Felhasznált és ajánlott forrásanyag:

[1], [2] V. F. Weisskopf: Embertelen-e a fizika ? (Természet Világa 2006/1 különszám 93-95, 116 old.
[3] Gyertyán Ervin: József Attila alkotásai és vallomásai, Szépirodalmi Kk.,Bp.1966
[4] Toró Tibotr: József Attila transznegatívum-elmélete és az anyag-antianyag-szimmetria (sértés) (Természet Világa 2006/1 különszám, 88-92), Tuska Ágnes: „működésben van a nyugalom" J. A. viszonya a fizikához, Fizikai Szemle 1980.11., Trevoda György: József Attila költeszetének kozmológiai vonatkozásai, Irodalom-történeti Közlemények, 1979.
[5] Marx György:A modern fizika forradalma és József Attila , Fizikai Szemle 2012/5
[6] Csoóri Sándor Nomád napló, Magvető Kk., 1978
[7] Mezei Judit: A tudomány és a művészet összecsengése J.A. költészetében (Ponticulus Hungaricus IX.évf.3.sz. 2005. március

Idézetek költeményekből: http://jozsefattila.elte.hu/v1/vs193702.htm#45
Prózai idézetek a szövegben pontatlan utalással: hu.wikipedia.org/wiki/József_Attila, Ponticus, IV.évf. 6 7-8.szám

Máthé Enikő

LEGO robotok

XV. rész

8. Feladat

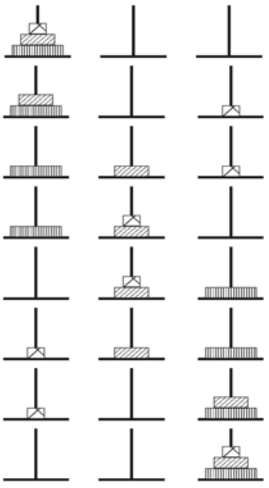
Tervezzük meg egy olyan robot programját, amelyik meg tudja oldani a Hanoi tornyai feladatot!

Hanoi tornyainak legendája

Sok ezer évvel ezelőtt Indiában, Fo Hi uralkodása alatt, a benaresi Siva-templom közepén volt egy márványtábla, amelyből három gyémánttű állt ki. Az első tűn 64 vékony aranykorong helyezkedett el, alul a legnagyobb átmérőjű, majd rajta egyre kisebbek. Egyszer Siva isten megparancsolta a templom papjainak, hogy rakják át a korongokat az első tűről a másodikra. Két fontos szabályt azonban be kellett tartaniuk:

- A korongok igen sérülékenyek, ezért egyszerre csak egyet lehet mozgatni,
- valamint nem kerülhet a szent korongokból magasabb értékű alacsonyabb értékű fölé.

A szerzetesek természetesen használhatták a harmadik tűt is a rakodás közben. Amikor az utolsó korong a helyére kerül, a templom porrá omlik össze, és a világ véget ér.



130. ábra: Hanoi tornyai

A Hanoi tornyai játék leírását először egy bizonyos N.

Claus de Siam, a Li-Sou-Stian egyetem oktatója publikálta egy párizsi újságban [1.]. Később kiderült, hogy az 1883-ban megjelent cikk szerzője valójában Edouard Lucas francia matematikus, a Lyceé Saint-Louis tanára (Az álnév a Lucas d'Amiens név betűiből született). A játékot Lucas Hanoi tornyának keresztelte el.

A megoldás

Mi a játék megoldása? Tételezzük fel, hogy a papok a lehető leghatékonyabban, hiba nélkül dolgoznak. A kérdés tehát a következő: legalább hány lépésben lehet mind a 64 korongot átjuttatni az első tűről a másodikra?

Érdemes a problémát először kevesebb korongra megvizsgálni, hátha tapasztalunk valami összefüggést a korongok és a lépések száma között (egy lépés alatt természetesen egy korong áthelyezését értjük). Magától értetődően egy korong esetén egy, és könnyen végiggondolható, hogy kettő esetén három lépésre van szükségünk. Három korongra már nem ennyire egyszerű a kép, de némi próbálkozás árán megtalálhatjuk azt a hét áthelyezést, amely a leggyorsabb megoldást adja. A korongok számát K-val, a lépésekét L-lel jelölve megsejthetjük a következő összefüggést: $L = 2^K–1$.

