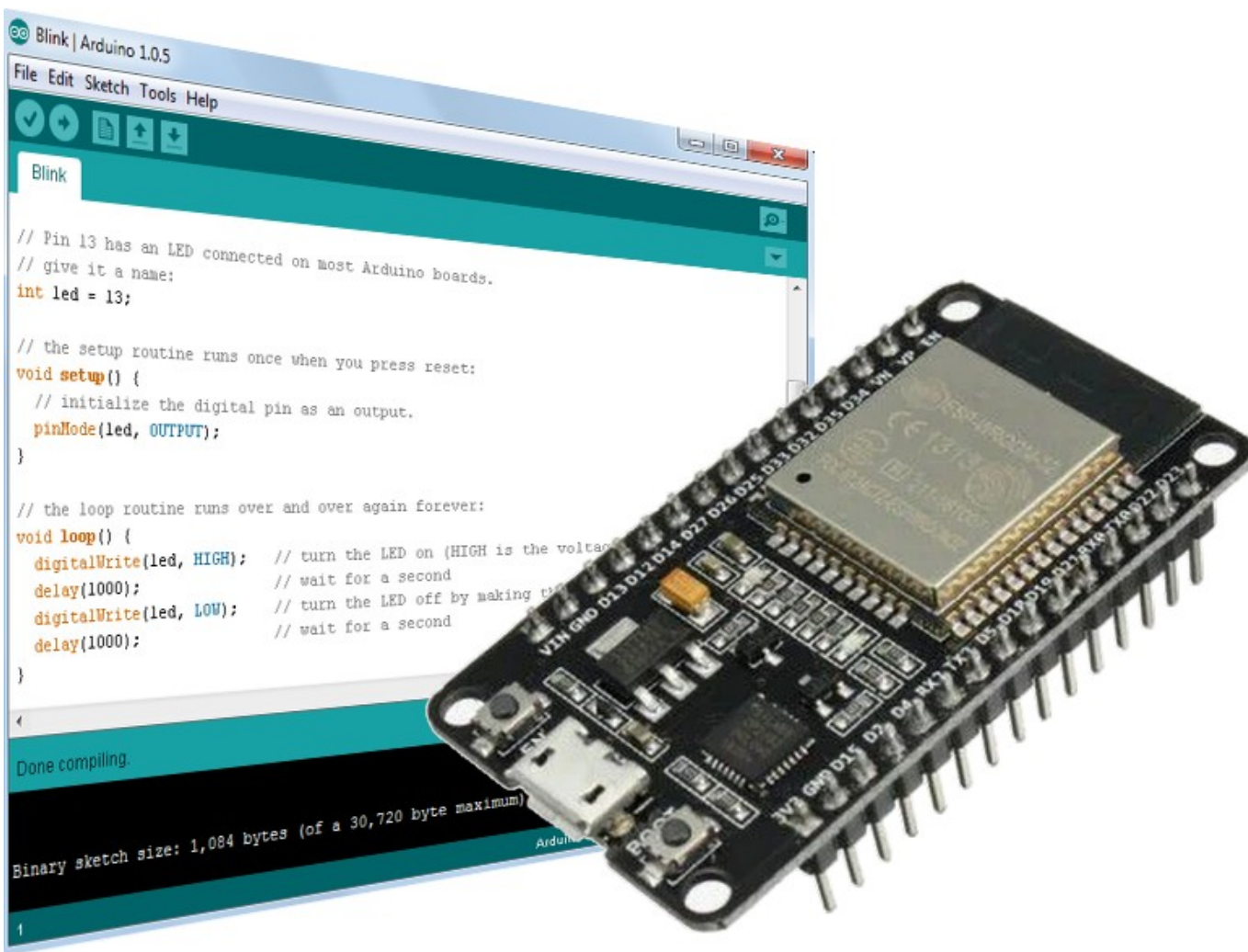


ESP32 mikrovezérlők programozása Arduino környezetben



2. Változók típusai, feltételvizsgálat, programelágazás

Példaprogramok

ESP32_variable_types.ino – változók típusai

valtozok.ino – változók túl- és alulcsordulása

ESP32_interrupt – programmegszakítás

ESP32_blink_control – LED villogás vezérlése

ESP32_multitasking – nem blokkoló időzítések

ESP32_gombvezerelt_ledek – LED
villogás vezérlése másképpen



Változók

- Programfutás alatt a mért vagy számított adatokat ideiglenesen el kell valahova helyezni, erre a RAM memóriában kell helyet foglalni
- A lefoglalt helyeket névvel látjuk el, ezzel hivatkozhatunk rájuk, amikor újra elővesszük, vagy felülírjuk az adatot
- A RAM memória csak a kikapcsolásáig őrzi meg az adatokat
- A változókat felfoghatjuk úgy, mint egy sokfiókos doboz (a memória) egy-egy fiókját, amibe egyvalamit beletehetünk, vagy kivehetünk
- A változó neve kötelezően kis- vagy nagybetűvel kezdődjön de tartalmazhat számjegyet vagy aláhúzásjelet is



Változó deklaráció és értékadás

- Változó deklarálásánál meg kell adni a típusát és a változó nevét például: `int a, b, c;` három előjeles egész változót hoz létre
- Módosítók: `signed` vagy `unsigned` (előjeles vagy előjel nélküli)
- Deklaráláskor nem kötelező, de megadhatunk kezdőértéket is például: `int a = 2, b = -3, c = 120;`
- Az Arduino C/C++ környezete azt is megengedi, hogy a változó deklarációja ne a program elején, hanem az első használatbavétel helyén legyen, mint például az alábbi programrészletben

```
void setup() {  
  pinMode(21, INPUT_PULLUP);    // GPIO21 digitális bemenet belső felhúzással  
}  
  
void loop() {  
  boolean sw = digitalRead(21); // Nyomógomb állapotának beolvasása  
  int adc = analogRead(34);     // ADC csatorna beolvasása  
  float temp = (adc*3300/4096-500)/10.0; //Adatok átszámítása mv-ra, °C-ra  
  ...  
}
```

