Programozási alapismeretek

Tóth Zsolt

Programozási alapismeretek

Tóth Zsolt

KÉSZÜLT A TÁMOP-4.1.2.A/1-11/1-2011-0096 TÉMÁJÚ, FELSŐFOKÚ SZAKKÉPZÉSEK FEJLESZTÉSE AZ NYME TTK-N PROJEKT KERETÉBEN

Tartalom

1. fejezet - Bevezetés

Tapasztalataink szerint a programozási ismeretek oktatása – az informatikai szakközépiskolai és mérnöki képzéseket leszámítva –meglehetősen mostohagyereknek számít mind a köz-, mind a felsőoktatásban. Az oktatásban részvevők többsége a programozást felesleges, nehéz, speciális területeken alkalmazható tárgynak érzi.

A programozástól való idegenkedés lényegében mind az oktatók, mint a diákok részéről érthető. A diákok többsége talán a tanulmányok befejezése után sohasem fog „élesben", általános programozási nyelveken programokat írni, a későbbi munkához kapcsolódó alkalmazások szerepe pedig rendkívül megnőtt, ezért a kevés informatikai órát a közkeletű felfogás szerint érdemesebb inkább ezekre szánni.

A programozási alapok elsajátítása azonban meggyőződésünk szerint méltatlanul mellőzött diszciplína, s általában véve minden képzési szinten és szakterületen helye lehet az oktatásban. Miért tartjuk fontosnak, hogy a nem programozónak készülő diákok programozást tanuljanak?

1. A legtöbb felhasználói program személyes és szervezeti igényekhez szabása vagy akár professzionális használata alapvető algoritmizálási alapismeretek nélkül gyakran elképzelhetetlen. Programozási alapismeretek birtokában például sokkal könnyebb a táblázatkezelők feltételes elágazásainak vagy egymásba ágyazott függvényeinek használata, egy összetett makró írása pedig eleve igényli a makró programnyelvének ismeretét.

2. Szinte nincs olyan szakterület, ahol ne használnának ún. nagyon magas szintű, speciális célra létrehozott programozási nyelveket. Számos ipari, szimulációs, gazdasági vagy statisztikai szoftver, de bizonyos megközelítésben lényegében a legtöbb felhasználói szoftver is ebbe a kategóriába tartozik. A magas szintű programozási nyelvek elsajátítását a programozási alapismeretek elsajátítása nagyban megkönnyíti.

3. A különböző online tartalommenedzsment rendszerek (CMS) használata rendkívül elterjedt az utóbbi időben, használatuk többnyire nem igényel programozási előismereteket. Azonban számos esetben felmerülhetnek olyan problémák, amelyek a programkód kisebb átalakítását igénylik. Alapvető programozási ismeretek birtokában pl. egy PHP-kód sokkal kevésbé tűnik idegennek, mint anélkül.

A fenti, programozásoktatás melletti érvek ellen természetesen fel lehet hozni ellenérvként, hogy mindegyikre alkalmazható képzett programozó. Ez részben talán igaz, azonban az adott szakterület specialistája vagy a szervezeti viszonyok ismerője hatékonyabban elláthatja ezeket a feladatokat. Ráadásul napról napra felmerülő apróbb problémákra – különösen kisebb szervezetekben – nem lehet folyamatosan „professzionális" programozókat igénybe venni, az időkorlátok és a költséghatékonysági szempontok ezt nem engedik meg. Ezenkívül egyáltalán nem biztos, hogy az adott, nagyon magas szintű programozási nyelvet a legtöbb, általános informatikai képzettséggel rendelkező, és esetleg más szakterületen tevékenykedő programozó behatóan ismeri.

Célzottan nem soroltuk fel azokat a programozási feladatokat, amelyeket véleményünk szerint alapvetően „professzionális" programozók láthatnak el. Azonban tapasztalataink szerint megfelelő elszántság és némi logikai érzék birtokában bárki programozóvá válhat, s akár nagyon magas szinten művelheti ezt a területet.

A programozásoktatás mellett természetesen számos egyéb propedeutikai érv felhozható, a programozási guruk talán az egész informatikai oktatást kizárólag programozásra építenék. Azonban a realitások talaján maradva is biztosak vagyunk abban, hogy a programozás – akár csak „játékként", agytornaként, a logikai készségek fejlesztéseként – mindenki számára hasznos lehet.

A jegyzetet elsősorban olyan leendő – jelenleg elsősorban a felsőfokú-felsőoktatási szakképzésben tanuló – szakembereknek szánjuk, akiknek munkájuk során informatikai, statisztikai és gazdasági ismereteikre kell majd hagyatkozniuk. A jegyzet nyelvezetét és felépítését ezért az informatikai-programozói képzésekhez képest leegyszerűsítettük. Az alapozáshoz használt programozási nyelvet

is a célcsoport várható igényeihez szabtuk, de törekedtünk a minél játékosabb alkalmazások bemutatására.