A bizonyítás egy ügyes trükkre épül. Vegyük észre, hogy K értékétől függetlenül biztos lesz egy olyan lépés, amikor a legnagyobb korong átkerül az elsőről a második tűre. Ekkor nyilván az összes többi a harmadik tűn van, mégpedig a szabályokból következően nagyság szerinti sorrendben. A feladat tehát a következőképpen módosul: helyezzük át a harmadik tűről a másodikra a korongokat úgy, hogy az elsőt is felhasználhatjuk. Egy újabb Hanoi-tornyot kaptunk, immár K–1 koronggal. Ezek szerint még annyi lépésre van szükségünk, mintha eggyel kevesebb koronggal kezdtük volna a játékot.

A fentiek segítségével felírhatjuk a következő képletet: $L = L' + 1 + L'$, ahol L' a lépések száma K–1 korong esetén. Az összeadás három tagja a megoldás három lépését jelöli:

1. K–1 korong az első tűről a harmadikra,
2. legnagyobb korong a helyére,
3. a K–1 korong vissza a másodikra.

Legyen S_K–1 a K darab koronghoz tartozó lépések száma, azaz $L = S_K$–1, illetve $L' = S_{K-1}$–1. A fenti képletet átalakítva: $L = 2 \cdot L' + 1$, azaz $S_K – 1 = 2 \cdot (S_{K-1}–1) + 1$, amiből a zárójel felbontása és az egyenlet rendezése után $S_K = 2 \cdot S_K$–1. A feladat elején már megállapítottuk, hogy S_1–1 = 1, azaz $S_1 = 2$. Mivel a fentiek szerint minden következő S kétszerese az előzőnek, ezért kimondhatjuk, hogy $S_K = 2^K$, ezzel pedig beláttuk a kiinduló állítást, hiszen $L = S_K$–1 = 2^K–1.

A szerzeteseknek tehát 2^{64}–1 (= 18 446 744 073 709 551 615) áthelyezést kell végrehajtaniuk. Ha feltesszük, hogy egy korongot átlagosan egy másodperc alatt tesznek át, munkájuk akkor is több mint 590 000 000 000 évig tartana (összehasonlításképpen, becslés szerint, a Világegyetem 13,7 milliárd éves)!

Kétségtelen, hogy a kellő ügyességgel megépíthető egy olyan LEGO robot, amely át tudja tenni a korongokat az egyik rúdról a másikra, azonban a robot vezérlésére szükség van a háttéralgoritmusra. Ezt fogjuk a következőkben megvizsgálni.

A Hanoi tornyai feladat a Divide et impera („oszd meg és uralkodj”) technika segítségével oldható meg klasszikus módon.

A probléma itt csupán az, hogy a LEGO MINDSTORMS EV3 Home Edition nem ismeri a rekurziót, tehát más megoldást kell keresnünk.

A kérdés tehát az, hogy létezik-e iteratív algoritmus az optimális lépéssorozat kigenerálásához?

Igen létezik, és az alábbi észrevételeken alapszik (nevezzük el a három tornyot „kicsi”-nek, „közepes”-nek és „nagy”-nak attól függően, hogy miként viszonyulnak egymáshoz méretük szerint a legfelső korongok):

- A három szóba jöhető lépés: kicsi → nagy, kicsi → közepes, közepes → nagy.
- Ha utoljára a kicsi korong lépett, akkor nem léphet újra az, mert vagy visszakerülne oda ahonnan jött (hurok), vagy ahova lépne oda direktbe (egy lépésből) is léphetett volna (ami nyilván „optimálisabb” lett volna).
- Tehát a kicsi és közepes korongok felváltva lépnek.
- Ha a közepes korong van soron, akkor egyértelmű, hogy „közepes → nagy” lépést kell tenni.
- Bizonyítható továbbá, hogy attól függően, hogy az *n* páratlan vagy páros a kicsi korong vagy az (*a*, *b*, *c*, *a*, *b*, *c*, ...) vagy az (*a*, *c*, *b*, *a*, *c*, *b*, ...) lépésmintát kell kövesse. Tehát ez esetben is egyértelműen meghatározható melyik a következő lépés.

