LEGO robotok

XX. rész

III.3.4.6. A kimenet programozása

Az EV3 tégla kimeneteit, vagyis a motorokat az ev3_output.h és ev3_output.c modulokban (kell használni az #include "C:\Apps\Bricx\API\ev3_output.h"-t) lévő függvények segítségével programozhatjuk.

```
bool OutputInit(void);
```
Inicializálja a kimenetet.

```
bool OutputOpen(void);
```
Megnyitja a kimenetet.

```
bool OutputClose(void);
```
Lezárja a kimenetet.

```
bool OutputExit(void);
```
Kilép az output modulból. Lezárja az összes kimenetet.

```
bool OutputProgramStop(void);
```
Megállítja a motorokat.

```
bool OutputInitialized(void);
```
Visszatéríti, hogy a kimenet inicializálva volt-e vagy sem.

```
bool OutputStop(byte Outputs, bool useBrake);
```
Megállítja a megadott kimeneteken található motorokat (`Outputs`), be lehet állítani, hogy azonnal megálljanak (brake), vagy, hogy a motorok áramellátása kikapcsoljon, és azok addig forogjanak tehetetlenségből szabadon, amíg meg nem állnak (coast).

```
bool OutputSetType(byte Output, char DeviceType);
```
Beállítja a megfelelő kimenet (`Output`) típusát (`DeviceType`).

```
bool OutputSetTypesArray(char* pTypes);
```
Egy tömb segítségével állítja be a kimenetek (motorok) típusait.

```
bool OutputSetTypes(char OutputA, char OutputB, char OutputC, char
OutputD);
```
Egyenként beállítja a kimenetek típusait.

```
bool OutputReset(byte Outputs);
```
Újraállítja (reszetálja) a kimeneteket, vagyis nullára állítja a pozícióikat.

```
bool OutputSpeed(byte Outputs, char Speed);
```

Beállítja a megfelelő kimenetek sebességét (a polaritáshoz viszonyítva). Lehetővé teszi a szabályozást, ha a kimenet fordulatszámmérővel (tachométerrel) rendelkezik.

```
bool OutputPower(byte Outputs, char Power);
```

Beállítja a megfelelő kimenet teljesítményét, erejét. Felfüggeszti a szabályozást és a pozicionálást.

```
bool OutputStartEx(byte Outputs, byte Owner);
```

A megadott kimeneteket indítja el.

```
bool OutputStart(byte Outputs);
```

`OWNER_NONE` beállítással indítja el az `Outputs`-szal megadott kimeneteket.

```
bool OutputPolarity(byte Outputs, char Polarity);
```

Beállítja a megadott kimenetek polaritását. Ha a `Polarity` 1, a motor előre, ha –1, akkor hátra fog forogni, a 0-val pedig vált, megfordítja az irányát.

```
bool OutputRead(byte Output, char* Speed, int* TachoCount, int* TachoSensor);
```

A megadott kimenet adatait olvassa ki. A `Speed` a sebesség, a `TachoCount` tachométer fordulatainak száma, a `TachoSensor` pedig a tachoérzékelő értéke. Ezeket az értékeket az utolsó visszaállítástól (reszetálástól) számolja.

```
bool OutputTest(byte Outputs, bool* isBusy);
```

Ellenőrzi, hogy melyik kimenet foglalt.

```
bool OutputState(byte Outputs, byte* State);
```

Beállítja az összes foglalt kimenet bitmaszkját (`State`).

```
bool OutputClearCount(byte Outputs);
```

Ha a motort érzékelő üzemmódban használjuk, törli a tachoszámot, vagyis lenullázza a fordulatszámmérőt.

```
bool OutputGetCount(byte Output, int* Tacho);
```

Ha a motort érzékelő üzemmódban használjuk, visszatéríti a fordulatszámmérő értékét (`Tacho`).

```
bool OutputGetTachoCount(byte Output, int* Tacho);
```

Ha a motort érzékelő üzemmódban használjuk, visszatéríti a fordulatszámmérő értékét az utolsó visszaállítás óta (`Tacho`).

```
bool OutputGetActualSpeed(byte Output, char* Speed);
```

Visszatéríti a motor aktuális sebességét (`Speed`).

```
bool OutputStepPowerEx(byte Outputs, char Power, int Step1, int Step2, int Step3, bool useBrake, byte Owner);
```

