Arduino C/C++ programozás

1. C program felépítése, függvények
2. Digitális kimenetek kezelése
3. Adat típusok, változók, műveletek, ciklusok
4. While ciklus, digitális kimenetek 2.
5. Elágazás, for ciklus, függvények
6. Digitális bemenetek kezelése
7. Soros kommunikáció, soros monitor
8. Analóg bemenetek kezelése

Felhasznált forrás, és ajánlott irodalom:
Ruzsinszki Gábor: Programozható Elektronikák

Felhasznált és ajánlott online szimulációs felület: www.tinkercad.com

1.1. C/C++ program felépítése

Egy C/C++ nyelvű program szerkezete:

- függvényekből (alprogramokból) áll
- egy függvény felépítése:
  - fejrész
    - visszatérési érték típusa függvény neve (paraméterek, ha vannak)
  - függvény törzse
    - { definíciók és utasítások }

- egy függvény tehát így néz ki:

```
típus függvénynév(paraméter1,paraméter2,...)
 {
    definíció1;
    definíció2;
    ...
    utasítás1;
    utasítás2;
    utasítás3;
    ...
 }
```

1.2. C/C++ program felépítése

Egy C/C++ nyelvű program szerkezete:

- függvény neve betűkből, számokból és az aláhúzás karakterből állhat, (hasonlóan a változó nevekhez),
- ékezetes karakter nem lehet nevekben, utasításokban !
- kisbetű, nagybetű különbözőnek számít ! ('f' nem ugyanaz mint 'F')
- utasítások, definíciók után → **;** (pontosvessző !)
- megjegyzések írása → // után (ezek nem hajtódnak végre !)

egy függvény tehát így néz ki:
pl.

```
void fuggveny1( )          // void → nincs visszatérési érték
 {
    byte i;               // sorok végén megjegyzések lehetnek  '//' után  !!!
    int szam=2;           // definíciók  után pontosvessző kell (';')
    i=3;
    szam=i+4;             //  utasítások után szintén  pontosvessző kell !!
    i++;
    ...
 }
```

megjegyzések

1.3. C/C++ program felépítése

Arduino esetén a C/C++ nyelvű program szerkezete:

- minimálisan két függvény !! → „setup” és „loop”
- a „setup” függvénnyel kezdődik a program végrehajtása, csak egyszer fut le
- ezután a „loop” függvény hajtódik végre, az viszont folyamatosan újra és újra végrehajtódik !! (végtelen ciklus)
- plusz függvényeket természetesen létrehozhatunk

egy egyszerű program Arduino esetén tehát így néz ki:

```
void setup( )            // először ez fut le (egyszer)
   {
        utasitas1;
        utasitas2;
        ...
   }

void loop( )       // majd ez fut folyamatosan, újra és újra
   {
        utasitas1;
        utasitas2;
        ...
   }
```

1.4. Arduino függvények

Sok speciális beépített Arduino függvény van, amelyek vagy a hardverrel kapcsolatban végeznek el valamit, vagy valami egyéb hasznos funkciójuk van.
Majd folyamatosan ismerkedünk meg a leggyakrabban használtakkal.
Ezek már meg vannak írva, nekünk csak használni kell azokat -->
meg kell hívni őket

„pinMode” függvény

Kivezetés (pin) konfigurálására szolgál.
Az Arduino „lábainak”, kivezetéseinek nagy része ugyanis lehet digitális bemenet és digitális kimenet is, nyilván egyidőben csak az egyik ! A függvény segítségével állítjuk be, hogy éppen bemenet vagy kimenet legyen egy adott kivezetés
Meghívása: pinMode(pin, mode);
mode: INPUT vagy OUTPUT

```
pinMode(2, OUTPUT);        //  2-es láb  kimenet lesz

pinMode(4, INPUT);         //  4-es láb  bemenet lesz
```

persze a lábszámot változóval is megadhatjuk !

1.5. Arduino függvények

„digitalWrite” függvény

Digitális kimenet írása. A digitális kimenetként beállított kivezetésre egy digitális értéket (0 vagy 1) küld, és az annak megfelelő feszültségszint jelenik meg a lábon.
Meghívása: digitalWrite(pin, value);
value: HIGH (1) vagy LOW (0)

HIGH (1) -> körülbelül a tápfeszültség kerül a lábra (+5V vagy +3,3V)
LOW (0) -> körülbelül 0 feszültség kerül a lábra

```
digitalWrite(1, HIGH);        //  1-es lábra 1-es érték  (+tápfeszültség)

digitalWrite(6, LOW);         //  6-os lábra 0-ás érték  (kb. 0V)
```

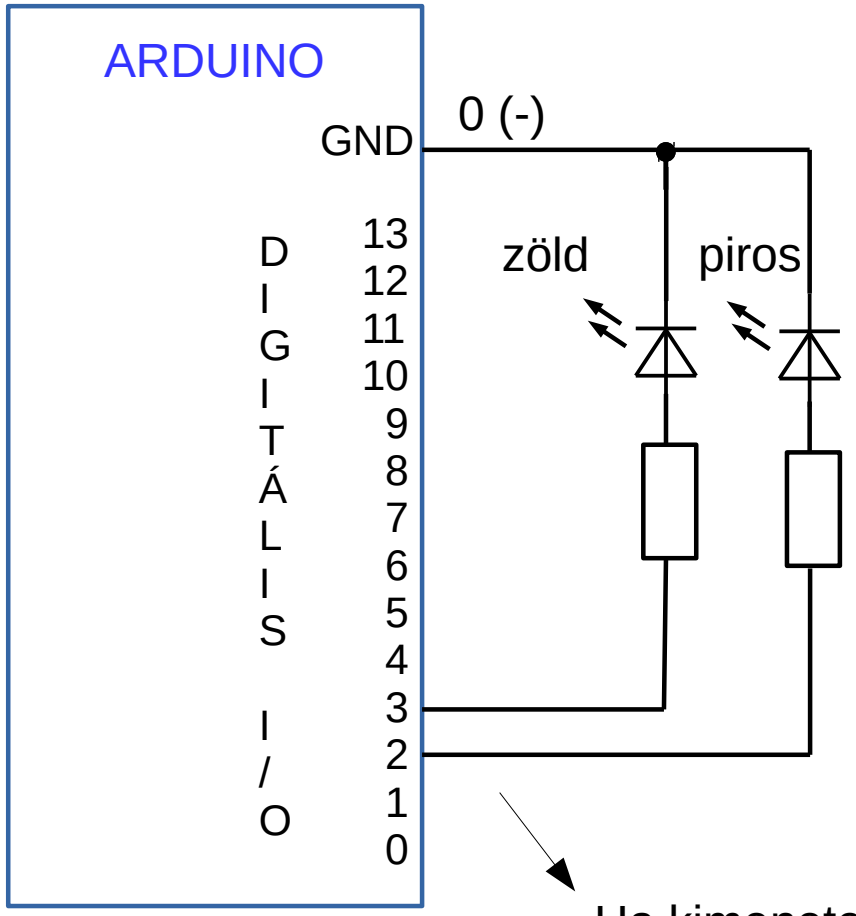
„delay” függvény

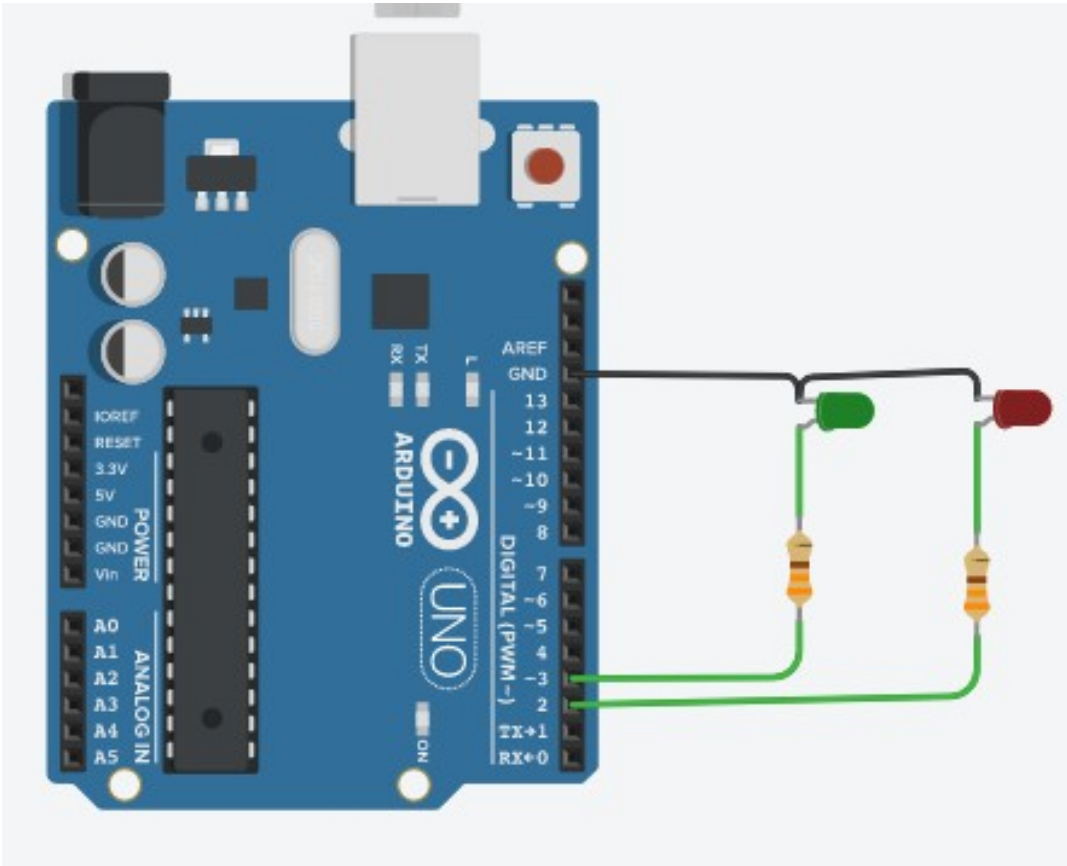
Késleltetés. Ennyi ideig várunk adott helyen a programban, és csak ha letelt az idő akkor hajtódik végre a következő utasítás.
Paramétere az időtartam milliszekundumban (ezredmásodpercben !).

```
delay(2000);          //  késleltetünk 2000 milliszekundumot (2 másodpercet)
```