Változók típusai az Arduino nyelvben

- A legfontosabb változó típusok az alábbi táblázatban láthatók

Típus	Méret [bájt]	Min. érték	Max. érték	STDINT típus (ANSI C90)	ESP32
boolean	1	false (0)	true (1)		
char	1	-128	127	int8_t	0..255
unsigned char <i>vagy byte</i>	1	0	255	uint8_t	0..255
int	2	-32 768	32 767	int16_t	int32_t
unsigned int <i>vagy word</i>	2	0	65 535	uint16_t	uint32_t
long	4	-2,147,483,648	2,147,483,647	int32_t	int32_t
unsigned long	4	0	4,294,967,295	uint32_t	uint32_t
float	4	-3.4028235E+38	3.4028235E+38		single prec.
double	erőforrás hiányában ugyanaz, mint a float típus				double prec.

ESP32_variable_types.ino

- Az alábbi programmal azt vizsgáljuk, hol csordulnak túl a változók

```
char c = 127;
int i = 32767;
long l = 2147483647;
double d = 1.0 / 7.0;
float f = 1.0 / 7.0;
void setup() {
  Serial.begin(115200);
  Serial.print("char c = ");
  Serial.print(c);
  Serial.print(" c+1 = ");
  Serial.println(c+1);
  c = 127;
  int64_t cmax = 127;
  do {
    cmax = c;    c++;
  } while(c > cmax);
  Serial.print("char maxvalue = ");
  Serial.println(cmax);
  c = 1;
  int64_t cmin = 1;
  do {
    cmin = c; c--;
  } while(c < cmin);
  Serial.print("char minvalue = ");
  Serial.println(cmin);
```

```
Serial.print("int i = ");
Serial.print(i);
Serial.print(" i+1 = ");
Serial.print(i+1);
Serial.print(" i(65535)+1 = ");
i = 65535;
Serial.print(i + 1);
Serial.print(" i(2147483647)+1 = ");
i = 2147483647;
Serial.println(i + 1);
i = 2147483640; cmin = 0; cmax = 0;
do {
  cmax = i; i++;
  Serial.println(i);
} while (i > cmax);
//--- Long and float típusok ----
Serial.print("long l = ");
Serial.print(l);
Serial.print(" l+1 = ");
Serial.println(l + 1);
Serial.print("float f(1/7) = ");
Serial.println(f,10);
Serial.print("double d(1/7) = ");
Serial.println(d,15);
}
```

ESP32_variable_types.ino

- Az ábrán a program futási eredményét láthatjuk

Eredmény:

char:
0 .. 255

int:
32 bit

long:
32 bit

float:
egyszeres
pontosság

double:
dupla
pontosság

```
COM4
Változók vizsgálata
char c = 127  c+1 = 128 ← nincs túlsordulás
char maxvalue = 255      ← signed char c-vel orvosolható!
char minvalue = 0
int i = 32767  i+1 = 32768  i(65535)+1 = 65536  i(2147483647)+1 = -2147483648
2147483641
2147483642
2147483643
2147483644
2147483645
2147483646
2147483647
-2147483648
long l = 2147483647  l+1 = -2147483648 ← 31. bitnél túlsordulás
float f(1/7) = 0.1428571492 ← egyszeres pontosság
double d(1/7) = 0.142857142857143 ← dupla pontosság
```

31. bitnél túlsordulás



31. bitnél túlsordulás

egyszeres pontosság

dupla pontosság

Autoscroll Show timestamp

Carriage return ▾

115200 baud ▾

Clear output

Túlcsordulás, számábrázolási pontosság

- **Túlcsordulás** akkor következik be, ha a változóban maximálisan ábrázolható számot meghaladja a művelet eredménye

```
byte a = 255;
```

```
a = a + 1; //nem 256, hanem 0 lesz!
```

$$\begin{array}{r} 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ \hline \end{array} = 255_{10}$$

$$+ \begin{array}{r} 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1 \\ \hline \end{array} = 1$$

$$\begin{array}{r} \textcircled{1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\ \hline \end{array} = 0 \text{ (8 biten, mivel a 9. bit elveszett)}$$



- Lebegőpontos (float) típusú változóknál a **számábrázolási pontosságot** két tényező korlátozza:

- ❖ Az értékes jegyek száma kb. 6-7, így például

`float a = 3.1234567E10;` esetén **a** és **a+1** nem lesz különböző

- ❖ Bizonyos számok (például 0.1) kettes számrendszerben nem ábrázolható véges számú számjeggyel – ahogyan pl. **1/3** vagy **1/7** sem ábrázolhatók véges tizedestörttel