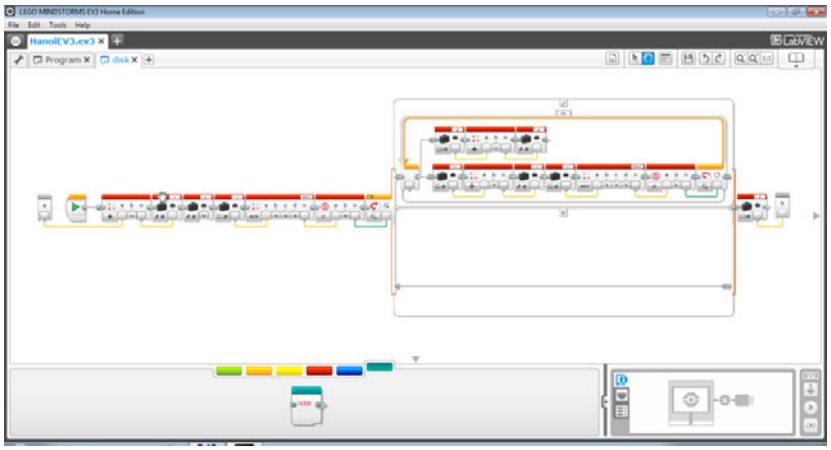
```
int disk(int i)
{
  int g=0, x = i+1;
  while(x%2==0)
  {
    ++g;
    x/=2;
  }
  return g;
}

int main()
{
  int n=3;
  int limit = pow(2, n) - 1;
  for (int i = 0; i < limit; ++i)
  {
    int d = disk(i);
    int source = (((i/(int)pow(2,d)+1)/2)*(2-(n+d)%2))%3;
    int dest = (source + (2 - (n + d)%2))%3;
    printf("disk %d: %c->%c\n", d, source+'a', dest+'a');
  }
  return 0;
}
```

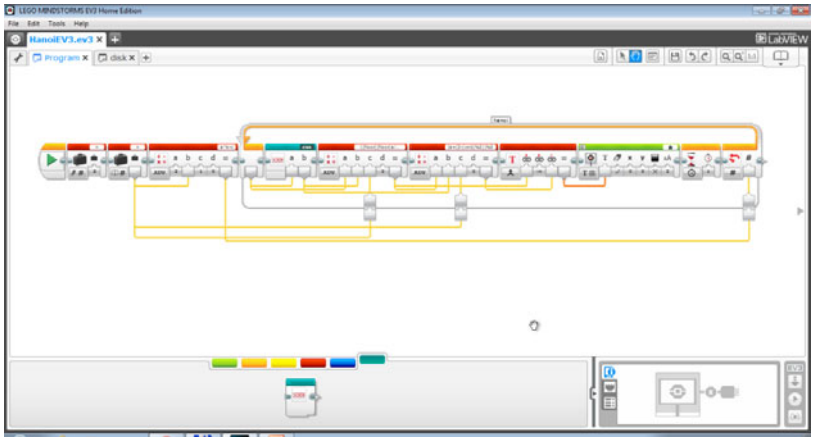
131. ábra: *Hanoi tornyai C nyelven*

A fentieket figyelembe véve a játék megoldását a 131. ábrán látható C nyelven írt programban foglalhatjuk össze.

Ha vizuális környezetbe szeretnénk átültetni, hogy robotunk is „értse”, először tervezzük meg a 132. ábrán látható disk nevű saját blokkot, majd rakjuk össze a 133. ábrán látható programot.



132. ábra: *A disk saját blokk*



133. ábra: *A Hanoi tornyai program*

III.2. Programozás a téglán

Az I.5. fejezetben már bemutattuk a tégla lehetőségeit, itt most a gyárilag telepített *Tégla program* pontot fogjuk részletezni.

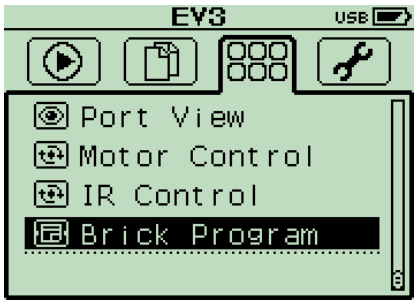
Az EV3 tégla programozási alkalmazása hasonló a számítógépünkre telepített LEGO MINDSTORMS EV3 Home Edition környezethez. Az itt található blokkok megadják nekünk a kezdéshez szükséges alapismereteket. Természetesen bonyolult programokat csakis a számítógépen tudunk megtervezni, de a tégla programok lehetőséget nyújtanak arra, hogy a robotunk alapszinten működhessen számítógép nélkül.

A téglán nem lehet egymásba ágyazott ciklusokat, bonyolult elágazásokat, több szálon futó programokat tervezni, nem lehet adatdrótokkal összekötni a blokkokat, nem lehet változókat, konstansokat használni vagy tömböket programozni, hanem csak nagyon kezdetleges érzékelő és motorműveleteket végrehajtani.