2.1. Digitális kimenetek kezelése 1.

LED-ek vezérlése, hardver





Ha kimenetek, akkor

HIGH (1) -> körülbelül a tápfeszültség kerül a lábra (+5V)

LOW (0) -> körülbelül 0 feszültség kerül a lábra

2.2. Digitális kimenetek kezelése 1.

LED-ek vezérlése, 2.1. mintafeladat

```
// Először felkapcsoljuk a piros LED-et (3 másodpercig),
// majd a zöld LED-et (4 másodpercig)

void setup( )          // először ez fut le (egyszer)
 {
        pinMode(2, OUTPUT);      //  2-es láb  kimenet lesz (piros LED)
        pinMode(3, OUTPUT);      //  3-as láb  kimenet lesz (zöld LED)

        digitalWrite(2, HIGH);       // 2-es lábra 5V → piros LED-et  felkapcsoljuk
        delay(3000);                 // késleltetés
        digitalWrite(2, LOW);        // 2-es lábra 0V → piros LED-et  lekapcsoljuk
        digitalWrite(3, HIGH);       // 3-as lábra 5V → zöld LED-et  felkapcsoljuk
        delay(4000);                 // késleltetés
        digitalWrite(3, LOW);        // 3-as lábra 0V → zöld LED-et  lekapcsoljuk
 }

void loop( )       // majd ez fut folyamatosan, újra és újra
 {
                   // de mivel üres !! →  nem történik semmi
 }
```

2.3. Digitális kimenetek kezelése 1.

LED-ek vezérlése, 2.2. mintafeladat

```
//  a piros és a zöld LED felváltva világít,
//  a piros 2 másodpercig, a zöld  3 másodpercig

  void setup( )          // először ez fut le (egyszer)
    {
        pinMode(2, OUTPUT);      //  2-es láb  kimenet lesz
        pinMode(3, OUTPUT);      //  3-as láb  kimenet lesz

    }

  void loop( )       // majd ez fut folyamatosan, újra és újra
    {
        digitalWrite(2, HIGH);       // 2-es lábra 5V → piros LED-et  felkapcsoljuk
        delay(2000);                 // késleltetés
        digitalWrite(2, LOW);        // 2-es lábra 0V → piros LED-et  lekapcsoljuk

        digitalWrite(3, HIGH);       // 3-as lábra 5V → zöld LED-et  felkapcsoljuk
        delay(3000);                 // késleltetés
        digitalWrite(3, LOW);        // 3-as lábra 0V → zöld LED-et  lekapcsoljuk

    }
```

2.4. Digitális kimenetek kezelése 1.

LED-ek vezérlése, 2.3. mintafeladat

```
// Először felkapcsoljuk a piros LED-et (3 másodpercig), majd le kapcsoljuk
// ezután a zöld LED folyamatosan villog: 1 másodpercig világít, majd 1 másodpercig nem,
//   és újra világít, ...

void setup( )
   {
        pinMode(2, OUTPUT);      //  2-es láb  kimenet lesz
        pinMode(3, OUTPUT);      //  3-as láb  kimenet lesz

        digitalWrite(2, HIGH);    // 2-es lábra 5V → piros LED-et  felkapcsoljuk
        delay(3000);              // késleltetés
        digitalWrite(2, LOW);     // 2-es lábra 0V → piros LED-et  lekapcsoljuk
   }

void loop( )
   {
        digitalWrite(3, HIGH);    // 3-as lábra 5V → zöld LED-et  felkapcsoljuk
        delay(1000);              // késleltetés
        digitalWrite(3, LOW);     // 3-as lábra 0V → zöld LED-et  lekapcsoljuk
        delay(1000);              // késleltetés
   }
```

2.5. Feladatok

Írj programokat az Arduino 2,3,4,5 lábaira kapcsolt 4 db LED vezérlésére

1. feladat

- A két szélső LED (L2,L5) felvillantása 1 másodpercre,
- majd a két középső (L3,L4) LED felvillantása 1 másodpercre !
- és ismétlődjön elölről az egész folyamatosan !

2. feladat

- futófény: egyszerre mindig egy LED világít 1 másodpercig,
  és folyamatosan ismétlődik elölről
  L2 - L3 - L4 - L5 - L2 - L3 - L4 - ……

3. feladat

- futófény2: hasonló az előző feladathoz, de most egyszerre mindig két LED világítson !
  L2,L3 - L3,L4 - L4,L5 - L2,L3 - L3,L4 - ……

3.1. Adat típusok, változók

Adat típusok

a legfontosabb adattípusok:

- byte → 8 bites, előjel nélküli egész szám (0 – 255)
- char → egy karakter tárolására szolgál, pl. 'g' 'k' '5' '+'
  vagy előjeles !! 8 bites egész szám (-128 – +127)
- int → előjeles egész szám, 16 bites (-32768 – +32767)
- word → 16 bites, előjel nélküli egész szám (0 – +65536)
  más néven: unsigned int
- boolean → logikai változó, két érték !
  true (igaz, 1) vagy false (hamis, 0)
  (de valójában 8 bites szám)

- long → előjeles egész szám, 32 bites
- unsigned long → előjel nélküli egész szám, 32 bites

- float → valós szám, lebegőpontos, 32 bites
  ($\pm$ 1,18*10^{-38} – $\pm$ 3,4*10^{38})

...

3.2. Adat típusok, változók

Változó

- egy adatot tárol, értéke (a tárolt adat) általában változik a program futása során
- van neve (betűkből, számokból és az aláhúzás karakterből állhat) → igazából a memóriában tárolódik
- van típusa ! (milyen típusú adatot tárol)

- használatuk előtt deklarálni, definiálni kell őket → típus név;
  vagy típus név = érték;

pl.

```
int  szam1;          // szam1  nevű,  egész típusú változó
byte  sorszam=2;     // sorszam  nevű  egész, 2 kezdő értékkel
char betu;           // betu nevű, karakter típusú változó
char betu2='B';      // betu2 nevű változó, 'B' kezdő értékkel
float atlag=6.5;     // atlag  nevű  valós  6,5 kezdő értékkel
```

- értékadás, később bármikor a programban '=' használatával

pl.

```
szam1=5;             // most már szam1 értéke 5 lesz, amíg
                     // meg nem változtatjuk
betu2='C';           // most már betu2 értéke C lesz (nem B)
sorszam=6;           // sorszam értéke 6 lesz
```

3.3. Adat típusok, változók

Konstans

- értéke állandó, nem változtatható meg !!
- lehet: egész, karakter, valós, ...
- használatuk előtt deklarálni kell őket → const típus név = érték;

pl.

```
const byte SZAM=200;    // 8 bites  egész konstans
const int SZAM2=1200;    //  16 bites egész konstans
const float MAX=4.5;        //   valós konstans

const char BETU='T';          // karakter konstans
```

Globális változók

Ha még a program legelején minden függvényen kívül definiáljuk a változókat, konstansokat, azok minden függvényben használhatóak lesznek –>
Ezek a globális változók, konstansok

Ha egy függvényen belül definiáljuk azokat –> csak abban a függvényben lesznek láthatóak (ezek a lokális változók)

3.4. Műveletek, operátorok, kifejezések

Aritmetikai műveletek

- a négy alapművelet operátorai: + - * /
- maradékos osztás: %
- növelés 1-el (increment): ++ - csökkentés 1-el (decrement): --

Értékadás → változónév=kifejezés;

először kiértékelődik az egyenlőség jel jobb oldalán lévő kifejezés → és a kapott értéket veszi fel a baloldali változó

```
int szam1, szam3;   // két egész változó létrehozása
szam1=4+5*2;        // szam1 értéke 14 lesz !!
                    // (először a szorzás lesz elvégezve)
szam3=(szam1-2)/2;     // szam3 értéke (14-2)/2=6  lesz
szam1=9%4;          // szam1 értéke 1 lesz
                    // (az egész osztás maradéka)
szam1++;            // szam1 értéke 1-el növelődik !  → 2 lesz
szam3=szam1*szam1;     // szam3 értéke 2^2=4  lesz
szam3--;            // szam3 értéke 1-el csökken !  → 3 lesz
```

3.5. Műveletek, operátorok, kifejezések

Logikai műveletek, operátorok

- ÉS (AND): && pl. a&&b
- VAGY (OR): || pl. c||d
- NEM (NOT): ! pl. !d

Relációs műveletek

- nagyobb: >
- kisebb: <
- egyenlő: ==
- nem egyenlő: !=
- nagyobb vagy egyenlő: >=
- kisebb vagy egyenlő: <=

pl.

```
x<10          // igaz értéket ad ha x kisebb mint 10
y==5          // igaz értéket ad ha y egyenlő 5-el
(x>10)&&(x<20)          // igaz értéket ad ha x  10 és 20 közé  esik
                        // (nagyobb mint 10  ÉS kisebb mint 20)
```

C program nyelven ha valahol logikai értékre van szükség, de szám van ott →
a számok automatikusan átkonvertálódnak logikai értékre !!

