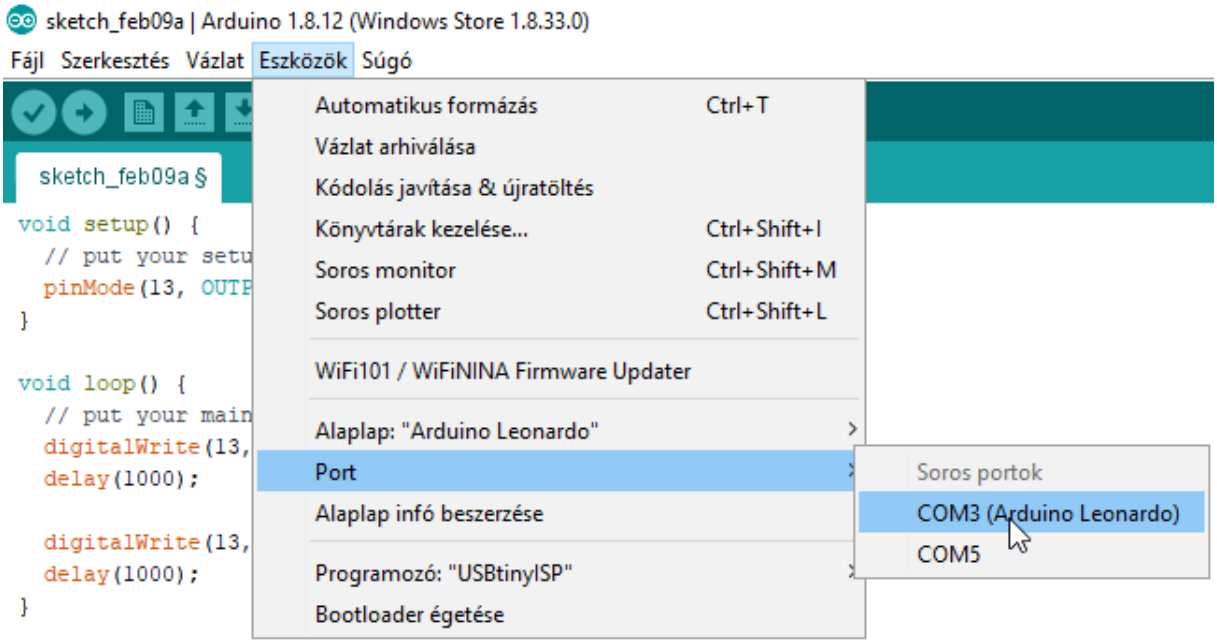
függvény várakozást iktat a programba. Paraméterként a várakozási időt kell megadni ezred másodpercben (millisec). Az itt beírt 1000-es érték 1 másodperces várakozást jelent. Ezután az újabb `digitalWrite()` függvénnyel lekapcsoljuk a LED-et, majd újra várunk egy másodpercet. Mivel a `loop()` függvény ismétlődően lefut, így a második várakozás után újra a `digitalWrite(13,HIGH)` függvény működik. Emiatt a LED bekapcsol, aztán egy másodperc múlva kikapcsol, majd újabb másodperc múlva megint bekapcsol, vagyis folyamatosan villog, egészen addig, amíg az Arduino-áramkör működik.

Az elkészült kódot a menüsor alatti pipa jelre kattintva tudjuk leellenőrizni (*9. ábra*). Ilyenkor az Arduino IDE program lefordítja a kódot, de nem tölti fel az Arduino-áramkörre. A jobbra mutató nyíl lenyomására a lefordított gépi kód át is töltődik az Arduino-lapkára, ami ezután futtatja is azt. A következő gombbal egy új skiccet lehet létrehozni. Az ezután következő felfelé nyíl lenyomásával egy korábbi skiccet lehet megnyitni, míg a lefelé nyílra kattintva menthetjük a számítógépünkre az aktuális skiccet. A még jobb érthetőség érdekében a gombok után olvasható az éppen kiválasztott funkció.

9. ábra: Műveletek a kóddal

A feltöltés előtt be kell állítanunk, hogy milyen alaplapra töltjük fel a kódot, és melyik portot használjuk ehhez. Először az **`Eszközök`** menü **`Alaplap`** menüpontjában beállítjuk az alaplap típusát, majd a **`Port`** menüpontjában az alaplap által használt portot (*10. ábra*).

10. ábra: Az alaplap és a port beállítása



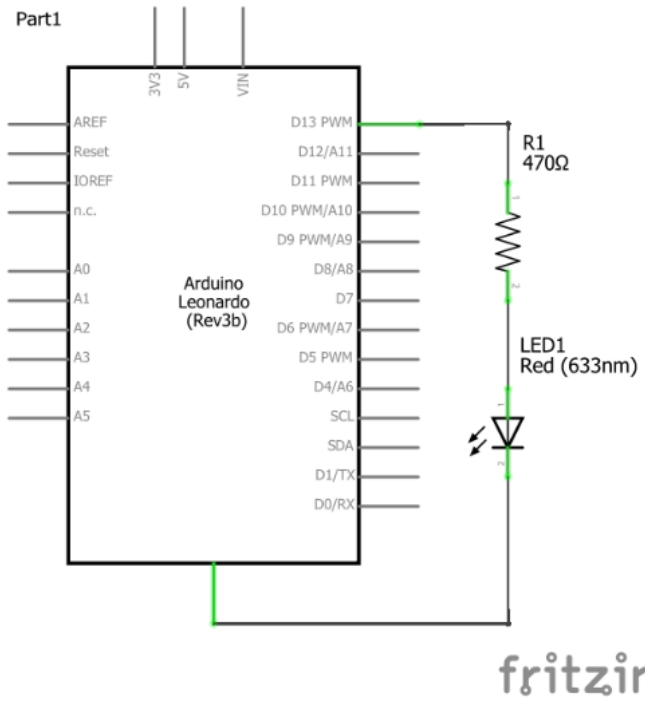
A lefordított gépi kód feltöltése közben az Arduino-lapkán villog az *RX* és *TX* LED, ez jelzi a soros vonali kommunikációt a számítógéppel. A gépi kód feltöltése után az *L* jelzésű LED villogni kezd, ami mutatja, hogy az első programunk működik.

Az alaplapra feltöltött programot az Arduino tárolja, a tápfeszültség kikapcsolása után sem veszíti azt el. A következő bekapcsolás után újra elindul a tárolt program. A feltöltés után már nem szükséges a soros vonali kapcsolat, így nemcsak a számítógép USB csatlakozójáról, hanem máshonnan, például egy külső akkumulátorról (PowerBank) vagy hálózati tápegységről is biztosíthatjuk a tápellátást az Arduino számára. A feltöltött program addig marad meg egy lapkán, amíg egy másikat nem töltünk rá.

Az első áramkör

Az előzőleg megírt kóddal nem csak az alaplapi LED-et, hanem az ahhoz kapcsolt külső LED-et is kapcsolgathatjuk. Az áramkör kapcsolási rajza a *11. ábrán* látható. A LED1 katódja (-) a Leonardo-alaplap GND-lábára van kötve. A GND (Ground – föld) a viszonyítási „nullapont", földpotenciálú pont egy áramkörben, megfelel egy szárazelem negatív pólusának, a rajta mért feszültség mindig 0 V (nulla Volt). A programban a GND feszültségének megfelelő értéket a LOW (alacsony) szóval vagy 0 számmal adhatjuk meg. A LED1 anódja (+) a 470 ohmos ellenállás (R1) egyik lábára csatlakozik. Az ellenállás másik lába az alaplap 13-as kivezetésére van kötve. Ez a kivezetés a programban OUTPUT-ként lett beállítva, és az kóddal vezérelhetően HIGH (magas) vagy LOW értékű lehet. A HIGH (vagy 1) érték esetén a kimeneten 5 V, LOW esetén 0 V feszültség mérhető. Ha a kivezetésre kapcsolt feszültség HIGH, akkor a LED világít, ha LOW, akkor pedig nem. A programban tehát a lábon mérhető feszültség értékét változtathatjuk meg.

11. ábra: LED-es villogó kapcsolási rajza

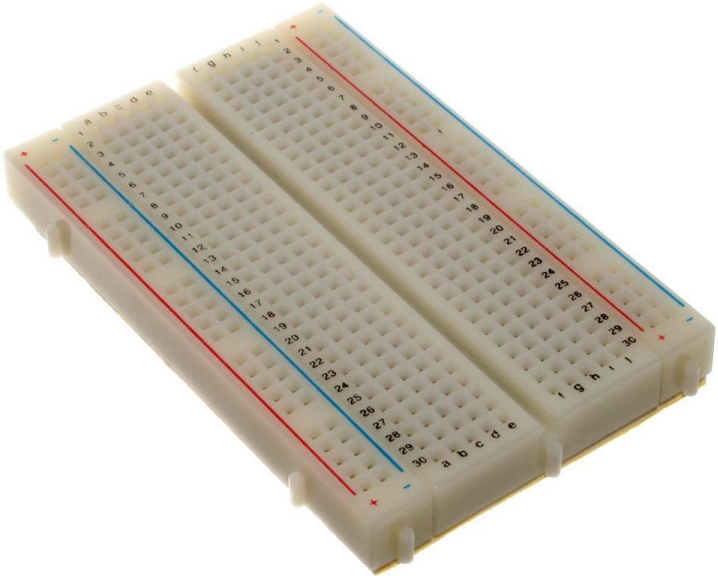


A kapcsolási rajz a *Fritzing programban* készült. Az elvi kapcsolásokat a gyakorlatban is meg kell valósítani. A fejlesztési szakaszban a kapcsoláshoz még nem készül nyomtatott áramkör (NYÁK), hanem úgynevezett próbapanelre kerül. A próbapanel (*12. ábra*) az áramkörök kísérleti felépítésére és kipróbálására szolgál. Angolul breadboard-nak (kenyérvágó deszkának) nevezik.

A próbapanelen az oszlopokat '+' és '–' jellel, valamint betűvel, a sorokat pedig számmal jelölik. A *13. ábrán* látható a próbapanel belső összeköttetése, vezetékezése. A panel két szélén egy-egy oszlopban pozitív és negatív kapcsolósor (szakzsargonban tápbusz) található. A sorokban az `a, b, c, d, e` és az `f, g, h, i, j` oszlopok pontjai kapcsolódnak egymáshoz.

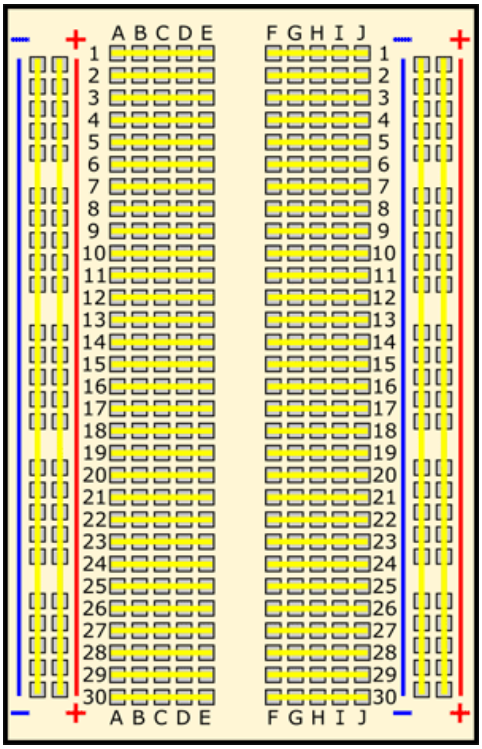
A próbapanelbe az alkatrészek lábait egyszerűen bele lehet dugni, az összekötésekhez pedig olyan vezetékeket lehet használni, amelyeknek a két végén szintén dugaszolható pinek vannak, ezeket *jumper wire-nek*, áthidaló vezetéknek hívják (*14. ábra*).

12. ábra: Próbapanel



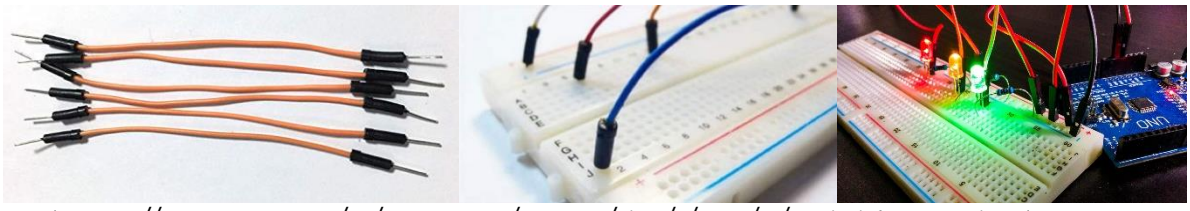
Forrás: https://hu.wikipedia.org/wiki/Szerel%C5%91lap#/media/F%C3%A1jl:400_points_breadboard.jpg

13. ábra: Próbapanel belső huzalozása



Forrás: https://osoyoo.com/wp-content/uploads/2017/06/breadboard-row-connections.png

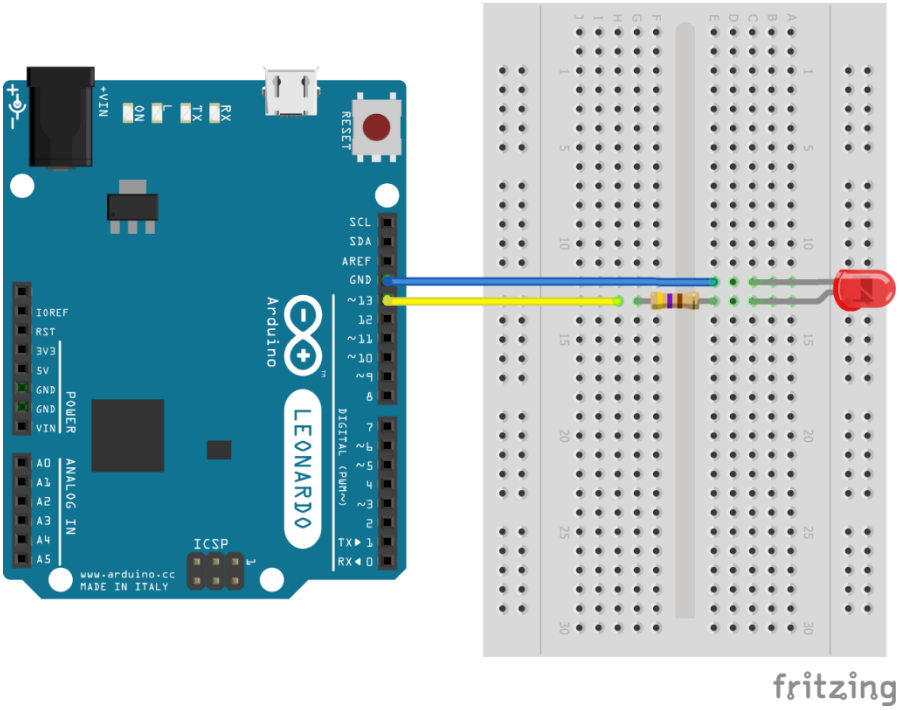
14. ábra: Összekötő kábelek használata



Forrás: https://dealroaster.com/us/wp-content/uploads/sites/2/2020/01/717zjari7flacsx679.jpg és https://www.pcboard.ca/image/cache/catalog/products/breadboard/proto-jumper-wire-800x800.jpg és https://i.ytimg.com/vi/fq6U5Y14oM4/maxresdefault.jpg

A kapcsolási rajz alapján a Fritzing programban elkészíthető a próbapanelen kialakítandó áramkör rajza (*15. ábra*). Ez alapján bárki összeállíthatja a LED-es villogó kapcsolását. Eleinte célszerű teljes mértékben lemásolni az elvi vázlatot, később, amikor a próbapanel felépítése világossá válik, más elosztásban is elhelyezhetők az alkatrészek. Az áramkör építésekor szüntessük meg az USB-csatlakozást! Először az alkatrészeket dugjuk be a panelre, majd a színes vonalakkal jelölt vezetékeket, ezek a megvalósítás során természetesen más színűek is lehetnek, mint az ábrán.

15. ábra: LED bekötése próbapanelen



Ha a kapcsolást helyesen állítottuk össze, akkor az USB-re való csatlakozás után az alaplapon és a próbapanelen lévő LED villogni fog. Az előzőekben már áttöltöttük a programot az alaplapra, ezért azt nem kell újra megtenni.

Ha már összeállítottuk az első áramkört, írjunk hozzá programot! Az SOS (*Save Our Souls* – Mentsd meg lelkeinket) nemzetközi vészjelzés Morse-kódja három rövid, három hosszú és

újabb három rövid jelzésből áll (··· — — — ···). Írjunk programot, ami az SOS „ritmusára” villogtatja a LED-et.

A feladat megoldásához tudnunk kell a jelzések és a szünetek hosszát. A rövid jel (ti) 1 egység hosszú, a hosszú jel (tá) 3 egység hosszú, a rövid szünetjel (jelköz) 1 egység hosszú, a hosszú szünetjel (betűköz) 3 egység hosszú, és végül a nagyon hosszú szünetjel (szóköz és mondatköz) 7 egység hosszú.

Az előző programhoz képest többet kell gépelnünk, de a `setup()` függvény ugyanaz, és a `loop()` függvényben sincs újabb utasítás. Mivel többször kell be és kikapcsolni a LED-et, ezért ehhez több utasítássort kell beírnunk, és az időzítésekből is használjuk mind az ötöt. Ha a rövid jel hosszát 200 milli szekundumnak állítjuk be, akkor a többi időzítés ebből számítható.

2. kód

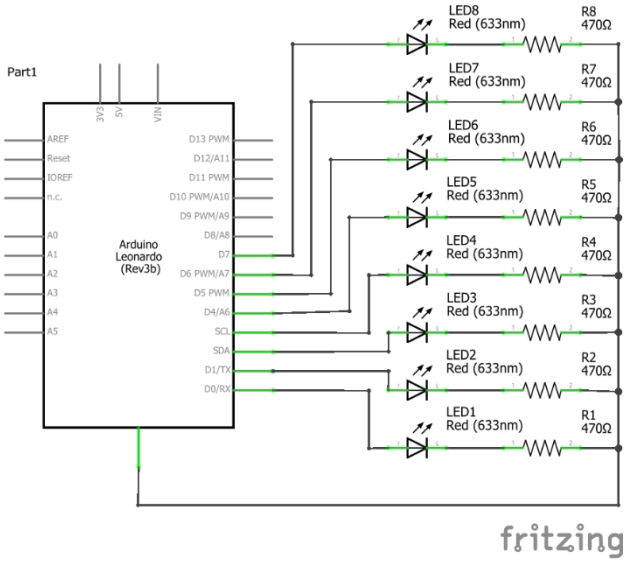
```
void setup() {

  pinMode(13, OUTPUT);
}

void loop() {

  digitalWrite(13, HIGH); //ti
  delay(200);

  digitalWrite(13, LOW);  //jelköz
  delay(200);

  digitalWrite(13, HIGH); //ti
  delay(200);

  digitalWrite(13, LOW);  //jelköz
  delay(200);

  digitalWrite(13, HIGH); //ti
  delay(200);

  digitalWrite(13, LOW);  //betűköz
  delay(600);

  digitalWrite(13, HIGH); //tá
  delay(600);

  digitalWrite(13, LOW);  //jelköz
  delay(200);

  digitalWrite(13, HIGH); //tá
  delay(600);

  digitalWrite(13, LOW);  //jelköz
  delay(200);

  digitalWrite(13, HIGH); //tá
  delay(600);

  digitalWrite(13, LOW);  //betűköz
  delay(600);
```

```
  digitalWrite(13, HIGH); //ti
  delay(200);

  digitalWrite(13, LOW);  //jelköz
  delay(200);

  digitalWrite(13, HIGH); //ti
  delay(200);

  digitalWrite(13, LOW);  //jelköz
  delay(200);

  digitalWrite(13, HIGH); //ti
  delay(200);

  digitalWrite(13, LOW);  //szóköz
  delay(1400);
}
```

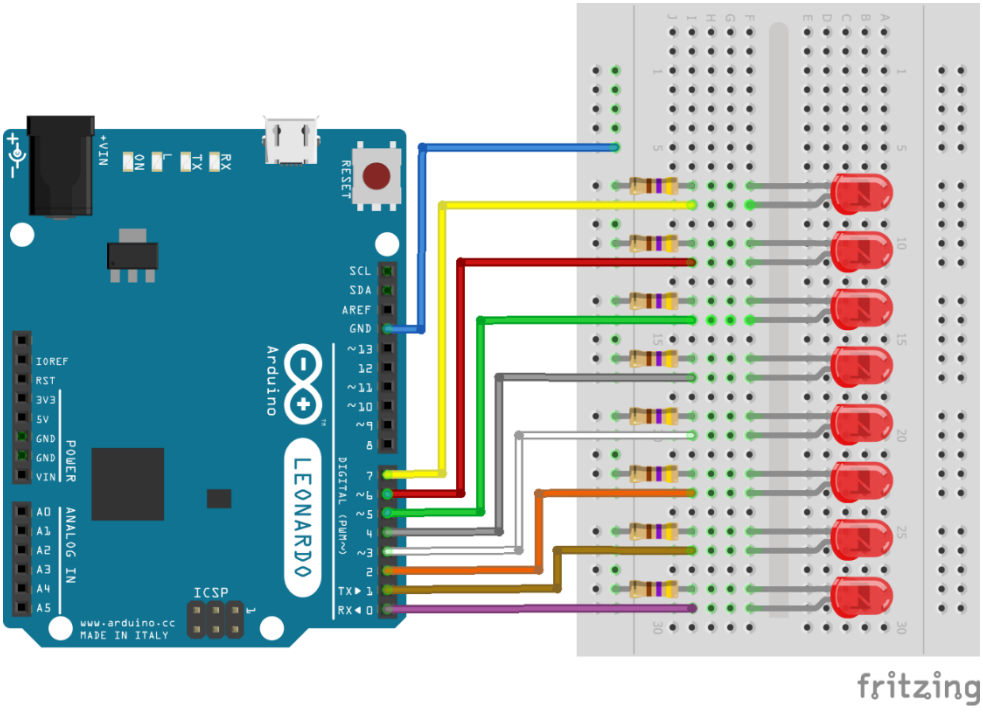
LED-es futófény

Ez az áramkör csak a kapcsolt LED-ek számában tér el az előzőtől, itt 8 LED-ből és természetesen a hozzá tartozó ellenállásokból áll a kapcsolás. A LED-ek anódját a 0–7 digitális kimenetekre kötjük, a katódjukhoz kapcsolódnak az ellenállások, amelyeknek a másik lába a GND-re van bekötve (*16. ábra*). Az áramkört most is a BreadBoard-on építjük meg (*17. ábra*)

16. ábra: LED-es futófény kapcsolási rajza



17. ábra: LED-es futófény megépítése próbapanelen



A kapcsoláshoz tartozó kód is nagyon hasonlít az előzőre, a különbség itt is csak annyi, hogy 8-szor kell megismételnünk a parancsokat.