Az általános, alapozó célú programozási nyelv és programfejlesztői környezet kiválasztása során el akartuk kerülni a programozási képzésekben gyakran tapasztalt csapdákat. Nem akartunk a régi, „jól bevált" (gyakran szinte évtizedek óta használt) programozási nyelvek és programozási környezetek közül válogatni, mert ezek többnyire már az első percektől elidegenítik a nem kimondottan programozónak készülő diákokat. A divatos és összetett programozási nyelvek és programozási környezetek közül pedig azért nem akartunk kiválasztani egyet, mert gyakran már a nyelv összetettsége és a fejlesztőkörnyezet lenyűgöző, funkcionális gazdagságának bemutatása meghaladja egy alapozó kurzus kereteit.

Hosszas mérlegelés után ezért – elnézést kérve a szabad szoftverek híveitől, akikkel minden szempontból rokonszenvezünk – a Microsoft Small Basic programozási környezetére esett a választásunk. A Small Basic által használt BASIC nyelv egyszerű, kimondottan kezdő programozóknak készült, a programfejlesztői környezet pedig letisztult, esztétikus, barátságos. A Small Basic ráadásul ingyenes – bár nem nyílt forráskódú – alkalmazás.

A jegyzet sokkal inkább emlékeztet egy felhasználói kézikönyvre, mint egy hagyományos programozási jegyzetre. Úgy gondoltuk, hogy – igazodva a kezdő programozók számára „emészthetőbb" módszertanhoz, amelyet a Microsoft által közzétett Small Basic-leírások is követnek – helyesebb, ha a felhasználók minél hamarabb programokat tudnak írni, mint hogy előbb tisztában legyenek a programozás teljes elméleti hátterével. A programozáselméletet amúgy is csak a szükséges és elégséges mértékben kívánjuk érinteni.

2. fejezet - A Small Basic és a BASIC

A Small Basic kialakítása során arra törekedtek, hogy a programozás inkább öröm, mint terhes kötelesség legyen a kezdő felhasználó számára.

A Small Basic fejlesztőkörnyezet a BASIC nyelv egy alapvetően 15 kulcsszóra épülő egyszerűsített változatára és Microsoft .NET platformjára épül, első változata 2008-ban jelent meg. Számos programkönyvtárral kiegészíthető, így a fejlesztőkörnyezet nemcsak a kezdő, hanem a professzionális programozók számára is tartogat lehetőségeket.

A BASIC – Beginner's All-purpose Symbolic Instruction Code, azaz a kezdők bármely célra használható szimbolikus utasítási kód – nyelvet az amerikai magyar Kemény János György és Thomas Eugene Kurtz fejlesztette ki 1964-ben.

A '70-es, '80-as évek Commodore, ZX, HT és egyéb „tévékomputereinek" felhasználói számára az adott géphez szabott – és többnyire a többivel inkompatibilis – BASIC-változat nem egy volt a programozási nyelvek közül, hanem maga volt „A" programozási nyelv. A különböző alkalmazások – gyakran játékok – többsége BASIC-ben készült.

A '90-es évek elejétől kibontakozó PC-forradalom némileg háttérbe szorította a BASIC-et, bár az oktatásban DOS alatt futtatható QBasic (QuickBasic) sokáig népszerű maradt. Az 1991-ben kialakított, az ún. objektumorientáltság (számunkra ez a fogalom csak a későbbiekben válhat érdekessé) felé elmozduló Microsoft Visual Basic, majd a 2002-ben megjelent, és azóta is folyamatosan fejlesztett, már teljesen objektumorientált, a .NET keretrendszerbe integrált Visual Basic .NET pedig beemelte-visszahozta a BASIC-et a professzionális programozás világába. A különböző Microsoft-alkalmazásokba (Excel, Word, Access stb.) beépített VBA (Visual Basic for Applications) makrónyelv és a Windows operációs nyelv scriptnyelveként funkcionáló Visual Basic Script szintén növelte a BASIC népszerűségét. Emellett számos egyéb, általános vagy speciális célra használt „BASIC-klón" létezik a tulajdonosi és a szabad szoftverek világában.

BASIC-et tanulni tehát nemcsak pedagógiai okokból érdemes, a BASIC nyelv alapjai közvetlenül számos környezetben felhasználhatóak.

3. fejezet - Ismerkedés a Small Basic-kel és a programozással

A fejlesztőkörnyezet letöltése

Első lépés, hogy hozzájussunk a Small Basic fejlesztőkörnyezethez. A szoftver – e sorok írásakor – legfrissebb v1.0 változatát legegyszerűbben a http://smallbasic.com oldal *Download* pontjára kattintva lehet letölteni. A letöltött SmallBasic.msi fájl közvetlenül futtatható. Windows XP, Windows Vista, Windows 7 és (vélelmezhetően) Windows 8 operációs rendszer alatt telepíthető, ha a .NET keretrendszer legalább 3.5-ös verziója is fut az adott gépen.