valtozok.ino

- Az alábbi programban egyszerű példákat mutatunk a változók használatára

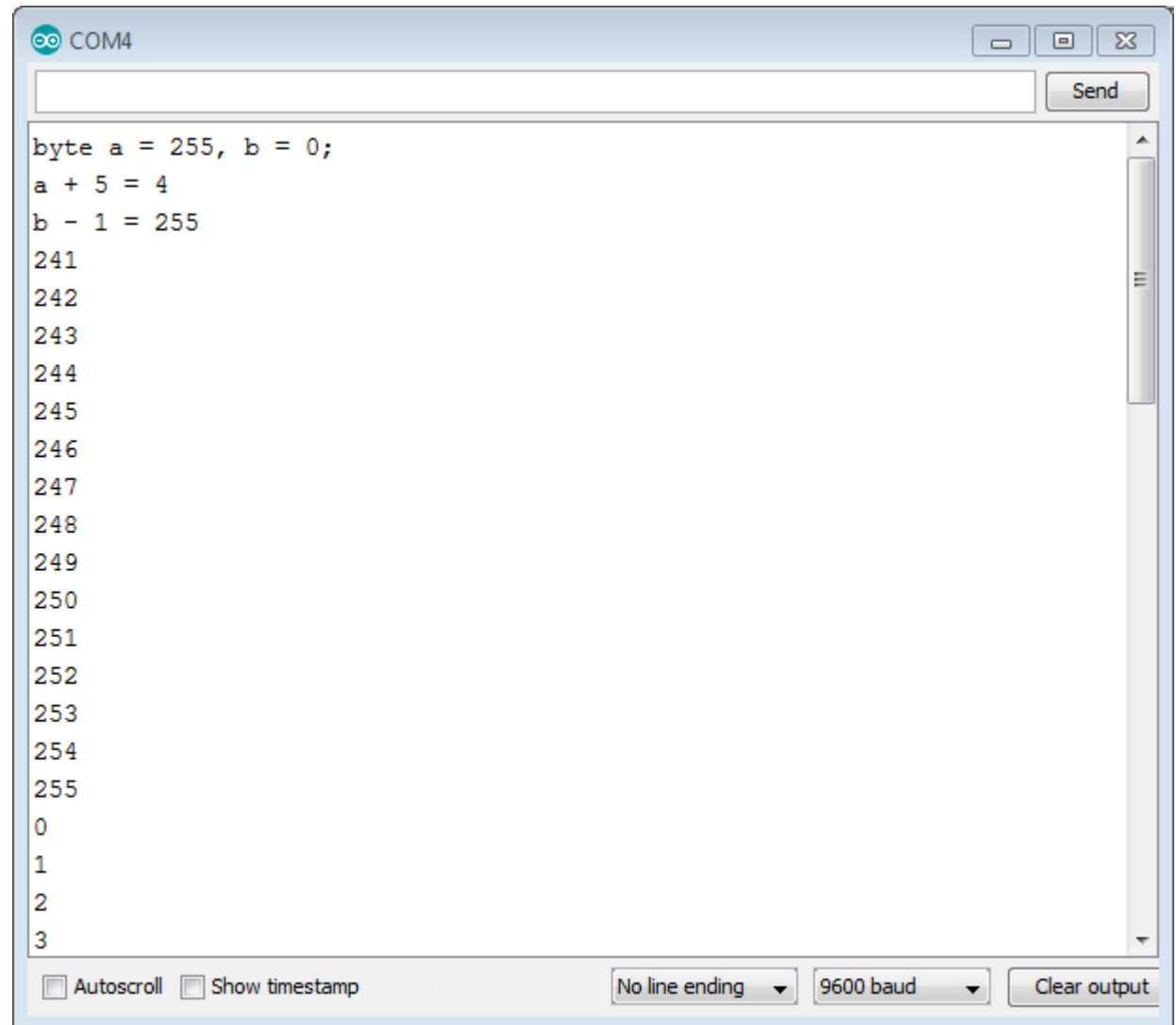
```
byte szamlalo = 240;           // szamlalo változó deklarálása kezdőértéke 240
byte a = 255, b = 0;

void setup() {
  Serial.begin(9600);         // A soros porton fogunk kiíratni
  Serial.println("byte a = 255, b = 0;");
  Serial.print("a + 5 = ");
  a = a + 5;                  // Túlcsordulás: 255 + 5 = 4 lesz (260 - 256)
  Serial.println(a);
  Serial.print("b - 1 = ");
  b = b - 1;                  // Alulcsordulás: 0 - 1 = 255 lesz (256 - 1)
  Serial.println(b);
}

void loop() {
  szamlalo++;                 // Egy hozzáadása minden ciklusban
  Serial.println(szamlalo);   // Kiíratjuk szamlalo aktuális tartalmát
}
```

valtozok.ino futási eredménye

- A program eredményének megtekintéséhez meg kell nyitnunk a terminál ablakot és újra kell indítani a programot
- **a+5** túlcsorduláshoz vezetett
- **b-1** alulcsorduláshoz vezetett
- A **szamlo** változó növelgetése 255 elérése után túlcsordulást okoz



```
COM4
byte a = 255, b = 0;
a + 5 = 4
b - 1 = 255
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
0
1
2
3
```

Konstans vagy #define?

- A `const` előtag konstans értéket jelent, az így deklarált változó csak olvasható lesz, értéke nem módosítható pl:
`const int RED_LED = 23;`
- A `const` módosítóval deklarált változó megőrzi típusát és érvényességi körét
- A `const` „változó” tömb is lehet
- A `#define` direktíva egy szimbólumot definiál a fordító minden előfordulásnál behelyettesíti a forráskódba pl.: `#define RED_LED 23`
- A `#define` direktívával létrehozott szimbólum nem definiál típust és globális érvényességű

```
const float pi = 3.14;
float x;
...
//it's fine to use consts in math
x = pi * 2;

//Illegal, you can't modify a constant
pi = 7;
```

```
#define RED_LED 23
#define GREEN_LED 22
#define BUTTON 21

// Hibás definíciók:
#define LED = 23 // Nem lehet '='
#define LED 23; // Nem lehet ';'
```