3. kód

```
void setup() {

  pinMode(0, OUTPUT);
  pinMode(1, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
}

void loop() {

  digitalWrite(0, HIGH);
  delay(1000);
  digitalWrite(0, LOW);
  delay(1000);

  digitalWrite(1, HIGH);
  delay(1000);
  digitalWrite(1, LOW);
  delay(1000);

  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
  delay(1000);

  digitalWrite(3, HIGH);
  delay(1000);
  digitalWrite(3, LOW);
  delay(1000);

  digitalWrite(4, HIGH);
  delay(1000);
  digitalWrite(4, LOW);
  delay(1000);

  digitalWrite(5, HIGH);
  delay(1000);
  digitalWrite(5, LOW);
  delay(1000);

  digitalWrite(6, HIGH);
  delay(1000);
  digitalWrite(6, LOW);
  delay(1000);

  digitalWrite(7,HIGH);
  delay(1000);
  digitalWrite(7, LOW);
  delay(1000);
}
```

A kód hosszú, sokat kell gépelni. A vágólap használatával fel lehet gyorsítani a gépelést. Egyszer begépeljük a kód részletét, amit kijelölés után a vágólapra másolunk (*Ctrl+C*), ezután azt még 7-szer beillesztjük az első kódrészlet után (*Ctrl+V*). Végül írjuk át a digitális kimenetek számát!

Szerencsére a vágólapos eljárásnál van egyszerűbb megoldás is. Az ilyen ciklikusan ismétlődő feladatokhoz a programnyelvek biztosítanak egy eszközt, amit *ciklusnak* nevezünk. Több féle ciklus létezik, amelyek az Arduino programozásához is felhasználhatók. Most a legegyszerűbb `for` ciklust ismerjük meg ezek közül. Az Arduino honlapján természetesen erről is találunk leírást a https://www.arduino.cc/reference/en/language/structure/control-structure/for/ linken.

```
for (initialization; condition; increment) {
  // statement(s);
}
```

A `for` ciklus egy vezérlési szerkezet, ami a kapcsos zárójelek közötti utasítás(ok) (`statement(s)`) ismétlésére szolgál. A ciklus fejléce három részből áll, amelyek pontos vesszővel vannak elválasztva. Az inicializálás (`initialization`) egyszer történik meg, itt rögzítjük a ciklusváltozó kezdőértékét. A ciklus addig fut, amíg a feltétel (`condition`) igaz, annak ellenőrzése az utasításblokk minden egyes lefutása után megtörténik. Amikor a feltétel hamis, a ciklus véget ér. A növekmény (`increment`) azt írja le, hogy a ciklusváltozó mennyivel nőjön vagy csökkenjen. Nézzünk egy példát:

```
for (int i = 0; i < 8; i++) {
   // utasítás(ok);
}
```

Az inicializálásnál adjuk meg a ciklusváltozó kezdőértékét. Itt találkozunk először a *változó* fogalmával, amit névvel azonosítunk, és a típusától függően valamilyen értéket tartalmaz, ami változhat. Hasonlatként vegyük a zoknisfiókot, aminek a neve zoknisfiók és zokni típusú elemek vannak benne, amelyek száma változhat. (A változó ellentéte az *állandó* (konstans), ami nem változik, például PI = 3,14.)

A példában a ciklusváltozó (`i`) egész (integer: `int`) típusú, vagyis csak pozitív és negatív egész számokat tartalmazhat, kezdő értéke egyenlő nullával: (`int i = 0`). Már említettük, hogy a számolást 0-val kezdjük. A ciklusunk addig fut, amíg a ciklusváltozó kisebb, mint 8 (`i < 8`), emiatt i értéke nullától hétig változik. Így összesen nyolc alkalommal hajtódik végre az utasításblokk. A feltételt megadhattuk volna így is: `i <= 7`, hiszen ekkor is végrehajtódnak az utasítások, ha a ciklusváltozó értéke egyenlő héttel. A ciklusváltozót egyesével növeljük (`i++`). Más programnyelvekkel ellentétben, a C nyelvben egy változó értékének növelését nem az `i = i + 1` formulával kell megadni. Az egyesével való növelést `i++`, a csökkentést `i--` utasítással adjuk meg. Az egytől eltérő értékkel, például 2-vel való növelést az `i+= 2`, a csökkentést az `i-= 2` formulával adjuk meg.

Nézzük most meg a `for` ciklussal leírt lényegesen rövidebb kódot:

4. kód

```
void setup() {

  for (int i = 0; i < 8; i++) {  //0-tól 7-ig
    pinMode(i, OUTPUT);          //minden láb kimenet
  }
}

void loop() {

  for (int i = 0; i < 8; i++) {  //0-tól 7-ig
    digitalWrite(i,HIGH);        //bekapcsoljuk a kimeneteket
    delay(1000);
    digitalWrite(i,LOW);         //kikapcsoljuk a kimeneteket
    delay(1000);
  }
}
```

A `for` ciklus törzsében, az utasításblokkban konkrét számok helyett már a ciklusváltozót kell beírnunk a kimenetek számozására, az Arduino futás közben sorban behelyettesíti annak értékét a `pinMode()` és a `digitalWrite()` függvényekbe.

Változtassuk most meg a késleltetés értékét a felére, legyen az fél másodperces, amit a `delay(500)` utasítással tehetünk meg. A késleltetés a program futása közben állandó, így azt célszerű konstansként, állandóként megadni, a következőképpen:

```
const int WAIT = 500;
```

Először megadjuk, hogy állandót deklarálunk (`const`), ezután azt, hogy milyen típusú (egész: `int`), majd a nevét (várakozás: `WAIT`), végül az értékét (`500`). Ezt az utasítást is pontosvesszővel kell lezárni. A C-ben – így az Arduino-programokban is – a változók neveit kisbetűvel, a konstansok neveit nagybetűvel szokás írni. Az `OUTPUT`, a `HIGH` és `LOW` előre megadott rendszerállandók.

Egészítsük ki az előző kódot a várakozási idő megadásával, ehhez csak a `loop` függvényt kell módosítani:

```
void loop() {

const int WAIT = 500;

  for (int i = 0; i < 8; i++) {
    digitalWrite(i,HIGH);
    delay(WAIT);
    digitalWrite(i,LOW);
    delay(WAIT);
  }
}
```

Ha ezután más időzítést is ki szeretnénk próbálni, akkor elég csak a `WAIT` konstans értékét átírni és újra lefordítani a kódot.

A várakozási időt megadhattuk volna egy változóban is, hiszen nem kötelező, hogy annak valóban változnia kell a futás során.

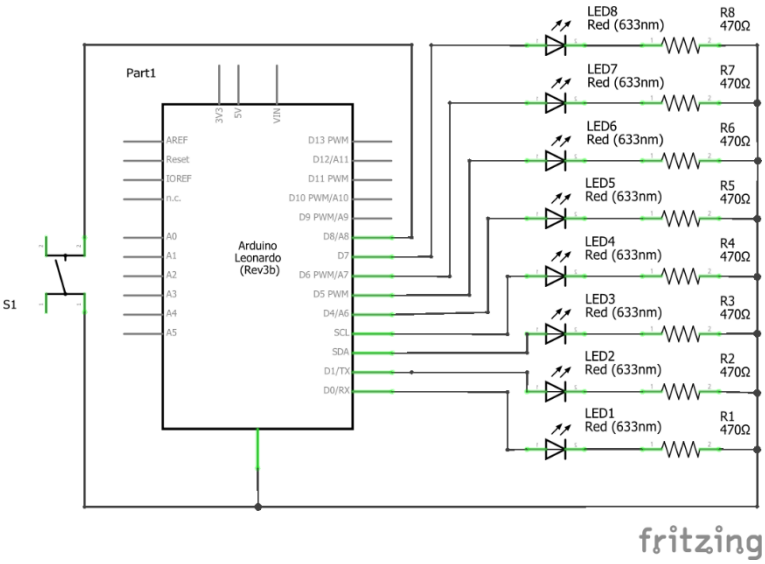
Az előzőekben írt SOS programban is használhatnánk konstanst az alap várakozási idő megadására, ekkor a többi időzítést előállíthatnánk szorzási művelettel.

LED-es futófény nyomógombbal

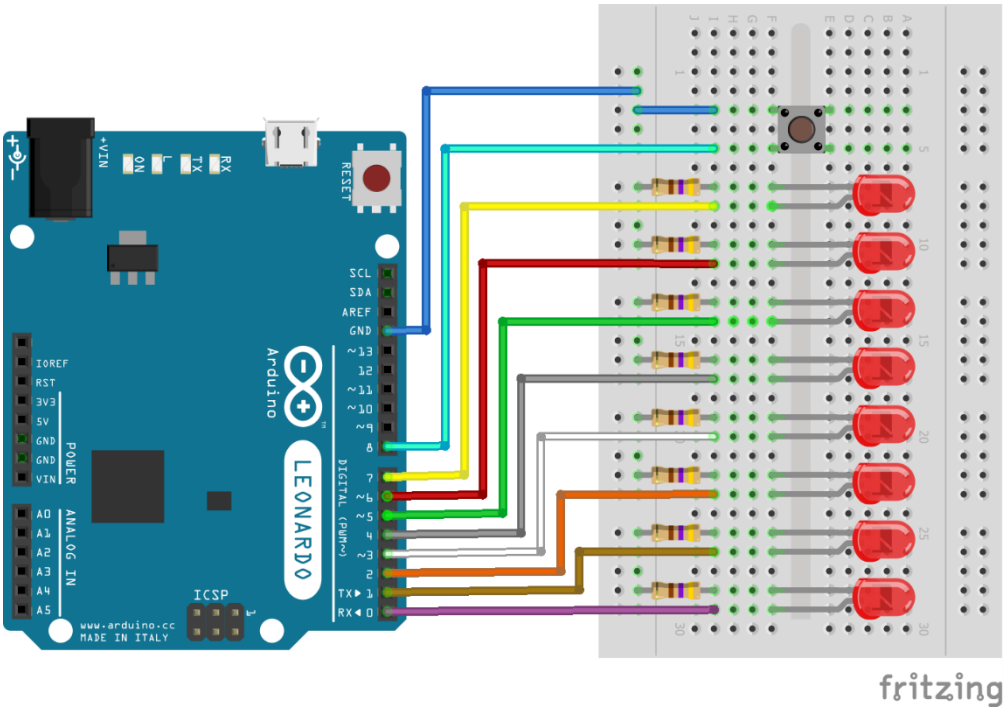
Az előző kapcsolást kiegészítjük egy nyomógombbal, aminek lenyomásakor megváltoztathatjuk a várakozási időt vagy a futófény irányát (*18,19 ábra*).

A nyomógombot a 8-as digitális lábra kötjük, amit most bemenetként (`INPUT`) használunk. A nyomógomb másik lábát a GND-re kötjük be. A nyomógomb alapesetben nincs benyomva, így a bemenet értéke magas (`HIGH`). Ha nyomógombot lenyomjuk, akkor a 8-as bemenet értéke alacsony (`LOW`) lesz, ha elengedjük, akkor ismét magas értéket vesz fel.

18. ábra: A LED-es futófény nyomógombbal kapcsolási rajza

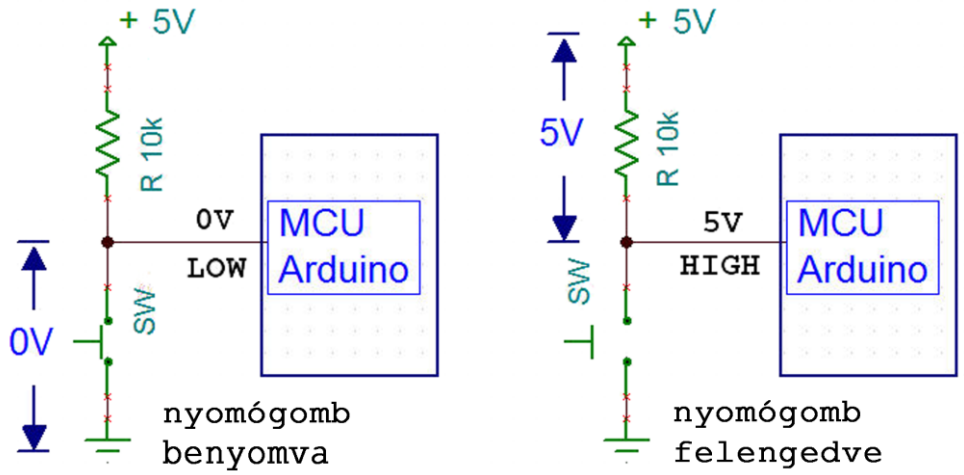


19. ábra: LED-es futófény nyomógombbal kapcsolás megvalósítása próbapanelen



Szokás az ilyen kapcsolások esetén úgynevezett *felhúzó ellenállás* használata, ami felengedett nyomógomb esetén biztosítja a bemenet magas szintjét (*20. ábra*). A felhúzó ellenállás nélkül a bemenetre kötött hosszú vezeték antennaként működik, és mindenféle zavaró jelet szed össze a környezetből, ezáltal a bemenet állapota bizonytalan lesz, nem mindig egyezik meg a nyomógomb állapotával. A mi kapcsolásunkban nincsen felhúzó ellenállás, mert az Arduino-áramkör tartalmazza azokat.

20. ábra: Felhúzó ellenállás beépítése



Forrás: http://fabacademy.org/2019/labs/berytech/students/nagi-abdelnour/images/WEEK7/pull-up-resistor.jpg alapján

A `setup()` függvényben most a 8-as lábat bemenetként kell megadnunk, és a belső felhúzó ellenállást is aktiválni kell a következőképpen:

```
pinMode(8, INPUT);      //8-as lab bemenet
digitalWrite(8,HIGH);   //belso felhuzoellenallas bekapcsolasa
```

Érdekesség, hogy a bemenetre úgy írunk, mintha kimenet lenne.

Az első kódban a nyomógomb állapotától, ezáltal a 8-as bemenet értékétől függően gyors vagy lassú lesz a futófény sebessége. Ehhez definiálunk egy egész típusú (`int`) változót, ami a késleltetési időt tárolja, és mindjárt értéket is adunk neki (`int wait = 500;`). A fél másodperces késleltetés esetén alapállapotban (nyomógomb felengedett) gyorsabb lesz a futás, mint eddig. A késleltetés akkor változik meg, amikor a nyomógombot benyomjuk.

A nyomógomb állapotát `digitalRead(pin)` függvény olvassa be, ami a `digitalWrite()` ellentéte. A függvény beolvassa egy bemenet (pin) állapotát, ami magas (`HIGH`) vagy alacsony (`LOW`) értékű lehet. A függvény `digitalRead()` ezt az értéket szolgáltatja eredményként, amit egy változóban tárolhatunk. Ez az első olyan függvényünk, amelyik ad vissza egy eredményt, ami egy egész (`integer` típusú) érték. (Már volt szó róla, hogy a `HIGH = 1`, a `LOW = 0`.) A `digitalRead()` tehát az első olyan függvény, ami nem `void`, hanem `int` típusú.

Ezután meg kell vizsgálni ezt a változót, hogy az értékétől függően eldöntsük, mit tegyen a program. Erre az `if` (ha) vezérlési szerkezetet használjuk:

```
if (condition) {
  //statement(s)
}
```

Az `if` megvizsgál egy feltételt (`condition`), és amennyiben az igaz (`true`), akkor végrehajtja az utasítás(oka)t (`statement(s)`). Megjegyzendő, hogy bár a `true` és a `false` is konstans mégis kisbetűvel kell írni őket!

A teljes kód a következő:

5. kód

```
void setup() {

  for (int i = 0; i < 8; i++) {
    pinMode(i,OUTPUT);
  }
  pinMode(8, INPUT);
  digitalWrite(8,HIGH);
}

void loop() {

  int wait = 500; //várakozási idő alaphelyzetben
  int pb_value;   //push button value: a nyomógomb értéke

  pb_value = digitalRead(8);  //nyomógomb állapot beolvasása
  if (pb_value == LOW) {      //ha a nyomógomb benyomott állapotban van
    wait = 1000;              //akkor 1 sec legyen várakozás
  }

  for (int i = 0; i < 8; i++) {
    digitalWrite(i,HIGH);
    delay(wait);
    digitalWrite(i,LOW);
    delay(wait);
  }
}
```

Mivel a `loop()` mindaddig ismétlődik, amíg az Arduino működik, ezért a `wait` változó érték-megadása (inicializálása), minden futás esetén megtörténik. A `wait` változó értéke csak akkor változik meg, ha a nyomógomb be lett nyomva.

A C nyelv törekszik a tömör kódra, így az előbbi bemenetbeolvasás és feltételmegadás egyszerűbben is leírható:

```
  if (digitalRead(8) == LOW) {
    wait = 1000;
  }
```

Mind a két kód ugyanazt végzi el. A második esetben egy utasítás sorban olvassuk be a bemenetet és vizsgáljuk meg annak állapotát.

Láthatjuk, hogy ebben az esetben a `pb_value` változóra nincs szükség, így az nem foglal helyet a memóriában. Most ennek nincs jelentősége, de bonyolultabb kapcsolások esetén szükség lehet erre 2 byte tárterületre is.

Eleinte célszerűbb talán a hosszabb, egyben jobban értelmezhető kód írása, a későbbi érthetőség miatt.

Következő programunkban a nyomógomb állapotától függően előre vagy hátra halad a futófény. Ehhez ismerkedjünk meg az `if` vezérlési szerkezet helyett a bővebb `if - else if - else` szerkezettel! Itt nem csak egy feltételt adhatunk meg az `if` (ha) kulcsszóval és a

hozzátartozó utasításokkal (`do Thing A`), hanem egyéb feltétel(eke)t is az `else if` (egyébként, ha) után, amit a hozzátartozó utasításblokk (`do Thing B`) követ. Ha az előzőleg megadott feltételek egyike sem teljesül, akkor az `else` (egyéb) kulcsszó után írhatjuk az ide tartozó utasításokat (`do Thing C`). A feltételes szerkezetben mindig kell, hogy szerepeljen az `if` utasítás. Az `else if` többször is szerepelhet, ha több feltételt is vizsgálunk, de el is hagyható, ha csak egyet. Az `else` is elhagyható, ahogy azt az előző példában láthattuk.

```
if (condition1) {
  // do Thing A
}
else if (condition2) {
  // do Thing B
}
else {
  // do Thing C
}
```

Az egyszerűség kedvéért az előre és a hátra futtató kódokat nem a `loop()` függvénybe írjuk, hanem külön saját függvényt írunk nekik. Nézzük a `hatra()` függvényt, ami nem ad vissza eredményt és nem is kap meg semmiféle adatot a hívó függvénytől. Emiatt lesz a függvény void és üres a zárójelek közötti rész. A `for` ciklusban a kezdő érték i = 7 lesz, az utolsó LED-et kapcsoljuk be először. A feltétel azt mondja, hogy addig fusson a ciklus, amíg a ciklusváltozó nagyobb nullánál (i >= 0) és a ciklusszámlálót egyesével csökkentjük (i--). Maga a ciklusblokk ugyanaz, mint az előre számolásnál.

```
void hatra()  {

  for (int i = 7; i >= 0; i--) {
    digitalWrite(i, HIGH);
    delay(100);
    digitalWrite(i, LOW);
    delay(100);
  }
}
```

A teljes kódban az `if` után csak egy `else` található. Ha a nyomógomb nincs benyomva, akkor előre fut, egyébként hátra.

6. kód

```
void setup() {

  for (int i = 0; i < 8; i++) {
    pinMode(i, OUTPUT);
  }

  pinMode(8, INPUT);
  digitalWrite(8, HIGH);
}
void loop() {

  if (digitalRead(8) == HIGH) {
    elore();
  }
```

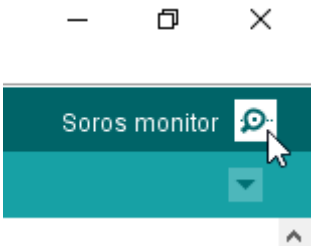
```
  else {
    hatra();
  }
}

void elore()  {

  for (int i = 0; i < 8; i++) {
    digitalWrite(i, HIGH);
    delay(100);
    digitalWrite(i, LOW);
    delay(100);
  }
}

void hatra()  {

  for (int i = 7; i >= 0; i--) {
    digitalWrite(i, HIGH);
    delay(100);
    digitalWrite(i, LOW);
    delay(100);
  }
}
```

Ugyanezt a kapcsolást használjuk fel a következő feladathoz is: Ha a nyomógomb benyomott állapotban van, akkor a késleltetési idő legyen 500 ms, egyébként pedig egy véletlen szám, ami 50 ms és 500 ms közötti érték lehet. A véletlen szám értékét meg fogjuk nézni az úgynevezett soros monitoron.