A telepítés során általában csak a *Next* gombra kell kattintani, de a második oldalon ne felejtsünk az „I accept the terms in the License Agreement" előtti jelölőnégyzetbe kattintani (s ezzel a felhasználás feltételeit elfogadni). A harmadik oldalon elvileg különböző nyelvi kiegészítések közül is válogathatunk, de jelenleg a magyar nyelv nem választható (ez természetesen a későbbiekben változhat).

Telepítés után a Start menüben elérhető a Small Basic angol nyelvű fejlesztőkörnyezet, nyissuk meg! (A fejlesztőkörnyezet angol nyelvű, azonban olyan kevés menüpontot és kulcsszót tartalmaz, hogy nyelvtudás nélkül is elboldogulunk vele, néhány angol mondat megtanulása pedig bárkinek hasznos lehet. Természetesen más nyelv is választható.)

A fejlesztőkörnyezet felépítése és első programunk

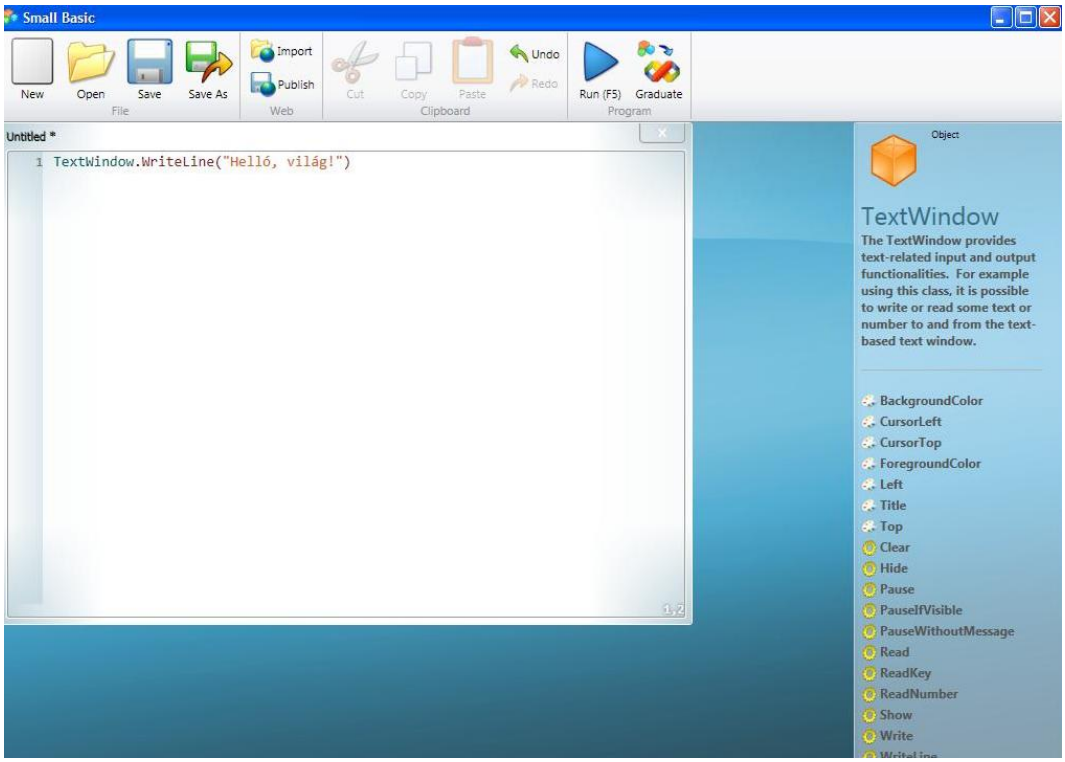
A fejlesztőkörnyezet letisztultsága első ránézésre is szembetűnő. Az *Eszköztár* a *New* (Új), *Open* (Megnyitás), *Save* (Mentés), *Save As* (Mentés másként), *Import* (Importálás), *Publish* (Közzététel), *Cut* (Kivágás), *Copy* (Másolás), *Paste* (Beillesztés), *Undo* (Vissza), *Redo* (Előre), *Run* (Futtatás) és a Visual Basic-kóddá történő átkonvertálást végző *Graduate* menüpontokat tartalmazza.

Készítsük el első programunkat! A programozási kurzusokban már szinte hagyománnyá vált, hogy első lépésben mindig a „Helló, világ!" című „elmés" szöveget íratjuk ki, mi sem szakítjuk meg ezt a „nemes" hagyományt.