0 → hamis, bármilyen 0-tól különböző szám → igaz

3.6. Elágazás, ciklus

Egy program alapvető struktúrákból épül fel

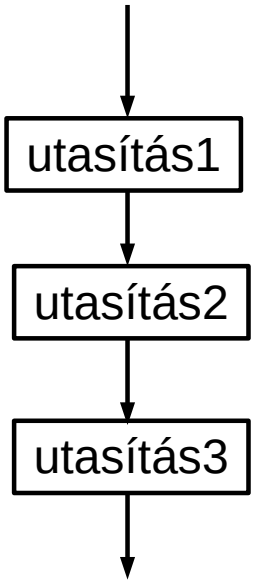
szekvenciális (sorrendi)

feltételes (elágazás)

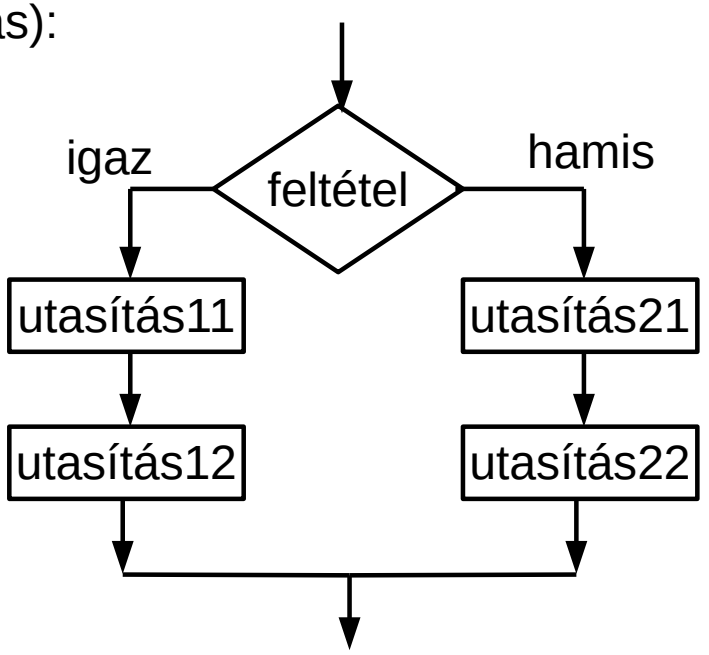
hurok (ismétlődés, ciklus)

index

-szekvenciális struktúra: az utasítások egymás után sorban (ez az alap)

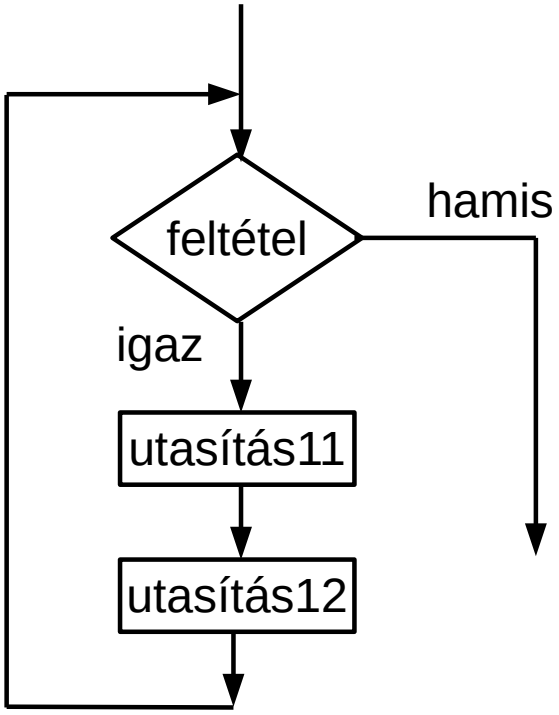


-feltételes struktúra (elágazás): a program végrehajtás két úton megy tovább (igaz és hamis ág)



3.7. Elágazás, ciklus

-hurok struktúra (ciklus): feltételtől függően ismétlünk utasításokat



-index struktúra: hasonló mint a feltételes struktúra, de a program végrehajtás nem két úton hanem több úton futhat tovább az 'index' értékétől függően

3.8. Ciklusok

Ciklus utasítás: ha többször ismételni akarunk utasításokat

- nyilván ha csak 2-szer, 3-szor kell ismételni kevés (1, 2) utasítást akkor simán le is írhatjuk egymás után többször azokat, de sok utasítás esetén, vagy ha sokszor kell ismételni, akkor valami jobb módszer kell
  → erre vannak a ciklusok
- többféle is van ! → while, for, do-while

while ciklus

while (feltétel) { ismétlendő utasítások }

amíg a feltétel igaz addig ismétli az utasításokat

pl. ismétlés 3-szor

```
byte i=0;     // ciklus változó, ezzel számolunk
while(i<3)    // meddig számolunk  (i=0,i=1,i=2)
 {
    ....      // ismétlendő utasítások
    ....
    i++;          //   változó növelése
 }
```

3.9. while ciklus

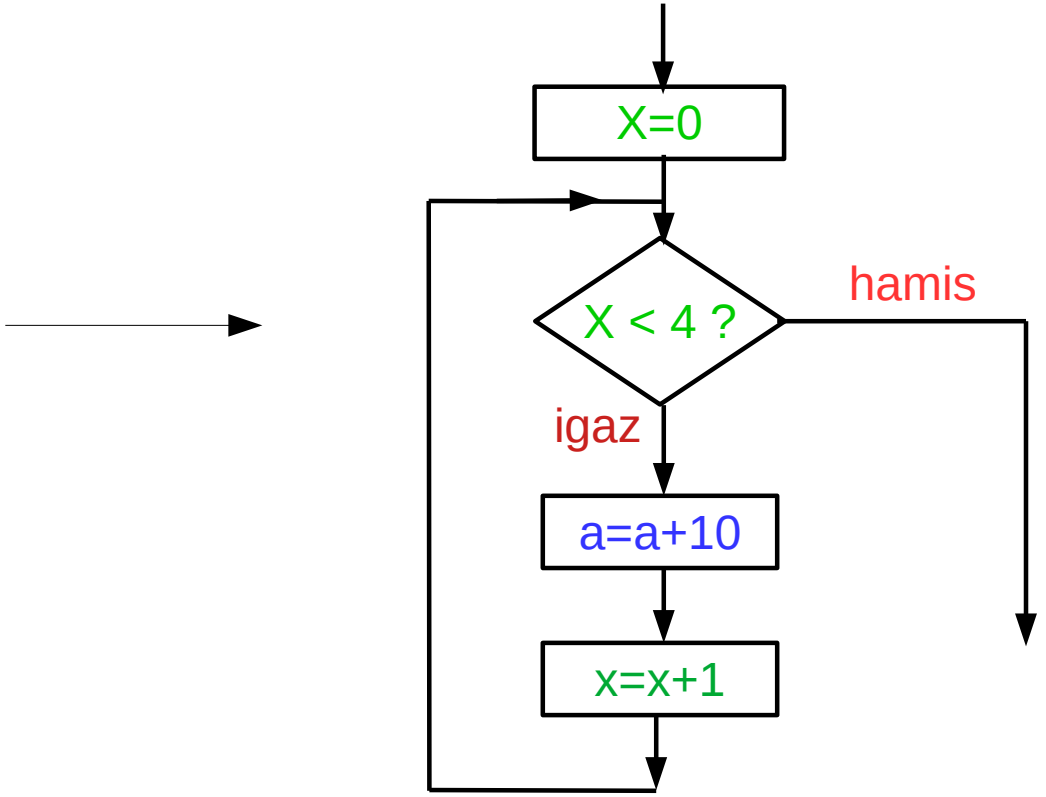
while ciklus

```
while (feltétel)
 {
     ismétlendő
     utasítások
 }
```

amíg a feltétel igaz,
addig ismétli az utasításokat

pl.

```
...
x=0;
while (x<4)
 {
     a=a+10;
     x++;
 }
...
```



3.10 while ciklus

While ciklus használata

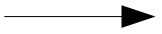
pl. adjuk össze a számokat 1-től 8-ig → ismétlés 8-szor

pl.

```
byte i=1;
byte ossz=0;

while(i<9)
 {
    ossz=ossz+i;
    i++;

 }
```



Megfelel ennek:

```
ossz=0;
ossz=ossz+1;
ossz=ossz+2;
ossz=ossz+3;
ossz=ossz+4;
ossz=ossz+5;
ossz=ossz+6;
ossz=ossz+7;
ossz=ossz+8;
```

ossz=ossz+i; → a ossz változó jelenlegi értékéhez hozzáadjuk i értékét → ez lesz a ossz új értéke

3.11. while ciklus

While ciklus használata

pl. adjuk össze az 1 és 101 közötti páros számokat
(tehát 2+4+6+8+……...+100)

```
byte szam=2;
int osszeg=0;

while(szam<101)    // csak 100-ig adjon össze
 {
    osszeg=osszeg+szam;
    szam=szam+2;                  // szam  növelése 2-vel
 }
```

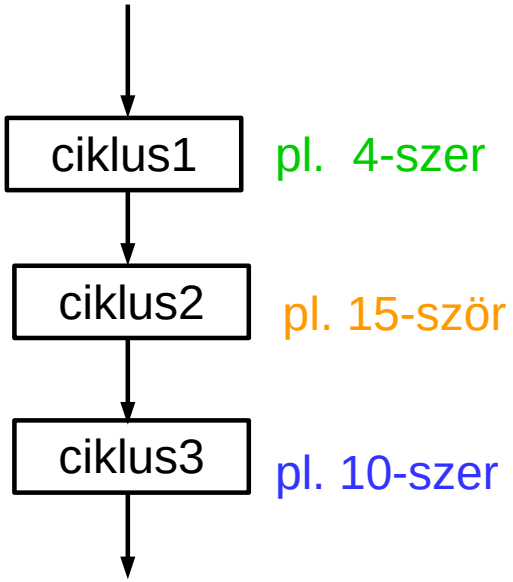
Végtelen ciklus

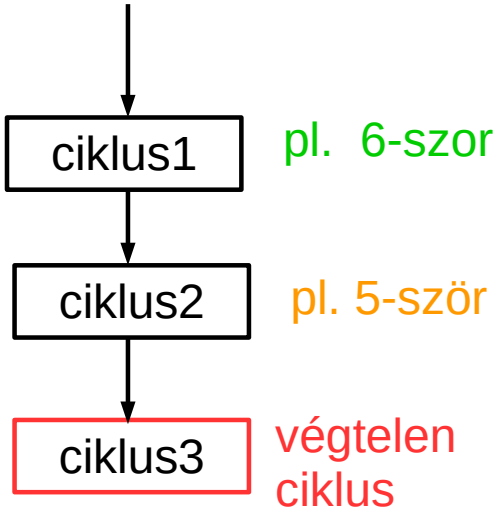
- ha a ciklus feltétele mindig igaz → soha nem lesz vége az ismétlésnek !!
- ez gyakran hiba eredménye, de lehet hogy mi szeretnénk ezt
- mikrovezérlőknél a fő program résznek végtelen ciklusban célszerű futnia → Arduino esetén ezt a loop függvény csinálja meg !