A portok kiosztása is alapértelmezett módon történik, ezen változtatni nem tudunk:

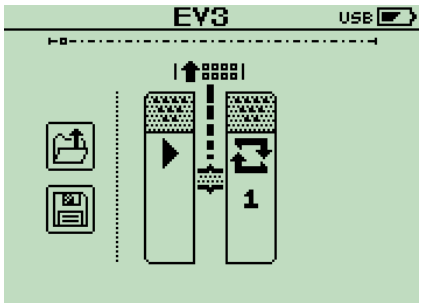
- 1-es port: érintésérzékelő
- 2-es port: giroszkópikus érzékelő
- 3-as port: színérzékelő
- 4-es port: infravörös érzékelő
- A port: közepes motor
- B port és C port: két nagy motor
- D port: egy nagy motor

A 134. ábrán látható módon keressük meg a téglagombok segítségével, és nyissuk meg a Brick Program (Tégla program) alkalmazást.



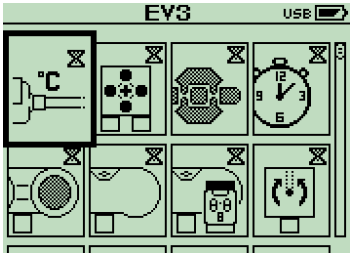
134. ábra: *A Tégla program alkalmazás*

Ha elindítottuk az alkalmazást, megjelenik a 135. ábrán látható környezet, amely nagyon egyszerű módon igyekszik reprodukálni a LEGO MINDSTORMS EV3 Home Edition blokkjait.



135. ábra: *A környezet*

Megfigyelhető a Start blokk és a Hurok blokk – amely segítségével a programot tudjuk megismételni 1, 2, …, 10 alkalommal vagy végtelenszer – egy sorrendi huzallal összekötve. Közöttük megjelenik a függőleges, megtört Blokk hozzáadás vonal. Ez jelzi, hogy további blokkokat adhatunk hozzá a programunkhoz. A tégla Fel gombjának megnyomásával tudjuk ezt megtenni, ekkor megjelenik a 136. ábrán látható blokkpaletta.



136. ábra: *A blokkpaletta*

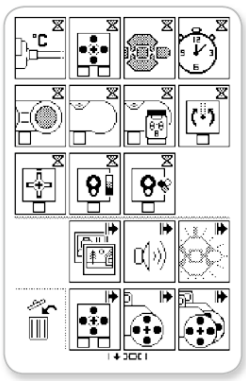
A blokkpalettán a Le, Fel, Jobbra, Balra téglagombok segítségével navigálhatunk, itt található az aktuális blokkot kitörlő kuka gomb is.

Ha végignavigálunk az egész palettán, akkor visszajutunk a programhoz.

Általában véve kétféle blokk van: a Cselekvő (Action) és a Váró (Wait). A cselekvő blokkot egy kis nyíl, a váró blokkot egy homokóra jelzi a blokk jobb felső sarkában. Összesen hat különböző cselekvő és tizenegy különböző váró blokk közül választhatunk.

A teljes blokkpaletta a 137. ábrán látható és a következő blokkokat foglalja magában (balról jobbra és fentről le):

- Váró blokkok
  - Hőmérsékletérzékelő
  - Motorérzékelő
  - Téglagombok
  - Várakozás egy adott ideig
  - Ultrahangos érzékelő
  - Infravörös érzékelő
  - Távirányító
  - Motorforgás érzékelő
  - Érintésérzékelő
  - Fényérzékelő
  - Színérzékelő
- Cselekvő blokkok
  - Kijelző
  - Hang
  - Fények a gombok körül
  - Közepes motor
  - Nagy motor
  - Két nagy motor (tank vagy kormányozás)
- Kuka



137. ábra: *A teljes blokkpaletta*

Ha megtaláltuk azt a blokkot amelyikre szükségünk van, navigáljunk rá, és nyomjuk meg a tégla középső gombját, így visszakerülünk a programunkhoz, és a kiválasztott blokk beszúródik az aktuális helyre.

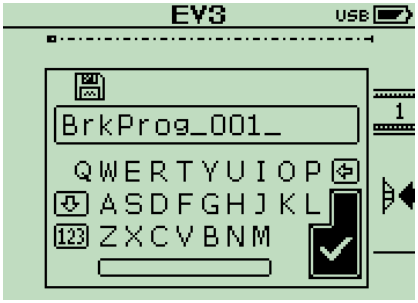
A programunkban a blokkok között a tégla bal és jobb gombjával navigálhatunk. A középső gomb megnyomásával módosíthatjuk a kijelölt blokk beállításait (mindig a képernyő közepén látható blokk) vagy új blokkot vehetünk fel, ha a sorrend vonal van kijelölve és a blokk hozzáadás vonal látható.