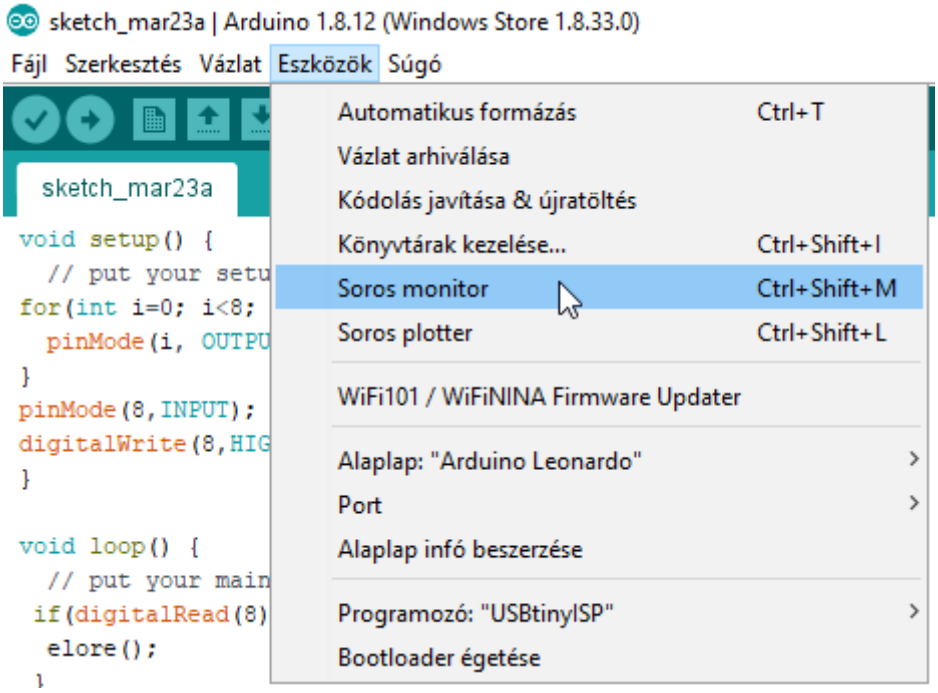
A soros monitor egy soros vonali kommunikációs lehetőség, melynek a segítségével az Arduino és a PC kapcsolatba tud lépni egymással. Nagy segítség a hibakeresésben, hiszen az Arduino változóit a program futása közben láthatjuk, de akár adatokat is küldhetünk neki. A kommunikáció fizikailag az USB kábelen történik, amely a PC soros portjához csatlakozik.

A soros monitort a fejlesztőkörnyezet ablakának jobb felső sarkában található gombra (*21. ábra*) kattintva vagy az **Eszközök** menü **Soros monitor** menüjével (*22. ábra*) vagy a *Ctrl+Shift+M* billentyűkombinációval nyithatjuk meg.

21. ábra: Soros monitor megnyitása ikonnal



22. ábra: Soros monitor megnyitása menüvel



A programozás során a soros port működtetéséhez szükséges utasításokat a *Serial* objektum metódusaival (függvényeivel) érhetjük el. Ez számunkra most azt jelenti, hogy a soros portot működtető függvények a programban a `Serial.` szöveg után következnek. Az objektumokról később még lesz szó.

A programban először a `setup()` függvényben kell megadni, hogy milyen sebességű legyen a kommunikáció:

```
Serial.begin(speed)
```

A sebesség (`speed`) mértékegysége a baud, azaz bit/secundum, az 1 másodperc alatt átvitt bitek száma. A 9.600 baud sebesség általában megfelelő érték.

A soros monitorra két függvénnyel írhatunk.

```
Serial.print(val)
Serial.println(val)
```

A `val` paraméter maga a kiírandó üzenet. Ha az üzenet szöveg, akkor azt idézőjelek közé („`Hello World`”), ha egy karakter (írásjel), akkor aposztrófok közé (`'N'`) kell tenni, míg a számokat csak egyszerűen be kell írnunk. Változó kiírásakor a változó nevét kell a zárójelek közé írni. A `Serial.println()` kiírja a benne lévő üzenetet és a következő sorba lép, legközelebb ott folytatja a kiírást. A `Serial.print()` szintén kiírja az üzenetet, csak nem vált sort, tehát a következő kiírás is ugyanabba a sorba kerül, amíg az be nem telik, utána folytatódik a következő sorban.

A programunkban a függvényeken is változtatnunk kell. Az `elore()` és `hatra()` függvények továbbra sem adnak vissza értéket, a típusuk marad `void`, viszont megkapják paraméterként a várakozási időt, ami egész szám és a neve wait: `void elore(int wait)`. A függvényeken belül a `delay()` függvénybe a `wait` változó nevét kell írnunk.

```
void elore(int wait) {

  for (int i = 0; i < 8; i++) {
    digitalWrite(i, HIGH);
    delay(wait);
    digitalWrite(i, LOW);
    delay(wait);
  }
}
```

A függvény hívása a `loop`-ból az `elore(ido);` utasítással történik. Látszik, hogy a késleltetés értékét tartalmazó változónak különböző neve van a két helyen. Ha a program olvasását, érthetőségét ez nem zavarja, akkor ezt meg lehet tenni, nem kapunk hibajelzést. A név eltérhet, de a típusnak egyeznie kell!

A véletlen számot előállító függvény nem kap értéket a hívótól, viszont eredményként visszaadja a generált számot. Mivel a visszaadott érték típusa egész, a függvény típusa is egész. A függvényt emiatt nem a `void`, hanem az `int` kulcsszó vezeti be: `int veletlenszam()`.

Mivel a soros vonalon is ki szeretnénk írni a számot, ezért értékét a `vel_szam` változóba tesszük. A számot a random függvény szolgáltatja, amit kétféle módon használhatunk:

```
random(max)
random(min, max)
```

Az első függvény nulla és a megadott maximumnál egyel kisebb érték (max-1) közötti tetszőleges számot ad vissza. A második függvénynél a minimumot is megadhatjuk, a visszaadott érték a min és max-1 közötti érték lehet. Ezután kiírjuk a kapott számot a soros portra, majd jön az eredmény visszaadása, ami a `return vel_szam;` utasítással történik. A `return` (visszatérés, megfordulás) utasításnak is két formája van:

```
return;
return value;
```

Ha függvény ad vissza értéket, akkor kell, hogy szerepeljen benne a `return value;` sor, ha nem ad vissza értéket, akkor a `return` utasítással visszatérhetünk a hívó függvényhez, még azelőtt, hogy annak a végéhez értünk volna. Ez általában akkor történik így, ha valamilyen feltétel miatt nem kell végigmenni a hátralévő utasításokon. Ha egy függvényben nem szerepel a `return`, akkor az az utolsó utasítássor végrehajtása után tér vissza a hívóhoz.

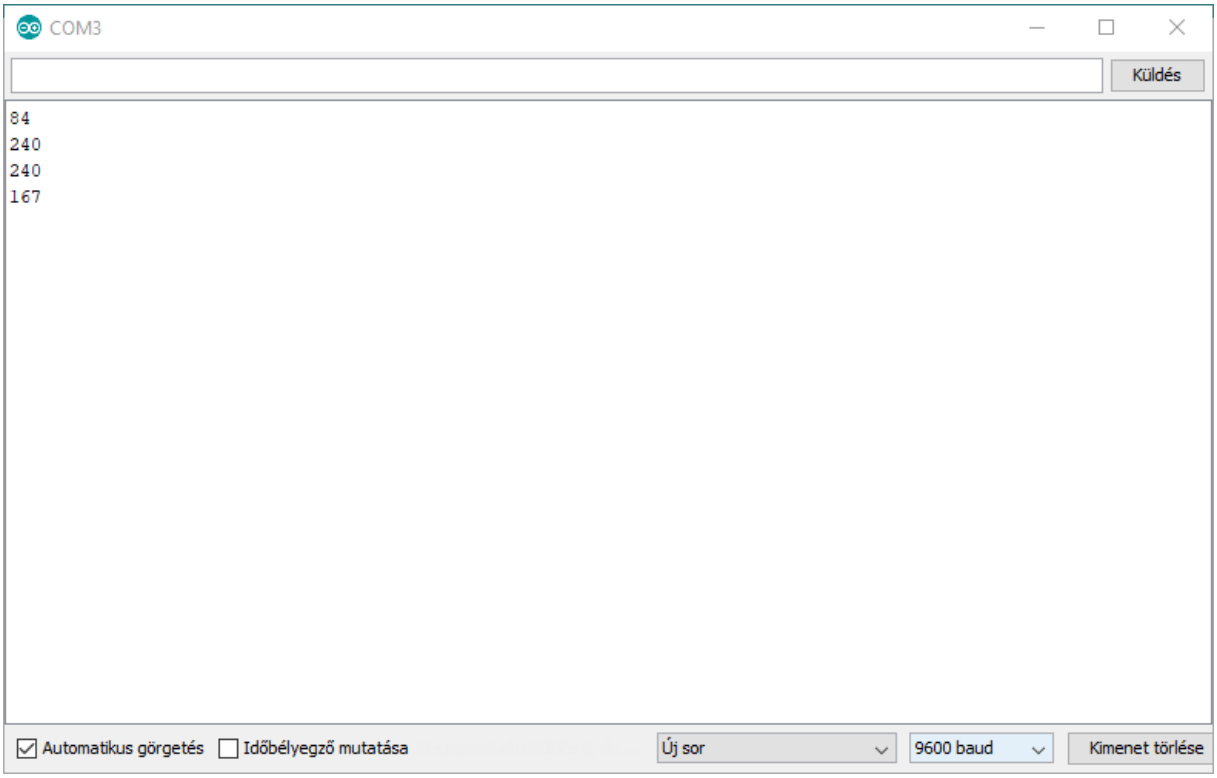
7. kód

```
void setup() {

  for (int i = 0; i < 8; i++) {
    pinMode(i, OUTPUT);
  }

  pinMode(8, INPUT);
  digitalWrite(8, HIGH);

  Serial.begin(9600);
}
```

```
void loop() {

  int ido;

  if (digitalRead(8) == LOW) {
    ido = 500;
  }
  else {
    ido = veletlenszam();
  }
  elore(ido);
  hatra(ido);
}

int veletlenszam() {

  int vel_szam;

  vel_szam = random(25, 300);
  Serial.println(vel_szam);
  return vel_szam;
}

void elore(int wait) {

  for (int i = 0; i < 8; i++) {
    digitalWrite(i, HIGH);
    delay(wait);
    digitalWrite(i, LOW);
    delay(wait);
  }
}

void hatra(int wait) {

  for (int i = 7; i >= 0; i--) {
    digitalWrite(i, HIGH);
    delay(wait);
    digitalWrite(i, LOW);
    delay(wait);
  }
}
```

A soros monitoron a program futása közben (*23. ábra*) láthatók a generált véletlenszámok. A számokat az Arduino-ban futó program generálja, az abban szereplő változó értékeit jelenítettük meg, tehát az onnan érkező adat látható a soros monitoron. A kommunikáció kétoldalú, mi is küldhetünk adatot az Arduino számára. A felül található beviteli mezőbe kell beírnunk kimenő adatot majd a jobbra található Küldés gombra kattintva küldhetjük el azt. A soros monitor fejlécében látható a használt port, ami most a COM3.

23. ábra: Soros monitor



LED-es futófény potenciométerrel

Az előző kapcsolásokban digitális be- és kimeneteket használtunk. Ezeknek az a jellemzője, hogy diszkrét (egyedi) értéket vehettek fel, vagy nulla Voltot (0, LOW) vagy 5 Voltot (1, HIGH), a kettő közötti feszültség értéket pedig nem. A kapcsoló vagy be volt nyomva vagy nem, a LED vagy világított vagy nem.

A világon a jelenségek nem egyedi, hanem folytonos értékeket vesznek fel, a hőmérséklet például folyamatosan nő vagy csökken. A hőmérsékletet hőmérővel mérjük, amely egy folyadékszál hosszával mutatja meg annak értékét. A hőmérséklet és a folyadékszál hossza együtt változik, ha a hőmérséklet nő, a folyadékszál hossza is nő, csökkenő hőmérséklet esetén a folyadékszál hossza is csökken. A két érték között arányosság, hasonlóság, analógia van, emiatt a hőmérőben lévő folyadék hosszát analóg jelnek nevezzük. Ha a hőmérséklet értékét Arduino-val szeretnénk feldolgozni, akkor itt is valamilyen vele analóg jelet kellene használnunk. Ez a jel nem lehet diszkrét, hanem folytonos, mert a hőmérséklet sem diszkrét, vagy van vagy nincs. A hőmérsékletnek megfeleltethetünk vele arányos feszültséget is, amit az Arduino A0–A5 analóg lábain mérhetünk.

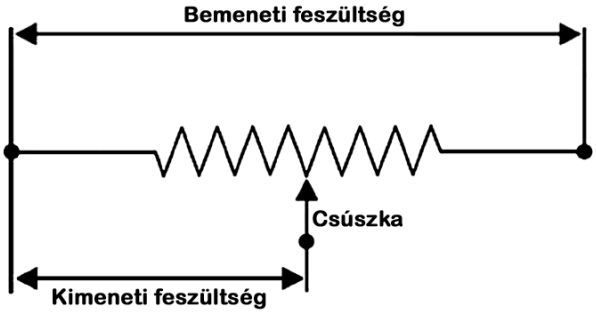
Ebben a kapcsolásban az A0 analóg bemenetet fogjuk használni. Egy potenciométer (*24. ábra*) segítségével 0-tól 5 Voltig folyamatosan állíthatjuk a feszültséget, amit az analóg bemenet érzékel.

24. ábra: Potenciométer felépítése

Forrás: https://www.electrical4u.com/images/2018/december18/rotary-potentiometer.jpg

A kapcsolásunkban a potenciométer ellenállásának egyik sarka a +5V-ra, a másik a GND-re (0 Volt) lett kötve. A potenciométer csúszkája (csúszó érintkezője) az ellenállás felületén mozog a két sarok között, és a helyzetének megfelelő feszültséget mérhetünk rajta. A potenciométer szélső helyzeteiben tehát 0 és 5 Voltot mérhetünk, közötte pedig a csúszka helyzetével arányosan a két szélsőérték közöttit (*25. ábra*).

25. ábra: Potenciométer működési elve

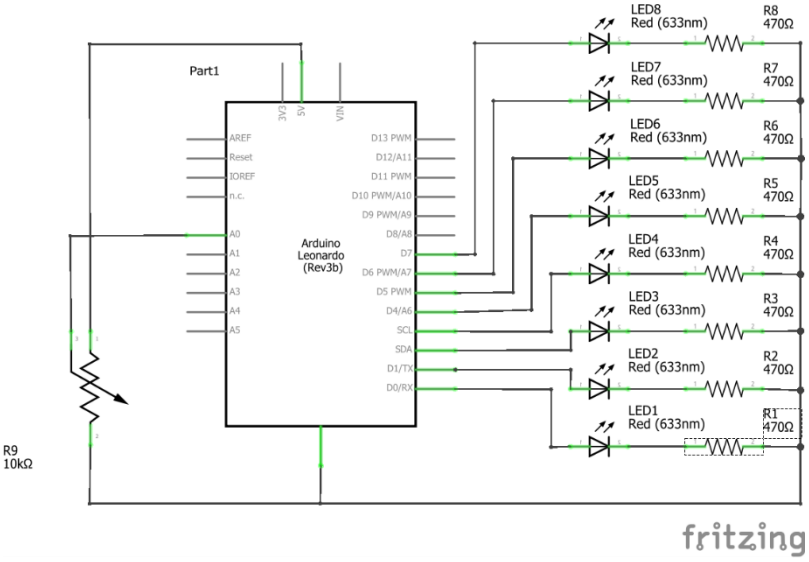


Forrás: https://www.electrical4u.com/images/2018/december18/potentiometer-circuit.jpg alapján

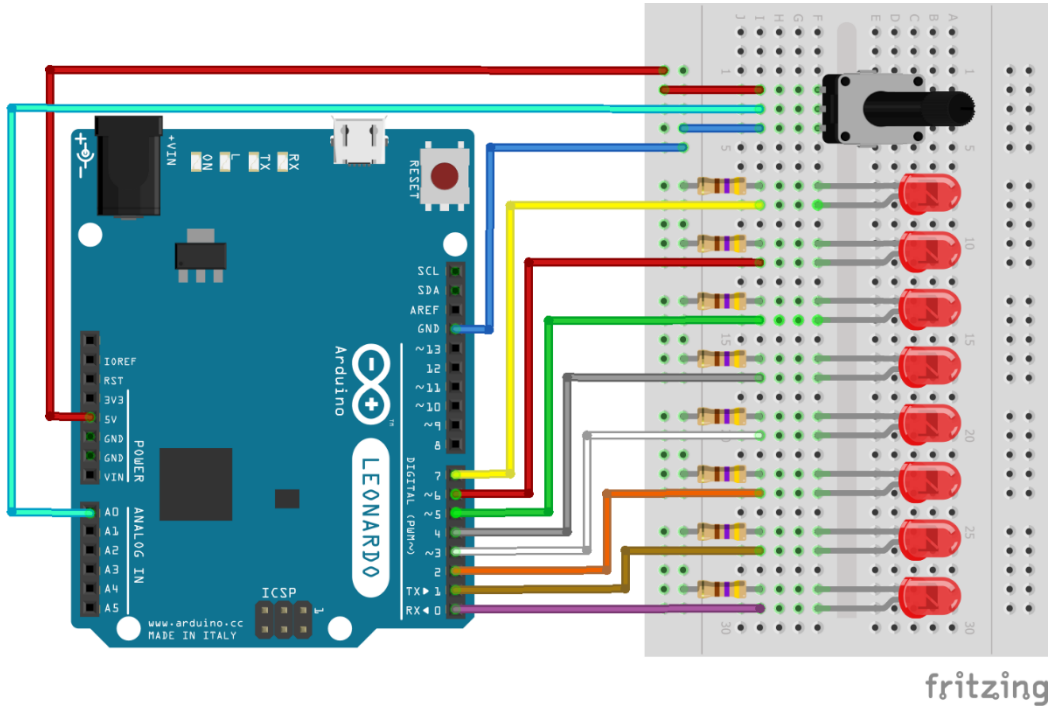
Az analóg bemenet érzékeli a rajta mérhető feszültséget, és azt egy a feszültséggel arányos számmá alakítja át, vagyis minden feszültség értékhez egy meghatározott szám tartozik. A szám 0 és 1023 közötti érték lehet, 0 Volthoz tartozik a 0, míg az 5 Volthoz az 1023 érték. Ezt az átalakítást analóg-digitális (A/D) átalakításnak nevezik.

Az áramkörben az előző futófénykapcsolást most egy potenciométerrel egészítjük ki (*26. ábra*), a generált véletlenszám helyett ezzel állítjuk be az időzítést.

26. ábra: LED-es futófény potenciométerrel kapcsolási rajza



27. ábra: LED-es futófény potenciométerrel megvalósítása próbapanelen



A kapcsolás elkészítése után írjuk meg hozzá a kódot, olvassuk be az A0 bemenet értékét, majd írjuk ki azt a soros monitoron, és legyen az érték a futófény késleltetési ideje milliszekundumban. Valójában a potenciométerrel a futófény sebességét tudjuk így vezérelni. A kapcsoláshoz írt kódban látható, hogy `setup()` függvényben az A0 bemenethez nem tartozik `PinMode()` függvény. Az A0-A5 alaphelyzetben analóg bemenet, ezért ezt nem kell újra megadni. Az előzőekhez képest csak az `analogRead(pin)` függvény újdonság, ami egy 0 és 1023 közötti egész számot ad vissza eredményként, amit egy `int` típusú változóban tárolunk.

8. kód

```
void setup() {

  for (int i = 0; i < 8; i++) {
    pinMode(i, OUTPUT);
  }
  Serial.begin(9600);
}

void loop() {

  int analog_bemenet;

  analog_bemenet = analogRead(A0);  //analóg bemenet beolvasása
  Serial.println(analog_bemenet);   //kiírás soros monitoron
  elore(analog_bemenet);
  hatra(analog_bemenet);
}

void elore(int wait) {
  for (int i = 0; i < 8; i++) {
    digitalWrite(i, HIGH);
    delay(wait);
    digitalWrite(i, LOW);
    delay(wait);
  }
}

void hatra(int wait) {
  for (int i = 7; i >= 0; i--) {
    digitalWrite(i, HIGH);
    delay(wait);
    digitalWrite(i, LOW);
    delay(wait);
  }
}
```

A következő példában a potenciométer azt vezérli, hogy hány LED-et kapcsoljon be az Arduino a kapcsolásban. Ehhez először a beolvasott értéket a 0–1023 tartományból át kell számítani a 0–7 tartományba, amit a következő sor ír le:

```
ertek = analog_bemenet * (8.0 / 1023.0);
```

A 8.0/1023.0 hányados megadja a két skála arányát, ezt megszorozva a mért bemeneti értékkel megkapjuk a 0–7 skálán a bemenetnek megfelelő értéket. A számolás közben típuskonverziót is végre kell hajtani, hiszen a hányados eredménye nem egész, hanem tizedes

tört. Ezt a törtet egésszel szorozva még mindig nem egészet kapunk, az eredmény tartalmaz tizedes értékeket, amelyek az int típusú változóból egyszerűen elmaradnak.
A LED-ek közül az `ertek` változónak megfelelő számút bekapcsolunk, a többit pedig ki. Ezt két `for` ciklussal tudjuk megtenni:

```
  for (int i = 0; i < ertek; i++) digitalWrite(i, HIGH);
  for (int i = ertek; i < 8; i++) digitalWrite(i, LOW);
```

Az első ciklus az `ertek` változóig számol el és kapcsolja be a LED-eket, a második pedig kikapcsolja a többit.

A soros monitorra történő kiírást is nézzük át:

```
  Serial.print(analog_bemenet);
  Serial.print("\t");
  Serial.println(ertek);
```

Először az `analog_bemenet` változóértékét írjuk ki. Ezután a `Serial.print("\t");` utasítás ugyanabba a sorba egy tabulátort (8 üres karaktert) „ír ki”, hogy a kiírt számokat így válassza el egymástól. Végül az 1–7 tartományba átszámolt `ertek` változót írjuk ki úgy, hogy a következő kiírás új sorba kerüljön. Az `ertek` változónak megfelelő számú LED kell, hogy világítson.

