A szelenoantionint napjainkban étrend és takarmánykiegészítőként alkalmazzák.

A szelén hiánya okozta egészségügyi rendellenességek (pajzsmirigy működésben, daganatos betegségek, immunhiány) elkerülhetők a szeléntartalmú adalékanyagok fogyasztásával (napi szükséglet 20–70 μg, max. 400 μg Se). Az alapos kutatási eredmények felfedték, hogy a szelén nagyon ellenmondásosan viselkedik az élő szervezetben. **Míg kis mennyiségben, bizonyos anyagi környezetben jótékony hatású, nagyobb mennyiségek esetén mérgező, rákkeltő hatása válik meghatározóvá.** A természetes (a mi vidékeinken termesztett) élelmiszerek az egészségesen működő szervezetek számára tartalmazzák a megfelelő szeléntartalmat. Ezért a reklámozott, nagy nyomelem tartalmú készítményeket csak orvosi szakvélemény után szabad fogyasztani.

Forrásanyag:

Pais I.: A szelén és az antioxidánsok, Természet Világa 128., 9 (1997) 422
Lente G.: MKL. 2018, dec. 395

M. E.

LEGO robotok

XIX. rész

III.3.4.5. A gombok programozása

Az EV3 tégla gombjait a következő *ev3_button.h* és *ev3_button.c* modulokban (kell használni az `#include "C:\Apps\Bricx\API\ev3_button.h"`-t) lévő függvények segítségével programozhatjuk:

```
bool ButtonLedInit();
```

A függvény inicializálja a tégla gombjait. Hamis értéket térít vissza, ha a folyamat nem volt sikeres.

```
bool ButtonLedOpen();
```

A függvény a gombok állapotával inicializál a nulla értékek helyett. Ez megakadályozza a hamis gombnyomási eseményeket a program indításakor.

```
bool ButtonLedClose();
```

Lezárja a munkafolyamatot, újrainicializálja a LED-eket.

```
bool ButtonLedExit();
```

A függvény lezárja a tégla gombjait, megszakítja a kommunikációt.

```
bool ButtonLedInitialized();
```

Visszatéríti, hogy a tégla gombjai inicializálva voltak-e vagy sem.

```
float HardwareVersion();
```

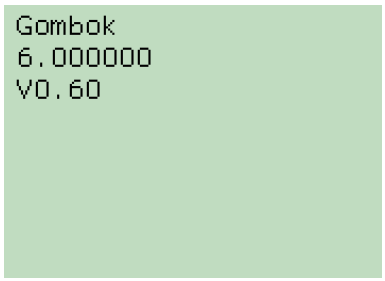
Visszatéríti a hardver verziószámát. A mi tesztesetünkben ez 6.0 volt. Részletek a 159. ábrán.

```
char* HardwareVersionString();
```

Visszatéríti a hardver verziószámát szöveges formátumban. A mi tesztesetünkben ez V0.60 volt.

A következő program a 159. ábrán látható kimenetet eredményezi:

```
1.  #include <stdio.h>
2.  #include <unistd.h>
3.  #include "C:\Apps\Bricx\API\ev3_lcd.h"
4.  #include "C:\Apps\Bricx\API\ev3_command.h"
5.  #include "C:\Apps\Bricx\API\ev3_button.h"
6.
7.  int main()
8.  {
9.    LcdInit();
10.   LcdText(1, 5, 5, "Gombok");
11.   ButtonLedInit();
12.   ButtonLedOpen();
13.   Wait(SEC_1);
14.   float hv = HardwareVersion();
15.   char v[15];
16.   sprintf(v, "%f", hv);
17.   LcdText(1, 5, 20, v);
18.   LcdText(1, 5, 35, HardwareVersionString());
19.   Wait(SEC_1);
20.   LcdClean();
21.   LcdExit();
22.   return 0;
23. }
```



159. ábra: Verziószám

```
void SetLedWarning(bool Value);
```

Bekapcsolja vagy kikapcsolja (a `TRUE` vagy `FALSE` paraméter szerint) a tégla figyelmeztető (narancssárga) LED-jeit.

```
byte LedWarning();
```

Visszatéríti, hogy a figyelmeztető (narancssárga) LED-ek be vannak-e kapcsolva (1) vagy sem (0).

```
void SetLedPattern(byte Pattern);
```

Beállítja a LED-ek mintázatát.