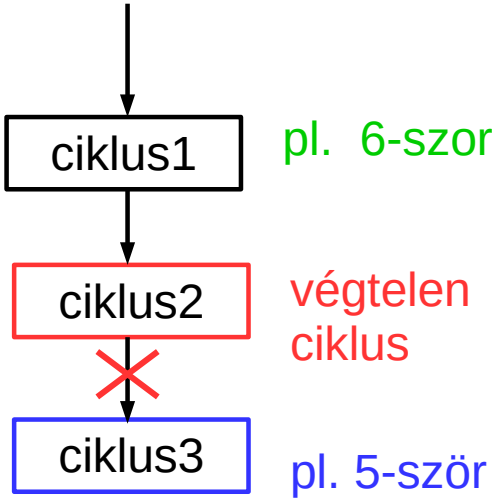
3.12. Több ciklus

Több ciklus egymás után

- Akkor használunk ilyen szerkezeteket, ha egymás után különböző dolgokat kell ismételni, elvileg bármennyi ciklust használhatunk egymás után
- viszont nyilvánvalóan csak az utolsó ciklus lehet végtelen ciklus !! →
  → mert a végtelen ciklus utáni utasítások (így a ciklusok is) már soha nem fognak végrehajtódni



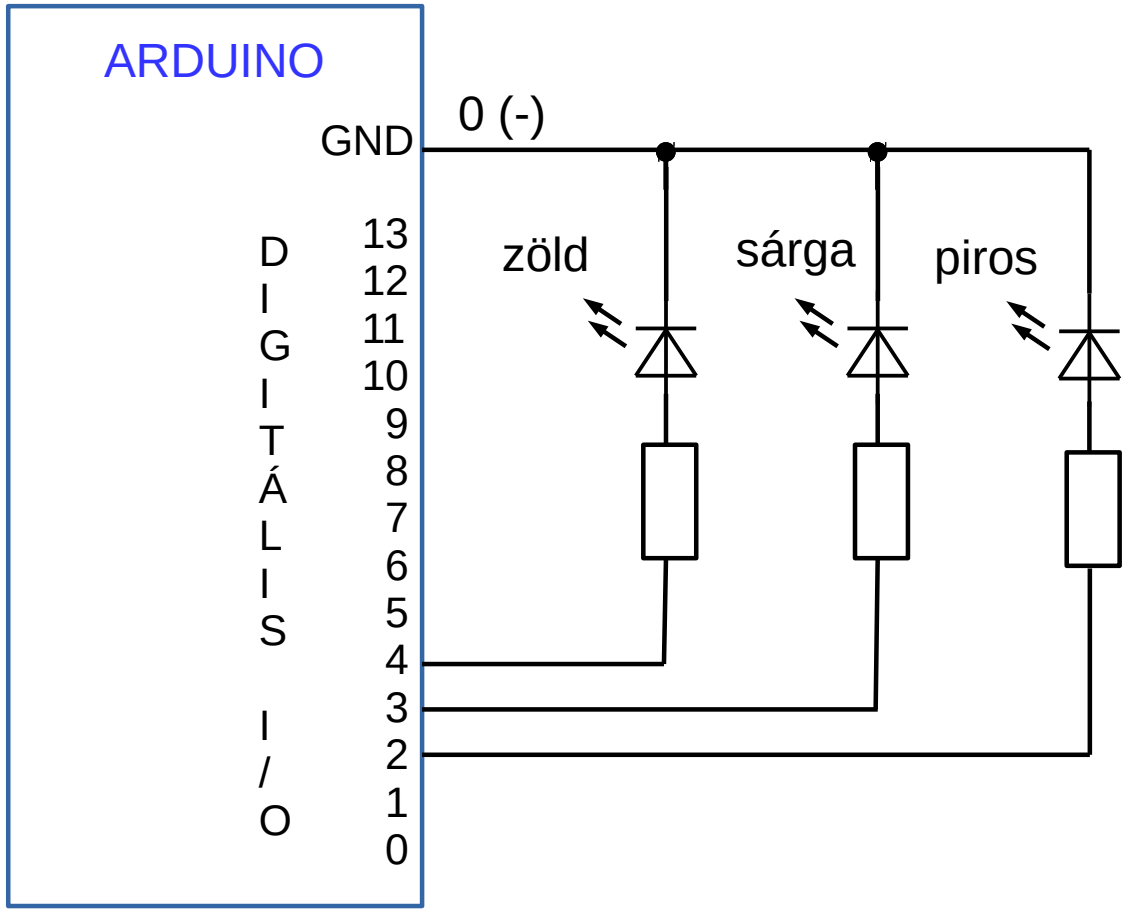




De ez már egyszer sem fog végrehajtódni !!

4.1. While ciklus, digitális kimenetek 2.

A hardver



4.2. Változók, konstansok használata

Arduino lábak megadása változókkal

Célszerű nem fixen beírni a lábszámokat a függvényekbe, hanem változókat, vagy konstansokat használni, több okból:

- így érthetőbb, jobban átlátható, hogy éppen mit kapcsolunk fel vagy le (legalábbis ha jó változó/konstans neveket adunk)
- ha változik az áramkör, mégis másik lábra kell átkötni valamit → csak egy helyen kell módosítani a programban !!

Még a program legelején minden függvényen hozzuk létre ezeket a változókat, konstansokat, így azok minden függvényben használhatóak lesznek (globális változók)

```
// globális  konstansokkal
const byte piros=2;          // piros  LED  a 2-es lábon !
const byte sarga=3;          //  sárga  LED  a 3-as lábon !
const byte zold=4;           //  zöld  LED  a 4-es lábon !

//vagy  globális  változókkal
byte piros=2;        // piros  LED  a 2-es lábon !
byte sarga=3;        //  sárga  LED  a 3-as lábon !
byte zold=4;         //  zöld  LED  a 4-es lábon !
```

4.3. Ciklus, digitális kimenetek

LED-ek vezérlése, 4.1. mintafeladat

```
// kapcsoljuk fel 8-szor a piros LED-et
// ( 1 másodpercig világítson, 1 másodpercig ne, …)
   const byte piros=2;          // piros  LED  a 2-es lábon !
   void setup( )
      {
            pinMode(piros, OUTPUT);        //  2-es láb  kimenet lesz, piros LED
            byte i=0;                      // számláló
            while(i<8)                     // ismétlés 8-szor  (0,1,2,3,4,5,6,7)
             {
                 digitalWrite(piros, HIGH);        //  piros LED-et  felkapcsoljuk
                 delay(1000);                      // késleltetés
                 digitalWrite(piros, LOW);         //  piros LED-et  lekapcsoljuk
                 delay(1000);                      // késleltetés
                 i++;                              // számláló növelése 1-el
             }
      }

   void loop( ) {
                    //  mivel üres →  nem csinál semmit !!
      }
```

4.4. Ciklus, digitális kimenetek

LED-ek vezérlése, 4.2. mintafeladat

kapcsoljuk fel 6-szor a piros LED-et (1 s-ig világítson, 1 s-ig ne, …),
majd kapcsoljuk fel 9-szor a sárga LED-et, hasonló időzítésekkel

```
const byte piros=2;            // piros  LED  a 2-es lábon !
const byte sarga=3;    //  sárga  LED  a 3-as lábon !

void setup( )            // először ez fut le (egyszer)
{
      pinMode(piros, OUTPUT);          //  2-es láb  kimenet lesz, piros LED
      pinMode(sarga, OUTPUT);          //  3-as láb  kimenet lesz, sárga LED

      byte i=0;                        // számláló
      while(i<6)                       // ismétlés 6-szor  (0,1,2,3,4,5)
       {
            digitalWrite(piros, HIGH);          //  piros LED-et  felkapcsoljuk
            delay(1000);                        // késleltetés
            digitalWrite(piros, LOW);           //  piros LED-et  lekapcsoljuk
            delay(1000);                        // késleltetés
            i++;                                // számláló növelése 1-el
       }
      i=0;                             // számláló  nullázása !!
      while(i<9)                       // ismétlés 9-szer  (0,1,2,3,4,5,6,7,8)
       {
            digitalWrite(sarga, HIGH);     //  sarga LED-et  felkapcsoljuk
            delay(1000);                        // késleltetés
            digitalWrite(sarga, LOW);      //  sarga LED-et  lekapcsoljuk
            delay(1000);                        // késleltetés
            i++;                           // számláló növelése 1-el
       }
 }

void loop( )       {  }          //  üres !!
```

4.5. Ciklus, digitális kimenetek

LED-ek vezérlése, 4.3. mintafeladat

Induláskor kapcsoljuk fel 5-ször a piros LED-et (1 s-ig világítson, 1 s-ig ne, …),
majd ezután folyamatosan ismétlődve először a sárga LED villogjon 4-szer (0,5 s-ig be, majd ki), utána a zöld LED villogjon 6-szor, majd újra a sárga, majd a zöld ….

A piros LED kapcsolgatását a setup függvénybe tettük, mert csak először kell végrehajtani

```
const byte piros=2;              // piros  LED  a 2-es lábon !
const byte sarga=3;              //  sárga  LED  a 3-as lábon !
const byte zold=4;               //  zöld  LED  a 4-es lábon !
byte i;                          // globális számláló (minden függvényben látszik!)

void setup( )
 {
      pinMode(piros, OUTPUT);     //  2-es láb  kimenet lesz, piros LED
      pinMode(sarga, OUTPUT);     //  3-as láb  kimenet lesz, sárga LED
      pinMode(zold, OUTPUT);      //  4-as láb  kimenet lesz, zöld LED

      i=0;                    // globális számláló nullázása
      while(i<5)              // ismétlés 5-ször  (0,1,2,3,4)
       {
            digitalWrite(piros, HIGH);       //  piros LED-et  felkapcsoljuk
            delay(1000);                     // késleltetés
            digitalWrite(piros, LOW);        //  piros LED-et  lekapcsoljuk
            delay(1000);                     // késleltetés
            i++;                        // számláló növelése 1-el
       }
 }
```

folytatás

4.6. Ciklus, digitális kimenetek

4.3. mintafeladat folytatása

a sárga és zöld LED-ek kapcsolgatását viszont folyamatosan ismételni kell, ezért a loop függvénybe kell tenni !

```
void loop( )
 {
       i=0;                        // globális számláló nullázása
       while(i<4)                        // ismétlés 4-szer  (0,1,2,3)
        {
             digitalWrite(sarga, HIGH);       //  sárga LED-et  felkapcsoljuk
             delay(500);
             digitalWrite(sarga, LOW);       //  sárga LED-et  lekapcsoljuk
             delay(500);
             i++;                              // számláló növelése 1-el
        }
       i=0;                        // globális számláló nullázása
       while(i<6)                        // ismétlés 6-szor  (0,1,2,3,4,5)
        {
             digitalWrite(zold, HIGH);         //  zöld LED  fel
             delay(500);
             digitalWrite(zold, LOW);         //  zöld LED le
             delay(500);
             i++;                              // számláló növelése 1-el
        }
 }
```

4.7. Feladatok

Írj programokat az Arduino 2,3,4,5,6,7 lábaira kapcsolt LED-ek vezérlésére

1. feladat

- A két szélső LED (L2,L7) felvillantása felváltva (2s-2s), négyszer ismételve !
- ezután villanjon fel ötször a két középső (L4,L5) LED (3s-3s)
- majd a program leáll

2. feladat

- A két középső (L4,L5) LED felvillantása egyszerre (1s), majd a két mellettük lévő (L3,L6) egyszerre 1s-ig, majd a két szélső (L2,L7)
- És ismétlődjön elölről az egész folyamatosan !