9. kód

```
void setup() {

  for (int i = 0; i < 8; i++) {
    pinMode(i, OUTPUT);
  }
  Serial.begin(9600);
}

void loop() {

  int analog_bemenet;
  int ertek;

  analog_bemenet = analogRead(A0);
  ertek = analog_bemenet * (8.0 / 1023.0);
  Serial.print(analog_bemenet);
  Serial.print("\t");
  Serial.println(ertek);

  for (int i = 0; i < ertek; i++) digitalWrite(i, HIGH);
  for (int i = ertek; i < 8; i++) digitalWrite(i, LOW);
}
```

LED-ek fényerejének szabályozása

Ebben a gyakorlatban az elektromos eszközök szabályozására alkalmazható PWM (pulse-width modulation), magyarul impulzusszélesség-moduláció technikát alkalmazzuk.

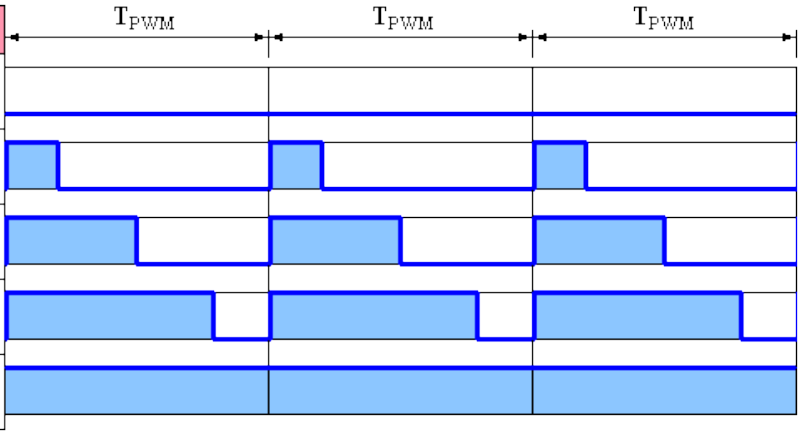
Az eddigi gyakorlatainkban a LED-eket vagy bekapcsoltuk és teljes fényerővel világítottak, vagy nem kapcsoltunk rájuk feszültséget, és így egyáltalán nem világítottak. Ha olyan gyorsan tudnánk ki- és bekapcsolni a LED-eket, hogy a két állapotot a szemünk már ne tudja megkülönböztetni, akkor úgy tűnne, mintha a LED nem világítana teljes fényerővel. Ha a LED bekapcsolt állapota hosszabb, mint a kikapcsolt állapot, akkor fényesebb lesz, ha a kikapcsolt állapota hosszabb, akkor pedig halványabb lesz a fénye.

Ezt az eljárást hívják PWM-nek. A LED-re kapcsolt elektromos feszültséget egy kapcsoló gyors ütemű be- és kikapcsolásával szabályozzák. Minél hosszabb ideig van a kapcsoló a bekapcsolt állapotban a kikapcsolt állapot időtartamához képest, annál nagyobb lesz a LED fényereje. A be- és kikapcsolásnak nagyon gyorsnak kell lennie, az Arduino esetén ez a PWM képes kivezetések legtöbbjén 488 Hz, azaz 1 másodperc alatt 488 alkalommal ismétlődik meg a be és kikapcsolás, 488 periódus zajlik le. Ebből kiszámítható, hogy egy periódus mennyi ideig tart. A periódusidő egyenlő a frekvencia reciprokával: $T_{PWM}=1/f$. A 488 Hz frekvencia megközelítően 2 ezredmásodperc periódusidőnek felel meg.

A PWM-jeleket a frekvencián kívül a kitöltési tényező, kitöltési arány jellemzi. Ez azt „mondja meg”, hogy egy periódus (egy be- és egy kikapcsolás) alatt hány százalékban volt magas (high, 1, +5V) szintű a jel. A *28. ábrán* különböző kitöltési arányú PWM-jeleket láthatunk. A 0% kitöltési arány kikapcsolt állapotot jelent, a 100% pedig bekapcsoltat. A LED-re 0%-os PWM esetén 0V feszültség jut, 100%-osnál pedig +5V, a kettő közötti kitöltési arányoknál pedig 0 és 5V közötti átlagfeszültségnek megfelelő feszültség mérhető.

28. ábra: Átlagfeszültség különböző kitöltési tényezőknél

Kitöltési arány	Átlagfeszültség
0%	0V
20%	1V
50%	2,5V
80%	4V
100%	5V



Forrás: http://www.t-es-t.hu/elokep/pic/felhkk/kk/16/pwm01.gif

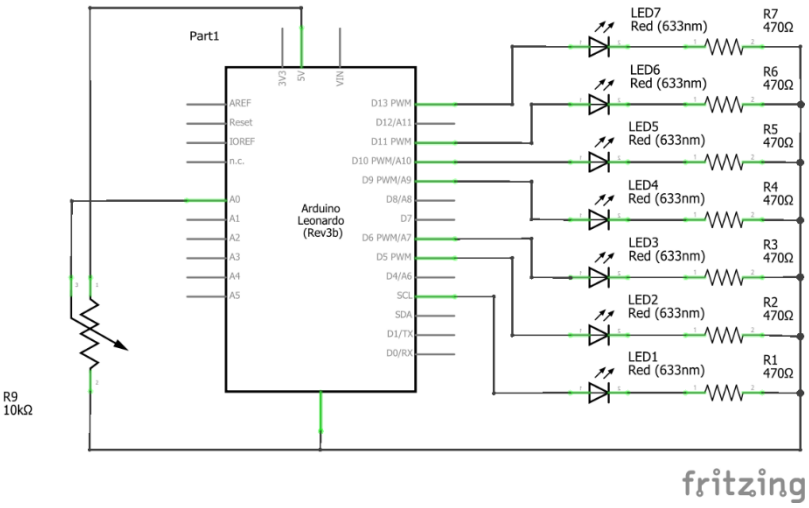
Az Arduino Leonardo lapkán a 3, 5, 6, 9, 10 és 11 lábakon állítható be PWM-jel, amelynek frekvenciája az 5. és 6. lábon 976 Hz, a többin, a már említett 488 Hz. A PWM-lábakat az alaplapon a ~ jellel látják el (*1. ábra*).

Az Arduino programozása során az `analogWrite(pin, value)` függvénnyel állíthatunk elő PWM-jelet. A függvénynek két paramétert kell megadni: a lábszámot (`pin`) és a kitöltési tényezőt (`value`).

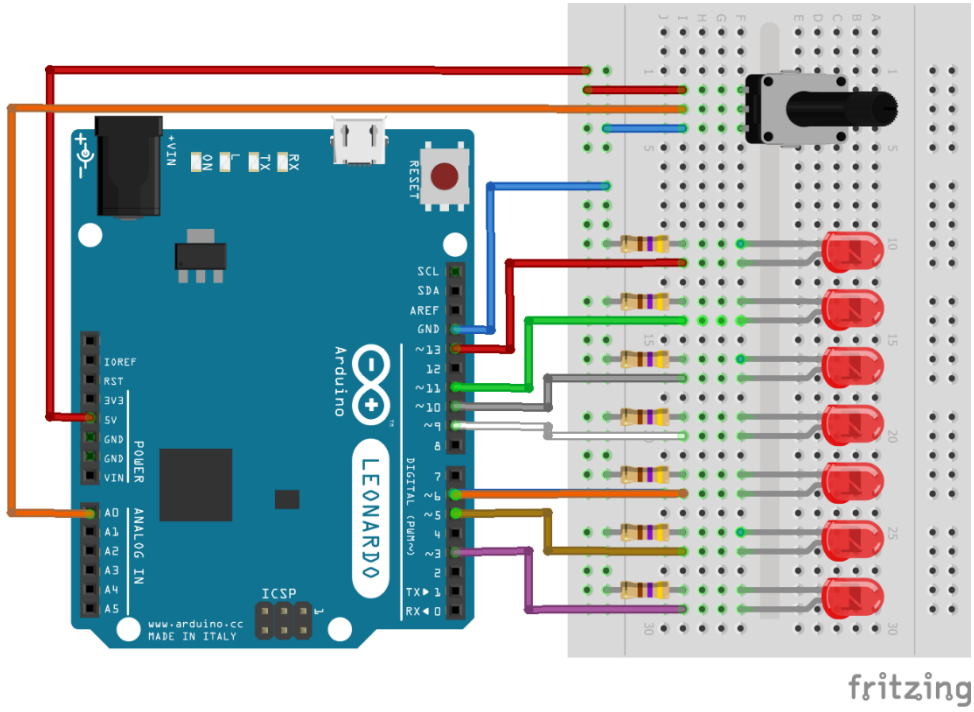
A kitöltési tényezőt egy 0 és 255 közötti értékkel adhatjuk meg. A `value = 0`, azaz 0%-os, a `value = 127` – 50%-os, a `value = 255` pedig 100%-os kitöltési tényezőt állít be.

Az előző áramkörünkön kell egy kicsit változtatnunk. Mivel csak hét PWM-láb van, ezért egy LED-et és egy ellenállást ki kell vennünk, majd a LED-eket sorban a PWM-lábakra kell kötnünk. A kapcsolással a LED-ek fényességét tudjuk változtatni a potenciométerrel. A próbapaneles kapcsoláson jól megfigyelhető a PWM-lábak száma mellet a ~ jel.

29. ábra: LED-fényerő szabályozására alkalmas kapcsolás



30. ábra: LED-fényerőt szabályozó kacsolás megvalósítása



Mivel a PWM-lábak nem egymás mellettiek, így a `for` ciklust sem tudjuk használni az eddigi módon. Mivel mégsem szeretnénk egyesével beállítani a lábakat, ezért létrehozunk számukra egy *tömböt*.

A tömbökben több összetartozó, azonos típusú értéket tárolhatunk. A tömbnek nevet adunk, és a benne lévő elemekre a tömbön belüli sorszámukkal hivatkozhatunk. A sorszámozás C nyelven mindig 0-val kezdődik.

A lábszámokat tartalmazó tömb megadása a következő:

```
int PWM_pins[7] = {3, 5, 6, 9, 10, 11, 13};
```

A `PWM_pins[7]` nevű tömb egész (`integer`) típusú elemeket tartalmaz, szám szerint hetet. A tömböt az egyenlőség után, kapcsos zárójelek között „feltöltjük" a PWM-kimenetek lábszámaival. A tömb értékeire a sorszámukkal lehet hivatkozni, az első elem sorszáma 0, a másodiké 1 stb. A példában a 3-as sorszámú elem értéke 9. Az így létrehozott tömb elemeire most már használhatjuk a `for` ciklust:

```
  for (int i = 0; i < 7; i++) {
    pinMode(PWM_pins[i], OUTPUT);
```

A ciklusban az `i` ciklusváltozó 0 és 6 közötti értékeket vesz fel, és a `PWM_pins` tömb `i`-edik elemében megadott lábat kimenetre állítja.

Az alábbi programban a beolvasott analóg értéket először átalakítjuk egy 0 és 255 közötti számra, majd azzal beállítjuk az összes PWM láb kitöltési tényezőjét.

10. kód

```
int PWM_pins[7] = {3, 5, 6, 9, 10, 11, 13};

void setup() {
  for (int i = 0; i < 7; i++) {
    pinMode(PWM_pins[i], OUTPUT);
  }
}

void loop() {
  int ertek;

  ertek = analogRead(A0) * (255.0 / 1023.0);

  for (int i = 0; i < 7; i++)
    analogWrite(PWM_pins[i], ertek);
}
```

Figyeljük meg, hogy a tömböt a `setup()` és a `loop()` függvényen kívül deklaráltuk, így azt mind a két függvény eléri. Az `érték` nevű változót a `loop`() függvényen belül deklaráltuk, így azt csak ott használhatjuk. A `PWM_pins[7]` tömb globális változó az `érték` pedig lokális változó.

Hétszegmenses kijelző

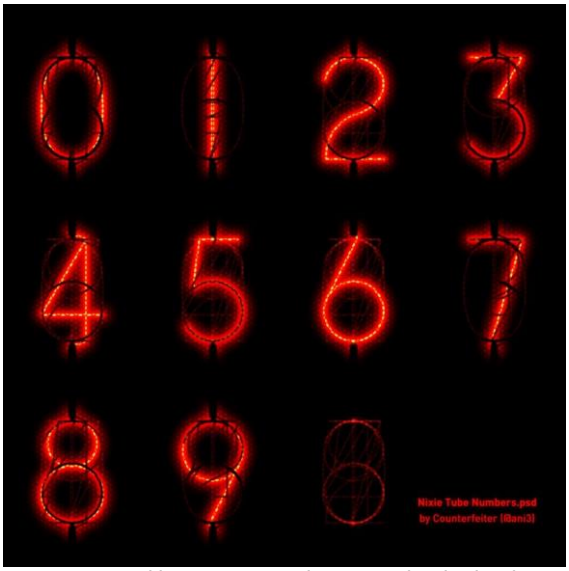
Az elektronika fejlődése során felmerült az igény, hogy a fizikai mennyiségeket ne csak analóg módon, mutatós műszerekkel, hanem számokkal is ki lehessen jelezni. Az elektroncsövek korszakában ezt számkijelző csövekkel, Nixie-csövekkel oldották meg (*31. ábra*).

31. ábra: Nixie cső

Forrás: https://upload.wikimedia.org/wikipedia/commons/thumb/6/68/ZM1210-operating_edit2.jpg/200px-ZM1210-operating_edit2.jpg

Egy zárt üvegcsőben egy dróthálóból készült anód és több szám, betű vagy egyéb szimbólum alakú katód található, amelyeket egy speciális kis nyomású gázkeverék vesz körül. A katód tetszőleges alakú lehetne, de a csöveket általában a számjegyek és a tizedespont kijelzésére használták (*32. ábra*). Ha a kiválasztott katódra feszültséget kapcsolunk, akkor körülötte gázkisülés alakul ki, ami színes fényt bocsát ki, így kirajzolva a szimbólumot. A kijelző színe a gázkeverék összetételétől függ, ami általában narancssárga és vörös között változik.

32. ábra: Nixie csővel kiírt számok



Forrás: https://i.pinimg.com/originals/ce/f8/cb/cef8cb2e4dec857ee5c58a1eece47865.jpg

A LED-ek megjelenése után lehetővé vált egyszerűbb, kisebb fogyasztású kijelző készítése. A *33. ábrán* egy hétszegmenses kijelző látható. Azért hívjuk ezt az eszközt hétszegmenses kijelzőnek, mert a nyolcas alakzatot hét részből, hét szegmensből állítottuk össze. (A szegmens jelentése: szakasz, rész, darab, szelet, szelvény.)

33. ábra: Hétszegmenses kijelző

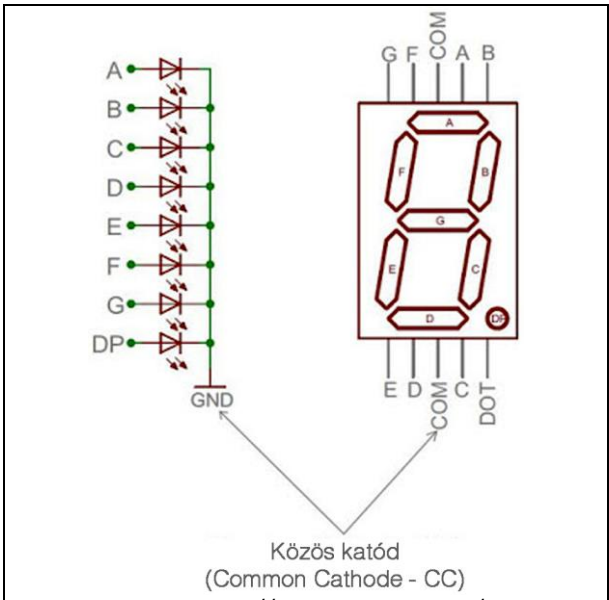
Forrás: http://www6.plala.or.jp/sands/dsc0004.jpg

Ilyen kijelzőt magunk is tudnánk készíteni. Az előző példákban használt nyolc LED negatív (katód) vagy pozitív (anód) lábait összekötve közös katódos (Common Cathode – CC) vagy közös anódos (Common Anode – CA) kapcsoláshoz juthatunk. Az összekötött hét LED-et az *33. ábra* szerint elrendezve egy 8-as számot formázó alakzatot, a fennmaradó nyolcadik LED-ből pedig egy tizedespontot lehet kialakítani. Azért nem tesszük ezt, mert számtalan méretben, színben kapható a kereskedelemben hétszegmenses kijelző.

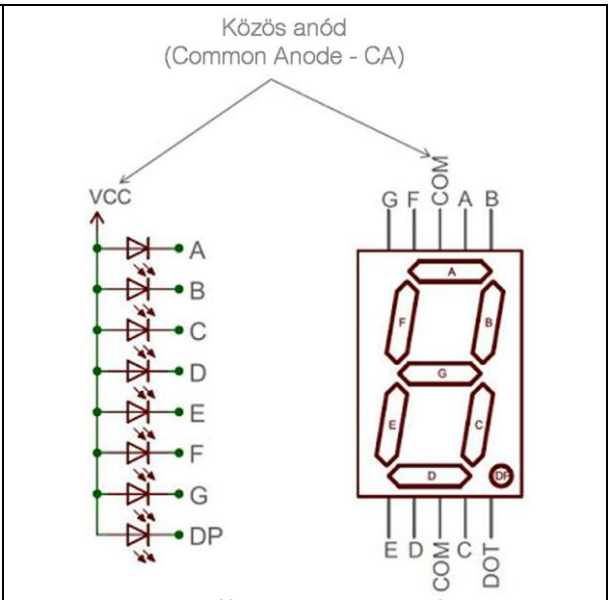
A kialakított szegmenseket szabvány szerint az A, B, C, D, E, F, G betűkkel jelölik, a tizedespont (dot point) jelölése DP vagy DOT. A közös katódot vagy anódot a COM (common – közös) felirat jelöli.

A hét LED-et egy tokba építik, és csak a bekötéshez szükséges lábakat vezetik ki. A kijelző belső kapcsolása és a tok bekötési rajza látható a *34. ábrán*.

34. ábra: A közös katódos és a közös anódos kijelző felépítése



Forrás: https://4.bp.blogspot.com/-PSFKPl2ebCM/VcPC5o3dnzI/AAAAAAAABQE/JMz-voKxY0vA/s400/CC.JPG alapján



Forrás: https://4.bp.blogspot.com/-BIlkLap-gXr0/VcPC6_XDCxI/AAAAAAAABQQ/PzEt-llv1oVo/s400/CA.JPG alapján

A kijelző alkalmas az összes szám és több betű megjelenítésére is (*35. ábra*), a gyakorlatban elsősorban a számok kijelzésére használják. Elterjedését az egyszerű felépítésének, jó láthatóságának és olcsóságának köszönheti.

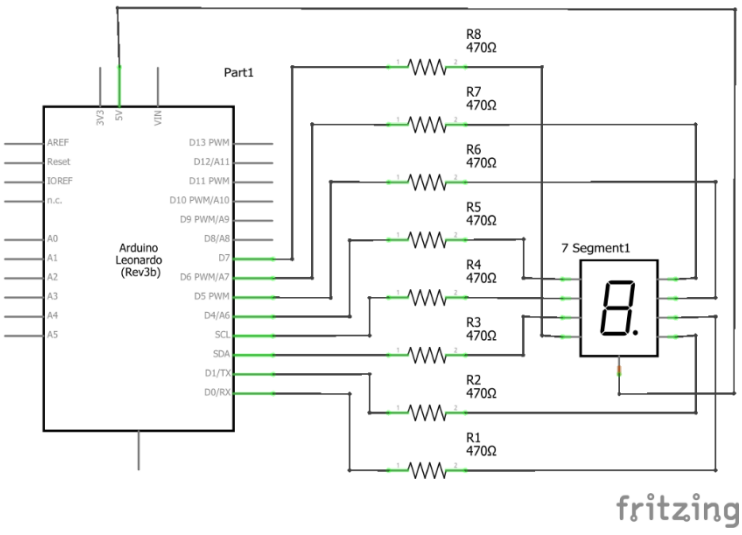
35. ábra: A hétszegmenses kijelzővel megjelenített számok és betűk

Forrás: https://lastminuteengineers.com/wp-content/uploads/2018/06/7-Segment-Display-Number-Formation-Segment-Contol.png

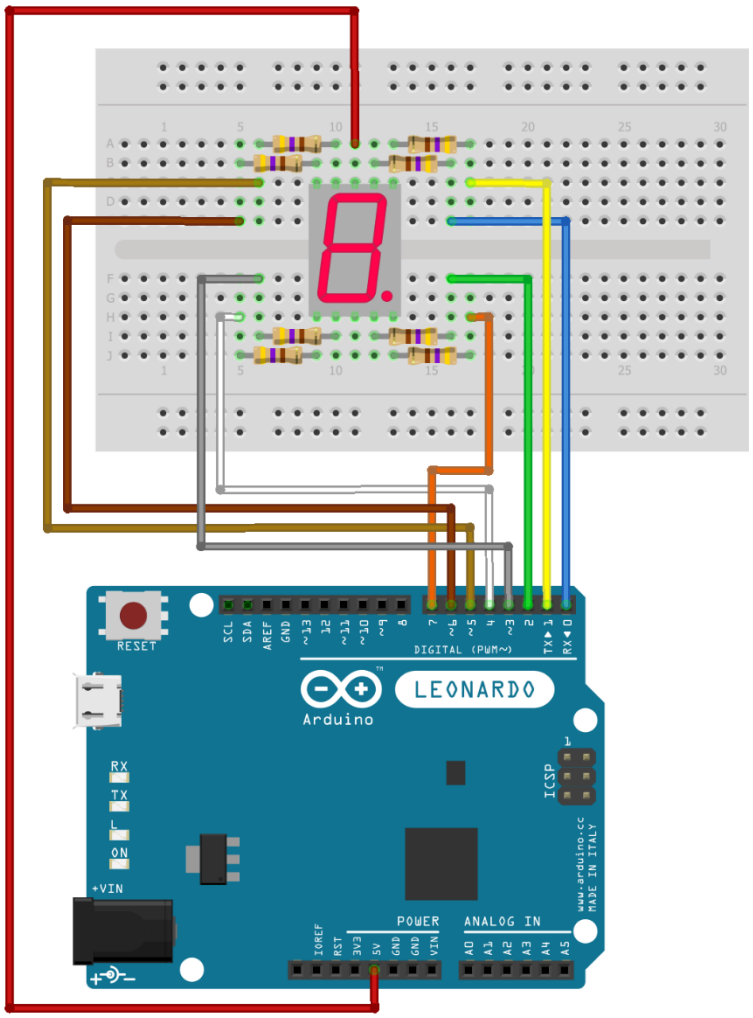
Az Arduinóval vezérelt hétszegmenses kijelző kapcsolásunk nagyon hasonlít a futófénykapcsolásra (*36–39. ábra*). Itt is a D0–D7 kimeneteket használjuk, amihez sorban az A–G szegmenseket és végül a DP-t kötjük egy-egy 470 ohmos ellenálláson keresztül. A kapcsolásunkban arra ügyelni kell, hogy a közös katódos (CC) és a közös anódos (CA) kijelző bekötése eltérő! A COM (közös) kivezetést közös katódos kijelző esetén az Arduino GND-lábára, míg közös anódos kijelzőnél a VCC (+5V) lábára kell kötni! Ez apró, de nagyon fontos eltérés.