Az `ev3_constants.h`-ban lévő következő konstansokat tudjuk használni:

```
LED_BLACK        0
LED_GREEN        1
LED_RED          2
```

```
LED_ORANGE       3
LED_GREEN_FLASH  4
LED_RED_FLASH    5
LED_ORANGE_FLASH 6
LED_GREEN_PULSE  7
LED_RED_PULSE    8
LED_ORANGE_PULSE 9
NUM_LED_PATTERNS 10
```

Amint látjuk, a LED-eknek 10 mintázata lehetséges: nincsenek bekapcsolva, zöld, piros, narancssárga, felvillanó zöld, felvillanó piros, felvillanó narancssárga, villogó zöld, villogó piros, villogó narancssárga.

A következő program a LED-ek mindegyik mintázatát bemutatja:

```
1.  #include <stdio.h>
2.  #include <unistd.h>
3.  #include "C:\Apps\Bricx\API\ev3_lcd.h"
4.  #include "C:\Apps\Bricx\API\ev3_command.h"
5.  #include "C:\Apps\Bricx\API\ev3_button.h"
6.
7.  int main()
8.  {
9.    LcdInit();
10.   LcdText(1, 5, 5, "Gombok");
11.   ButtonLedInit();
12.   ButtonLedOpen();
13.   int i;
14.   for(i = LED_BLACK; i < NUM_LED_PATTERNS; ++i)
15.   {
16.     SetLedPattern(i);
17.     Wait(SEC_1);
18.   }
19.   SetLedPattern(LED_BLACK);
20.   LcdClean();
21.   LcdExit();
22.   return 0;
23. }
```

```
byte LedPattern();
```

Visszatéríti, hogy a LED-ek éppen milyen mintázatot mutatnak.

```
word ButtonWaitForAnyEvent(unsigned int timeout);
```

A megadott `timeout` ideig vár egy akármilyen gombesemény bekövetkezésére. Visszatéríti a lenyomott, felengedett gomb, gombok értékét.

```
word ButtonWaitForAnyPress(unsigned int timeout);
```

A megadott `timeout` ideig vár egy akármilyen gomblenyomás bekövetkezésére. Visszatéríti a lenyomott gomb, gombok értékét.

Az alábbi program az Escape gomb lenyomásáig vár gombnyomásra, majd kiírja a kijelzőre, hogy melyik gomb volt lenyomva:

```
1.  #include <stdio.h>
2.  #include <unistd.h>
```

```
3.  #include "C:\Apps\Bricx\API\ev3_lcd.h"
4.  #include "C:\Apps\Bricx\API\ev3_command.h"
5.  #include "C:\Apps\Bricx\API\ev3_button.h"
6.
7.  int main()
8.  {
9.    LcdInit();
10.   ButtonLedInit();
11.   ButtonLedOpen();
12.   char v[20];
13.   sprintf(v, "Gombok");
14.   while (true)
15.   {
16.     LcdClearDisplay();
17.     LcdText(1, 5, 5, "Gombok");
18.     byte but = ButtonWaitForAnyPress(1000);
19.     if (but == 0) sprintf(v, "None");
20.     if ((but & BUTTON_ID_ENTER) != 0)
21.         sprintf(v, "Enter");
22.     if ((but & BUTTON_ID_LEFT) != 0)
23.         sprintf(v, "Left");
24.     if ((but & BUTTON_ID_RIGHT) != 0)
25.         sprintf(v, "Right");
26.     if ((but & BUTTON_ID_UP) != 0)
27.         sprintf(v, "Up");
28.     if ((but & BUTTON_ID_DOWN) != 0)
29.         sprintf(v, "Down");
30.     if ((but & BUTTON_ID_ESCAPE) != 0)
31.         sprintf(v, "Escape");
32.     LcdText(1, 5, 15, v);
33.     Wait(SEC_1);
34.     if (but == BUTTON_ID_ESCAPE) break;
35.   }
36.   LcdClean();
37.   LcdExit();
38.   return 0;
39. }
```

```
bool ButtonIsUp(byte Button);
```

Jelzi, hogy a megadott Button gomb fel van-e engedve. A Button a következők egyike lehet: BUTTON_ID_UP, BUTTON_ID_ENTER, BUTTON_ID_DOWN, BUTTON_ID_RIGHT, BUTTON_ID_LEFT, BUTTON_ID_ESCAPE, BUTTON_ID_ALL.

```
bool ButtonIsDown(byte Button);
```