3. feladat

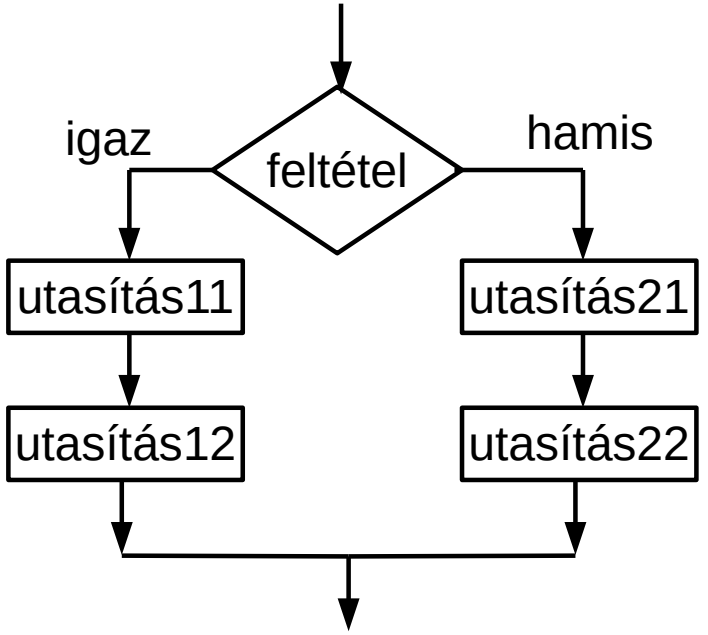
futófény: egyszerre mindig egy LED világít 1s ideig, DE !

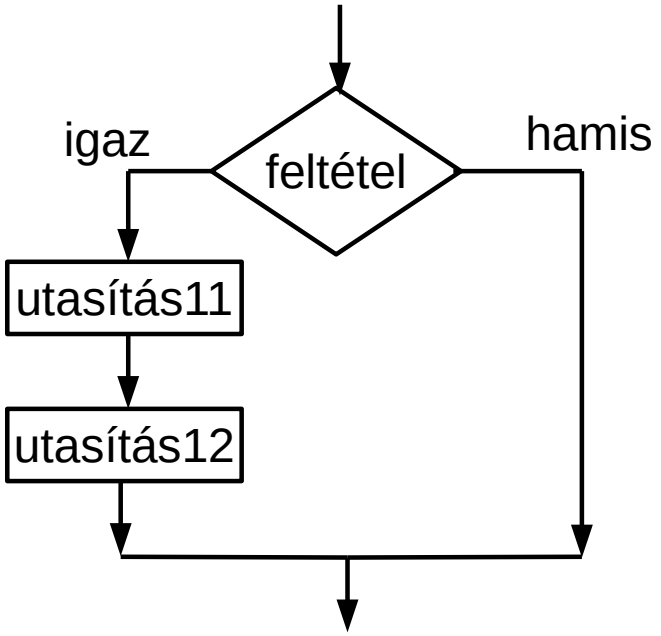
- 4-szer előre → L2 - L3 - L4 - L5 - L6 - L7 - L2 - L3 - L4 - ……
- ezután 4-szer hátra → L7 - L6 - L5 - L4 - L3 - L2 - L7 - L6 - L5 - ……
- És ismétlődjön elölről az egész folyamatosan !

5.1. Elágazás

Feltételes struktúra (elágazás):

- a program végrehajtás két úton megy tovább → igaz és hamis ág
- csak akkor hajtunk végre utasításokat ha valamilyen feltétel teljesül (igaz ág)
- vagy, ha a feltétel nem teljesül akkor más utasításokat hajtunk végre
- a hamis ág lehet üres is ! (nem tartalmaz utasítást)





5.2. Feltételes utasítás

Feltételes utasítás: if → elágazást hoz létre a programban

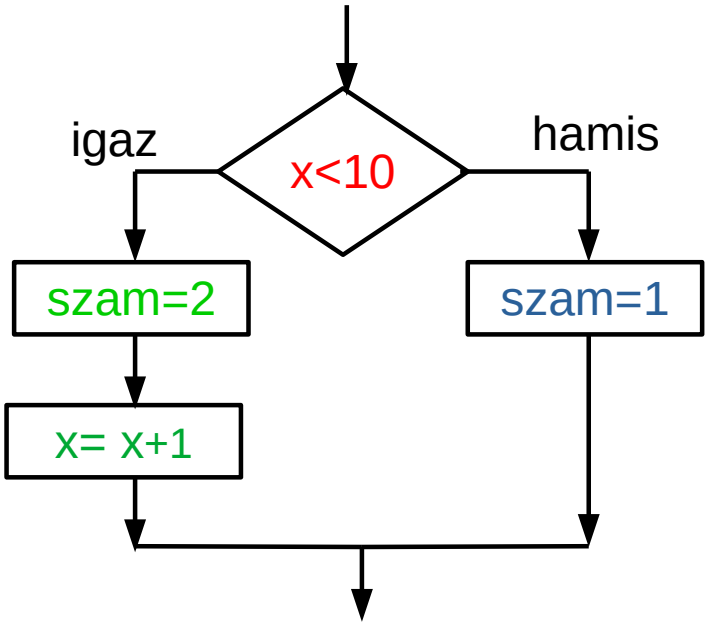
- szintaktikája:

```
if (feltétel) { igaz ág utasításai  }
else { hamis ág utasításai  }
```

// az else rész elhagyható !

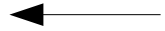
pl.

```
if(x<10)  { szam=2;   x++;  }
else { szam=1;  ... }
```



De gyakoribb a következő elrendezés:

```
if(x<10)
{
    szam=2;
    x++;
}
```



// igaz ág

```
else
{
    szam=1;
}
```


// hamis ág

5.3. Feltételes utasítás

Feltételes utasítás használata

pl. a programban folyamatosan számolni kell 1-től 8-ig, majd kezdeni elölről (1,2,3,4,5,6,7,8,1,2,3, ….)

```
byte szamlal=1;
      …
      ...
if(szamlal<8)  szamlal++;          // ha csak egy utasítás van
else szamlal=1;                    //  akkor  {  } elhagyható !
      …
      ...
```

megoldás másféleképpen:

```
      ...
szamlal++;                         // először növelünk
if(szamlal>8) szamlal=1;           //  ha túlléptük a határt →
      …                            //  kezdőérték beállítása újra
```

5.4. For ciklus

- amíg a feltétel igaz addig ismétli az utasításokat
- főleg akkor célszerű használni, ha adott (előre ismert) számú ismétlés szükséges
  (pl. tömbök kezelése)

Szintaktikája:

```
for (ciklus változó kezdőérték; feltétel; ciklus változó léptetése)
 {
      ismétlendő utasítások
 }
```

pl. a számok összeadása 1-től 20-ig for ciklussal

```
byte i;
int osszeg=0;
for(i=1;i<21;i++)
 {
      osszeg=osszeg+i;
 }
```

Ugyanez while ciklussal

```
byte i=1;
int osszeg=0;
while(i<21)
 {
      osszeg=osszeg+i;
      i++;
 }
```

5.5. Arduino függvények

„digitalRead” függvény

Digitális bemenet olvasása. A digitális bemenetként beállított kivezetésen lévő feszültségszintnek megfelelő digitális értéket (0 vagy 1) adja vissza.
Meghívása: digitalRead(pin);

```
byte b1;
byte b2;

b1 = digitalRead(1);          //  1-es láb állapotának beolvasása ’b1’ változóba
b2 = digitalRead(6);          //  6-os láb állapotának beolvasása ’b2’ változóba
```

„analogRead” függvény

Analóg bemenet olvasása. Az analóg bemenetként használható kivezetésen lévő feszültségszint digitalizált értéket adja vissza 10 bites számként ! (0 - 1023)
Meghívása: analogRead(pin);