A kijelző és a hang blokkok esetében az alapértelmezett képek és hangok állíthatók be. Ezeknek a listáját a IV. fejezetben találhatjuk meg.

A program futtatáshoz a tégla bal gombjával navigáljunk a program legelején lévő Start blokkra. Nyomjuk meg a középső gombot, és a programunk elindul.

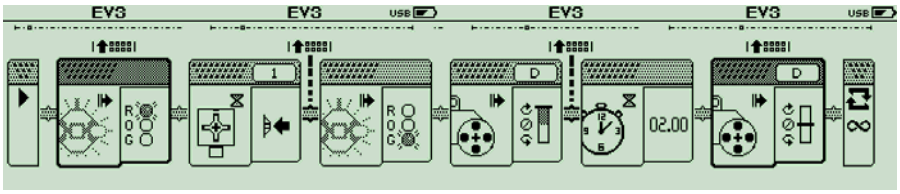
A Start blokk előtt lévő Megnyitás (Open) ikonra kattintva megnyithatjuk a lementett EV3 tégla programjainkat.

A Megnyitás alatti Mentés (Save) ikonra kattintva elmenthetjük a programot. A programnak nevet kell adni a 138. ábrán látható módon, majd az OK-ra kattintva tudjuk elmenteni a programot a *BrkProg_SAVE* mappába, amely az Állomány navigáció képernyőn érhető el.



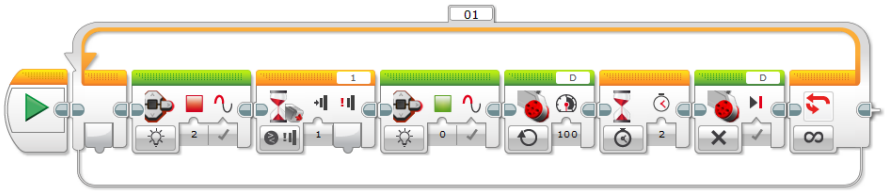
138. ábra: *Névadás és mentés*

A 139. ábrán látható példaprogram bekapcsolja a gombok piros fényét, vár, ameddig meg nem nyomjuk az érintésérzékelőt, ekkor bekapcsolja a zöld fényeket, beindítja a motort 2 másodperc erejéig, majd lezárja ezt. A beállított ciklusnak köszönhetően ezt a programot végtelenszer ismétli az EV3 tégla.



139. ábra: *Példaprogram – programozás a téglán*

Ha a 139. ábrán látható példaprogramot beimportáljuk a LEGO MINDSTORMS EV3 Home Edition *Tools* menüjének *Import Brick Program* parancsával, akkor a 140. ábrán látható programot kapjuk.



140. ábra: *Példaprogram – importálva*

III.3. Programozás imperatív nyelvekben

A Neumann-elvek felhasználásával megépített számítógépek programozása – az alacsony vagy magas szintű programozási nyelvek által – szorosan összefügg az imperatív (*imperative*: parancsoló, utasító) paradigmával.

Az imperatív paradigma tulajdonságai:

- *Algoritmikusság* – a programozó algoritmust kódol (forráskódot, programszöveget ír le), és ez az algoritmus működteti a processzort.
- *Utasítások használata* – a program utasítások sorozatából áll.
- *Változók használata* – legfőbb programozói eszköz a változó, amely a tár közvetlen elérését biztosítja, lehetőséget nyújt a tárban lévő érték közvetlen megváltoztatására. Az algoritmusok, utasítások változókat használnak, a változók értékeit módosítják, tehát a program a hatását közvetlenül a tárban lévő értékekre fejti ki.
- *Ciklikusság* – lehetséges az utasítások ismételt végrehajtása.
- *Elágazó programszerkezet* – létezik GOTO utasítás, a program végrehajtása több ág valamelyikén futhat.
- *Tükrözés* – a beolvasás és kiírás a memória direkt másolásával történik meg.

Az imperatív programozási nyelvek főbb elemei: a változók, konstansok, típusok, kifejezések, utasítások, vezérlési szerkezetek, programegységek.