Jelzi, hogy a megadott Button gomb le van-e nyomva. A Button a következők egyike lehet: BUTTON_ID_UP, BUTTON_ID_ENTER, BUTTON_ID_DOWN, BUTTON_ID_RIGHT, BUTTON_ID_LEFT, BUTTON_ID_ESCAPE, BUTTON_ID_ALL.

```
void ButtonWaitForPress(byte Button);
```

Várakozik a megadott `Button` gomb lenyomásáig.

```
void ButtonWaitForPressAndReleae(byte Button);
```
Várakozik a megadott `Button` gomb lenyomásáig és felengedéséig.

Megjegyzés: Az eljárás neve `ButtonWaitForPressAndRelease` kellene, hogy legyen, de minden valószínűség szerint elírták.

```
bool ButtonPressedEx(byte btn, bool resetCount);
```
NXC-stílusú API-függvény. Nincs lehetőség a gombnyomások megszámlálására, rövid vagy hosszú nyomás vagy felengedés időtartam érzékelésére. A `resetCount` paraméter is figyelmen kívül marad.

A `btn` paraméterben a lekérdezendő gombot kell megadni, ha ez le volt nyomva, akkor a függvény `TRUE` értéket térít vissza.

Tulajdonképpen megegyezik a `ButtonIsDown(btn)` függvénnyel.

```
bool ButtonPressed(byte btn);
```
A fenti függvény egyszerűsített változata, csak a `btn` paramétert kell megadni.

```
char ReadButtonEx(byte btn, bool reset, bool* pressed, word* count);
```
NXC-stílusú API-függvény. Tulajdonképpen a `pressed` paraméterben visszatéríti, hogy a `btn` gomb le volt-e nyomva vagy sem.

```
byte ButtonState(byte btn);
```
Ha a `btn` gomb le van nyomva, akkor visszatérít egy `BTNSTATE_PRESSED_STATE | BTNSTATE_PRESSED_EV` értéket, különben egy `BTNSTATE_NONE` értéket.

III.3.4.5. Az időzítő programozása

Vannak olyan esetek, amikor az EV3 robotok időben kell, hogy reagáljanak a bekövetkező eseményekre, vagy előre megadott ütemezés szerint kell, hogy végezzék feladataikat (pl. azonos időközönként megmérik a fényértékeket). Ebből következően az EV3 tégla képes kell legyen az alábbiakra:

- időtartam mérése,
- idő-alapú események generálása, ami lehet egyszeri vagy ismétlődő,
- megfelelő sebességgel reagálni az előre nem meghatározható időben bekövetkező eseményekre.

Ebből következik, hogy szükségünk van olyan eszközökre és módszerekre, amelyek lehetővé teszik a hatékony időalapú tevékenység végzését. Főbb elemei ennek az eszköztárnak az időzítők vagy időszámlálók, amelyek lehetővé teszik számunkra az idő mérését, illetve a feladatok ütemezését.

Az EV3 tégla időzítőit az *ev3_timer.h* és *ev3_timer.c* modulokban (kell használni az `#include "C:\Apps\Bricx\API\ev3_timer.h"`-t) lévő típusok és függvények segítségével programozhatjuk.

Az időt centiszekundumban (CS – 100 ütem, ketyegés másodpercenként), milliszekundumban (MS – 1000 ütem, ketyegés másodpercenként) vagy mikroszekundumban (US – 1 000 000 ütem, ketyegés másodpercenként) mérhetjük.

Mindhárom időzítőből (CS, MS, US) négy-négy állhat rendelkezésünkre.