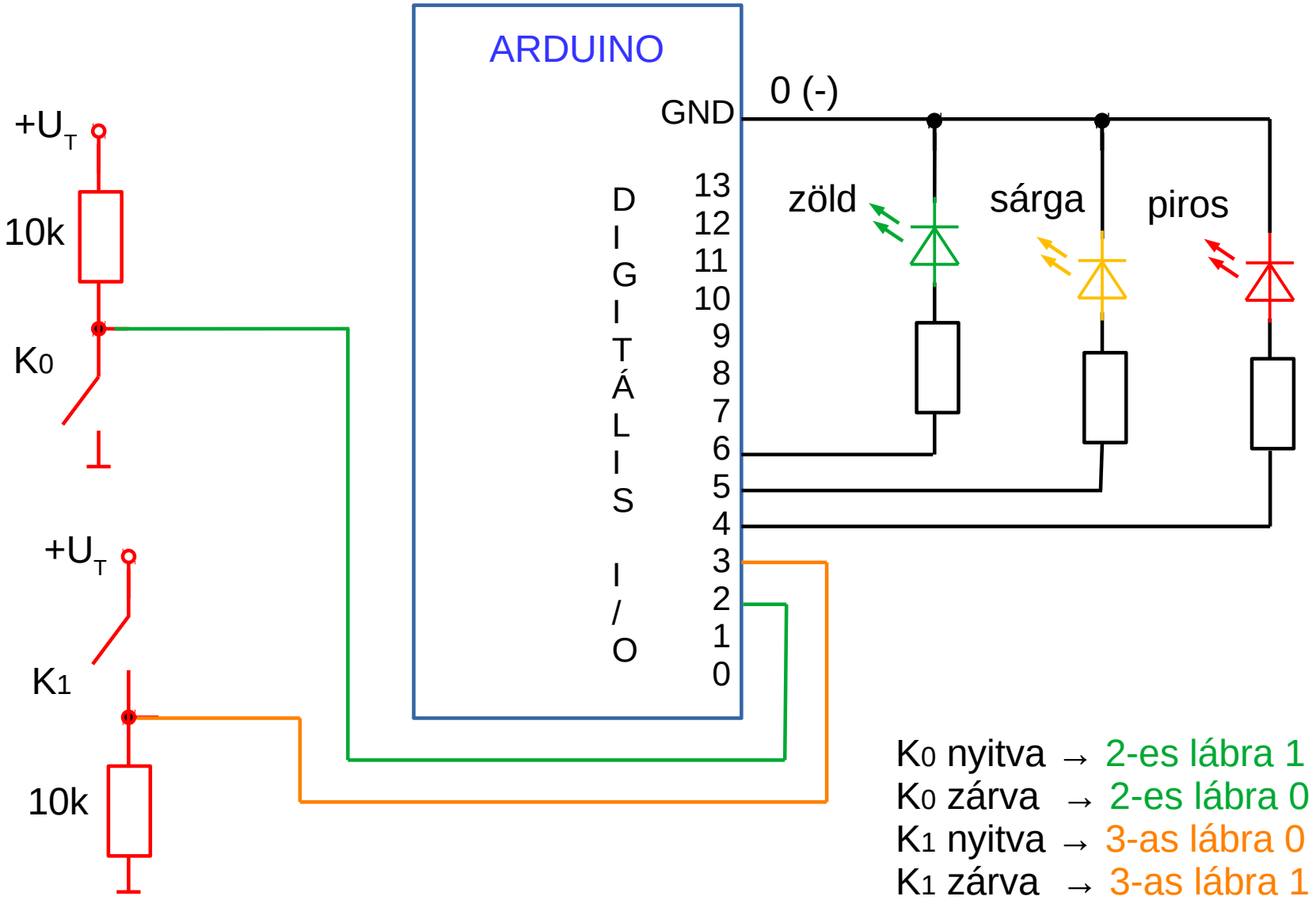
```
int a1;
int a2;

a1 = analogRead(14);    //  14-es (A0) láb állapotának beolvasása ’a1’ változóba
a2 = analogRead(15);    //  15-ös (A1) láb állapotának beolvasása ’a2’ változóba
```

Nem lehet akármelyik láb analóg bemenet !!!
Külön oldalon vannak kivezetve A0, A1 … A5 névvel (14-es, 15-ös, … 19-es lábak)

6.1. Digitális bemenetek kezelése

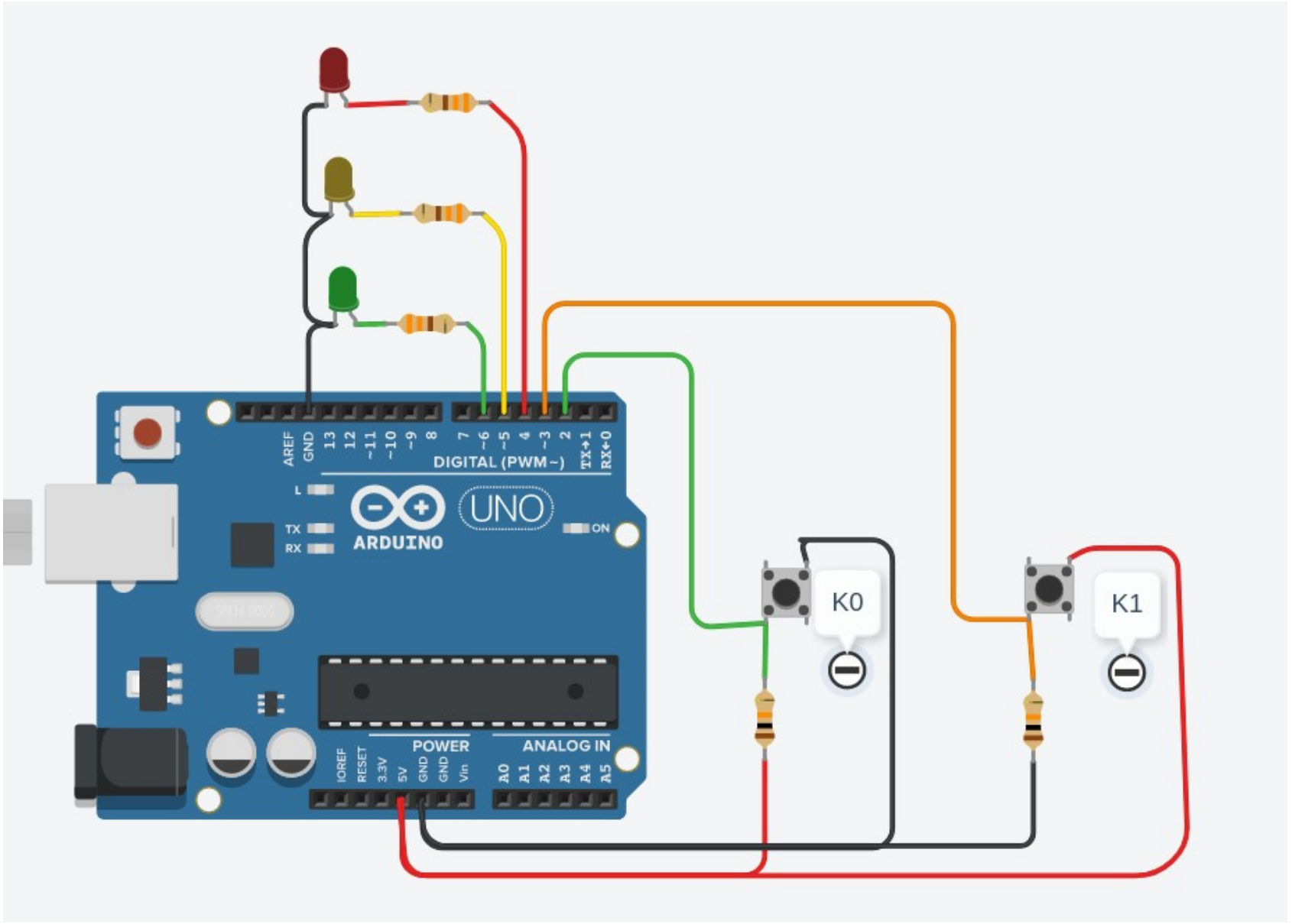
LED-ek vezérlése nyomógombbal, hardver



K0 nyitva → 2-es lábra 1
K0 zárva → 2-es lábra 0
K1 nyitva → 3-as lábra 0
K1 zárva → 3-as lábra 1

6.2. Digitális bemenetek kezelése

LED-ek vezérlése nyomógombbal, hardver



6.3. A kapcsolók érzékelése

Kapcsolók bekötése

Kétféleképpen köthetjük be a kapcsolókat
- kapcsolhatunk vele 1 szintet (tápfeszültséget) → a rajzon K_1
- kapcsolhatunk vele 0 szintet → a rajzon K_0

K_1 nyitva → 3-as lábra 0
K_1 zárva → 3-as lábra 1

K_0 nyitva → 2-es lábra 1
K_0 zárva → 2-es lábra 0

Digitális bemenet beállítása

A láb beállítása bemenetnek a „pinMode” függvénnyel

```
pinMode(3, INPUT);        //  3-as láb  bemenet lesz
pinMode(2, INPUT);        //  2-es láb  bemenet lesz
```

Bemenet értékének lekérdezése

```
byte k1;
byte k0;
k1 = digitalRead(3);      //  3-as láb állapotának beolvasása ’k1’ változóba
k0 = digitalRead(2);      //  2-es láb állapotának beolvasása ’k0’ változóba
```

6.4. Elágazás nyomógomb hatására

6.1. mintafeladat

- ha K1 le van nyomva → zöld LED villog folyamatosan
- ha K1 nincs lenyomva → piros LED villog folyamatosan

```
byte k1;     // változó, amely tárolja a nyomógomb állapotát
void setup( )
 {
        pinMode(3, INPUT);       //  3-as láb  bemenet lesz (K1)
        pinMode(4, OUTPUT);    //  4-es láb  kimenet lesz (piros LED)
        pinMode(6, OUTPUT);    //  6-os láb  kimenet lesz (zöld LED)
 }
void loop( )
 {
        k1 = digitalRead(3);              // K1 nyomógomb lekérdezése
        if(k1==1)                         // ha K1 le van nyomva
         {
              digitalWrite(6, HIGH);    // zöld LED felkapcs.
         }
        else                              // egyébként (ha K1 nincs lenyomva)
         {
              digitalWrite(4, HIGH);    // piros LED felkapcs.
         }
        delay(500);
        digitalWrite(4, LOW);       // piros LED lekapcs.
        digitalWrite(6, LOW);       // zöld LED lekapcs.
        delay(500);
 }
```

6.5. Nyomógomb felengedésére várunk

6.2. mintafeladat

- sárga LED felvillantása 1s-ra, sokszor (1s szünetekkel)
- ha K0 kapcsolót lenyomjuk → leáll a villogás, felengedése után folytatódik
- while(feltétel) { } → amíg a feltétel igaz → addig fut → várakozás egy eseményre → most a kapcsoló felengedésére

```
byte k0;     // változó, amely tárolja a nyomógomb állapotát
void setup( )
 {
        pinMode(2, INPUT);       //  2-es láb  bemenet lesz (K0)
        pinMode(5, OUTPUT);    //  5-ös láb  kimenet lesz (sárga LED)
 }
void loop( )
 {
        k0 = digitalRead(2);        // K0 állapotát lekérdezzük, és tároljuk
        while(k0==0)                  //  amíg K0 lenyomva → fut ez a ciklus
         {
              k0 = digitalRead(2);        // K0 állapotát lekérdezzük, és tároljuk
         }
        digitalWrite(5, HIGH);             // 5-ös lábra 5V → sárga LED-et  felkapcsoljuk
        delay(1000);
        digitalWrite(5, LOW);              // 5-ös lábra 0V → sárga LED-et  lekapcsoljuk
        delay(1000);
 }
```

6.6. Feladatok

Írj programokat az előző kapcsolásra, a LED-ek vezérlésére

1. feladat

- ha K1 le van nyomva → sárga LED villog folyamatosan fél másodpercenként
- ha K0 van lenyomva → piros LED villog folyamatosan fél másodpercenként