A futófény kapcsolásnál a LED-ek katódjait kötöttük össze, tehát az közös katódos kapcsolásnak felel meg. Abban eltér a két áramkör, hogy a futófénynél a 470 ohmos ellenállás a LED-ek katódja és a GND közé lett kötve, itt viszont nem. Az áramkör ugyanúgy működne akkor is, ha a LED-ek anódjára kötnénk az ellenállásokat, majd azokat a D0–D7 kimenetre. A lényeg, hogy az ellenállások sorba legyenek kötve LED-ekkel.

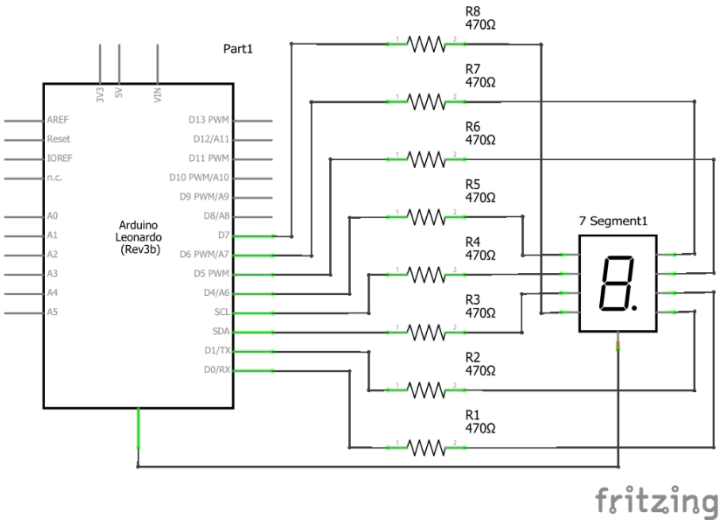
36. ábra: Közös anódos kapcsolás



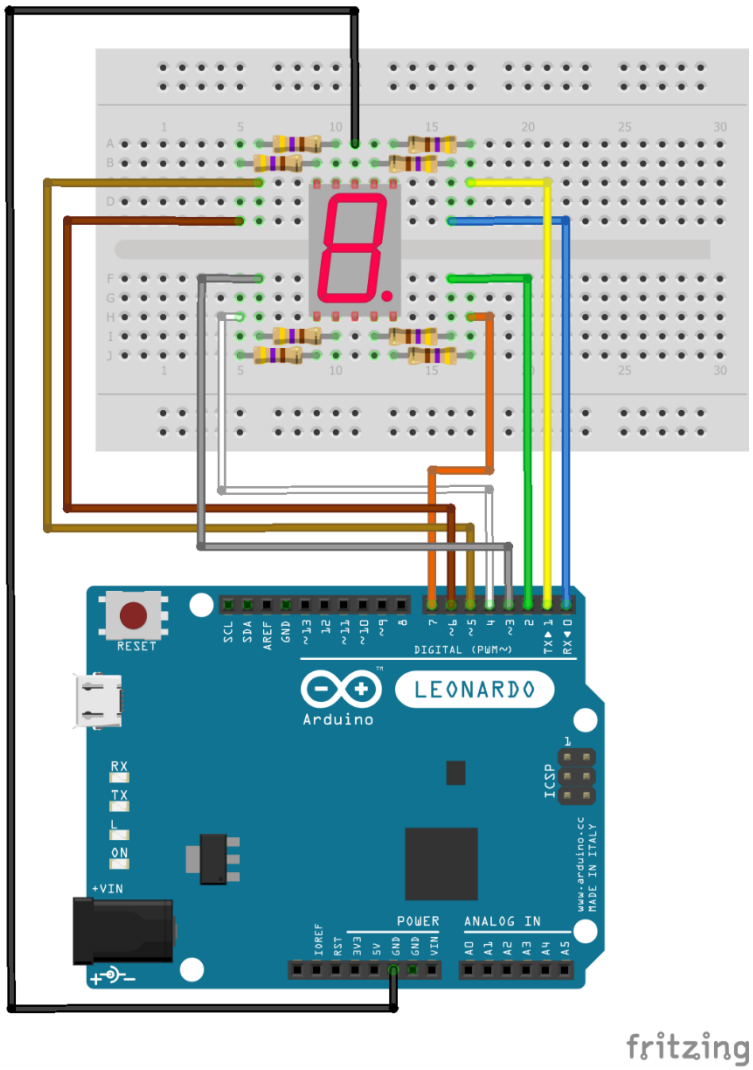
37. ábra: Közös anódos kapcsolás próbapanelen



38. ábra: Közös katódos kapcsolás



39. ábra: Közös katódos kapcsolás próbapanelen



A kapcsolás működésének kipróbálására használhatjuk az első futófényhez írt kódot is. Ha közös katódos kijelzőnk van, akkor sorban bekapcsolnak (világítanak), majd kikapcsolnak a szegmensek A-tól kezdődően a DP-ig. Közös anódos kijelzőnél éppen fordított a helyzet, először minden LED világít, majd sorban kikapcsolnak, és újra bekapcsolnak a szegmensek. Ez a kód alkalmas a rossz bekötés felderítésére és javítására. Ha a szegmenseket rosszul kötöttük be, akkor a LED-ek nem a szegmensek sorrendjében kapcsolnak be.

Közös katódos kapcsolás esetén az alábbi kód bekapcsolja az összes szegmenst, majd fél másodperc várakozás után kikapcsolja azokat, ezután vár egy másodpercet, és az egész kezdődik elölről. A közös anódosnál ez pont fordítva történik.

11. kód

```
void setup() {

  for (int i = 0; i < 8; i++) {
    pinMode(i, OUTPUT);
    digitalWrite(i, HIGH);
  }
}
```

```
void loop() {

  for (int i = 0; i < 8; i++) {
    digitalWrite(i, 0);
    delay(500);
    digitalWrite(i, 1);
    delay(500);
  }
  delay(1000);
```

A kipróbálás után jelezzük ki a számokat 0-tól 9-ig. Ehhez szükséges felírni a kijelző igazságtáblázatát, vagyis azt, hogy az egyes szegmensek közül melyiknek kell bekapcsolt, illetve kikapcsolt állapotúnak lenni a számok megjelenítéshez. Az *1. táblázat* a közös katódos kijelző igazságtábláját mutatja. A számjegyek megjelenítéséhez a szükséges szegmenseket bekapcsoljuk, a kimenetre HIGH értéket kapcsolunk, a sötét szegmensekre pedig LOW értéket, a tizedespontot nem kapcsoljuk be, így az mindig LOW értékű. A közös anódos kijelző igazságtáblájában a HIGH helyet LOW, a LOW helyett pedig HIGH érték szerepelne.

1. táblázat: A közös katódos, hétszegmenses kijelző igazságtáblája

	a	b	c	d	e	f	g	dp
0	H	H	H	H	H	H	L	L
1	L	H	H	L	L	L	L	L
2	H	H	L	H	H	L	H	L
3	H	H	H	H	L	L	H	L
4	L	H	H	L	L	H	H	L
5	H	L	H	H	L	H	H	L
6	H	L	H	H	H	H	H	L
7	H	H	H	L	L	L	L	L
8	H	H	H	H	H	H	H	L
9	H	H	H	H	L	H	H	L

H = HIGH, L = LOW

A `setup()` függvény nem tartalmaz újdonságot. A számok kijelzése egyesével történik a `digitalWrite()` függvénnyel beállítva az egyes szegmensek értékeit HIGH vagy LOW értékre az 1. táblázatnak megfelelően. A kevesebb gépelés érdekében HIGH helyett 1-et, a LOW helyett 0-át írunk. Közös anódos kijelző esetén csak fel kellene cserélnünk az értékeket.

12. kód

```
void setup() {
  for (int i = 0; i < 8; i++) {
    pinMode(i, OUTPUT);
  }
}

void loop() {
  int wait = 500;   //Várakozási idő megadása

  // nulla
  digitalWrite(0, 1);  //a
  digitalWrite(1, 1);  //b
```

```
digitalWrite(2, 1);  //c
digitalWrite(3, 1);  //d
digitalWrite(4, 1);  //e
digitalWrite(5, 1);  //f
digitalWrite(6, 0);  //g
digitalWrite(7, 0);  //DP
delay(wait);

// egy
digitalWrite(0, 0);  //a
digitalWrite(1, 1);  //b
digitalWrite(2, 1);  //c
digitalWrite(3, 0);  //d
digitalWrite(4, 0);  //e
digitalWrite(5, 0);  //f
digitalWrite(6, 0);  //g
digitalWrite(7, 0);  //DP
delay(wait);

// kettő
digitalWrite(0, 1);  //a
digitalWrite(1, 1);  //b
digitalWrite(2, 0);  //c
digitalWrite(3, 1);  //d
digitalWrite(4, 1);  //e
digitalWrite(5, 0);  //f
digitalWrite(6, 1);  //g
digitalWrite(7, 0);  //DP
delay(wait);

// három
digitalWrite(0, 1);  //a
digitalWrite(1, 1);  //b
digitalWrite(2, 1);  //c
digitalWrite(3, 1);  //d
digitalWrite(4, 0);  //e
digitalWrite(5, 0);  //f
digitalWrite(6, 1);  //g
digitalWrite(7, 0);  //DP
delay(wait);

// négy
digitalWrite(0, 0);  //a
digitalWrite(1, 1);  //b
digitalWrite(2, 1);  //c
digitalWrite(3, 0);  //d
digitalWrite(4, 0);  //e
digitalWrite(5, 1);  //f
digitalWrite(6, 1);  //g
digitalWrite(7, 0);  //DP
delay(wait);

// öt
digitalWrite(0, 1);  //a
digitalWrite(1, 0);  //b
digitalWrite(2, 1);  //c
digitalWrite(3, 1);  //d
digitalWrite(4, 0);  //e
digitalWrite(5, 1);  //f
digitalWrite(6, 1);  //g
digitalWrite(7, 0);  //DP
delay(wait);
```

```
  // hat
  digitalWrite(0, 1);  //a
  digitalWrite(1, 0);  //b
  digitalWrite(2, 1);  //c
  digitalWrite(3, 1);  //d
  digitalWrite(4, 1);  //e
  digitalWrite(5, 1);  //f
  digitalWrite(6, 1);  //g
  digitalWrite(7, 0);  //DP
  delay(wait);

  // hét
  digitalWrite(0, 1);  //a
  digitalWrite(1, 1);  //b
  digitalWrite(2, 1);  //c
  digitalWrite(3, 0);  //d
  digitalWrite(4, 0);  //e
  digitalWrite(5, 0);  //f
  digitalWrite(6, 0);  //g
  digitalWrite(7, 0);  //DP
  delay(wait);

  // nyolc
  digitalWrite(0, 1);  //a
  digitalWrite(1, 1);  //b
  digitalWrite(2, 1);  //c
  digitalWrite(3, 1);  //d
  digitalWrite(4, 1);  //e
  digitalWrite(5, 1);  //f
  digitalWrite(6, 1);  //g
  digitalWrite(7, 0);  //DP
  delay(wait);

  // kilenc
  digitalWrite(0, 1);  //a
  digitalWrite(1, 1);  //b
  digitalWrite(2, 1);  //c
  digitalWrite(3, 1);  //d
  digitalWrite(4, 0);  //e
  digitalWrite(5, 1);  //f
  digitalWrite(6, 1);  //g
  digitalWrite(7, 0);  //DP
  delay(wait);
}
```

Látjuk, hogy így a kód elég hosszú és nehezen olvasható. Mivel a DP szegmens mindig kikapcsolt, így felesleges minden számhoz beleírnunk a `digitalWrite(7, 0); //DP` sort, azt elég lenne csak a setup() függvénybe egyszer beírni.

Jelentősebb egyszerűsítésre az ad lehetőséget, hogy a kijelző állapotait egy tömbbe foglalhatjuk. A tömbre már láttunk példát, a PWM-kódnál, ahol egydimenziós tömböt készítettünk, az adatok egy sorban voltak. Most is így járunk el, minden számjegyhez készítünk egy tömböt, amelyben a bekapcsolt számjegy szegmenseinek adatait tároljuk. A könnyebb olvashatóság miatt definiáljuk a `BE` és a `KI` konstansokat, amelyek értékét attól függően állítjuk 1 vagy 0 értékre, hogy közös katódos vagy közös anódos kijelzőt használunk. Ezenkívül létrehozunk minden számjegyhez egy függvényt, amelyik az adott tömb értékeinek megfelelően kapcsolja be a szükséges szegmenseket. Tömb használatával már alkalmazhatunk

ciklust, ami jelentősen leegyszerűsíti a kódot. A függvények csak abban különböznek, hogy melyik tömbből veszik ki az értékeket.

13. kód

```
const int BE=1; // Ha közös katód, akkor az 1 a bekapcsolt (BE=1)
const int KI=0; // Ha közös anód, akkor a 0 a bekapcsolt (BE=0)

// A különböző számokat tároljuk egy tömbben,
// BE annak a szegmensnek az értéke, ami világít

                  //   a, b, c, d, e, f, g,dp
int nulla[8] =      { BE,BE,BE,BE,BE,BE,KI,KI};    // 0
int egy[8] =        { KI,BE,BE,KI,KI,KI,KI,KI};    // 1
int ketto[8] =      { BE,BE,KI,BE,BE,KI,BE,KI};    // 2
int harom[8] =      { BE,BE,BE,BE,KI,KI,BE,KI};    // 3
int negy[8] =       { KI,BE,BE,KI,KI,BE,BE,KI};    // 4
int ot[8] =         { BE,KI,BE,BE,KI,BE,BE,KI};    // 5
int hat[8] =        { BE,KI,BE,BE,BE,BE,BE,KI};    // 6
int het[8] =        { BE,BE,BE,KI,KI,KI,KI,KI};    // 7
int nyolc[8] =      { BE,BE,BE,BE,BE,BE,BE,KI};    // 8
int kilenc[8] =     { BE,BE,BE,BE,KI,BE,BE,KI};    // 9

void setup()
{
  for(int i=0 ; i<8; i++)
    {
     pinMode(i, OUTPUT);
    }
 }

void loop()
{
  int wait = 500;   //Várakozási idő megadása

  //itt adjuk meg, hogy a tömb melyik sorát kell kiírni a függvénnyel
  Display_Nulla();
  delay(WAIT);
  Display_Egy();
  delay(WAIT);
  Display_Ketto();
  delay(WAIT);
  Display_Harom();
  delay(WAIT);
  Display_Negy();
  delay(WAIT);
  Display_Ot();
  delay(WAIT);
  Display_Hat();
  delay(WAIT);
  Display_Het();
  delay(WAIT);
  Display_Nyolc();
  delay(WAIT);
  Display_Kilenc();
  delay(WAIT);
}
```

```
void Display_Nulla() // a függvény, ami kiírja a nullát
{
  for (int i=0; i < 8; i++)
    {
     digitalWrite(i, nulla[i]);
    }
}

void Display_Egy()
{
  for (int i=0; i < 8; i++)
    {
     digitalWrite(i, egy[i]);
    }
}

void Display_Ketto()
{
  for (int i=0; i < 8; i++)
    {
     digitalWrite(i, ketto[i]);
    }
}

void Display_Harom()
{
  for (int i=0; i < 8; i++)
    {
     digitalWrite(i, harom[i]);
    }
}

void Display_Negy()
{
  for (int i=0; i < 8; i++)
    {
     digitalWrite(i, negy[i]);
    }
}

void Display_Ot()
{
  for (int i=0; i < 8; i++)
    {
     digitalWrite(i, ot[i]);
    }
}

void Display_Hat()
{
  for (int i=0; i < 8; i++)
    {
     digitalWrite(i, hat[i]);
    }
}

void Display_Het()
{
  for (int i=0; i < 8; i++)
    {
     digitalWrite(i, het[i]);
    }
}
```

```
void Display_Nyolc()
{
  for (int i=0; i < 8; i++)
    {
     digitalWrite(i, nyolc[i]);
    }
}

void Display_Kilenc()
{
  for (int i=0; i < 8; i++)
    {
     digitalWrite(i, kilenc[i]);
    }
}
```

A kód egyszerűbb és jobban olvasható lett. További egyszerűsítésre is lehetőségünk nyílik. Az egydimenziós tömb helyett készíthetnénk egy olyan *kétdimenziós tömböt*, ahol egy sorban ismét egy számhoz tartozó adatok lennének, de a tömb most összesen tíz sort tartalmazna, minden sorban egy számhoz tartozó adatokkal.

A következő példában a kijelzővel tíz különböző számot akarunk megjeleníteni, 0-tól 9-ig. Ezenkívül szeretnénk kijelezni a hexadecimális számokat is. A hexadecimális számokról eddig nem esett szó, de már láttuk azokat a *35. ábrán*.

A *hexadecimális* (tizenhatos) *számrendszer* kulcsfontosságú az informatikában. A tizenhatos számrendszerben a `0, 1, 2, 3, 4, 5, 6, 7, 8, 9` számjegyeket és az `A, B, C, D, E, F` betűket használjuk, ezek segítségével ábrázoljuk a számokat 0-tól 15-ig, vagyis összesen 16 különböző értéket. A tizenhat érték miatt be kell vezetnünk újabb számjegyeket, amelyeket jobb híján `A, B, C, D, E, F` betűkkel jelölünk. A `0-9` számjegyek használata ugyanolyan, mint a tízes számrendszerben, a 10-et az `A`, a 11-et a `B`, a 12-t a `C`, a 13-at a `D`, a 14-et az `E`, és a 15-öt az `F` „számjegy” jelöli.

A bináris (kettes) számrendszer szintén kulcsfontosságú az informatikában. Láttuk már, hogy a digitális kivezetéseken 0 vagy 1 értéket lehet beállítani. Az áramkörökben nagyon könnyen értelmezhető ez a két különböző érték: nincs feszültség vagy van feszültség. Egy vezetéken két különböző érték lehet, 0 vagy 1. Kettő vezetéken már négy különböző érték lehet, mindegyiken kettő-kettő. Belátható, hogy három vezetéken 8, négy vezetéken 16, n vezetéken 2^n különböző kombináció állhat elő. Ezeket a kombinációkat jól le lehet írni a 0 és 1 számjegyekkel, vagyis a kettes (bináris) számrendszerbeli számokkal. Az is könnyen belátható, hogy a bináris számokkal nagy számokat nehézkes leírni, ezért alakult ki a hexadecimális számrendszer, amelyben a négyjegyű bináris számot egy számjeggyel meg lehet adni. A *2. táblázatban* 0-tól 16-ig szerepelnek a számok a decimális, bináris és hexadecimális számrendszerben kifejezve.

2. táblázat: A decimális, bináris és hexadecimális számok

Decimális számok	Bináris számok	Hexadecimá-lis számok
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

A 16 „számjegyen” kívül szeretnénk most kijelezni csak a tizedes pontot (DP) és az üres kijelzőt is, amikor nem világít egyik LED sem. Ennyi különböző értékhez összesen 18 soros kétdimenziós tömböt kell létrehoznunk. A két dimenziós tömböt az `int digit[18][8]` utasítással definiáljuk, a tömb 18 sorból áll, és minden sor nyolc elemet tartalmaz (a hét szegmens és a DP értékét). A számok kijelzésére az előző példából idemásoljuk a számokat kiíró függvények valamelyikét és egy kicsit átalakítjuk. Az új függvény megkapja, hogy a tömb hányadik sorából (`int sor`) kell vennie az értékeket, majd a `for` ciklusban a `digitalWrite(oszlop, digit[sor][oszlop]);` utasítás a megfelelő lábra kiírja a tömb megfelelő elemét. Ez azért lehetséges, mert a szegmenseket a 0–7 lábakra kötöttük be, így azok megegyeznek a tömb oszlopainak sorszámával.

```
void Display_Nulla()
{
  for (int i=0; i < 8; i++)
    {
     digitalWrite(i, nulla[i]);
    }
}
```

```
void Display_Segment(int sor)
{
  for (int oszlop=0; oszlop < 8; oszlop++)
    {
     digitalWrite(oszlop, digit[sor][oszlop]);
    }
}
```

A teljes kód:

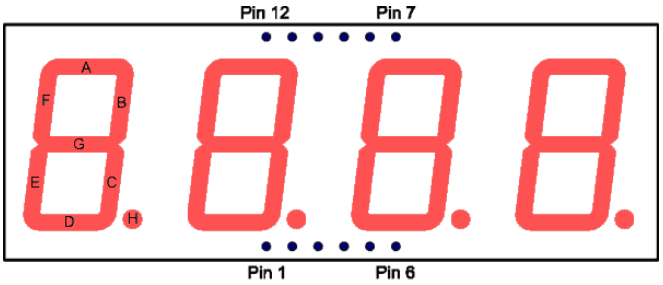
14. kód

```
const int BE=1;
const int KI=0;

// A különböző számokat tároljuk egy tömbben,
// BE annak a szegmensnek az értéke, ami világít

//                      a, b, c, d, e, f, g,dp
int digit[18][8] = {
                      {BE,BE,BE,BE,BE,BE,KI,KI},     // 0
                      {KI,BE,BE,KI,KI,KI,KI,KI},     // 1
                      {BE,BE,KI,BE,BE,KI,BE,KI},     // 2
                      {BE,BE,BE,BE,KI,KI,BE,KI},     // 3
                      {KI,BE,BE,KI,KI,BE,BE,KI},     // 4
                      {BE,KI,BE,BE,KI,BE,BE,KI},     // 5
                      {BE,KI,BE,BE,BE,BE,BE,KI},     // 6
                      {BE,BE,BE,KI,KI,KI,KI,KI},     // 7
                      {BE,BE,BE,BE,BE,BE,BE,KI},     // 8
                      {BE,BE,BE,BE,KI,BE,BE,KI},     // 9
                      {BE,BE,BE,KI,BE,BE,BE,KI},     // A
                      {KI,KI,BE,BE,BE,BE,BE,KI},     // B
                      {BE,KI,KI,BE,BE,BE,KI,KI},     // C
                      {KI,BE,BE,BE,BE,KI,BE,KI},     // D
                      {BE,KI,KI,BE,BE,BE,BE,KI},     // E
                      {BE,KI,KI,KI,BE,BE,BE,KI},     // F
                      {KI,KI,KI,KI,KI,KI,KI,BE},     // Tizedespont
                      {KI,KI,KI,KI,KI,KI,KI,KI}      // Üres kijelző
                    };

void setup()
{
  for(int i=0 ; i<8; i++)
    {
     pinMode(i, OUTPUT);
    }
 }

void loop()
{

  //itt adjuk meg, hogy a tömb melyik sorát kell kiírni a függvénnyel
  for (int sor = 0; sor < 18; sor++)
    {
     delay(1000);
     Display_Segment(sor); //átadjuk a függvénynek a sor számát
    }
  delay(2500);
}

void Display_Segment(int sor)  //a függvény, ami kiír egy értéket
{
  for (int oszlop=0; oszlop < 8; oszlop++)
    {
     digitalWrite(oszlop, digit[sor][oszlop]);
    }
}
```

Négydigites hétszegmenses kijelző

Ha a decimális 9-nél vagy a hexadecimális 15-nél nagyobb számot szeretnénk megjeleníteni, akkor több számjegyre (digitre) van szükségünk. Leggyakoribb a 2 és 4 digites kialakítás. A kijelzőkön az egyes digitek közös lába (közös katód vagy anód) is ki van vezetve, emiatt a kivezetések száma több lesz, mint az egydigites kialakításnál. Négydigites kialakításnál a hét szegmensen és a tizedesponton kívül még a digitek közös pontjait is ki kell vezetni, emiatt legalább 12 lábú egy ilyen áramkör (*40. ábra*). Egy kijelző itt is közös anódos vagy közös katódos lehet (*41. ábra*).