Beállítja a kimenetek teljesítményét (`Power`), a rámpalépéseket (fel, állandó, le), valamint a megállási módot (`useBrake`).

```
bool OutputStepPower(byte Outputs, char Power, int Step1, int Step2,
int Step3);
```
Az előbbi függvényt hívja alapértelmezett `useBrake` `TRUE` és `Owner` `OWNER_NONE` beállításokkal.

```
bool OutputTimePowerEx(byte Outputs, char Power, int Time1, int Time2,
int Time3, bool useBrake, byte Owner);
```
Az `OutputStepPowerEx` függvényhez hasonló beállító függvény, csak nem a lépéseket, hanem az időt állítja be.

```
bool OutputTimePower(byte Outputs, char Power, int Time1, int Time2,
int Time3);
```
Az előbbi függvényt hívja alapértelmezett `useBrake` `TRUE` és `Owner` `OWNER_NONE` beállításokkal.

```
bool OutputStepSpeedEx(byte Outputs, char Speed, int Step1, int Step2,
int Step3, bool useBrake, byte Owner);
```
Az `OutputStepPowerEx` függvényhez hasonló beállító függvény, csak nem a teljesítményt, hanem a sebességet állítja be.

```
bool OutputStepSpeed(byte Outputs, char Speed, int Step1, int Step2,
int Step3)
```
Az előbbi függvényt hívja alapértelmezett `useBrake` `TRUE` és `Owner` `OWNER_NONE` beállításokkal.

```
bool OutputTimeSpeedEx(byte Outputs, char Speed, int Time1, int Time2,
int Time3, bool useBrake, byte Owner);
```
Az `OutputStepSpeedEx` függvényhez hasonló beállító függvény, csak nem a lépéseket, hanem az időt állítja be.

```
bool OutputTimeSpeed(byte Outputs, char Speed, int Time1, int Time2,
int Time);
```
Az előbbi függvényt hívja alapértelmezett `useBrake` `TRUE` és `Owner` `OWNER_NONE` beállításokkal.

```
bool OutputStepSyncEx(byte Outputs, char Speed, short Turn, int Step,
bool useBrake, byte Owner);
```
Kormányzási módot valósít meg két motor segítségével. Az `Outputs` paraméter tehát csak `OUT_AB`, `OUT_AC`, `OUT_AD`, `OUT_BC`, `OUT_BD`, vagy `OUT_CD` lehet. Beállítja a sebességet (`Speed`), a fordulás szögét (`Turn`), a lépést (`Step`), a megállási módot (`useBrake`), valamint a tulajdonost (`Owner`).

```
bool OutputStepSync(byte Outputs, char Speed, short Turn, int Step);
```
Az előbbi függvényt hívja alapértelmezett `useBrake` `TRUE` és `Owner` `OWNER_NONE` beállításokkal.

```
bool OutputTimeSyncEx(byte Outputs, char Speed, short Turn, int Time,
bool useBrake, byte Owner);
```

Kormányzási módot valósít meg két motor segítségével, tehát a OutputStepSync függvényhez hasonló beállító függvény, ám nem a lépést, hanem a motor működési idejét (Time) állítja be.

```
bool OutputTimeSync(byte Outputs, char Speed, short Turn, int Time);
```

Az előbbi függvényt hívja alapértelmezett `useBrake` `TRUE` és `Owner` `OWNER_NONE` beállításokkal.

```
void SetOutputEx(byte Outputs, byte Mode, byte reset);
```

Beállítja a kimenet módját (`Mode`). Ez a mód `OUT_FLOAT`, `OUT_OFF` vagy `OUT_ON` lehet. Az `OUT_ON` bekapcsolja a motort, az `OUT_OFF` kikapcsolja, az `OUT_FLOAT` pedig kikapcsolja az áramellátást, de hagyja a motort szabadon megállni. A `reset` segítségével a fordulatszámmérőt állíthatjuk vissza, reszetálhatjuk.

```
void SetOutput(byte Outputs, byte Mode);
```

Az előbbi függvényt hívja alapértelmezett `reset` `RESET_NONE` beállítással.

```
void SetDirection(byte Outputs, byte Dir);
```

Beállítja a motor irányát. Ha a `Dir` 1 (`OUT_FWD`), a motor előre, ha –1 (`OUT_REV`), akkor hátra fog forogni, a 0-val (`OUT_TOGGLE`) pedig vált, megfordítja az irányát.

```
void SetPower(byte Outputs, char Power);
```

Beállítja a motor teljesítményét.

```
void SetSpeed(byte Outputs, char Speed);
```

Beállítja a motor sebességét.

```
void OnEx(byte Outputs, byte reset);
```

Bekapcsolja a motort.

```
void OffEx(byte Outputs, byte reset);
```

Kikapcsolja a motort.

```
void FloatEx(byte Outputs, byte reset);
```

Kikapcsolja a motor áramellátását, és hagyja magától megállni.

```
void On(byte Outputs);
```

Az `OnEx` függvényt hívja alapértelmezett `RESET_NONE` paraméterrel.

```
void Off(byte Outputs);
```