2. feladat

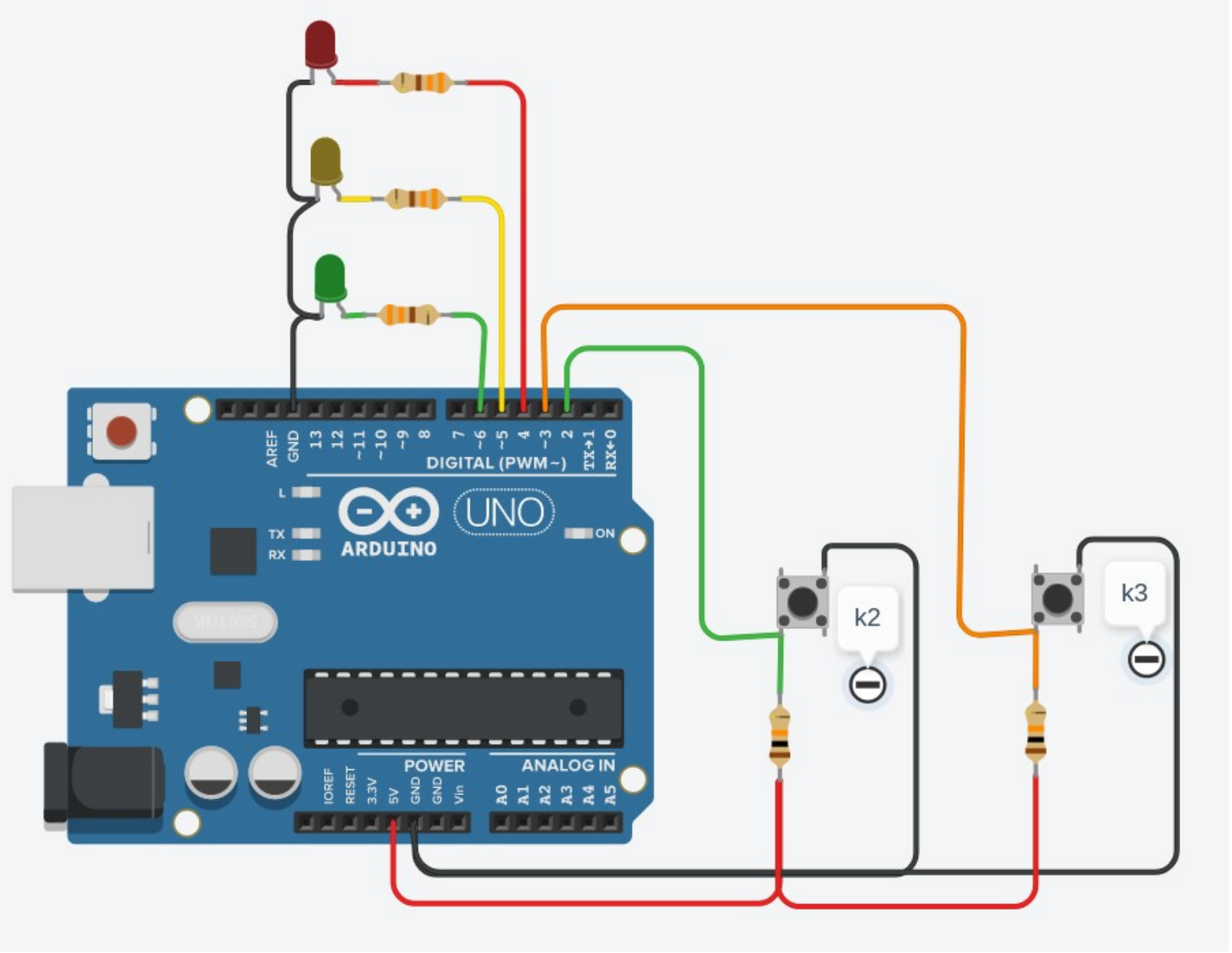
- A három LED egymás után világít folyamatosan, fél másodpercenként (piros-sárga-zöld-piros-sárga-zöld-piros-…)
- ha K1 nyomógombot nyomva tartjuk → leáll a futófény, felengedése után folytatódik

3. feladat

- ha K0 és K1 egyszerre le van lenyomva → a zöld és piros LED egyszerre villog

6.7. Digitális bemenetek, kimenetek kezelése

LED-ek vezérlése nyomógombokkal, hardver



6.8. Digitális bemenetek, kimenetek kezelése

6.3. mintafeladat

```
//  először mindhárom LED felkapcsol, majd lekapcsol, majd a piros LED lassan villog ! (5-5 sec.)
// K2-őt (stop) megnyomva, a villogás leáll
// K3-at (start) megnyomva, a villogás újra indul

const byte pirosL=4;
const byte sargaL=5;
const byte zoldL=6;
const byte k2=2;        // stop
const byte k3=3;       // start
byte villog=1;    // változó, amely tárolja hogy villog vagy nem
byte i=1;           // számláló

void setup( )
{
        pinMode(pirosL,OUTPUT);
        pinMode(sargaL,OUTPUT);
        pinMode(zoldL,OUTPUT);
        pinMode(k2,INPUT);
        pinMode(k3,INPUT);
       digitalWrite(pirosL,HIGH);
        digitalWrite(sargaL,HIGH);
        digitalWrite(zoldL,HIGH);
        delay(1000);
        digitalWrite(pirosL,LOW);
        digitalWrite(sargaL,LOW);
        digitalWrite(zoldL,LOW);
       Serial.begin(9600);

}
```

6.9. Digitális bemenetek, kimenetek kezelése

6.3. mintafeladat

```
void loop( )
{
        if(digitalRead(k2)==0)              // K2-STOP lenyomva
                { villog=0;  }              // villogás leáll
        if(digitalRead(k3)==0)              // K3-START lenyomva
                { villog=1; i=1; }          // villogás indul

       if(villog==1)                        // ha  villogás van, akkor
        {
                if(i<51)                    // amíg a számláló (i) értéke 0-50
                        digitalWrite(pirosL,HIGH);        // a LED világít
                else                                      //ha i ->  51-100
                        digitalWrite(pirosL,LOW);   //a LED nem világít
        }
        else                                // ha nincs villogás, akkor soha nem világít
                digitalWrite(pirosL,LOW);

        delay(100);
        i++;                        // számláló léptetése
        if(i>100) i=1;              // csak 100-ig számolunk
        Serial.println(i);
}
```

7.1. Soros kommunikációt vezérlő függvények

Arduino programunkba alapból importálva van a soros kommunikációt megvalósító osztálykönyvtár, így nincs több teendőnk, csak használnunk kell a „Serial” objektum metódusait („függvényeit”). Sok függvény van, most csak a néhány legfontosabb, amit használni fogunk, hogy adatot küldjünk a programunkból

„begin” metódus

Soros kommunikáció engedélyezése, indítása, és a sebesség megadása (bit/s)

```
Serial.begin(speed);                // indítás, sebesség megadásával
                // speed: 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
        pl.  Serial.begin(9600);
```

„print” és „println”metódus

Adat küldése.

```
Serial.print(val);
   // val: szöveg vagy változó (egész, karakter, szöveg, lebegőpontos)
                pl.    Serial.print("Hello !");   Serial.print(12);  Serial.print(x);
```

```
Serial.println(val);
```

Mint a „print” de új sort is kezd a végén

„end” metódus

Soros kommunikáció befejezése

```
Serial.end();
```

7.2. Soros monitor használata

Soros monitor

Az Arduino IDE szoftverben van egy beépített soros kommunikációt támogató eszköz, a „soros monitor”. Ez a funkció a Tinkercad online szimulációs felületen is elérhető. Ezt felhasználva olvasni tudjuk az Arduino hardveren futó programunk által a soros porton küldött üzeneteket, adatokat. Ezt nagyon jól lehet használni a programunk tesztelésére is például, a változók aktuális értékét elküldve, látható hogy mi történik.

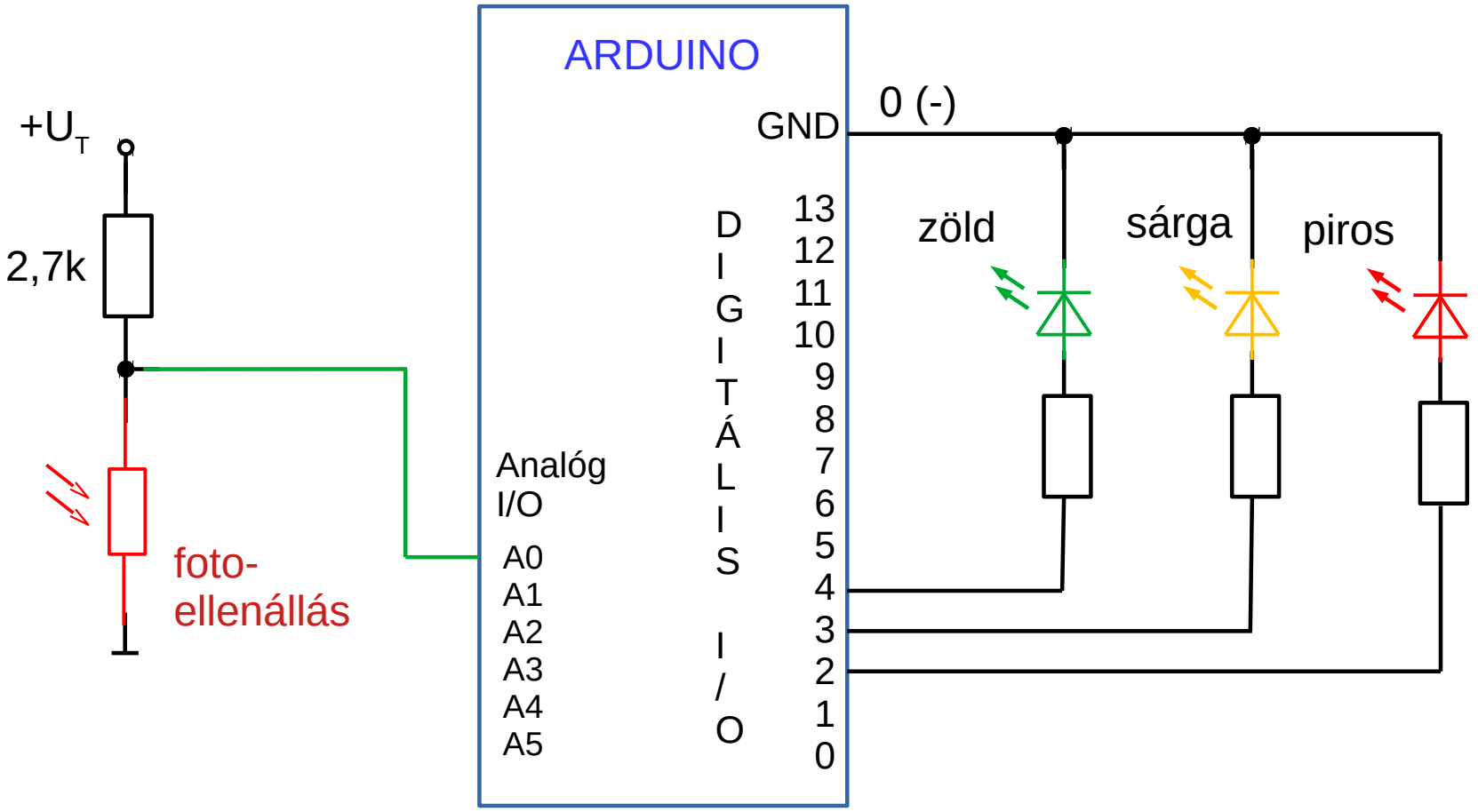
7.1. mintafeladat

```
// Küldjük el egy 1-30 ciklus ciklus változóját soros porton
  byte i=1;
  void setup( )
  {
        Serial.begin(9600);        //  soros kommunikáció beállítása
        Serial.println("Hello");   //  soros üzenet küldése
        while(i<31)                // ismétlés 30-szor
         {
            Serial.println(i);     //  soros üzenet küldése
            i++;                   // számláló növelése 1-el
            delay(400);
         }
  }
 void loop( )
  {

  }
```