Változók érvényességi köre

- A változók egyik fontos jellemzője az érvényességi kör (scope)
- Arduino környezetben minden olyan változó *globális*, amit a függvényeken kívül deklarálnak
- A függvényekben vagy a *for* ciklus inicializálás részében deklarált változók *lokálisak*, azaz csak az adott függvény, vagy a *for* ciklus törzsében érhetők el

```
int PWMduty; // ez a változó bármelyik függvényből elérhető

void setup() {
    // ...
}

void loop() {
    int i; // az "i" változó csak a "loop" függvényben érhető el
    float f; // az "f" változó csak a "loop" függvényben érhető el
    // ...
    for (int j = 0; j < 100; j++) {
        // a "j" változó csak a "for" cikluson belül érhető el
    }
}
```

Statikus és illékony változók

- A *static* módosító azt jelzi, hogy a lokális változó értékét meg kell őrizni a függvényhívások között

```
int randomWalk(int moveSize) {  
    static int place; // variable to store value in random walk  
    place = place + (random(-moveSize, moveSize + 1));  
    return place;  
}
```

- A *volatile* módosító azt jelzi, hogy a változó értékét minden hivatkozáskor a tárhelyről kell elővenni (ideiglenesen tárolt értéke nem használható). Akkor van erre szükség, ha más programszál vagy megszakítást kiszolgáló rutin módosíthatja az értékét

```
volatile byte state = LOW; ESP32_interrupt.ino  
void setup() {  
    pinMode(LED_BUILTIN, OUTPUT); // LED_BUILTIN itt GPIO2  
    attachInterrupt(digitalPinToInterrupt(21), blink, FALLING);  
}  
void loop() { digitalWrite(LED_BUILTIN, state); }  
  
void blink() { state = !state; } // change state at each interrupt
```


Aritmetikai műveletek

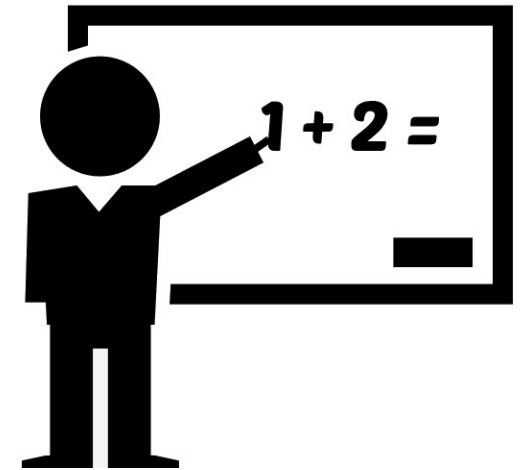
- Számítási (aritmetikai) műveletek

```
a = a + 5; // összeadás
```

```
a = c - b; // kivonás
```

```
a = a * 7; // szorzás
```

```
a = b / 2; // osztás
```



- Fontos, hogy milyen típusú változót használunk, mert egész típusoknál $10 / 4$ eredménye nem 2.5, hanem 2 lesz!

- **Osztási maradék** előállítása a `%` művelettel történik

Legyen pl. $a = 15$, akkor az $a/6$ osztás eredménye 2 lesz, az $a\%6$ művelet eredménye pedig 3 lesz (mivel $15 = (2 * 6) + 3$)

- Összevonások (rövidebb jelölésmód)

```
x++ // ugyanaz, mint: x = x + 1
```

```
x += y // ugyanaz, mint: x = x + y
```

```
x *= y // ugyanaz, mint: x = x * y
```

```
x-- // ugyanaz, mint: x = x - 1
```

```
x -= y // ugyanaz, mint: x = x - y
```

```
x /= y // ugyanaz, mint: x = x / y
```

Relációs és logikai operátorok

- Logikai kifejezésekben, illetve feltételvizsgálatoknál az alábbi operátorok (műveletek) állnak rendelkezésre:

Relációs operátorok

`==` egyenlő pl. `x == y`
`!=` nem egyenlő pl. `x != y`
`<` kisebb mint pl. `x < y`
`>` nagyobb mint pl. `x > y`
`<=` kisebb, vagy egyenlő
`>=` nagyobb, vagy egyenlő

Logikai operátorok

Logikai változókon vagy logikai kifejezéseken végzett műveletekhez használhatók

`!` Tagadás (negáció) pl. `!a`
`&&` ÉS művelet pl. `a && b`
`||` VAGY művelet pl. `a || b`

- **Megjegyzések:**

- ❖ A relációs, illetve a logikai műveletek eredménye egy logikai (boolean) mennyiség, ami csak 0 (ha hamis) vagy 1 (ha igaz) értéket vehet fel.
- ❖ A C fordítók kiterjesztett értelmezéssel bármilyen 0-tól különböző számot is elfogadnak igaz értéként. Így például `if(x != 0)` helyett `if(x)` -et is írhatunk.
- ❖ Ne keverjük össze a logikai operátorokat a bitenként logikai operátorokkal!
Bitenkénti műveletek: `~`, `&`, `|`, `^` Logikai műveletek: `!`, `&&`, `||`