A *Szerkesztőbe*, a fejlesztőkörnyezet baloldali-középső részébe – a valószínűleg megjelenő *Untitled** felirat alatt, az 1-es sorszámmal kezdődő részbe – az első sorba írjuk be a következő szöveget:

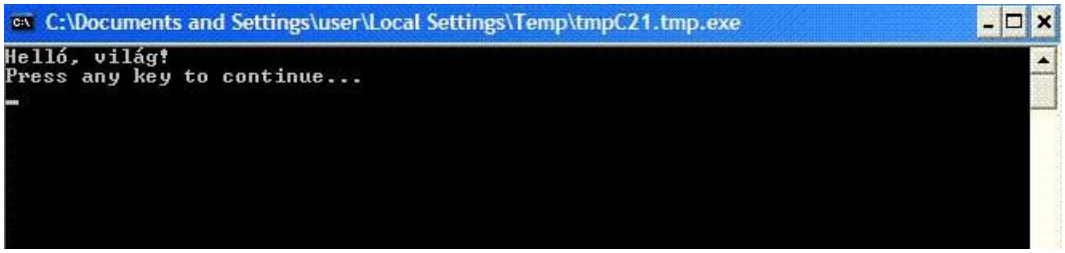
```
TextWindow.WriteLine("Helló, világ!")
```

Talán már a gépelés alatt feltűnt, hogy beíráskor egy felugró ablakból válogathatunk az egyes elemek közül, amelyekről már az ablakban kapunk némi leírást, illetve a választható elemek (később ezt a fogalmat még finomítjuk) továbbiakkal egészíthetők ki a jobb szélső *Információs sávban*.



1. ábra: A Small Basic futtatókörnyezet első programunkkal

Ha beírtuk Szerkesztőbe első programkódunkat, futtassuk! Kattintsuk az eszköztáron található *Run* gombra vagy nyomjunk az F5 funkcióbillentyűre! Ha mindent jól írtunk be, egy új ablakban látjuk programunk futás utáni eredményét. (2. ábra)



2. ábra: Első futtatott programunk

Mentsük el a programkódot a Save gombra kattintva vagy a **CTRL+S** billentyűkombinációval! (Különösen összetettebb programokat érdemes rendszeres időközönként menteni, közhely, de tényleg érdemes erre odafigyelni.) Ha a programkódot „elso” néven mentjük, a Szerkesztő felett láthatjuk, hogy a kód *.sb* kiterjesztést kapott.

Ha fájlkezelővel megvizsgáljuk a könyvtárat, ahová az .sb kiterjesztésű fájlt mentettük, megállapíthatjuk, hogy a mentés után három másik fájl is látható a könyvtárban:

- *elso.exe*: a kód lefordított és futtatható verziója,
- *elso.pdb*: a program megfelelő működéséhez tartozó adatbázis,
- *SmallBasicLibrary.dll*: a Small Basic futásidejű programkönyvtára, olyan elemeket/függvényeket tartalmaz, amelyek nélkülözhetetlenek a megfelelő futáshoz.

Bár az első program – törvényszerűen – nem tűnik túl „izmosnak”, a legfontosabb lépésen túl vagyunk, s már ez az egyszerű programocska is érdemes a tanulmányozásra.

Programszerkezet

Technikailag a későbbiekben is nagyon hasonló – csak összetettebb és több – lépésből állnak majd össze programjaink. Valamilyen úton-módon *instrukciókat* (végrehajtandó feladatokat) fogalmazunk meg BASIC nyelven és ezek a szükséges logikai rendbe szervezve a megfelelő eredményt adják.

Az egyes instrukciók a BASIC programozási nyelvben *utasításoknak* feleltethetők meg. A program minden egyes sora egy-egy utasítás. Ha futtatjuk a programkódot, a Small Basic végrehajtja azokat egészen a program végéig. Pontosabban: a program *fordítója* (*fordítóprogramja*, *compilerje*) létrehoz egy .exe kiterjesztésű fájlt, amelyet futtat.

Ha megvizsgáljuk az első programkódunkat, az első utasításunk alapvetően három részből áll:

- TextWindow
- WriteLine
- „Helló, világ!”

Természetesen a különböző egyéb elemek – pontok, zárójelek – is fontosak, és megváltoztatásuk kihatással van a program működésére, de a logikailag elkülönülő elemek a fentiek.

A *TextWindow* (az angol „szöveg” és „ablak” szavakból) az első programunk futása során felugró, fekete hátterű ablaknak felel meg, amelyet szokás *konzolnak* is hívni.

Más megközelítésben a TextWindow az első *objektumunk*, amit használtunk. A Small Basicben – mint általában minden programozási nyelvben, nemcsak a szigorúan objektum-orientáltakban (erről későbbi stúdiumaikban még lehet szó) – számos objektum van. Bár az objektum megragadása egzakt módon nem is olyan könnyű, talán kezdő szinten elégséges – és korántsem tökéletes – definíció, hogy az objektum tulajdonságokkal (paraméterekkel) és műveletekkel-viselkedésekkel-eljárásokkal felruházott egység.

A Small Basicben az egyes objektumoknak lehetnek *tulajdonságai* (*Property*), ill. *műveletei* (*Operation*). A műveletek egy része a magyar informatikai szaknyelvben *metódusként, eljárásként*, mások inkább *függvényként* közelíthetők meg. Mi azonban nem térünk el a Microsoft szóhasználatától, ezért a későbbiekben egységesen tulajdonságokról és műveletekről beszélünk.