Az `ev3_constants.h`-ban lévő következő konstansokat tudjuk használni:

```
CS_TIMER_1     0          MS_TIMER_1     0          US_TIMER_1     0
CS_TIMER_2     1          MS_TIMER_2     1          US_TIMER_2     1
CS_TIMER_3     2          MS_TIMER_3     2          US_TIMER_3     2
CS_TIMER_4     3          MS_TIMER_4     3          US_TIMER_4     3
NUM_CS_TIMERS  4          NUM_MS_TIMERS  4          NUM_US_TIMERS  4
```

Az *ev3_timer.h* típusai:

```
typedef enum {
  ti10ms,
  ti50ms,
  ti100ms,
  ti250ms,
  ti500ms,
  ti1sec
} TimerInterval;
```

```
typedef void (*TimerCallback)(int sig);
```

Az *ev3_timer.h* függvényei:

```
void TimerInit();
```

Inicializálja az időzítőket.

```
void ClearTimer(byte Timer);
```

Törli a CS alapú időzítőt. A `Timer` az időzítő száma lehet 0 és 3 között vagy a `CS_TIMER_1`, `CS_TIMER_2`, `CS_TIMER_3`, `CS_TIMER_4` konstansok valamelyike.

```
void ClearTimerMS(byte Timer);
```

Törli az MS alapú időzítőt. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `MS_TIMER_1`, `MS_TIMER_2`, `MS_TIMER_3`, `MS_TIMER_4` konstansok valamelyike.

```
void ClearTimerUS(byte Timer);
```

Törli az US alapú időzítőt. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `US_TIMER_1`, `US_TIMER_2`, `US_TIMER_3`, `US_TIMER_4` konstansok valamelyike.

```
void SetTimer(byte Timer, unsigned long Value);
```

Beállítja a CS alapú időzítőt. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `CS_TIMER_1`, `CS_TIMER_2`, `CS_TIMER_3`, `CS_TIMER_4` konstansok valamelyike. A `Value` a beállítandó érték.

```
void SetTimerMS(byte Timer, unsigned long Value);
```

Beállítja az MS alapú időzítőt. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `MS_TIMER_1`, `MS_TIMER_2`, `MS_TIMER_3`, `MS_TIMER_4` konstansok valamelyike. A `Value` a beállítandó érték.

```
void SetTimerUS(byte Timer, unsigned long Value);
```

Beállítja az US alapú időzítőt. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `US_TIMER_1`, `US_TIMER_2`, `US_TIMER_3`, `US_TIMER_4` konstansok valamelyike. A `Value` a beállítandó érték.

```
unsigned long Timer(byte Timer);
```

Egy lassú, másodpercenként 10 ütemű, ketyegésű időzítőértéket térít vissza.

```
unsigned long FastTimer(byte Timer);
```

Visszatéríti a CS alapú időzítő értékét. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `CS_TIMER_1`, `CS_TIMER_2`, `CS_TIMER_3`, `CS_TIMER_4` konstansok valamelyike.

```
unsigned long TimerMS(byte Timer);
```

Visszatéríti az MS alapú időzítő értékét. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `MS_TIMER_1`, `MS_TIMER_2`, `MS_TIMER_3`, `MS_TIMER_4` konstansok valamelyike.

```
unsigned long TimerUS(byte Timer);
```

Visszatéríti az US alapú időzítő értékét. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `US_TIMER_1`, `US_TIMER_2`, `US_TIMER_3`, `US_TIMER_4` konstansok valamelyike.

A következő példaprogram az időzítők inicializálása után kiírja a lassú, gyors (CS), MS, valamint US időzítők első 11 értékét. A program eredményét a 160. ábrán láthatjuk. Megfigyelhetjük, hogy a grafikus képernyőre való írás eléggé időigényes, az időzítők értékeit késlelteti.

```
1.  #include <stdio.h>
2.  #include <unistd.h>
3.  #include "C:\Apps\Bricx\API\ev3_lcd.h"
4.  #include "C:\Apps\Bricx\API\ev3_command.h"
5.  #include "C:\Apps\Bricx\API\ev3_button.h"
6.  #include "C:\Apps\Bricx\API\ev3_timer.h"
7.
8.  int main()
9.  {
10.   LcdInit();
11.   LcdClearDisplay();
12.   TimerInit();
13.   ClearTimer(0);
14.   ClearTimerMS(0);
15.   ClearTimerUS(0);
16.   unsigned long t, ft, ms, us;
17.   int i;
18.   char s[10];
19.   for (i = 0; i <= 10; ++i)
20.   {
21.     t = Timer(0);
22.     ft = FastTimer(0);
23.     ms = TimerMS(0);
24.     us = TimerUS(0);
25.     sprintf(s, "%d", t);
26.     LcdText(1, 0, 10 * (i+1), s);
27.     sprintf(s, "%d", ft);
28.     LcdText(1, 28, 10 * (i+1), s);
29.     sprintf(s, "%d", ms);
30.     LcdText(1, 65, 10 * (i+1), s);
31.     sprintf(s, "%d", us);
32.     LcdText(1, 112, 10 * (i+1), s);
```

```
33.     Wait(SEC_1);
34.   }
35.   LcdClean();
36.   LcdExit();
37.   return 0;
38. }
```

```
0   0    0     27
10  103  1024  1024147
20  205  2047  2047145
30  307  3070  3069999
40  409  4093  4092807
51  512  5116  5115835
61  614  6139  6138348
71  716  7162  7161607
81  819  8185  8184564
92  921  9208  9207220
102 1023 10231 10230355
```