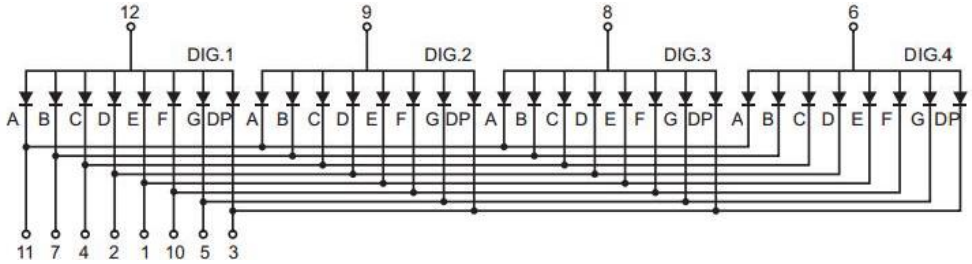
40. ábra: Négydigites hétszegmenses kijelző



Forrás: https://makbit.com/web/wp-content/uploads/2016/01/LED4x7Pins.png

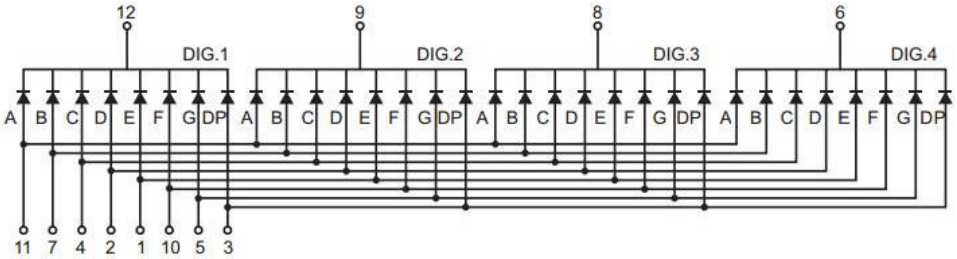
41. ábra: A négydigites kijelzők belső kapcsolási rajza és lábszámozása

Közös anód (Common Anode -CA)



Forrás: https://osoyoo.com/wp-content/uploads/2017/08/anode.jpg

Közös katód (Common Cathode - CC)

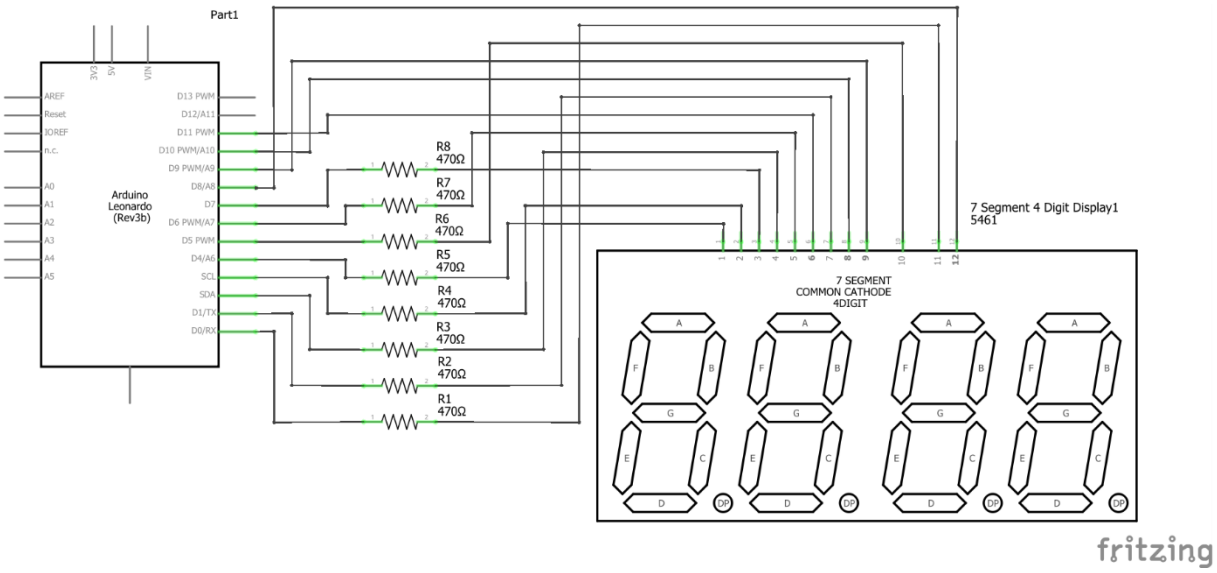


Forrás: https://osoyoo.com/wp-content/uploads/2017/08/cathnode.jpg

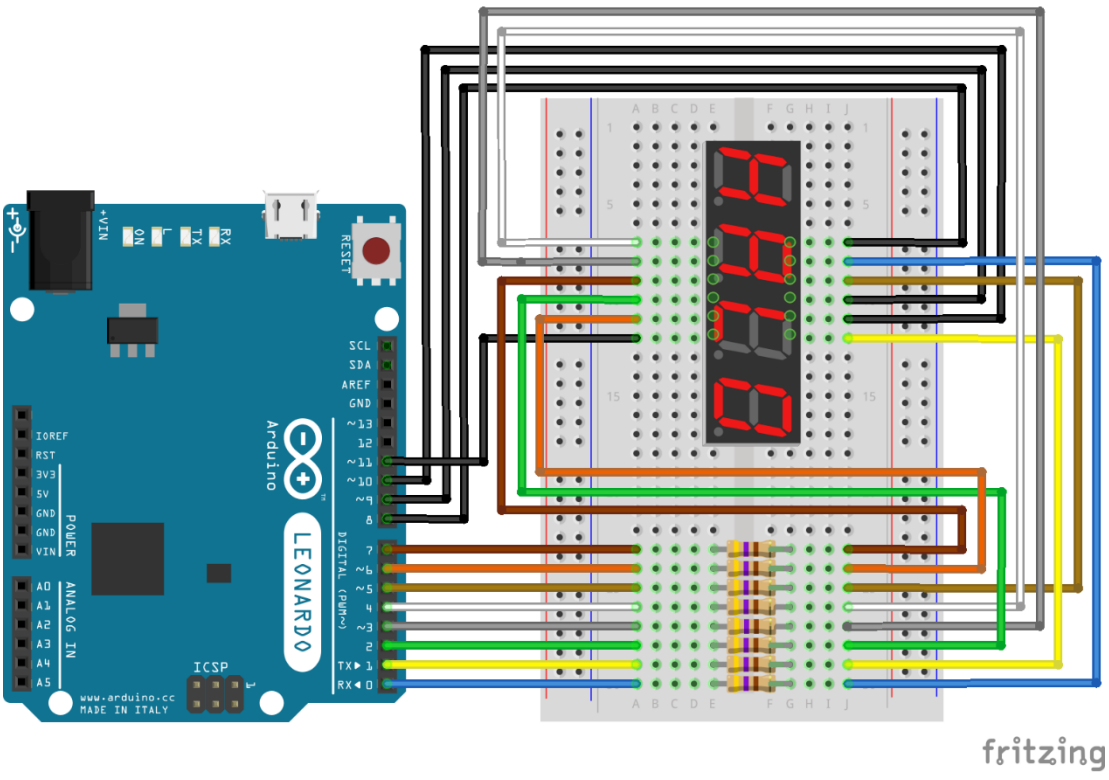
Egy digit egy szegmense akkor világít, ha a közös pont és a szegmens is a megfelelő feszültséget kapja. Közös katódos (CC) kapcsolás esetén a digit közös pontját a GND (0, LOW) feszültségre, a szegmens másik pontját pedig a +5V (1, HIGH) feszültségre kell kapcsolnunk. Minden más esetben a kijelző sötét marad, nem világít. Az eddigiek alapján belátható, hogy közös anódos (CA) kapcsolásnál a közös lábat +5V, a szegmenseket a GND-feszültségre kell kapcsolni. Az egy digites kijelző esetén a közös pontot egyszerűen egy vezetékkel rákötöttük az Arduino megfelelő lábára, most ezt is szoftveresen, programból kell megoldani.

Az Arduino-lapkához való bekötéskor – az egydigites kapcsoláshoz hasonlóan – itt is alkalmaznunk kell az előtét-ellenállásokat. A szegmenseket a D0–D7 lábakra, a négy számjegyet, digiteket (DIG1–DIG4) a D8–D11 lábakra kötjük be (*42. ábra*). Az ugyanolyan lábkiosztású kijelzők esetén megegyezik a közös anódos és közös katódos bekötés, csak a kódjuk lesz különböző. A kapcsolást próbapanelre építjük (*43. ábra*). Az áramkör bonyolultabb, és több alkatrészt is tartalmaz. Emiatt nehezebben férünk el a panelen, és több vezetékre van szükség, mint korábban.

42. ábra: Közös katódos kapcsolás



43. ábra: Közös katódos kapcsolás próbapanelen



Mivel a szegmensek össze vannak kötve, így egy időben csak egyforma számokat lehet kiírni az összes digitre. Ha nem tudnánk megoldani, hogy különböző számokat lássunk a különböző digiteken, akkor nem lenne értelme az ilyen kijelzőknek. A digitenkénti szegmenskivezetés túl sok lábat igényelne, és ehhez a megoldáshoz nem is kellene egybeépíteni a kijelzőket.

A megoldás az, hogy becsapjuk a szemünket. A filmeken is állóképeket látunk, csak olyan gyorsan váltakoznak egymás után, hogy azt a szemünk már mozgásként érzékeli. A kijelzőknél ezt *multiplex* (többcsatornás) *vezérlésnek* hívják. Ilyenkor az Arduino mindig csak egy digitet kapcsol be, és megjelenít rajta egy számot. Ezután vár egy kevés ideig (a másodperc tört részéig), aztán jön a következő digiten egy szám és így tovább. Ezt olyan gyorsan végzi, hogy mind a négy digitet bekapcsoltnak látjuk.

A kapcsolás megépítése után ellenőrizzük le azt az alábbi kóddal. Először `setup()`-ban beállítjuk, hogy a 0–11 lábak kimenetek legyenek. Ezután a digiteket kikapcsoljuk, amit a közös katódos kijelző miatt a `HIGH` érték beállításával tehetünk meg. A digitek összes szegmensét bekapcsoljuk, szintén a `HIGH` érték beállításával. A `loop()`-ban sorban bekapcsoljuk, majd várakozás után lekapcsoljuk a számjegyeket (digiteket). A bekapcsoláshoz elég csak a digitek közös lábára `LOW` értéket kapcsolni, mert a szegmensértékeket már beállítottuk a `setup()` függvényben.

15. kód

```
void setup() {
  int i;

  for (i = 0; i < 12; i++) {  //kimenetek beállítása
    pinMode(i, OUTPUT);
  }
  for (i = 8; i < 12; i++) {  //digitek kikapcsolása
    digitalWrite(i, HIGH);
  }
  for (i = 0; i < 8; i++)  {  //szegmensek bekapcsolása
    digitalWrite(i, HIGH);
  }
}

void loop() {
  int i, wait = 1000;  //várakozási idő

  for (i = 8; i < 12; i++) {
    digitalWrite(i, LOW);  //digitek bekapcsolása
    delay(wait);
    digitalWrite(i, HIGH); //digitek kikapcsolása
    delay(wait);
  }
}
```

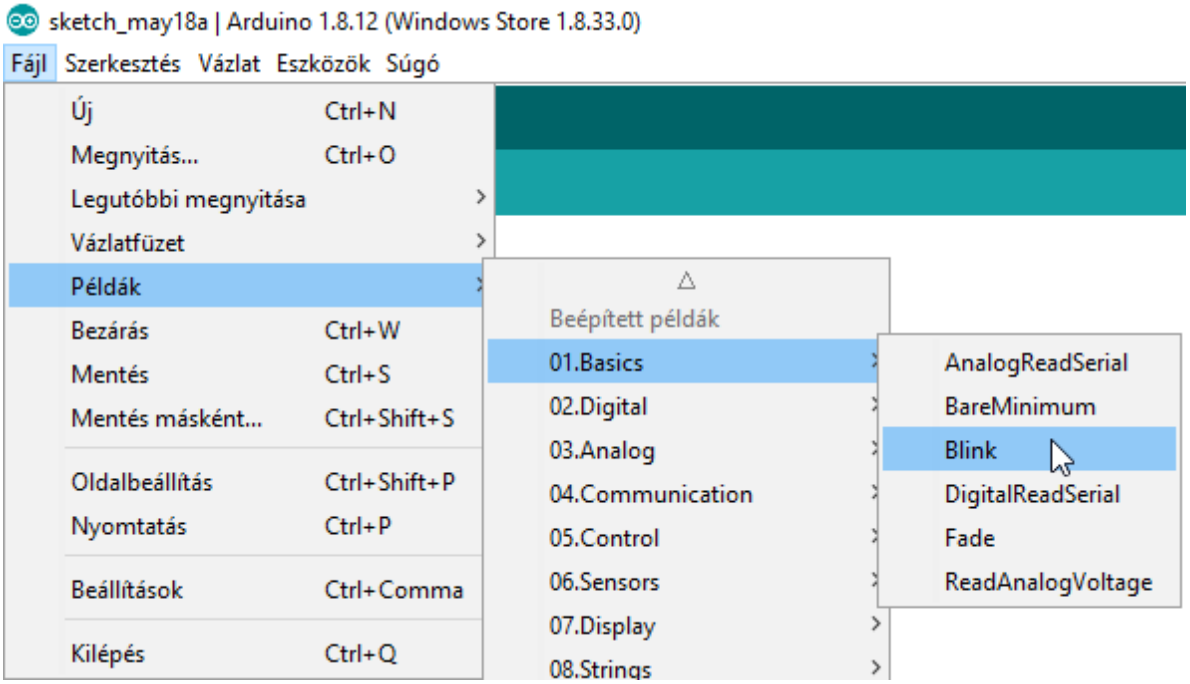
Ha a kapcsolásunkat helyesen építettük meg, akkor egy-egy másodpercre sorban egymás után bekapcsolódnak a digitek. A várakozási idő csökkentésekor azt látjuk, hogy a digitek be- és kikapcsolását már egyre nehezebb megkülönböztetni. Ha a várakozási időt 1-re állítjuk (a várakozás 1 ezred másodpercnyi, 1 msec), akkor már egyáltalán nem lehet a számok be- és kikapcsolását látni, az összes digit bekapcsoltnak tűnik.

Az eddig tanultak és az egydigites kódok alapján meg tudnánk írni egy olyan kódot, amellyel tetszőleges négyjegyű számot tudnánk kiírni a kijelzőnkre. Az eddigieken kívül a digitek kapcsolgatását kellene még megoldani.

Szerencsére az ilyen sokak által használt eszközre felhasználható bonyolultabb kódokat lelkes fejlesztők már megírták, és ráadásul közkinccsé is tették azokat. Bárki közreadhatja az általa fejlesztett programot, ha megtalálhatók benne a szükséges elemek, amelyeket egy *könyvtárban* kell tárolni. Vegyük példának egy *Program* nevű könyvtárat, ami C++ nyelven íródik, és a következőket tartalmazza: *fejléc fájl* (`Program.h`), *megvalósított kód* (`Program.cpp`), *kulcsszó fájl* (`keywords.txt`), *példa program* (`Program.ino`). A könyvtárkészítésről bővebben a következő linken kaphatunk információt: https://www.arduino.cc/en/Hacking/libraryTutorial.

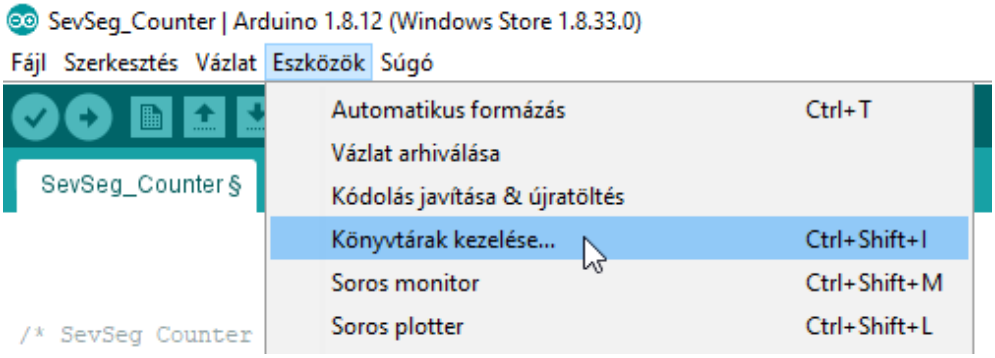
A telepített Arduino IDE már kezdetektől tartalmaz néhány ilyen könyvtárat, amelyeket fel tudunk használni a munkánk során. A hozzájuk tartozó példákat a **`Fájl`** menüpont **`Példák`** almenüpontjában lehet megnézni (*44. ábra*).

44. ábra: Példák elérése



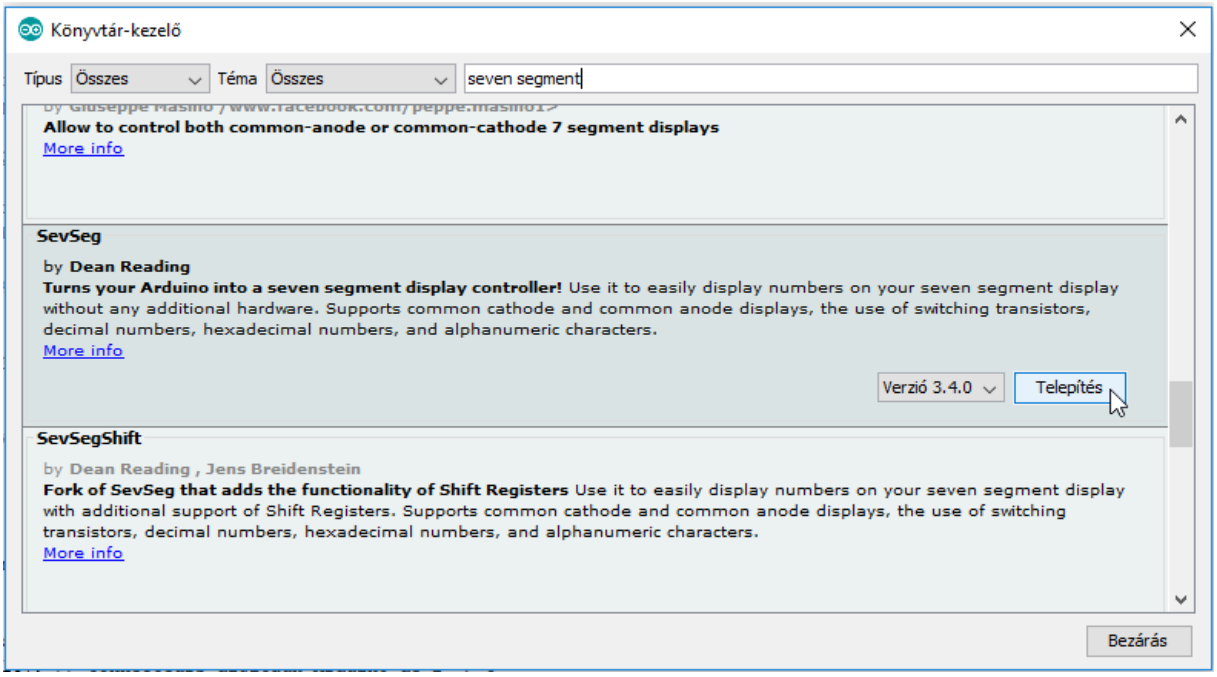
A nem telepített, de elérhető könyvtárakhoz az **`Eszközök`** menü **`Könyvtárak kezelése`**... menüpontból vagy a *Ctrl+Shift+I* billentyűkombinációval tudunk eljutni (*45. ábra*). Ugyancsak itt lehet elérni a könyvtárak újabb verzióit.