Az EV3-tégla számos imperatív nyelvben programozható, például:

- ROBOTC for MINDSTORMS EV3 (nem ingyenes)
- leJOS JAVA for EV3 (firmware csere)
- BRICXCC: C++ & C (ingyenes)
- EV3DEV (Debian, firmware csere): pár nyelv alája
- MONOBRICK: C# & .NET (firmware csere)
- PYTHON FOR EV3

Itt az ingyenes *Bricx CC* (*Bricx Command Center*) környezetet mutatjuk be, amely egy C-hez hasonló imperatív nyelvet tartalmaz. Nyilvánvaló, hogy a fent említett elemek mindegyike a C nyelvből öröklődött, a *Bricx CC* újdonsága csupán az, hogy új függvényeket tartalmaz az EV3-tégla programozásához.

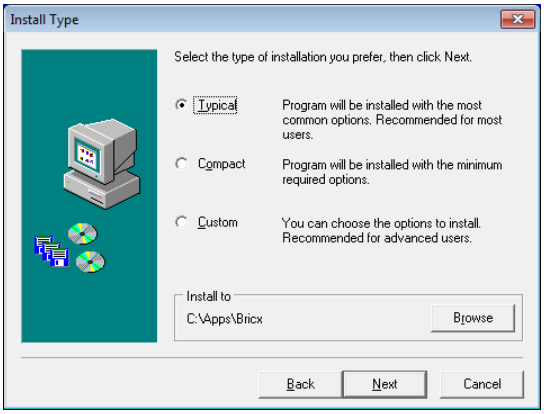
III.3.1. A Bricx CC telepítése

Töltsük le a *Bricx Command Center* (BricxCC) legutolsó verzióját a *http://bricxcc.sourceforge.net/* oldalról.

Kezdje el a BricxCC környezet (IDE) telepítését a letöltött, például *bricxcc_setup_33810_20130220.exe* futtatásával, majd kövessük a telepítés egyes lépéseit (Next gomb).

A telepítéshez válasszunk egy rövid, lehetőleg szóközöket és egyéb különleges karaktereket nem tartalmazó nevű mappát, például *C:\Apps\Bricx*, így később nem kell különösebb, úgynevezett Escape-szekvenciákkal (vezérlőkarakterekkel) törődnünk a programozás során.

Válasszuk ki a *Typical* (Tipikus) telepítési módot a 141. ábrának megfelelően.



141. ábra: *Telepítés*

Írjuk be a *http://bricxcc.sourceforge.net/test_releases/* címet a böngészőbe, majd töltsük le innen a legutolsó dátummal rendelkező *test_release.zip* csomagot. Legyen ez például a *test_release20131007.zip*.

Csomagoljuk ki a ZIP-et a BricxCC telepítési mappába, a fenti esetben ez a: *C:\Apps\Bricx*.

Ugyanebben a mappában keressük meg a *linux_tools.zip* csomagot, hozzunk létre a telepítési mappában egy *linux_tools* nevű mappát, majd csomagoljuk ki ide a ZIP-pet.

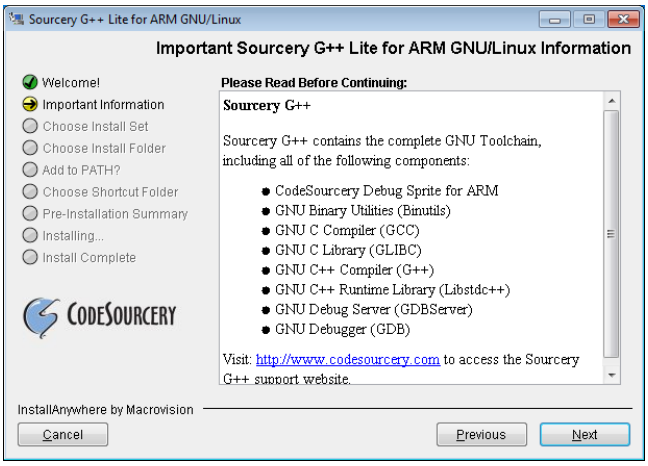
Ugyaninnen töltsük le az *lms_api.zip* csomagot is. A telepítési mappában hozzunk létre egy *API* nevű mappát, majd csomagoljuk ki ide az *lms_api.zip*-et.

Így a BicxCC telepítésével megvagyunk.

Telepítsük most a Sourcery G++ LiteToolchains for the EV3 segédprogramot is, amely letölthető a http://www.codesourcery.com/sgpp /lite/arm/portal/package4573/public/arm-none-linux-gnueabi/arm-2009q1-203-arm-none-linux-gnueabi.bin oldalról.

A telepítéshez indítsuk el a letöltött *arm-2009q1-203-arm-none-linux-gnueabi.exe* programot.