Az `OffEx` függvényt hívja alapértelmezett `RESET_NONE` paraméterrel.

```
void Float(byte Outputs);
```

A `FloatEx` függvényt hívja alapértelmezett `RESET_NONE` paraméterrel.

```
void Coast(byte Outputs);
```

A `FloatEx` függvényt hívja alapértelmezett `RESET_NONE` paraméterrel.

```
void Toggle(byte Outputs);
```
Megváltoztatja a motor forgási irányát.

```
void Fwd(byte Outputs);
```
Beállítja a motor előreforgását.

```
void Rev(byte Outputs);
```
Beállítja a motor visszaforgását.

```
void OnFwdEx(byte Outputs, char Power, byte reset);
```
Beállítja a motor előreforgását, teljesítményét, valamint a fordulatszámmérőjét. Bekapcsolja a motort.

```
void OnRevEx(byte Outputs, char Power, byte reset);
```
Beállítja a motor visszaforgását, teljesítményét, valamint a fordulatszámmérőjét. Bekapcsolja a motort.

```
void OnFwd(byte Outputs);
```
Bekapcsolja a motort előre irányba, `OUT_POWER_DEFAULT`, valamint `RESET_NONE` alapértelmezett paraméterekkel.

```
void OnRev(byte Outputs);
```
Bekapcsolja a motort vissza irányba, `OUT_POWER_DEFAULT`, valamint `RESET_NONE` alapértelmezett paraméterekkel.

```
void OnFwdRegEx(byte Outputs, char Speed, byte RegMode, byte reset);
```
Bekapcsolja a motort előre irányba, beállítja a sebességét és a fordulatszámmérőt. Ebben a fejlesztésben a `RegMode` paraméter nem játszik szerepet.

```
void OnRevRegEx(byte Outputs, char Speed, byte RegMode, byte reset);
```
Bekapcsolja a motort vissza irányba, beállítja a sebességét és a fordulatszámmérőt. Ebben a fejlesztésben a `RegMode` paraméter nem játszik szerepet.

```
void OnFwdReg(byte Outputs, char Speed);
```
Az `OnFwdRegEx` függvényt hívja `OUT_REGMODE_SPEED` és `RESET_NONE` alapértelmezett paraméterekkel.

```
void OnRevReg(byte Outputs, char Speed);
```
Az `OnRevRegEx` függvényt hívja `OUT_REGMODE_SPEED` és `RESET_NONE` alapértelmezett paraméterekkel.

```
void OnFwdSyncEx(byte Outputs, char Speed, short Turn, byte reset);
```
Kormányzási módot valósít meg két motor segítségével előre irányba, beállítja a sebességet, a fordulatot, valamint a fordulatszámmérőt.

```
void OnRevSyncEx(byte Outputs, char Speed, short Turn, byte reset);
```
Kormányzási módot valósít meg két motor segítségével vissza irányba, beállítja a sebességet, a fordulatot, valamint a fordulatszámmérőt.

```
void OnFwdSync(byte Outputs, char Speed);
```
Az `OnFwdSyncEx` függvényt hívja 0 fordulási szöggel és `RESET_NONE` alapértelmezett paraméterrel.

```
void OnRevSync(byte Outputs, char Speed);
```
Az `OnRevSyncEx` függvényt hívja 0 fordulási szöggel és `RESET_NONE` alapértelmezett paraméterrel.

```
void RotateMotorNoWaitEx(byte Outputs, char Speed, int Angle, short Turn, bool Sync, bool Stop);
```
Bekapcsolja a motort párhuzamos módon, vagyis a következő utasítás végrehajtása is elindul. Megadhatjuk a sebességet (`Speed`), a fordulat szögét (`Angle`), a fordulást (`Turn`), ha a `Sync true`, akkor kormányzási módba kerülünk, ha nem, akkor csak egy motort indít, megadhatjuk a megállás módját (`Stop`).

```
void RotateMotorNoWait(byte Outputs, char Speed, int Angle);
```
Az előbbi függvényt hívja `Turn = 0`, `Sync = true`, `Stop = true` alapértelmezett értékekkel.

```
void RotateMotorEx(byte Outputs, char Speed, int Angle, short Turn, bool Sync, bool Stop);
```
A `RotateMotorNoWaitEx` függvényhez hasonló függvény, ám míg a motor működik, más utasítás nem hajtható végre.

```
void RotateMotor(byte Outputs, char Speed, int Angle);
```
Az előbbi függvényt hívja `Turn = 0`, `Sync = true`, `Stop = true` alapértelmezett értékekkel.

```
void OnForSyncEx(byte Outputs, int Time, char Speed, short Turn, bool Stop);
```
Az `OutputTimeSyncEx` függvényt hívja.

```
void OnForSync(byte Outputs, int Time, char Speed);
```
Az előbbi függvényt hívja `Turn = 0`, `Stop = true` alapértelmezett értékekkel.

```
void OnForEx(byte Outputs, int Time, char Power, byte reset);
```
A megadott ideig (`Time`) működteti a motort a megadott teljesítménnyel (`Power`). Beállíthatjuk a fordulatszámmérőt is.

```
void OnFor(byte Outputs, int Time);
```
Az előbbi függvényt hívja `Power = OUT_POWER_DEFAULT`, `reset = RESET_NONE` alapértelmezett értékekkel.

```
void ResetTachoCount(byte Outputs);
```
Reszetálja (visszaállítja) a fordulatszámmérőt vagy mérőket.

```
void ResetAllTachoCounts(byte Outputs);
```
Reszetálja (visszaállítja) az összes fordulatszámmérőt.

```
void ResetBlockTachoCount(byte Outputs);
```
A `ResetTachoCount` függvény szinonimája.

```
void ResetRotationCount(byte Outputs);
```
Reszetálja (visszaállítja) a fordulatszámmérőt vagy mérőket.

```
void ResetCount(byte Outputs, byte reset);
```
Reszetálja (visszaállítja) a fordulatszámmérőt vagy mérőket. A reset paraméter `RESET_COUNT`, `RESET_BLOCK_COUNT`, `RESET_ROTATION_COUNT`, `RESET_BLOCKANDTACHO` vagy `RESET_ALL` lehet, s ennek függvényében hívja az előbbi függvények valamelyikét.

```
int MotorTachoCount(byte Output);
```
Visszatéríti a megadott motor fordulatszámmérőjének értékét.

```
int MotorBlockTachoCount(byte Output);
```
Az előbbi függvény szinonimája.

```
char MotorPower(byte Output);
```
Visszatéríti a motor aktuális sebességét.

```
char MotorActualSpeed(byte Output);
```
Az előbbi függvény szinonimája.

```
int MotorRotationCount(byte Output);
```
Visszatéríti a motor fordulatszámát.

```
bool MotorBusy(byte Output);
```
Visszatéríti, hogy a motor foglalt-e vagy sem.