45. ábra: Könyvtárak elérése



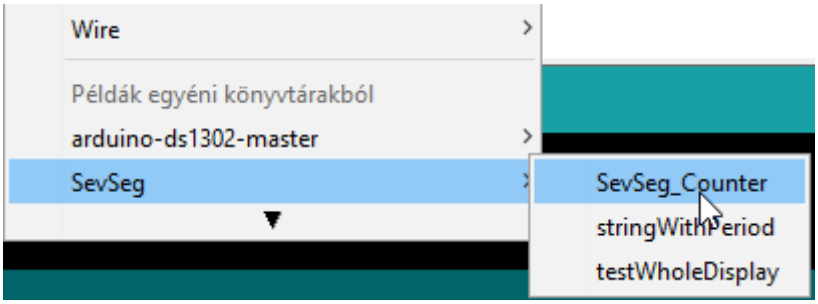
Mivel nagyon sok ilyen könyvtár van, ezért célszerű beírni a keresőmezőbe valamilyen kulcsszót. A *„seven segment”* kereső szavakra is több találatot kapunk. A leírásokat böngészve tudunk közülük választani. Előfordulhat, hogy érdemes többet is kipróbálni. Most a *SevSeg* könyvtárat fogjuk használni, ezért vigyük fölé az egérmutatót, ekkor megjelenik a **Telepítés** gomb, amire kattintanunk kell. A több verzióval rendelkező könyvtárak közül mindig a legutolsó verziót kínálja fel. Korábbi verziót a **Telepítés** gomb megnyomása előtt kell kiválasztanunk (*46. ábra*).

46. ábra: Könyvtár keresése kulcsszavakkal



Telepítés után használhatjuk ezeket a könyvtárakat, de előtte érdemes megismerkedni velük. A már leírt módon (**`Fájl`** menü, **`Példák`** almenü) megtalálhatjuk a könyvtárhoz mellékelt kódot, amivel tanulmányozhatjuk a működést (*47. ábra*).

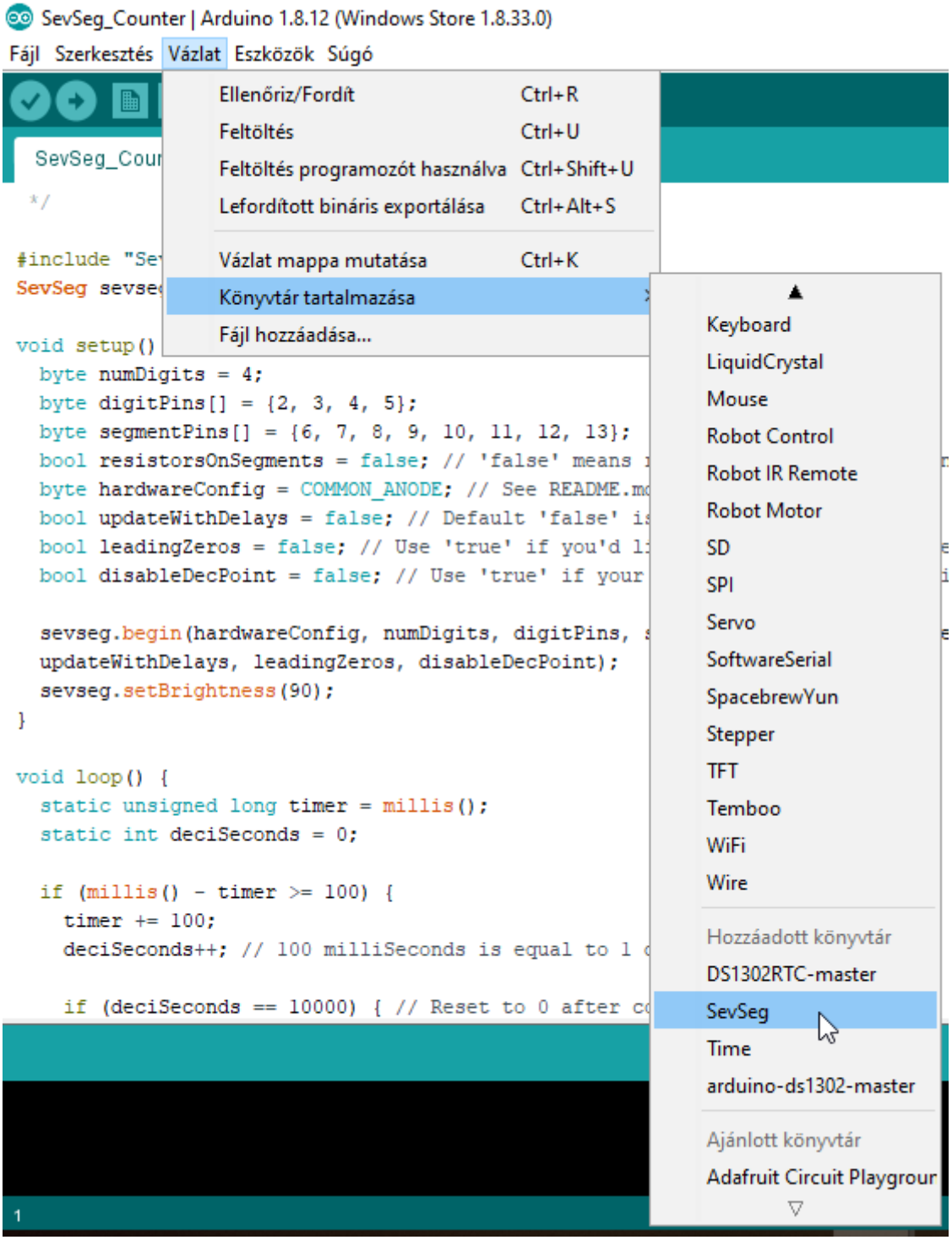
47. ábra: A SevSeg könyvtár példa kódjai



Most a példabeli `SevSeg_Counter` program egyszerűbb változatát készítjük el. A számláló egyesével számol, ami első hallásra nem bonyolult feladat. Ha egyszerűen csak növelnénk egy számláló tartalmát, és azt íratnánk ki, akkor az nagyon gyorsan növekedne, nagyon „pörögne” a kijelzőn. Ezért csak akkor jelenítünk meg egy számot, ha a számláló már elszámolt 1000-ig.

Először nézzük meg a `SevSeg` könyvtár miatti sajátosságokat. Elsőként meg kell adnunk a könyvtárhoz tartozó úgynevezett *header* fájlt: `#include <SevSeg.h>`. Ha ezt nem adjuk meg, akkor nem tudjuk használni a funkciókat, mert ez a `SevSeg.h` fájl tartalmazza a szükséges deklarációkat. A sort nem kell begépelni. A **`Vázlat`** menüpontban a **`Könyvtár tartalmazása`** almenüben kikeressük a szükséges nevet, és arra kattintva az első sorban megjelenik a szükséges szöveg (*48. ábra*).

48. ábra: A header fájl beíratása



Az Arduino a C++ nyelvre épül, ami *objektumorientált nyelv*. Ez azt jelenti, hogy nem az alapokról kell mindent felépíteni, hanem létre lehet hozni okos objektumokat, amelyeket aztán fel lehet használni a programjainkban. Sok már elkészített objektum is elérhető az Arduino fejlesztő környezetében (pl.: `Serial`).

Az objektumoknak vannak *tulajdonságaik* és *metódusaik*. A tulajdonságok valamilyen értéket jelentenek, mint az embernél a magasság, a metódusok pedig valamilyen viselkedést, mint például: az ember megy. Ezeket összefoglaljuk egy osztályba (`Class`). Az osztály tehát leírja az objektum általános jellemzőit. Ahhoz, hogy ezt használni tudjuk, az osztályból létre kell hoznunk egy konkrét objektumot, egy példányt, ami az osztály jellemzőit tartalmazza, amit ezután a programban el tudunk látni konkrét értékekkel, és használhatjuk a metódusaikat. Az objektum metódusaira az `objektum_név.metódus()`, a változóira az `objektum_név.változó_név` alakban lehet hivatkozni.

A `SevSeg.h` fájlban definiálták a `SevSeg` osztályt. Miután az `#include` sorban megadtuk a header fájlt, használhatjuk az abban megadott `SevSeg` osztályt. A használathoz először létre kell hozni egy objektumot az osztály alapján, ami egyszerűen úgy történik, hogy valamilyen nevet adunk neki, amivel később hivatkozni tudunk rá.

A `SevSeg display;` sorban definiáljuk a `SevSeg` típusú objektumot, amelynek a `display` nevet adtuk. A név szabadon választható, ezután viszont ezt a választott nevet használjuk az objektumműveletekben.

A `setup()` függvényben először megadjuk a digitek számát. Ezután rögzítjük egy tömbben, hogy melyik Arduino-lábra kötjük a digitek közös kivezetéseit, egy másikban pedig a szegmensek által használt lábakat. A `display.begin` metódussal tudjuk inicializálni az objektumot. Az előzőekben megadottakon túl a kijelző típusát kell beállítani, ami `COMMON_CATHODE` vagy `COMMON_ANODE` lehet.

A `loop()` függvényben először megnöveljük a `szamlalo` változóértékét, majd megnézzük, hogy így nagyobb lett-e 999-nél. Ha igen, akkor azonnal lenullázzuk, és a kiírandó számot növeljük meg 1-gyel. Figyelni kell még rá, hogy a kiírandó szám nem lehet ötjegyű, így a `kiir` változót is le kell nullázni, ha 9999-nél nagyobb lesz.

Ezután a `display.setNumber(kiir, 0);` sorban megadjuk a kiírandó számot (a `kiir` változóban) és azt, hogy hány tizedesjegyet szeretnénk használni. Mivel most egészként írjuk ki a számot, a tizedesek száma 0. A `display.refreshDisplay();` metódust ismételten le kell futtatni, hogy az újabb értéket kiírja.

16. kód

```
#include <SevSeg.h>

  int szamlalo = 0;
  int kiir = 0;

SevSeg display; //létrehozzuk a display objektumot

void setup() {

  byte numDigits = 4;  // a digitek száma
  byte digitPins[] = {8, 9, 10, 11};  //a digitek Arduino lábai
  byte segmentPins[] = {0, 1, 2, 3, 4, 5, 6, 7};  //a szegmensek lábai

  //A display objektum inicializálása
  display.begin(COMMON_CATHODE, numDigits, digitPins, segmentPins);
}
```

```
void loop() {

  szamlalo++;                  //megnöveljük a számláló értékét
  if (szamlalo > 999) {        //ha a számláló nagyobb 999-nél
    szamlalo = 0;              //akkor lenullázzuk
    kiir++;                    //és a kiírandó számot növeljük
  }


  if (kiir > 9999) kiir = 0; // ha a kiírandó szám már ötjegyű lenne,
                             // akkor lenullázzuk

  display.setNumber(kiir, 0); //szám és tizedes jegyek számának megadása
  display.refreshDisplay();   //frissítjük a kijelzőt
}
```

A kiírandó számot ennél egyszerűbben is elő lehet állítani. Ha a megnövelt számláló értékét elosztjuk ezerrel, akkor megkapjuk a kiírandó számot. Cseréljük ki a `loop()` függvényben az `if (szamlalo > 999)` feltételhez tartozó blokkot a következő sorra!

```
  kiir = szamlalo / 1000;
```

Ez a szám, mivel osztás eredménye, törtszám lesz, tartalmazni fog tizedesjegyeket. Az eredményváltozót egész típusúnak deklaráltuk, ezen nem változtat az sem, hogy egy törtszámmal tettük egyenlővé. Az eredményből csak az egész részt tárolja a `kiir` változó.

Töltsük fel a javított kódot az Arduino-ra. Azt tapasztaljuk, hogy a számok az eddigiekhez hasonlóan szépen növekednek. Amikor a kiírandó érték 32 lesz, akkor viszont ahelyett, hogy 33 lenne a következő szám, –32 látszik a kijelzőn, majd –31 és így tovább 32-ig.

Nem ezt az eredményt vártuk. Olvassuk el mit ír az *Arduino Reference* az `integer` típusról a https://www.arduino.cc/reference/en/language/variables/data-types/int/ linken:

Az egész szám az elsődleges adattípus a számok tárolásához, 16 bites (2 bájtos) értéket tárol. Ez -32 768 és 32 767 közötti tartományt jelent. A legkisebb értéke -2^{15}, a legnagyobb értéke pedig $2^{15} - 1$.

Ha a legnagyobb és a legkisebb értéket ezerrel elosztjuk, akkor 32 és –32 lesz az eredmény. Ha egy művelet eredménye meghaladja az azt tároló változó minimum vagy maximum értékét, akkor túlcsordulásról beszélünk. A `szamlalo` változót folyamatosan növeltük, míg elérte a maximumát, ezután a minimum értékét vette fel, és újra növekedett, a `szamlalo` „körbefordult”.

Mit lehetne tenni? Adjunk nagyobb számok tárolását is lehetővé tévő típust a `szamlalo` változónak. A `long` típus szintén egész számokat tárol, de nem két bájton, hanem négy bájton (32 biten), így a legkisebb tárolható érték –2 147 483 648 (-2^{31}), a legnagyobb 2 147 483 647 ($2^{31} - 1$). https://www.arduino.cc/reference/en/language/variables/data-types/long/

Cseréljük ki a `szamlalo` változó adattípusát, majd töltsük fel az új kódot az Arduino-ra!

```
  long szamlalo = 0;
```

Most már 9999-ig számol a számláló.

A C nyelvben lehetőségünk van a kód tömörebb formában írására.

```
  szamlalo++;
  kiir = szamlalo / 1000;
```

A fenti két sort egybe lehet írni az alábbi módon:

```
kiir = szamlalo++ / 1000;
```

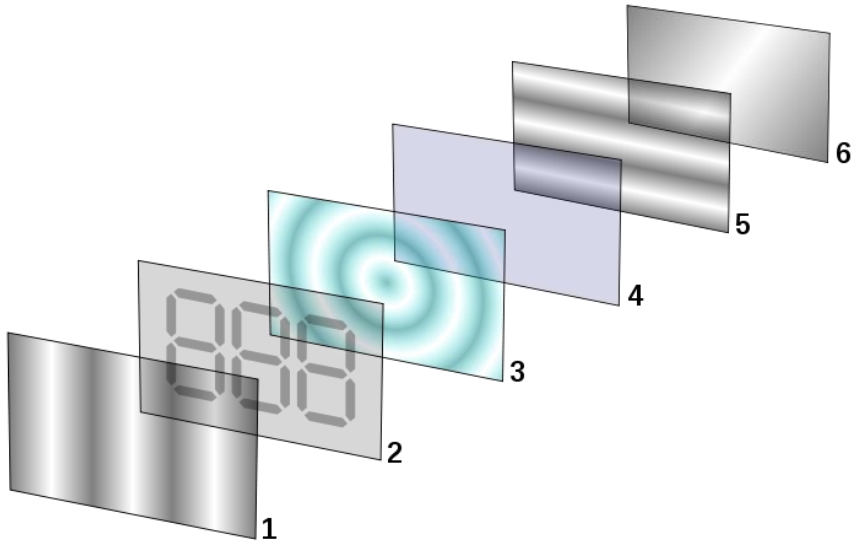
Ilyen esetben először a belső művelet, a `szamlalo` növelése hajtódik végre, ezután az osztás, és végül a `kiir` változóba kerül az osztás eredménye.

Ne törekedjünk a kód minél tömörebb megírására! Az is nagyon fontos szempont, hogy a legközelebbi használatnál gyorsan megértsük, mit is csináltunk korábban. Ahogy valaki gyakorlatot szerez, kialakul a saját stílusa, amelyik számára a legmegfelelőbb.

LCD

Az LCD (Liquid-crystal display – folyadékkristályos kijelző) kis fogyasztású kijelző. Mára a legelterjedtebb kijelzőfajta, ezt használják a TV-k, a számítógépek, mobiltelefonok stb. Kezdetben még csak egyszínűek voltak, és csak bizonyos alakzat kijelzésére voltak alkalmasak, hasonlóan a hétszegmenses kijelzőhöz (*49. ábra*). A működési elve nagyon leegyszerűsítve a következő: két átlátszó lap közé mikronméretű árkokban elhelyezett folyadékkristályos anyagot feszültséggel vezérelve azt átlátszó és át nem látszó állapotba lehet hozni, így világos és sötét alakzatot kapunk.

49. ábra: Az LCD felépítése

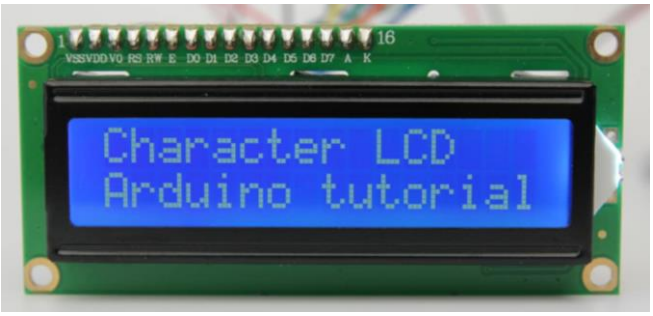


1 Függőleges tengelyű polarizáló szűrő; 2 Üveghordozó, a megjelenő alakzatokat formáló elektródákkal; 3 Folyadékkristály; 4 Üveghordozó, közös elektróda fóliával; 5 Vízszintes tengelyű polarizáló szűrő; 6 Fényvisszaverő felület és/vagy háttérvilágítás.

Forrás: https://upload.wikimedia.org/wikipedia/commons/thumb/d/dc/LCD_layers.svg/220px-LCD_layers.svg.png

A gyakorlatokon egy kétsoros, soronként 16 karakteres (16 oszlopos) LCD-t használunk majd. Egy karakter 5x8 képpontból áll (*50. ábra*).

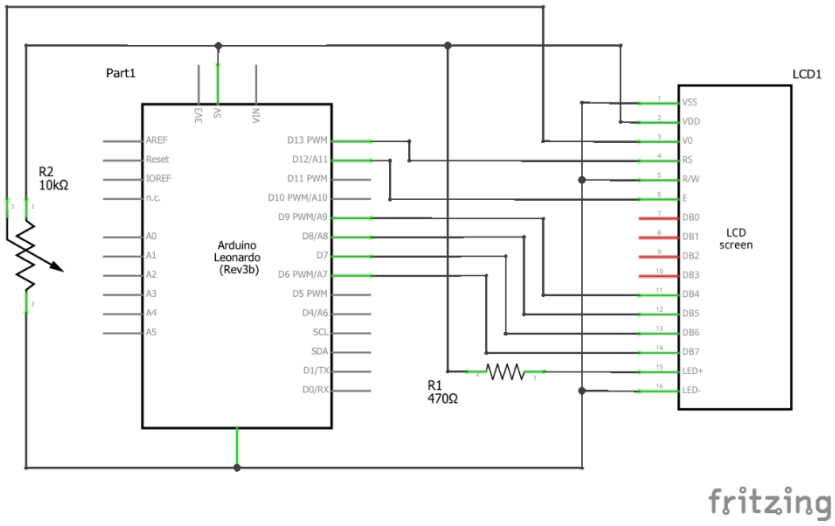
50. ábra: A felhasznált LCD



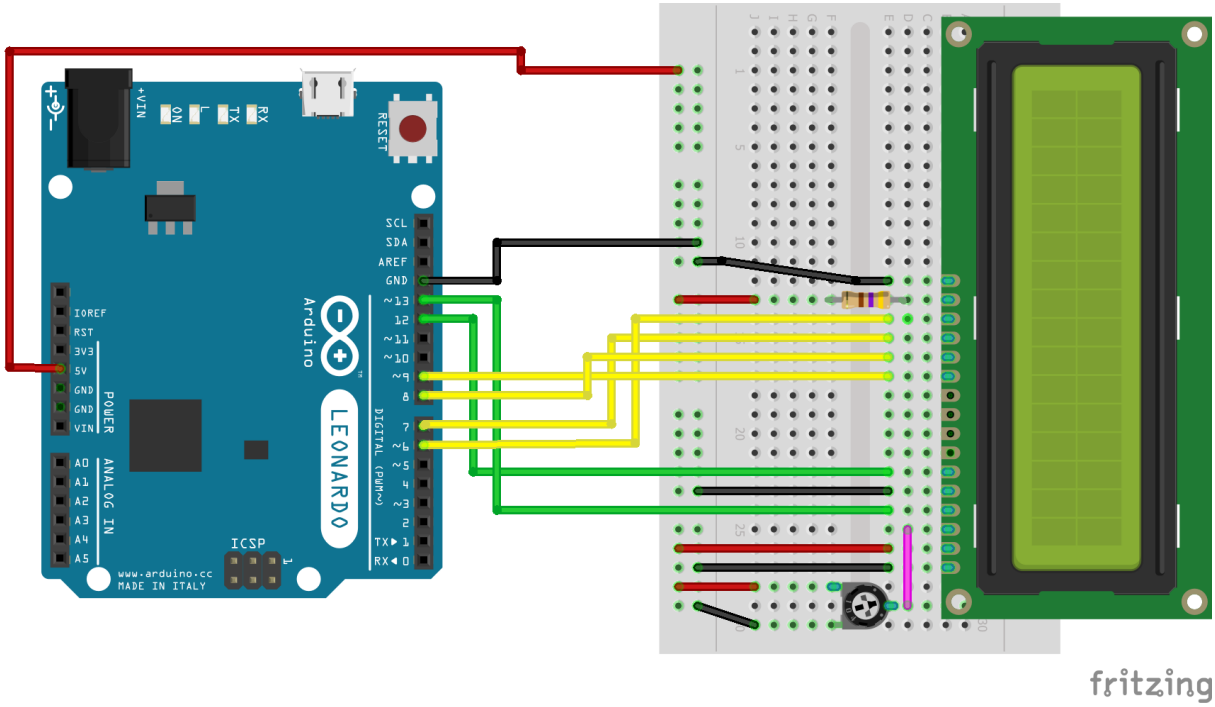
Forrás: https://www.makerguides.com/wp-content/uploads/2019/07/16x2-character-lcd-arduino-tutorial.jpg

A kijelző bekötése az *51. ábra* szerint történik, a kapcsolást próbapanelen állítjuk össze (*52. ábra*). A kapcsolásban a kijelzőn kívül egy ellenállás van, amelyik a háttérvilágítást biztosító LED előtét-ellenállása és egy potenciométer, amelyik a kijelzőn megjelenő szöveg kontrasztját szabályozza.