A TextWindow után ponttal elválasztva a *WriteLine(data)* – az angol „ír” „sor” és „adat” szavakból – elemet látjuk. A WriteLine a TextWindow objektum egyik művelete, s talán nem okoz meglepetést, hogy egy olyan műveletet takar, amely kiír egy sort (egy szöveget) a konzolra. A „Helló, világ!” pedig az az *input* (bemeneti érték, jel), amely az adott sorban megjelenik.

Második programunk

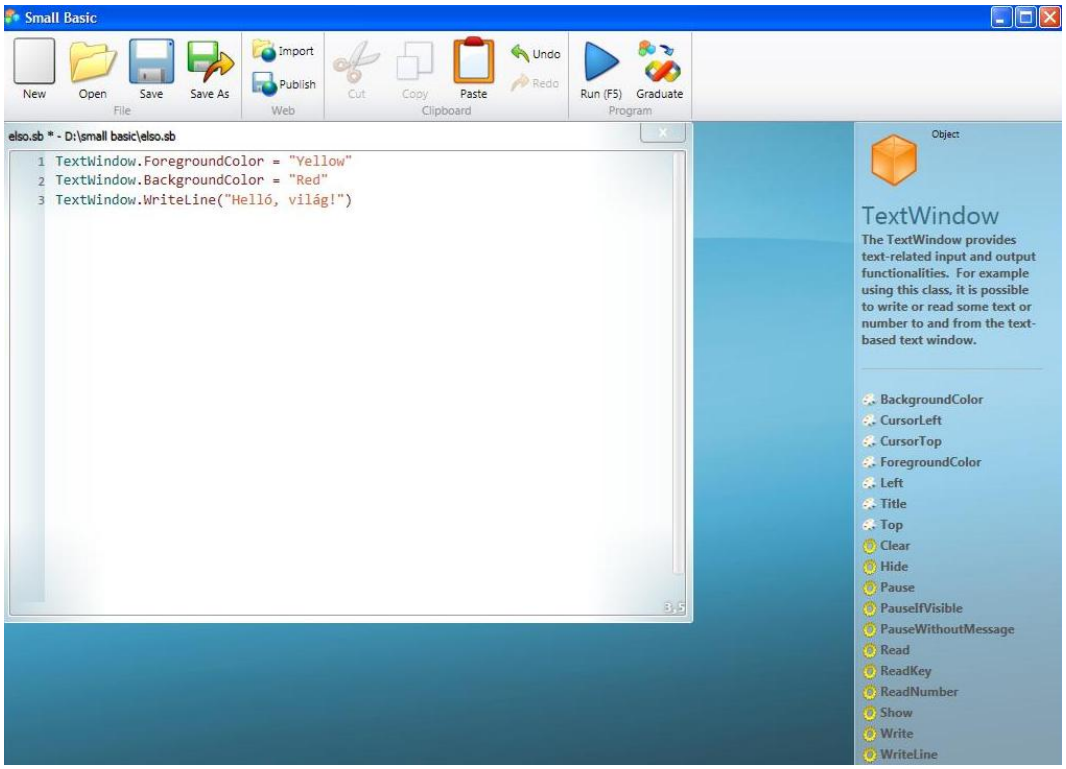
Második programkódunk alig bonyolultabb az elsőnél, de már három sorból áll. Írjuk be a Szerkesztőbe a következőt, majd futtassuk:

```
TextWindow.ForegroundColor = "Yellow"
TextWindow.BackgroundColor = "Red"
TextWindow.WriteLine("Helló, világ!")
```

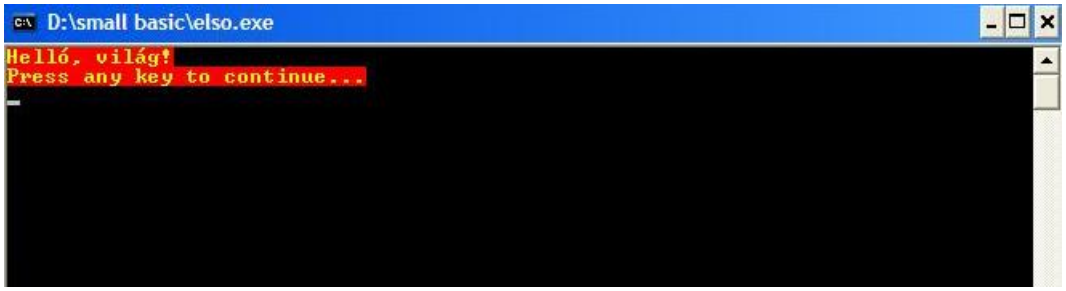
A program futtatása során azt tapasztaljuk, hogy a képernyőn feltűnő szöveg piros háttérrel és sárga betűkkel jelenik meg. Ezt a TextWindow objektum *ForegroundColor* (az angol „előtér” és „szín” szavakból) és *BackgroundColor* (az angol „háttér” és „szín” szavakból) *tulajdonság*ok megadásával értük el. A *Yellow* sárgára változtatta betűszínt, a *Red* vörösre festette a szöveg hátterét. Hasonló módon

a *Black, Blue, Cyan, Gray, Green, Magenta, Red, White, Yellow, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, DarkYellow* színeket is használhattuk volna.