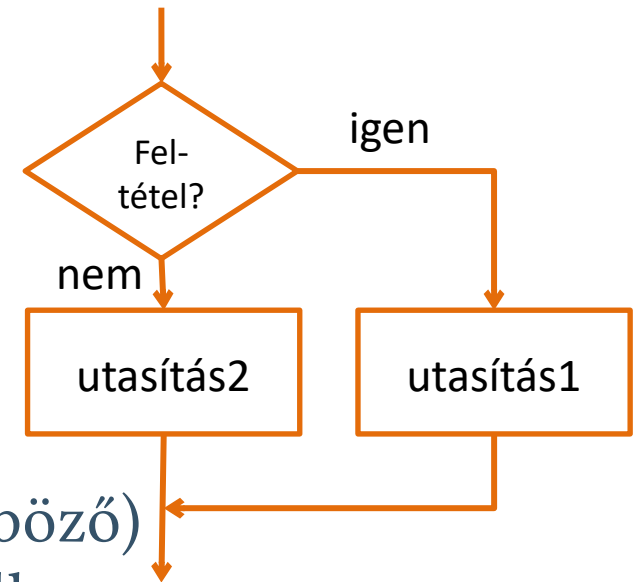
Feltételes elágazás: if – else utasítás

- Megvizsgál egy feltételt, s ezt követően a végrehajtás menete a feltételvizsgálat eredményétől függ

```
if(feltétel) utasítás1  
else utasítás2
```

(az else ág elhagyható)

- A **feltétel** egy kifejezés, ami aritmetikai és relációs operátorokat is tartalmazhat. A kiértékelés eredménye
igaz: ha a kifejezés értéke **true** (0-tól különböző)
hamis: ha a kifejezés értéke **false**, vagy nulla.



- Ha a **feltétel teljesül**, akkor **utasítás1** kerül végrehajtásra, **egyébként pedig utasítás2** lesz végrehajtva. Például:
Ha BUTTON nincs lenyomva

```
if(digitalRead(BUTTON)) { digitalWrite(LED,LOW); }  
else { digitalWrite(LED,HIGH); } //Ha a gomb le van nyomva, a LED világít
```

Ha BUTTON le van nyomva

PUSH2 olvasása **1**, ha nincs lenyomva, **0**, ha le van nyomva

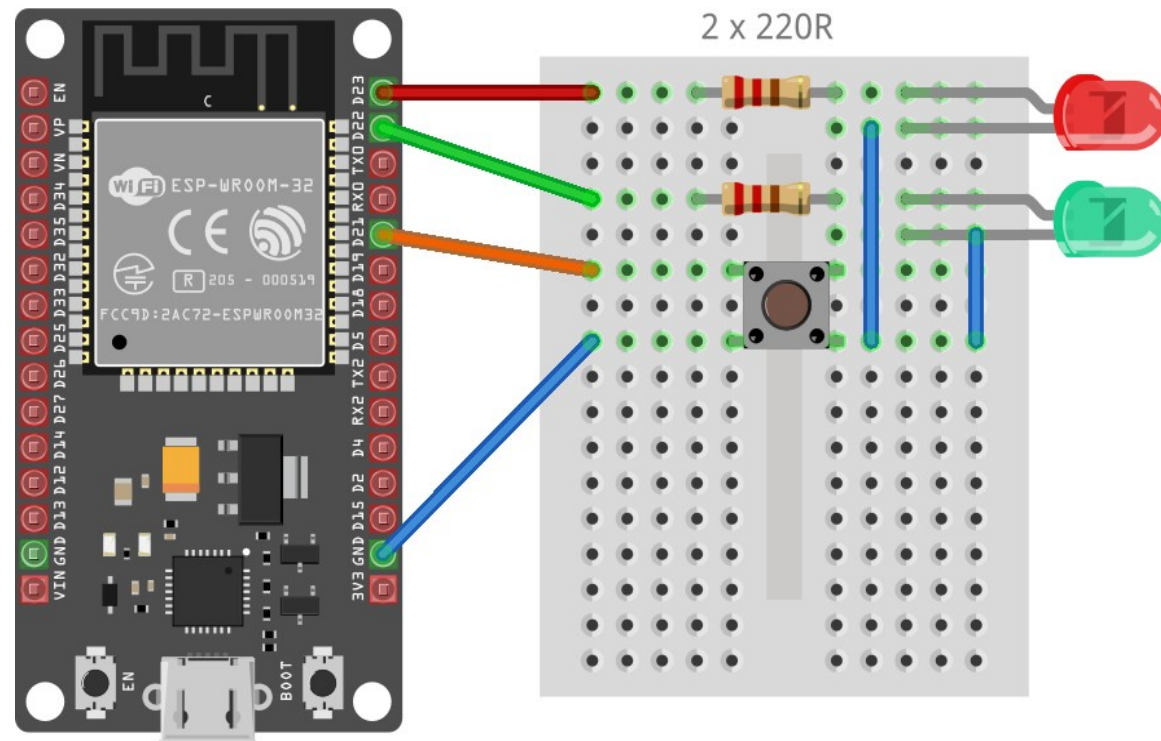
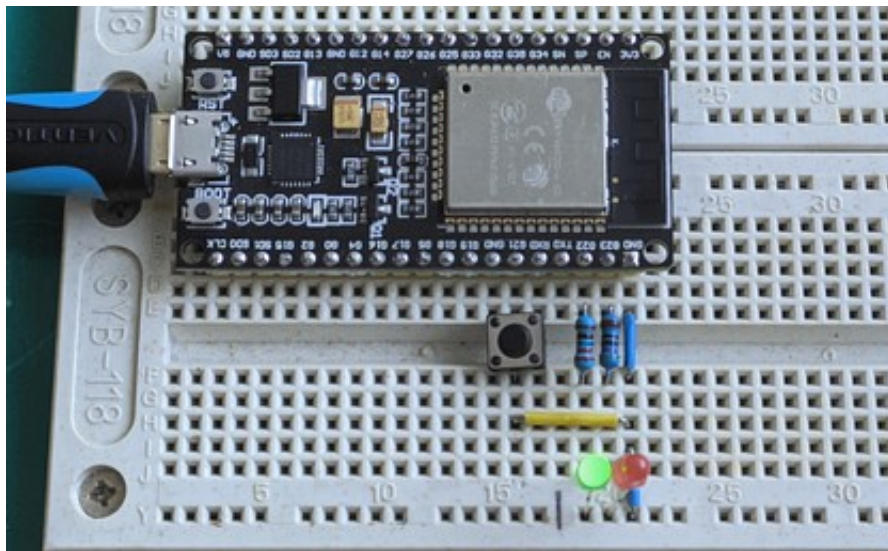
ESP32_blink_control.ino