Itt is válasszunk egy rövid, lehetőleg szóközöket és egyéb különleges karaktereket nem tartalmazó nevű mappát, például *C:\Apps\GPP*.



142. ábra: *A G++ telepítése*

Amint a 142. ábrán láthatjuk, a programcsomag egy teljes G++ fordítóprogramot, függvénykönyvtárakat stb. tartalmaz.

Válasszuk itt is a *Typical* (Tipikus) telepítési módot, állítsuk be a telepítési útvonalat, ez most: *C:\Apps\GPP*, majd telepítsük a csomagot.

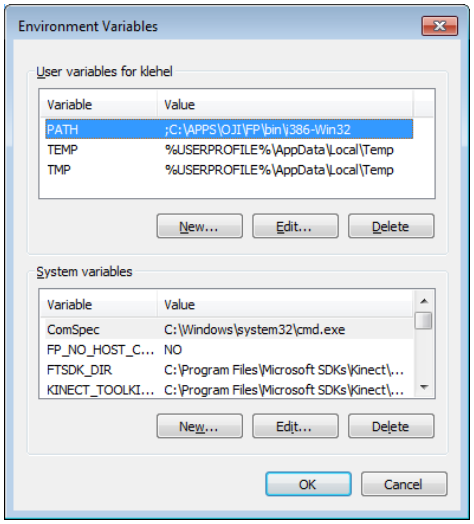
Rendszerünk beállításának következő lépése a telepített szoftverek elérési útvonalainak a megadása.

Ehhez menjünk a Windows kezelőpanel *\Control Panel\System and Security\System* lapjára, majd itt a bal oldali menüsorból kattintsunk az *Advanced System Settings* (Haladó rendszerbeállítások) sorra. A megjelenő párbeszédablak *Advanced* (Haladó) fülecskéjé-

ben nyomjuk meg az *Environment Variables* (Környezeti változók) gombot. Ekkor a 143. ábrán látható párbeszédablak fog megjelenni.

Itt a *PATH* (elérési útvonal) soron állva nyomjuk meg az *Edit…* (Szerkeszt) gombot.

A megjelenő párbeszédablakban a *Variable value* (Változó értéke) sorhoz adjuk hozzá pontosvesszővel elválasztva a BricxCC, a linux_tools, valamint a G++ programkönyvtárait például a következőképpen: *;C:\Apps\Bricx;C:\Apps\Bricx\linux_tools;C:\Apps\GPP\bin*.



143. ábra: *A környezeti változók beállítása*

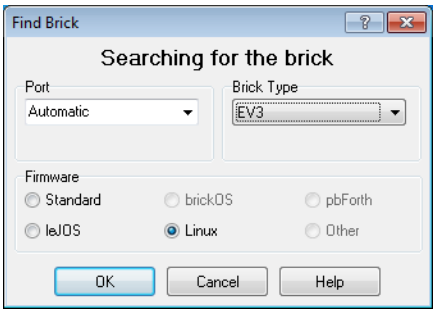
Végezetül a rendszerünk beállításának utolsó lépéseként töltsük le, és telepítsük a LEGO EV3 tégla legutolsó verziójú firmware-t. Az elektronikai rendszerekben és a számítástechnika területén *firmware* alatt azokat a rögzített, többnyire kisméretű programokat és/vagy adatstruktúrákat értjük, melyek különböző elektronikai eszközök vezérlését végzik el. Ez maga a LEGO EV3 robot operációs rendszere.

A firmware-t elérhetjük a *https://www.lego.com/en-gb/mindstorms/downloads* oldalon az EV3 MINDSTORMS FIRMWARE DOWNLOAD (PC/MAC) fejezetben. A cikk írásának pillanatában ez a V1.09H verziójú volt.

A firmware letöltése után a számítógépről ezt át kell telepítenünk a LEGO EV3 téglára. Ehhez elsősorban arra ügyeljünk, hogy a tégla elemei vagy akkumulátorai ne legyenek kifogyóban, legyen legalább 10 perc működésre elegendő energia bennük.