A művelet és a tulajdonság határait esetenként nem is olyan könnyű definiálni, hiszen mindkettő az objektumra „irányul”. Formailag azonban a Small Basicben egyszerű a megkülönböztetés. A műveletek inputjai zárójelben vannak, míg a tulajdonságok értékeit egyenlőségjel segítségével adjuk meg.



3. ábra: A Small Basic futtatókörnyezet második programunkkal



4. ábra: A második program futási eredménye

Ahhoz hogy a programjaink változatos feladatok megoldására legyenek alkalmasak, kicsit jobban bele kell ásnunk magunkat a programozáselméletbe és a BASIC nyelv szabályaiba, elemeibe.

4. fejezet - A változók használata

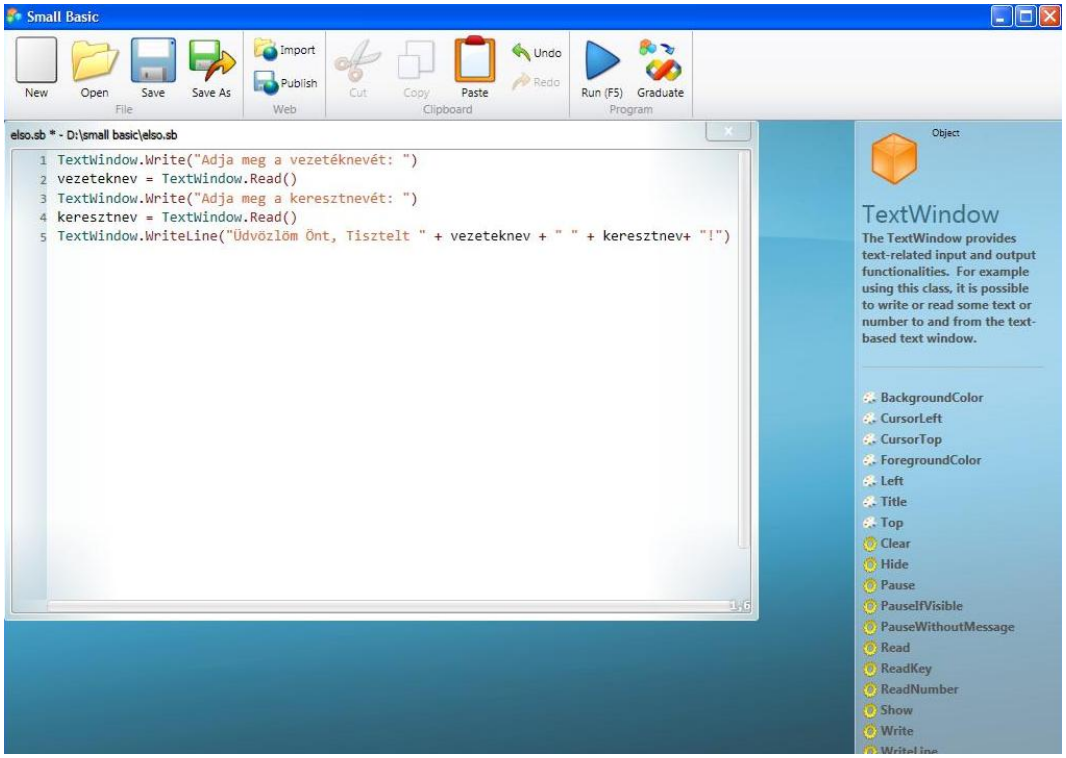
Névbekérő program

Fejlesszük tovább az első programunkat! A program kérje be a vezetéknevünket és keresztnevünket, majd köszöntsön bennünket teljes nevünkön!

Írjuk be a Szerkesztőbe a következő programkódot:

```
TextWindow.Write("Adja meg a vezetéknevét: ")
vezeteknev = TextWindow.Read()
TextWindow.Write("Adja meg a keresztnevét: ")
keresztnev = TextWindow.Read()
TextWindow.WriteLine(
    "Üdvözlöm Önt, Tisztelt " + vezeteknev + " " + keresztnev+ "!")
```

Ha a fenti kódot futtatjuk, programfutás közben némi interakcióra is lehetőségünk nyílik. Ehhez azonban *változó*kat kellett használnunk.



5. ábra: „Névbekérő” programunk

6. ábra: „Névbekérő” programunk futási eredménye

Vizsgáljuk meg a program új elemeit!

A program futása közben tapasztalhattuk, hogy az „Adja meg vezetéknevét:” és az „Adja meg keresztnevét:” szöveg megjelenítése után egy villogó kurzor jelent meg, amely jelezte, hogy a program egy érték megadását – a vezetéknév és a keresztnév beírását – várja a felhasználótól. Ezt a

```
vezeteknev = TextWindow.Read()
```

és a

```
keresztnev = TextWindow.Read()
```

sorokkal értük el.