- Villogtassunk két LED-et felváltva, de a villogás frekvenciája (az átkapcsolások közötti időtartam) a nyomógomb állapotától függjön:
 - ❖ Ha a gombot elengedjük, 500 ms-onként váltunk (1 Hz villogás)
 - ❖ Ha a gombot lenyomjuk, 100 ms-onként váltunk (5 Hz villogás)
- A kapcsolást megegyezik az előző előadás **ESP32_button2led.ino** példájában bemutatottal:

GPIO23: piros LED

GPIO22: zöld LED

GPIO21: nyomógomb



ESP32_blink_control.ino

```
#define RED_LED    23
#define GREEN_LED  22
#define BUTTON     21
```

Használjuk az `if` utasítást a nyomógomb állapota szerinti elágazáshoz

```
void setup() {
    pinMode(RED_LED, OUTPUT);           // legyen kimenet
    pinMode(GREEN_LED, OUTPUT);        // legyen kimenet
    pinMode(PUSH2, INPUT_PULLUP);      // legyen bemenet, belső felhúzással
}

void loop() {
    boolean sw;                         // Ebben tároljuk a nyomógomb állapotát
    sw = digitalRead(BUTTON);           // Beolvassuk a nyomógomb állapotát
    digitalWrite(RED_LED, HIGH);        // A piros LED világít
    digitalWrite(GREEN_LED, LOW);       // A zöld LED nem világít
    if(sw == HIGH) delay(500);          // Felengedett gomb esetén HIGH állapot van
    else delay(100);                    // Lenyomásnál 100 ms késleltetés
    digitalWrite(RED_LED, LOW);         // A piros LED nem világít
    digitalWrite(GREEN_LED, HIGH);     // A zöld LED világít
    if(sw == HIGH) delay(500);          // Felengedett gomb esetén több késleltetés
    else delay(100);                    // Lenyomásnál 100 ms késleltetés
}
```


if utasítások egymásba ágyazása

Egymásba ágyazott **if-else** szerkezeteknél, hiányzó **else** ágak esetén nem világos, hogy a meglévő **else** ág melyik **if** utasításhoz tartozik. Például az

```
if (n > 0)
    if (a > b) z = a;
    else z = b;
```

programrészletben az **else** a belső **if** utasításhoz tartozik.

Általános szabály: az **else** mindig a hozzá legközelebb eső, **else** ág nélküli

if utasításhoz tartozik. Ha nem így szeretnénk, akkor használjunk kapcsos zárójeleket! Például:

```
if (n > 0) {
    if (a > b) z = a;
}
else z = b;
```

A nem egyértelmű helyzet különösen zavaró az olyan szerkezetekben, mint az alábbi:

```
if (n >= 0)
    for (i = 0; i < n; i++)
        if (s[i] > 0) {
            Serial.println("...");
        }
else /* ez így hibás */
    Serial.println("n értéke negatív");
```

A tagolás ellenére a fordító itt az **else** ágat a belső **if** utasításhoz kapcsolja.

Az ilyen, nehezen felderíthető hibák megelőzésére használjunk kapcsos zárójeleket a második **if** utasítás körül!

Az else – if szerkezet

Az **else if** szerkezet adja a többszörös elágazások programozásának egyik legáltalánosabb lehetőségét.

A szerkezet úgy működik, hogy a program sorra kiértékeli a *kifejezéseket* és ha bármelyik ezek közül igaz, akkor végrehajtja a megfelelő *utasítást*, majd befejezi az egész vizsgáló láncot.

Itt is, mint bárhol hasonló esetben, az *utasítás* helyén kapcsos zárójelek között elhelyezett blokk is állhat.

Az utolsó **else** ág alapértelmezés szerint a „*fentiek közül egyik sem*” esetet kezeli. Ha ilyenkor semmit sem kell csinálni, ez az ág elhagyható.