8.1. Analóg bemenetek kezelése

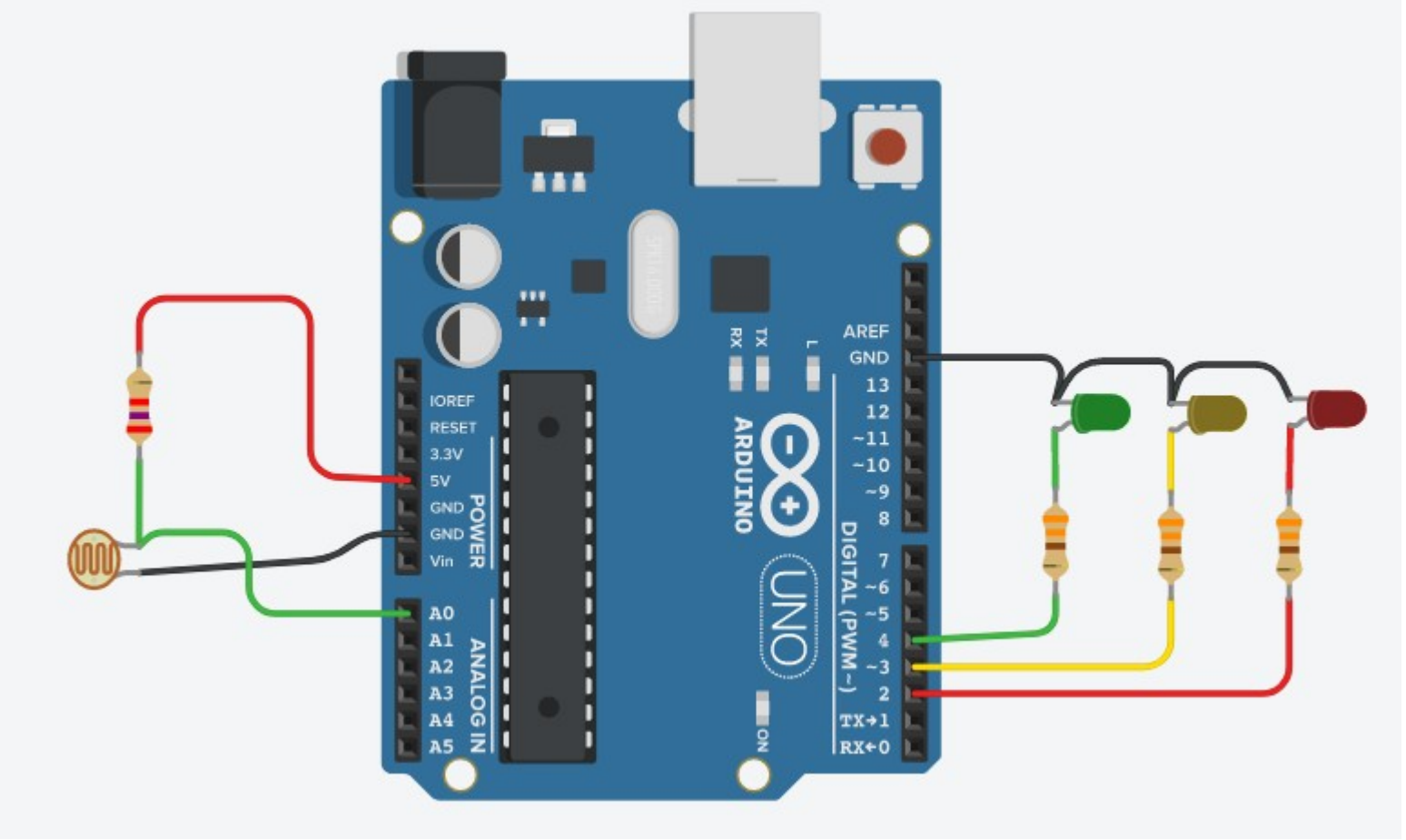
LED-ek vezérlése fotoellenállással, hardver



A0 → 14-es láb
A1 → 15-ös láb
....

8.2. Analóg bemenetek kezelése

LED-ek vezérlése fotoellenállással, hardver



8.3. Analóg bemenetek kezelése

8.1. mintafeladat

A fotoellenállást érő fény erősségétől függően változik az ellenállása → a sorba kötött fix értékű ellenállással egy feszültség osztót alkotnak, amelynek kimeneti feszültsége (ezt vezetjük az Arduino analóg bemenetére) így változik a fény változásának hatására !

Feladat: A fotoellenállást érő fény erősségétől függően más-más LED-ek villogjanak →
ha az analóg érték:
- nagyobb mint 450 → mindhárom LED villog
- 250-nél nagyobb, de 450-nél nem → zöld + sárga villog
- 250, vagy annál kisebb → csak a zöld villog

```
// változók, kimenetek, bemenetek elnevezése
const byte piros=2;          // piros LED a 2-es lábon
const byte sarga=3;          // sárga LED a 3-as lábon
const byte zold=4;           // zöld LED a 4-es lábon
const byte feny=14;          // A0 analóg bemeneten (14) fotoellenállás

int sotet=1023;              // tárolja az analóg bemenet értéket

void setup()      // kezdeti beállítások
{
   pinMode(piros, OUTPUT);        //  'piros'  kimenet lesz
   pinMode(sarga, OUTPUT);        //  'sarga'  kimenet lesz
   pinMode(zold, OUTPUT);         //  'zold'  kimenet lesz
   pinMode(feny, INPUT);          //  'feny'  bemenet lesz
}
```

8.4. Analóg bemenetek kezelése

8.1. mintafeladat, folytatás

```
void loop()             //  végtelen ciklus, folyamatosan ismétlődnek
{
    sotet=analogRead(feny);          // analóg bemenet olvasása
    if(sotet>450)
     {
        digitalWrite(zold, HIGH);        // zöld LED felkapcs.
        digitalWrite(sarga, HIGH);       // sárga LED felkapcs.
        digitalWrite(piros, HIGH);       // piros LED felkapcs.
     }
    else if(sotet>250)
     {
        digitalWrite(zold, HIGH);        // zöld LED felkapcs.
        digitalWrite(sarga, HIGH);       // sárga LED felkapcs.
     }
    else
        digitalWrite(zold, HIGH);        // zöld LED felkapcs.
    delay(500);
    digitalWrite(zold, LOW);             // zöld LED lekapcs.
    digitalWrite(sarga, LOW);            // sárga LED lekapcs.
    digitalWrite(piros, LOW);            // piros LED lekapcs.
    delay(500);

}
```

8.5. Analóg bemenetek kezelése

8.2. mintafeladat

A 8.1 feladatot egészítsük ki soros kommunikációval !
A beolvasott analóg értéket folyamatosan küldjük el a soros porton.

```
// változók, kimenetek, bemenetek elnevezése
const byte piros=2;          // piros LED a 2-es lábon
const byte sarga=3;          // sárga LED a 3-as lábon
const byte zold=4;           // zöld LED a 4-es lábon
const byte feny=14;          // A0 analóg bemeneten (14) fotoellenállás

int sotet=1023;              // tárolja az analóg bemenet értéket

void setup()      // kezdeti beállítások
{
    pinMode(piros, OUTPUT);        //  'piros'  kimenet lesz
    pinMode(sarga, OUTPUT);        //  'sarga'  kimenet lesz
    pinMode(zold, OUTPUT);         //  'zold'  kimenet lesz
    pinMode(feny, INPUT);          //  'feny'  bemenet lesz

   Serial.begin(9600);           //  soros kommunikáció beállítása
   Serial.println("Hello");      //  soros üzenet küldése
}
```

8.6. Analóg bemenetek kezelése

8.2. mintafeladat, folytatás

```
void loop()             //  végtelen ciklus, folyamatosan ismétlődnek
{
    sotet=analogRead(feny);          // analóg bemenet olvasása
    Serial.println(sotet);                  //  soros üzenet küldése

    if(sotet>450)
     {
        digitalWrite(zold, HIGH);        // zöld LED felkapcs.
        digitalWrite(sarga, HIGH);       // sárga LED felkapcs.
        digitalWrite(piros, HIGH);       // piros LED felkapcs.
     }
    else if(sotet>250)
     {
        digitalWrite(zold, HIGH);        // zöld LED felkapcs.
        digitalWrite(sarga, HIGH);       // sárga LED felkapcs.
     }
    else
        digitalWrite(zold, HIGH);        // zöld LED felkapcs.
    delay(500);
    digitalWrite(zold, LOW);             // zöld LED lekapcs.
    digitalWrite(sarga, LOW);                   // sárga LED lekapcs.
    digitalWrite(piros, LOW);            // piros LED lekapcs.
    delay(500);

}
```