160. ábra: Időzítők

unsigned long long `TimerGetCS();`

A `gettimeofday(&tv, 0);` C függvény segítségével visszatéríti a Unix Epoch (0:00:00 January 1, 1970 GMT) óta eltelt időt centiszekundumokban (CS).

unsigned long long `TimerGetMS();`

A `gettimeofday(&tv, 0);` C függvény segítségével visszatéríti a Unix Epoch (0:00:00 January 1, 1970 GMT) óta eltelt időt milliszekundumokban (MS).

unsigned long long `TimerGetUS();`

A `gettimeofday(&tv, 0);` C függvény segítségével visszatéríti a Unix Epoch (0:00:00 January 1, 1970 GMT) óta eltelt időt mikroszekundumokban (MS).

unsigned long `TimerWait(`**unsigned long** `Time);`

Visszatéríti a `TimerGetMS()` értékét, hozzáadva a Time értéket.

void `TimerReady(`**unsigned long** `Timer);`

Vár ameddig a `Time` paraméterrel milliszekundumban megadott idő lejár, vagyis `Timer > TimerGetMS()`.

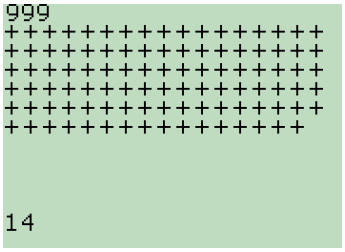
void `SetTimerCallback(TimerInterval interval, TimerCallback callback);`

A függvény segítségével az időzítők használatának másik nagy lehetőségét tudjuk leprogramozni.

Gyakran szükségünk lehet bizonyos időközönként (periodusként) megszakítani a program normális futását, elvégezni valamilyen műveletet, majd visszatérni a program normális futásához. Ezt is időzítővel tudjuk megoldani, mégpedig úgy, hogy a `SetTimerCallback` függvény segítségével megadunk egy időintervallumot (`interval` – milliszekundumokban), amely eltelte után újra és újra meghívódik a második paraméterként megadott függvény (`callback`).

A következő példaprogram 100 milliszekundomonként kirajzol egy „+” jelt a képernyőre. 10 ezer milliszekundum futás után a program eredményét a 161. ábrán láthatjuk.

```
1.  #include <stdio.h>
2.  #include <unistd.h>
3.  #include "C:\Apps\Bricx\API\ev3_lcd.h"
4.  #include "C:\Apps\Bricx\API\ev3_command.h"
5.  #include "C:\Apps\Bricx\API\ev3_button.h"
6.  #include "C:\Apps\Bricx\API\ev3_timer.h"
7.
8.  int x=0, y=10;
9.
10. void Handler(int sig)
11. {
12.   LcdText(1, x, y, "+");
13.   x+=10;
14.   if (x > 177)
15.   {
16.    X = 0;
17.    Y += 10;
18.   }
19.   char s[10];
20.   sprintf(s, "%d", sig);
21.   LcdText(1, 0, 110, s);
22. }
23.
24. int main()
25. {
26.   LcdInit();
27.   LcdClearDisplay();
28.   TimerInit();
29.   SetTimerCallback(ti100ms, &Handler);
30.   int i = 0;
31.   char s[10];
32.   while (i < 1000)
33.   {
34.        sprintf(s, "%d", i);
35.        LcdText(1, 0, 0, s);
36.        Wait(10);
37.        ++i;
38.     }
39.     LcdClean();
40.     LcdExit();
41.     return 0;
42.    }
```



161. ábra
Megadott időközönként ismétlődő esemény

Kovács Lehel István