A következő program segítségével a motorokat teszteljük.

```
1.  #include "c:\APPS\Bricx\API\ev3_command.h"
2.  #include "c:\APPS\Bricx\API\ev3_output.h"
3.
4.  int main()
5.  {
6.    OutputInit();
7.    ResetAllTachoCounts(OUT_ABCD);
8.    SetPower(OUT_A, 90);
9.    SetSpeed(OUT_B, 40);
10.   SetPower(OUT_C, 60);
11.   SetPower(OUT_D, -60);
12.   On(OUT_ALL);
13.   Wait(SEC_5);
14.   Float(OUT_ALL);
15.   OutputClose();
16.   OutputExit();
17.   return 0;
18. }
```

Sajnos John Hansen fejlesztései itt leálltak, az érzékelőket nem programozta le, ám saját fejlesztésű programjainkban jól tudjuk ezeket a modulokat használni, a többit pedig leprogramozzuk mi.

9. Feladat

Tervezzük meg egy olyan robot programját, amelyik meg tudja oldani rekurzívan a Hanoi tornyai feladatot!

Mivel most imperatív nyelven programozunk, nyugodtan használhatunk rekurzív hívást is. A program így a következő lesz:

```
1.  #include <stdio.h>
2.  #include <unistd.h>
3.  #include "C:\Apps\Bricx\API\ev3_timer.h"
4.  #include "C:\Apps\Bricx\API\ev3_lcd.h"
5.  #include "C:\Apps\Bricx\API\ev3_command.h"
6.  #include "C:\Apps\Bricx\API\ev3_output.h"
7.
8.  char mes[6];
9.
10. void hanoi(int k, char s, char d, char h)
11. {
12.   if (k == 1)
13.   {
14.     sprintf(mes, "%c -> %c\n", s, d);
15.     LcdText(1, 0, 0, mes);
16.     LcdRefresh();
17.     Wait(SEC_1);
18.   }
19.   else
20.   {
21.     hanoi(k-1, s, h, d);
22.     sprintf(mes, "%c -> %c\n", s, d);
23.     LcdText(1, 0, 0, mes);
24.     LcdRefresh();
25.     Wait(SEC_1);
26.     hanoi(k-1, h, d, s);
27.   }
28. }
29.
30. int main()
31. {
32.   LcdInit();
33.   LcdOpen();
34.   int n = 3;
35.   hanoi(n, '0', '1', '2');
36.   LcdExit();
37.   LcdClose();
38.   return 0;
39. }
```

Kovács Lehel István