51. ábra: Az LCD-kapcsolás



52. ábra: Az LCD-kapcsolás megvalósítása



A hardver megépítése után lássunk neki a működtetőprogramnak!

Az Arduino az LCD működtetéséhez is tartalmaz beépített könyvtárt. A **Könyvtár tartalmazása** menüvel beírhatjuk az első sort, megadva a header fájlt (`#include <LiquidCrystal.h>`). A LiquidCrystal osztály alapján létrehozzuk az `lcd` nevű objektumot. Ennek definiálásakor az Arduino lábaira való bekötést is meg kell adnunk. A `setup()` függvényben

megadjuk a kijelző típusát, majd kiírjuk az első üzenetünket, ami az első sor első oszlopában kezdődik, ez az alapértelmezett pozíció.

A `loop()` függvény a kijelző második sorának első oszlopába kiírja a bekapcsolás óta eltelt másodpercek számát.

Ahhoz, hogy a kiírás mindig ugyanonnan kezdődjön, pozícionálnunk kell a kurzort. A kurzor (helyőr) az aktuális pozíciót jelöli valamilyen módon a kijelzőn. A program írása közben az aktuális pozíciót egy függőleges villogó vonalka jelzi a számítógépünk monitorján. Az LCD-n egy vízszintes vonalka a kurzor, de csak akkor látható, ha azt bekapcsoljuk az `lcd.cursor()` metódussal, mert alapértelmezés szerint kikapcsolt állapotban van. Kikapcsolni az `lcd.noCursor()` metódussal lehet.

Az `lcd.setCursor()` függvényben először az oszlop, majd a sor számát kell megadnunk, figyelembe véve, hogy a számolást mindig nullával kezdjük. A számozás a bal felső saroktól kezdődik.

A `millis()` függvény ezredmásodpercekben számolja a bekapcsolás óta eltelt időt, ezért azt el kell osztanunk ezerrel, hogy másodpercet kapjunk. Az `lcd.print(millis()/1000);` sorban azt látjuk, hogy az egyik függvény argumentumában szerepel egy másik függvény és egy művelet is. Ezt *egymásba ágyazásnak* hívjuk. A kiértékelés belülről kifelé történik: először a `millis()` függvény visszaad egy értéket, ezután következik az osztás 1000-rel, amelynek az eredményét kapja meg az `lcd.print()` függvény argumentumként, amit az feldolgoz és kiírja az értéket.

17. kód

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(13, 12, 9, 8, 7, 6);  //LCD - Arduino láb:
                                        //RS-D12, E-D11, DB4-D9 ,DB5-D8,
                                        //DB6-7, DB7-6

void setup() {

  lcd.begin(16, 2);             //Az LCD 2 sor, 16 oszlopos
  lcd.print("Hello, vilag!");   //LCD-re kiírunk szöveget
}

void loop() {

  lcd.setCursor(0, 1);          //A kiírás helye: 0. oszlop, 1. sor
  lcd.print(millis()/1000);     //eltelt másodperc a bekapcsolás ota
}
```

A következő programban egy egyszerű órát készítünk, amitől ne várjunk atomórai pontosságot. Ha pontos órát akarunk készíteni, akkor be kell szereznünk egy kvarckristályos óramodult.

A könyvtártartalmazás és az `lcd` objektum definiálása után megadjuk a `h`, az `m` és az `s` integer változókat, ezekben tároljuk az órák, a percek és a másodpercek számát. Ezekbe először beírjuk a kezdeti értékeiket is, mégpedig azt, hogy éppen mennyi az idő. Mivel nem tudjuk pontosan, hogy a fordítás, feltöltés mennyi időt vesz igénybe, így már az óra indulásakor némi pontatlanság lép fel.

A `setup()` függvényben hasonlóan az előző programhoz megadjuk a kijelző sorainak és oszlopainak számát.

A `loop()` függvény először meghívja az `ora()` függvényt, ami a globális `h, m,` és `s` változók értékeit kezeli. A függvényben növeljük a másodperc értékét, majd a kód további részében azt figyeljük, hogy a változók elérik-e a maximális értéküket, ha igen, akkor lenullázzuk azokat. A maximum másodpercnél és percnél 59, óránál 23.

Ezután töröljük a kijelzőt. Miért kell ezt megtennünk? A kijelzőn a kiírt tartalom fent marad, amíg felül nem írjuk vagy le nem töröljük azt. A jelenlegi programban a 10-nél kisebb számokat csak egy számjeggyel írjuk ki, ezáltal az időtől függően nem lesznek egyforma hosszúak az értékek. Nézzünk példákat! A 9 óra 7 perc 2 másodperc kiírva így néz ki: 9:7:2, míg tíz óra huszonegy perc negyvennyolc másodperc így: 10:21:48. Az utóbbi 3 számjeggyel kevesebb. Az 59. másodperc kiírása után a következő szám 0 lesz, ami csak az 5-öst írja felül, a 9-est nem, így az továbbra is látszani fog egészen addig, amíg a 10-et el nem éri a másodperc érteke, ami ezáltal újra kétjegyű lesz. Ennek elkerülésére a legegyszerűbbnek az tűnik, hogy letöröljük a kijelzőt, és újra kiírunk mindent. Emiatt újra ki kell írnunk a „`Time:`” szöveget is, mert az is törlődik. Meg lehet oldani, hogy a tíznél kisebb perc és másodperc elé 0-t, a tíznél kisebb óra elé szóközt írjunk, erre a következő kódban látunk egy megoldást.

A kiírás után várunk egy másodpercet, majd a `loop()` függvény kezdődik elölről.

18. kód

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(13, 12, 9, 8, 7, 6);

int h = 9; //az óra alapértéke
int m = 59; //a perc alapértéke
int s = 46; //a másodperc alapértéke

void setup()
{
  lcd.begin(16, 2);
}

void loop()
{
  ora();  //meghívjuk az óra függvényt

  lcd.clear(); //töröljük a kijelzőt
  lcd.setCursor(0, 0);
  lcd.print("TIME: ");
  lcd.print(h);
  lcd.print(":");
  lcd.print(m);
  lcd.print(":");
  lcd.print(s);

  delay(1000); //várunk egy másodpercet
}
```

```
void ora() {

  s++;          //növeljük a másodpercek számát

  if (s == 60) //ha elértük az 1 percet
  {
    s = 0;      //a másodpercet nullázzuk
    m = m + 1; //a percet növeljük
  }
  if (m == 60) //ha elértük az 1 órát
  {
    m = 0;      //a percet nullázzuk
    h = h + 1; //az órát növeljük
  }
  if (h == 24) //ha elértük az 1 napot
  {
    h = 0;      //az órát nullázzuk
  }
}
```

A következő kódban kiküszöböljük az előző kiírási problémát.

Először ismerkedjünk meg röviden a `printf()` függvénnyel, ami formázott szöveget ír ki a standard kimenetre. A már említett beköszönés, a „Hello World" (Helló világ) kiírása C nyelven a következő:

```
printf("Hello, World!");
```

A printf() általános alakja:

```
printf("formátum vezérlő mező", változólista);
```

Ebből mind a formátumvezérlő mező, mind a változólista elhagyható, láttuk, hogy a beköszönésnél sem voltak változók. A formátumvezérlő mezőben kell megadnunk a kiírandó szöveget, amely tartalmazhat változókat is. A változókat formátumvezérlő karakterekkel adjuk meg, amelyek helyére a függvény a működése során mindig behelyettesíti az adott változók aktuális értékét. A formátumvezérlő karaktereket mindig megelőzi egy `%` jel, ezután a változó típusát és a kiíratás formátumát kell megadni. A legalapvetőbb formátumvezérlő karakterek: a `%d` az egész (int), a `%c` egy karakter, a `%s` egy karakterlánc (string) megadására szolgál. A karakterlánc szöveget jelent. A számok formátumánál megadható, hogy hány számjeggyel akarjuk azt megjeleníteni. A `%5d` azt jelenti, hogy a számot öt szám szélességben írjuk ki, ha a szám csak kétjegyű, akkor az első három hely üresen marad. Ha azt szeretnénk, hogy az üres helyeket töltse fel nullával a függvény, akkor a `%05d` formátumot kell megadni. A változólistában a formátumvezérlő mezőben megadott formátumvezérlő karakterek sorrendjében kell megadnunk a változókat.

```
printf("i értéke = %d",i);
```

Az Arduino esetén nem tudjuk használni a `printf()` függvényt, hiszen itt nincs olyan szabványos kimenet, amire ki lehetne írni egy szöveget. Szerencsére létezik az `sprintf` függvény, amivel egy stringbe lehet írni, ezt a stringet pedig már ki tudjuk írni az LCD-kijelzőre az `lcd.print()` függvénnyel. Az `sprintf()` abban tér el a `printf()`-től, hogy a formátumvezérlő mező elé meg kell adni annak a stringnek a nevét, amibe írunk.

Nézzük meg most a javított kódot. A `setup()` függvényben szerepel a TIME felirat kiírása, mert most nem töröljük a kijelzőt, így az végig a kijelzőn marad.

A `loop()` csak a kiírásban tér el az előző kódtól. Az `sprintf` függvény `time_string` változóba ír, amit globális, nyolcelemű `char` típusú tömbként definiáltunk. Ebbe először beírjuk az órát 2 számjegy szélességben (%2d), majd a percet és a másodpercet szintén 2 számjegy szélességben, de itt előírjuk azt is, hogy egyszámjegyű érték esetén a függvény írjon az elé egy 0-t (`%02d`). Az értékeket a kettőspont ':' választja el egymástól.

19. kód

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(13, 12, 9, 8, 7, 6);

int h = 9; //az óra alapértéke
int m = 59; //a perc alapértéke
int s = 46; //a másodperc alapértéke

char time_string[8]; //az időt tartalmazza 12:08:45 formátumban

void setup()
{
  lcd.begin(16, 2);
  lcd.setCursor(6, 0);  //első sor közepére
  lcd.print("TIME");    //kiírjuk
}

void loop()
{

  ora();       //meghívjuk az óra függvényt

  sprintf(time_string, "%2d:%02d:%02d", h, m, s);

  lcd.setCursor(4, 1);      //második sor közepére
  lcd.print(time_string);   //kiírjuk az időt

  delay(1000); //várunk egy másodpercet
}

void ora() {

  s++;         //növeljük a másodpercek számát

  if (s == 60) //ha elértük az 1 percet
  {
    s = 0;     //a másodpercet nullázzuk
    m = m + 1; //a percet növeljük
  }
  if (m == 60) //ha elértük az 1 órát
  {
    m = 0;     //a percet nullázzuk
    h = h + 1; //az órát növeljük
  }
  if (h == 24) //ha elértük az 1 napot
  {
    h = 0;     //az órát nullázzuk
  }
}
```

A következő programmal a számítógépről soros vonalon szöveget küldünk az Arduino-nak, amit az megjelenít az LCD-kijelzőn. Már láttuk, hogy a soros monitoron az Arduino-ban futó program kiválasztott változóinak értékét ki tudtuk írni a PC képernyőjére. A soros vonal azonban kétoldalú, vagyis a PC-ről is tudunk az Arduino-nak küldeni adatokat. Mondjuk, ha ez nem így lenne, akkor nem is tudnánk feltölteni rá a programjainkat.

A kiírás pillanatnyi pozícióját, a kurzor helyét, a sor- és az oszlopváltozóban tároljuk. Ezek egész számok, most azonban nem `int`, hanem `byte` típusúak. A `byte` (magyar helyesírással: bájt) olyan adattípus, ami előjel nélküli számokat képes tárolni 0-tól 255-ig. Egy bájt 8 bitből áll, amely $2^8 = 256$ különböző értéket képes tárolni, mivel a számolás nullával kezdődik, emiatt az utolsó érték 255 lesz.

Mivel a kijelzőnk kétsoros és soronként 16 karakteres (16 oszlopos), ezért a `byte` típus bőven elegendő a sor- és az oszlopváltozó számára. Láttuk, hogy az integer számok két bájtból állnak, így két bájt tárhelyet is meg tudtunk spórolni. Ennek a megtakarításnak most nincs szerepe, de bonyolultabb programoknál előfordulhat, hogy minden bájtnyi helyre szükségünk van a memóriában.

A `setup()` függvényben az `lcd.cursor()` bekapcsolja a kijelzőn a kurzort, így látni fogjuk rajta a következő karakter kiírási pozícióját. A kurzor kezdeti pozíciója nulladik sor nulladik oszlopa, vagyis a kijelző bal felső sarka. Ez az alapértelmezett (default) érték, ezért ezt nem kell külön megadnunk. Ha kiírást más helyen kezdjük, mint az előző példákban, akkor az `lcd.setCursor()` függvényt kell használnunk.

A `loop()` függvényben először megvizsgáljuk, hogy érkezett-e adat a soros vonalon. A `Serial.available()` függvény visszatérési értéke egy szám, ami megadja, hogy az Arduino hány karaktert kapott soros vonalon. A beérkezett karaktereket egy 64 bájtos pufferben (átmeneti tárolóban) helyezi el az Arduino. Ha a tárolt karakterek száma nagyobb nullánál, akkor beolvassuk a `'c'` változóba, majd kiírjuk a soros monitorra és az LCD-kijelző aktuális pozíciójába is.

Ezután már csak a kurzor következő pozíciójának beállítása van vissza. Először a soron belül növeljük meg az oszlop számát, ha az már nem fér ki, akkor a következő sorba írunk. Ha sorok is beteltek, akkor letöröljük a kijelzőt, és kezdjük a kiírást az első sor első pozíciójában.

20. kód

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(13, 12, 9, 8, 7, 6);

byte sor = 0;
byte oszlop = 0;

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

  lcd.begin(16, 2);
  lcd.cursor();
}
```

```
void loop()
{
  char c;

  if (Serial.available() > 0) //van beérkezett karakter?
  {
    c = Serial.read();        //beolvassuk a 'c' változóba
    Serial.println(c);        //kiírjuk a soros monitoron

    lcd.setCursor(oszlop, sor); //a kurzor pozícióba
    lcd.print(c);               //kiirjuk a 'c' értékét

    oszlop++;          //az oszlop növelése

    if (oszlop == 16) //ha elértük a sor végét
    {
      oszlop = 0;     //az oszlopot nullázzuk
      sor++;          //a sort növeljük
    }
    if (sor == 2)     //ha betelt a második sor is
    {
      sor = 0;        //a sort nullázzuk
      lcd.clear();    //a kijelzőt töröljük
    }
  }
}
```

A program feltöltése után be kell kapcsolnunk a soros monitort, mert azon keresztül tudunk karaktereket kiküldeni. A fejléc alatti beviteli mezőbe beírt karaktert a jobb oldalon látható Küldés gombra kattintva vagy az *Enter* billentyű lenyomásával lehet elküldeni. A küldés után a kijelzőn és a soros monitoron is megjelenik a karakter, amit már az Arduino küldött vissza. A karaktereket nemcsak egyesével lehet kiküldeni, hanem hosszabb szöveget is begépelhetünk. Ekkor az összes karakter egyenként átküldésre kerül, és a pufferben tárolódik. Onnan egyesével olvassuk ki és jelenítjük meg azokat. Ez jól látszik a soros monitoron, ahol a szöveg összes karaktere új sorba íródik.

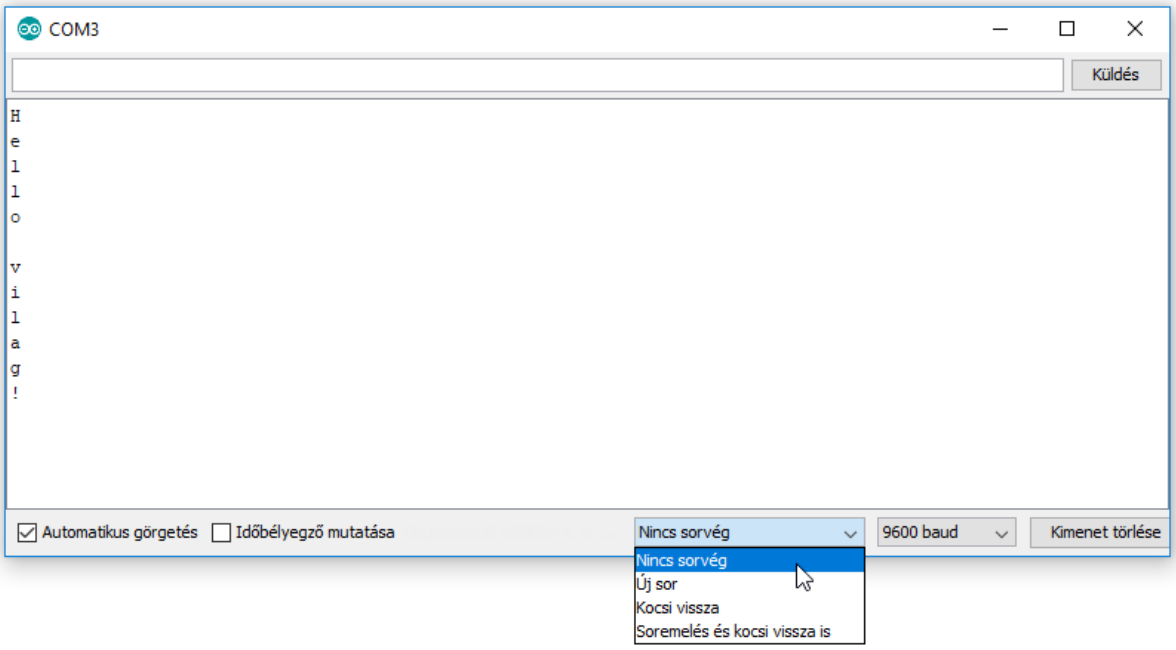
Írjuk ki most a „Hello vilag!” szöveget! A soros monitoron látszik a kiírt szöveg, viszont a kijelzőn a kiírt szöveg után két értelmezhetetlen karakter is megjelent (*53. ábra*). A kurzor ezek után helyezkedik el, ami azt jelenti, hogy ezt a két karaktert is kiküldtük a szöveggel együtt az Arduino-ra.

53. ábra: Az LCD-re kiírt szöveg a végén vezérlőkarakterekkel

Ez a két karakter úgynevezett vezérlőkarakter, ami a soros monitoron is „látszik”, a szöveg után következő két sor üres. A vezérlőkarakterek a monitoron nem látszanak, viszont az LCD-kijelzőn ez nem így van. Az egyik vezérlőkarakter a kocsi vissza (*carriage return*, CR), a másik a soremelés (*line feed*, LF). A kifejezések az írógép használatból erednek. A kocsin helyezkedett el a papír, és azt le, fel, jobbra és balra lehetett mozgatni, így beállítani a következő kiírandó betű helyét. A kocsi vissza a sor elejére viszi a kocsit, és a következő betű ugyanannak a sornak az elejére kerülne, amibe eddig is írtunk. Emiatt szükséges a soremelés, ami a következő sor elejére pozícionálja a kiírás helyét. Ezeket a fogalmakat megörökölték a számítógépek is, a vezérlőkarakterek elnevezésében és funkciójában. A soremelés és kocsi vissza általában együttesen szerepel.

Ezeket a vezérlőkaraktereket az adatátviteli protokollok (szabályok) hozzátehetik az üzenethez, anélkül, hogy mi azt a programban előírtuk volna. A soros monitoron be lehet állítani, hogy a sorvég jelzése benne legyen-e az üzenetben vagy sem (*54. ábra*).

54. ábra: A soros monitor beállítása



A négy lehetőség: Nincs sorvég, Új sor; Kocsi vissza; Soremelés és kocsi vissza is. Az utolsó beállítás esetén az LF és CR is kiküldésre kerül, a második harmadik esetén csak az egyik, míg az elsőnél egyik sem. Az *53. ábrán* látható kiíráshoz a legalsó (Soremelés és kocsi vissza is) beállítás volt érvényben a küldéskor. Az első (Nincs sorvég) beállítással úgy tudunk kiírni, hogy a szöveg végére nem kerül semmi (*55. ábra*).

55. ábra: Az LCD-re kiírt szöveg Nincs sorvég beállítással

Az Arduino-hoz közvetlenül is kapcsolható billentyűzet, és akkor az LCD mint monitor működhet (*56. ábra*).

56. ábra: Arduino-val vezérelt billentyűzet és LCD

Forrás: https://i1.wp.com/embedds.com/wp-content/uploads/2009/11/How-to-drive-USB-keyboard-from-Arduino.jpg?ssl=1

A kapcsolásokban felhasznált elektronikai elemek

Számos internetes oldalon beszerezhetők az Arduino-kapcsolások alkatrészei. Lehet rendelni kezdő csomagot (Starter kit), amelyek egy Arduino-lapkát, érzékelőket, kijelzőket, kapcsolókat, különböző elektronikai elemeket (ellenállás, LED, potenciométer stb.), próbapanelt és vezetékeket tartalmaznak. Az Arduino hivatalos lapján a készlethez egy 170 oldalas leírás is tartozik (https://store.arduino.cc/products/arduino-starter-kit-multi-language?selectedStore=eu). Vásárlás előtt érdemes szétnézni, mert az árakban nagy eltérések lehetnek.

A könyvünkben ismertetett kapcsolások kevés alkatrészből megépíthetők, és azok nem drágák. Az ismerkedéshez, az alapok elsajátításához, a bemutatott kapcsolások megépítéséhez elegendő az alábbi listában szereplő alkatrészeket beszerezni.

Arduino Leonardo panel	1 db
LED	8 db
7 szegmenses kijelző	1 db
4 digites 7 szegmenses kijelző	1 db
LCD 16x2 kijelző	1 db
Ellenállás, 470 Ω	8 db
Potenciométer 10 k	1 db
Nyomógomb	1 db
Próbapanel	1 db
Összekötő vezeték	20 db


MAGYAR AGRÁR- ÉS
ÉLETTUDOMÁNYI EGYETEM