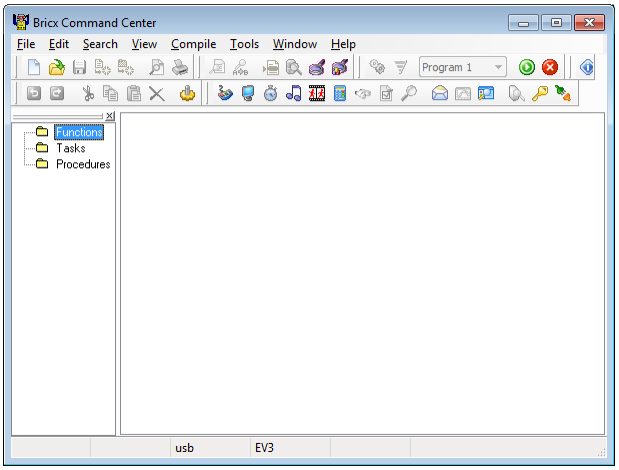
Kapcsoljuk be a LEGO EV3 téglát, és csatlakoztassuk a számítógéphez az USB kábel segítségével.

Indítsuk el a Bricx Command Center környezetet. Itt először a 144. ábrán látható beállításokkal (Port: Automatic, Brick Type: EV3, Firmware: Linux) keressük meg a téglánkat.



144. ábra: *A tégla megkeresése*

A tégla megkeresése után megjelenik a 145. ábrán látható környezet, amelyben a programozás mellett számos más lehetőség adódik a LEGO EV3 tégla beállításaira, kezelésére.



145. ábra: *A Bricx Command Center*

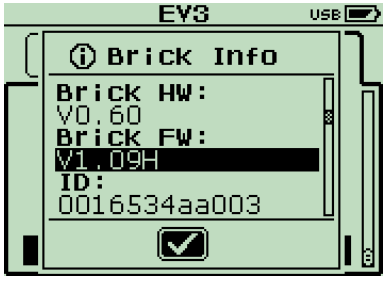
A firmware téglára való telepítése érdekében a *Tools* menü *Download Firmware* (Firmware letöltés) parancsát válasszuk ki. Ekkor egy párbeszédablak jelenik meg, amelyben beállíthatjuk, hogy melyik firmware-t szeretnénk telepíteni. Keressük meg és adjuk meg a LEGO oldalról letöltött firmware-t (pl. EV3 Firmware V1.09H.bin).

A párbeszédablak bezárása után a rendszer elkezdi telepíteni az új firmwaret, a tégla kijelzőjén is megjelenik az *Updating.* (frissítés) felirat. A firmware letöltése az EV3 téglára körülbelül 5–7 percet vesz igénybe. A letöltés befejezése után a tégla újraindul.

Természetesen a firmware frissítését el tudjuk végezni a LEGO MINDSTORMS EV3 Home Edition *Tools* (Eszközök) menüjének *Firmware Update* (Firmware frissítő) parancsával is.

Ha meg akarunk győződni a firmware frissítéséről, az EV3 tégla képernyőjének jobb szélén keressük meg a csavarkulcsot (a jobb téglagomb nyomogatásával), majd itt válasszuk ki a *Brick Info* (tégla információ) lehetőséget a lefelé gomb nyomogatásával. Itt a *Brick FW:* sorban megjelenik a firmware verziószáma a 146. ábra szerint.

A rendszerünk kész, használhatjuk.



146. ábra: *A firmware verziószáma*

Kovács Lehel István

Centrált rendszerek

III. rész

8. Centrált rendszerek egyesítése

Optikai eszközök készítésekor gyakran kerülünk olyan helyzetbe, hogy egyszerűbb optikai rendszerek egyesítésével tudunk a célnak megfelelő leképező rendszert kialakítani. Nyilvánvaló, ha két centrált rendszert úgy egyesítünk, hogy optikai tengelyeik egybeesnek, új centrált rendszert kapunk. Ismerve a részrendszerek adatait és egymáshoz viszonyított helyzetüket, meghatározhatjuk az egyesített rendszer adatait is. Ehhez azt a gyakran használt elvet alkalmazzuk, melynek értelmében az első rendszer képtere tárgytér a következő számára. Így az egyesített rendszer az első leképező eszköz tárgyterét a második képterébe képezi le.

A következetes tárgyalás érdekében megegyezünk abban, hogy az első rendszer adatait egy vesszővel (egyszer jelzett), a másodikét két vesszővel (kétszer jelzett) látjuk el, míg az egyesített rendszer adatait jelöletlenül hagyjuk. A két rendszer egymáshoz viszonyított helyzetét a második rendszer tárgytéri gyújtótávolságától az első rendszer képtéri gyújtótávolságáig mért irányított szakasz határozza meg. Ez, az 5. ábra jelölését felhasználva, a

$$\Delta = F_1''F_2' \tag{8.1}$$

optikai köz vagy a mikroszkópoknál használt elnevezés szerint optikai tubushossz.