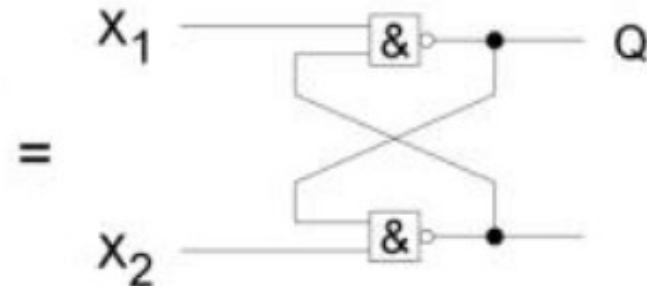
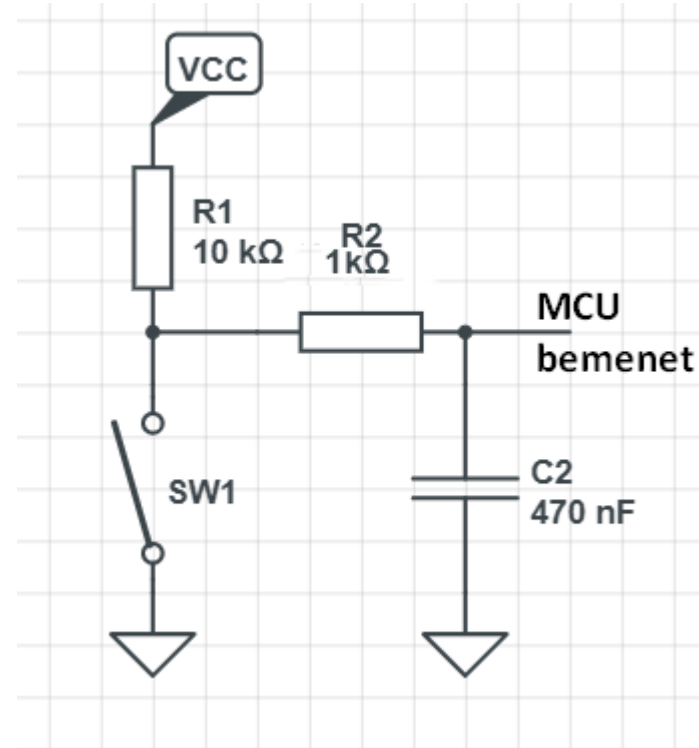
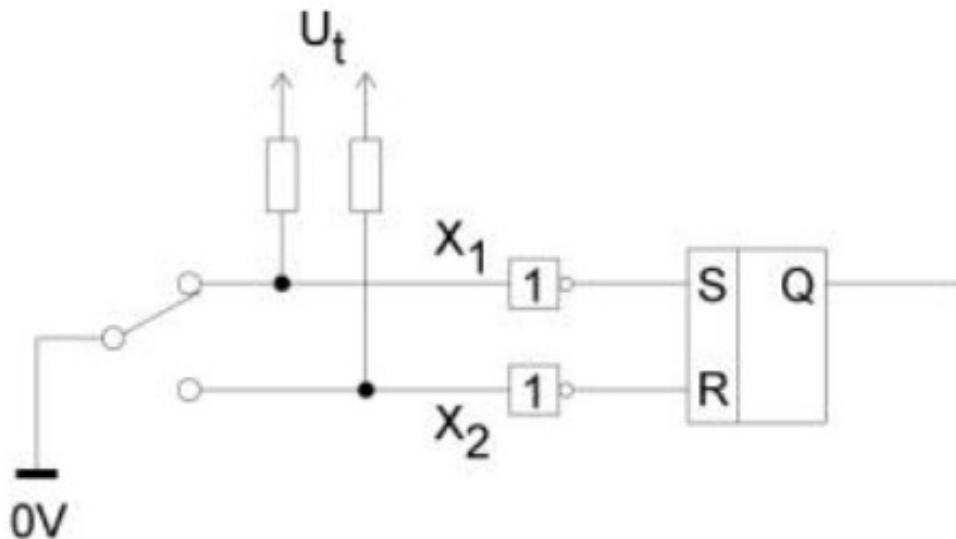
```
if (kifejezés)
    utasítás
else if (kifejezés)
    utasítás
else if (kifejezés)
    utasítás
else if (kifejezés)
    utasítás
.
.
.
else
    utasítás
```

Egy kis elektronika...

■ Nyomógomb, illetve kapcsoló hardveres pergésmentesítése

❖ Analóg módszer:
a jel integrálásával

❖ Digitális módszer:
S – R tárolóval



Időzítés – késleltetés nélkül

- Egy LED villogtatása késleltető függvényhívásokkal könnyen megvalósítható, de a program menetének megakasztása nem mindig elfogadható
- Mit tegyünk, ha több, különböző időzítést kívánó feladatot kell a programnak elvégeznie?
- A **millis()** függvényt fogjuk használni, amely a legutóbbi (újra-) indítástól számított futó idő értékét adja meg ezredmásodpercekben

```
unsigned long time;
```

```
void setup() {
```

```
  Serial.begin(115200);
```

```
}
```

```
void loop() {
```

```
  Serial.print("Time: ");
```

```
  time = millis();
```

```
  Serial.println(time); // kiírja az indítás óta eltelt időt
```

```
  delay(1000);          // tartunk egy kis szünetet, hogy ne írjunk túl sokat
```

```
}
```

Egyszerű példa a **millis()** függvény használatára
(forrás: [Arduino Reference](#))

ESP32_multitasking.ino

- Két LED-et villogtatunk, különböző frekvenciával
- A piros LED 1000 ms-onként vált állapotot, a zöld LED pedig 5000 ms-onként
- Mindig előre kiszámoljuk, és eltároljuk (redTime, greenTime változók) hogy mikor kerül sorra az adott esemény
- A ciklusban megnézzük, hogy mennyi az idő és összehasonlítjuk azt a feljegyzéseinkkel

```
#define RED_LED 23
#define GREEN_LED 22
uint32_t nowTime, redTime, greenTime;
boolean rState=LOW, gState=LOW;

void setup() {
  pinMode(RED_LED,OUTPUT);
  pinMode(GREEN_LED,OUTPUT);
  nowTime = millis();
  redTime = nowTime + 1000;
  greenTime = nowTime + 5000;
}
```

A kapcsolás ugyanaz, mint a 16. oldalon

```
void loop() {
  nowTime = millis();
  if(nowTime > redTime) {
    rState = !rState;
    digitalWrite(RED_LED,rState);
    redTime += 1000;
  }
  if(nowTime > greenTime) {
    gState = !gState;
    digitalWrite(GREEN_LED,gState);
    greenTime += 5000;
  }
}
```


Nyomógomb vezérelt LED villogtatás

- Bonyolítsuk az előző programot azzal, hogy a LED-ek villogási sebességét egy nyomógombbal lehessen szabályozni:
- Minden lenyomás felezze meg az átbillentési időket (az egyszerűség kedvéért egyformán mindkettőét)!
- Ha a piros LED átbillenési ideje már kisebb, mint 200 ms, akkor a következő gombnyomáskor állítsuk vissza a kiindulási átbillenési időket!
- Gondoljunk a nyomógomb pergésére! Egyszerű megoldásként csak 20 ms-onként vizsgáljuk meg a nyomógomb állapotát...