A *Read()* (az angol „olvas” szóból) a TextWindow objektum egyik művelete. A művelet a program futása során bekér egy karaktersort (egy sztringet, egy szöveget) a felhasználótól, s amíg a bevitt karaktersort a felhasználó az „Enter” billentyűre kattintva le nem zárja, a program nem lép a következő sorra.

A két egymással szembeállított zárójel arra utal, hogy műveletről van szó, de mivel közöttük semmi sincs, azt is jelzi, hogy az inputot esetünkben nem a programkód tartalmazza, hanem a felhasználótól a program futása közben kérjük be.

A Read() művelettel bekért-beolvasott szöveg azonban egyben egy változó értékadását is elvégzi. A „keresztnev” és „vezeteknev” változók elraktározzák a bevitt értéket, s a program futásának végéig – ha közben az értéket meg nem változtatják – meg is őrzik.

A fent vázolt tulajdonság alapján lényegében definiálhatjuk a változó jelentését: A változó névvel ellátott adattároló.

A programkód utolsó sorában a WriteLine művelet inputjának összetett megadására látunk példát:

```
TextWindow.WriteLine(
    "Üdvözlöm Önt, Tisztelt " + vezeteknev + " " + keresztnev+ "!")
```

A program a változók értékeit az idézőjelek közé rakott szöveges konstansok közé írja ki.

Az input a kódírás közben megadott szövegdarabokból és a program futása közben bekért változók értékeiből áll össze, ez jelenik meg köszöntésként a képernyőn.

A változókra vonatkozó szabályok

A változókra számos programnyelvben szigorú szabályok vonatkoznak, s általában a változó típusát – a változóban tárolható adatok jellegét (szám, szöveg, dátum stb.) – is definiálni kell. Ennek kényelmetlen jellege ellenére számos előnye van. Például a megfelelően kialakított változótípusokkal jelentősen csökkenhető a program erőforrásigénye.

A Small Basicben más programnyelvekkel összehasonlítva viszonylag kevés szabály vonatkozik a változókra:

- A változó nevének betűvel kell kezdődnie, s a név nem tartalmazhat *foglalt kulcsszavakat* (később lesz róluk szó), mint például az *if*, a *for* vagy a *then*.

- A változó nevében a betűk, a számok, és aláhúzásjelek (_) szabadon kombinálhatók.
- Érdemes a változók nevét úgy megfogalmazni, hogy utaljanak a bennük tárolt tartalomra.
- A változónév nem lehet hosszabb 40 karakternél.
- Bár a programozók között szinte hagyománnyá vált, hogy a változók – s a különböző címkéket, feliratokat leszámítva minden általuk definiált elem – nevét szinte kizárólag az ASCII-kódtábla (az angol nyelvben használt alapkarakter-készlet) karaktereiből állítják össze, elvileg a változók nevében is használhatók a magyar ékezetes betűk. Tehát a *vezetéknév* és a *keresztnév* változó használata sem vezetett volna hibához. Azonban a programozói tapasztalat szerint még a mai napig a karakterkódolási problémák vezetnek a legtöbb hibához, ezért a hagyomány tovább él. A jegyzetben ezért mi is csak kivételes esetben használunk ékezetes betűket a változónevekben.

Számos programozási nyelvben a változók érvényességi köre alapján pl. globális és lokális – vagy pl. modulszintű – változókat is megkülönböztetünk. A globális változók a program egészében, a többi típus csak egyes programrészekben (modulokban, szubrutinokban, a programozó által definiált függvényekben, eljárásokban stb.) értelmezett.

A Small Basicben azonban minden változó globális változó!

(Részben ebből fakad, hogy Small Basicben a számos programozási nyelvben ismert argumentumátadás is hiányzik.)

„Számtároló” program

A következő egyszerű példaprogram a számok tárolására ad példát:

```
TextWindow.Write("Adjon meg egy számot: ")
szam1 = TextWindow.ReadNumber()
TextWindow.Write("Adjon meg még egy számot: ")
szam2 = TextWindow.ReadNumber()
TextWindow.WriteLine("A két szám szorzata: " + szam1 * szam2)
```

A TextWindow objektum Read() művelete helyett a *ReadNumber* (az angol „szám” és „(be)olvas” szavakból) műveletet használtuk a számok bekérésére. Ennek segítségével elkerülhetjük, hogy a felhasználó pl. véletlenül betűt üssön a számok közé, s ez hibás eredményhez vezessen.

Bár a fenti programkód helyes eredményt ad, érdemesebb a két szám szorzatát külön változóként definiálni, a s a programkódot a következőre módosítani:

```
TextWindow.Write("Adjon meg egy számot: ")
szam1 = TextWindow.ReadNumber()
TextWindow.Write("Adjon meg még egy számot: ")
szam2 = TextWindow.ReadNumber()
szam3 = szam1 * szam2
TextWindow.WriteLine("A két szám szorzata: " + szam3)
```

Programozástechnikai okból elsősorban azért érdemes változókat használni ott, ahol csak lehet, mert a változó értéke a program futása közben módosulhat, s a változók használata rugalmasabbá teszi a programozást. (Ennél az egyszerű példánál „bonyolultabb” lett a programkód viszont a program esetleges bővítése esetén a „szam3” változó rendelkezésre áll egyéb jellegű felhasználásra, amire az első változatnál nem lenne lehetőség. A későbbiekben majd mélyebben is megértjük a változók használatának fontosságát.)