ESP32_gombvezereelt_ledek.ino

```
#define RED_LED    23
#define GREEN_LED 22
#define BUTTON    21
uint32_t nowTime, redTime, greenTime, buttonTime;
uINT32_t redInterval = 1000, greenInterval = 5000;
boolean rState = HIGH, gState = HIGH, bState = HIGH;
```

```
void changeSpeed() {
  if (redInterval < 200) {
    greenInterval = 5000;
    redInterval = 1000;
  }
  else {
    greenInterval /= 2;
    redInterval /= 2;
  }
}
```

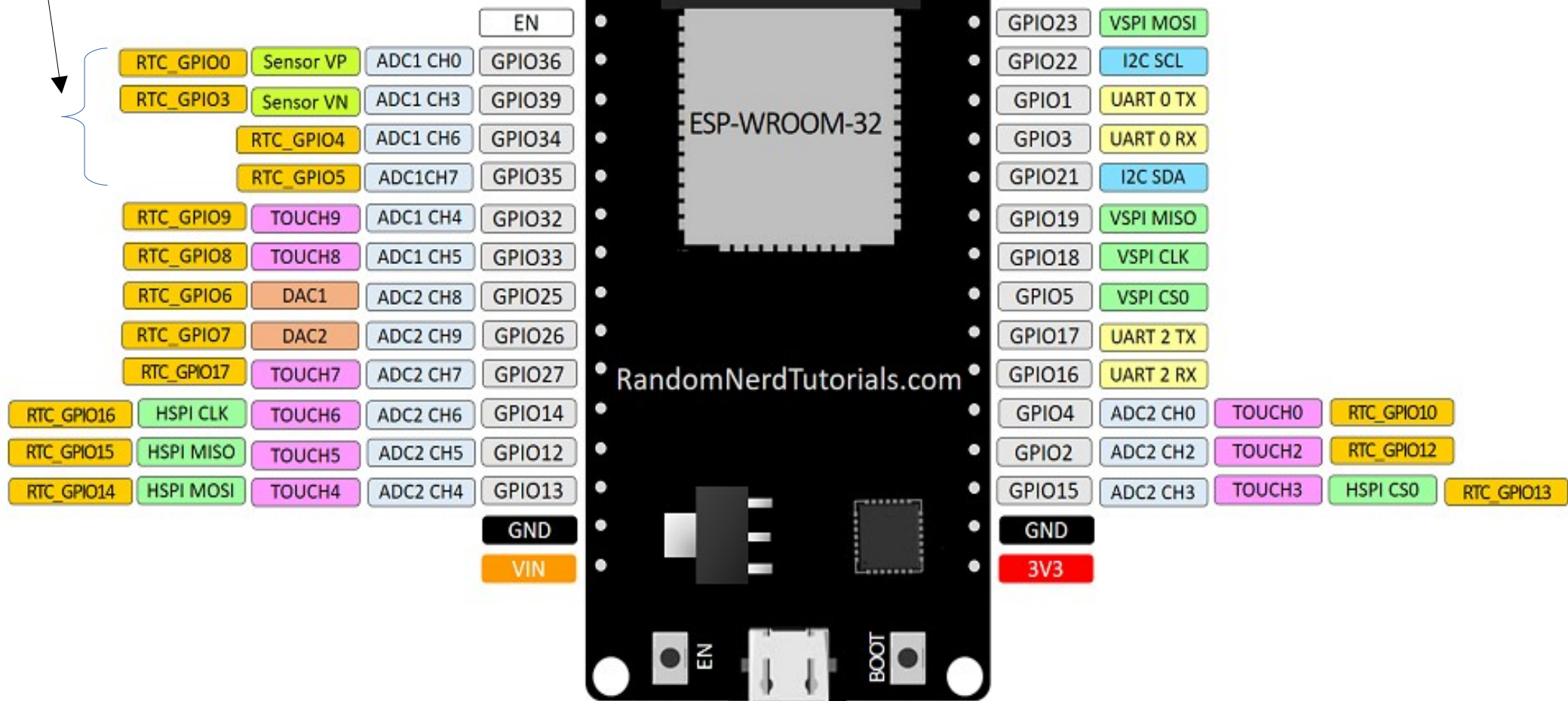
```
void setup() {
  pinMode(RED_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
  pinMode(BUTTON, INPUT_PULLUP);
  nowTime = millis();
  redTime = nowTime + redInterval;
  greenTime = nowTime + greenInterval;
  buttonTime = nowTime + 20;
}
```

```
void loop() {
  nowTime = millis();
  if (nowTime > redTime) {
    rState = !rState;
    digitalWrite(RED_LED, rState);
    redTime = nowTime + redInterval;
  }
  if (nowTime > greenTime) {
    gState = !gState;
    digitalWrite(GREEN_LED, gState);
    greenTime = nowTime + greenInterval;
  }
  if (nowTime > buttonTime) {
    buttonTime = nowTime + 20;
    if (digitalRead(BUTTON) != bState) {
      bState = !bState;
      if (bState == LOW) changeSpeed();
    }
  }
}
```

A DOIT ESP32 Devkit-1 kártya kivezetései

Csak bemenetek lehetnek!

GPIO6 - GPIO11: foglalt (SPI flash)



- Forrás: randomnerdtutorials.com/getting-started-with-esp32/

Ellenállás színkódok