5. fejezet - Feltételes elágazás

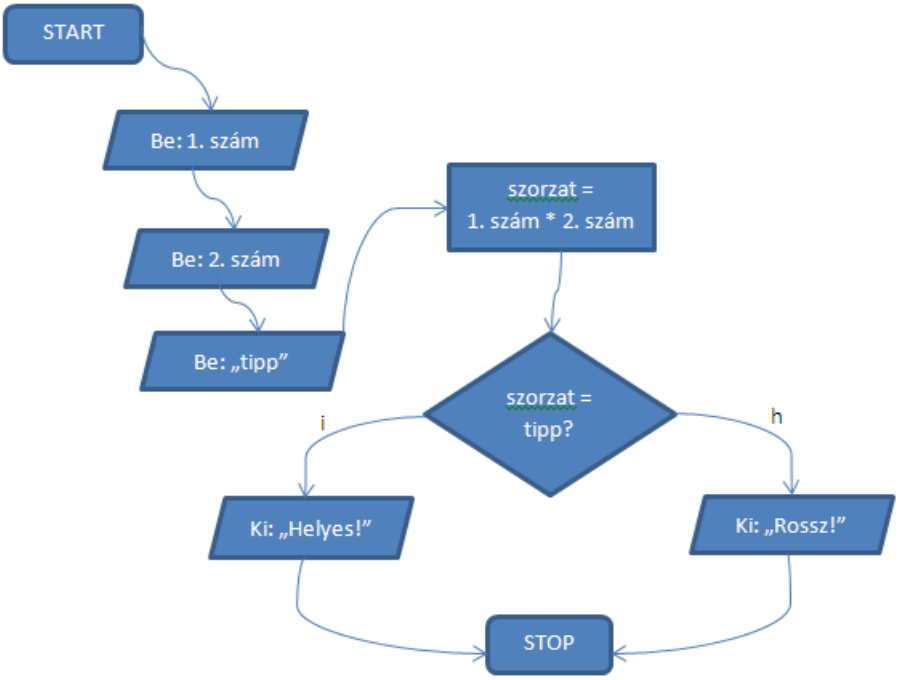
Az általános célú programozási nyelvek három fő ismérve, hogy a programok *szekvenciákra* (egymás utáni lépésekre – talán ezt az eddigiek alapján nem kell külön elmagyarázni), *feltételes elágazásokra* és *ciklusokra* oszlanak. Ebben a fejezetben a feltételes elágazásokkal ismerkedünk meg. A feltételes elágazások használata bármely *algoritmus* megfogalmazása során – az algoritmus egy adott cél elérése érdekében végrehajtott lépéssorozatot jelent – hasznos lehet, lényegében a logikus gondolkodás egyik alapja.

Algoritmus-tervezés

Az előző fejezetben két szám összeszorzására használt példát feltételes elágazás segítségével egy kicsit továbbfejlesztjük. Nemcsak az összeszorzandó két számot kérjük be a felhasználótól, hanem egy „tippet” is a megoldásra. Ha a tipp megegyezik a számított megoldással, a program kiírja, hogy a felhasználó helyes megoldást adott, ha nem, akkor jelzi a hibát.

A megoldás előtt érdemes a program algoritmusát vázolni.

A programozásban számos vizuális eszköz létezik az algoritmusok leírására. A legegyszerűbb, amikor ún. *pszeudokóddal* szövegesen leírjuk az algoritmust. Többnyire azonban *folyamatábrával*, *blokkdiagrammal*, *struktogrammal* vagy *Jackson-ábrával* dolgozunk.



7. ábra: Folyamatábra („i” – igaz, „h” – hamis)

Kezdő programozók számára véleményünk szerint ezek egy része kevésbé fontos (a bonyolultabb algoritmusoknál lehet rájuk szükség), de a későbbi stúdiumok során érdemes ezekkel megismerkedni, és minden programkód – kódrészlet – megírása előtt-során használni. Tapasztalataink szerint a programozók egy része előszeretettel használja a vizuális algoritmusleíró eszközöket, mások saját rendszert használnak, de elég népes azoknak a tábora, akik nem használnak ilyen eszközöket.

A 7. ábrán a fentiekben vázolt algoritmusunk egy lehetséges folyamatábráját látjuk (a szokással ellentétben nem teljesen függőlegesen rendezve, ezzel is érzékeltetve, hogy a saját algoritmusvázlataink sosem lesznek mérnöki pontosságúak). A jegyzet hátralévő részében nem közlünk ilyen ábrákat, de a tananyag feldolgozása során érdemes az egyes programok megírása előtt vázlatokat készíteni.

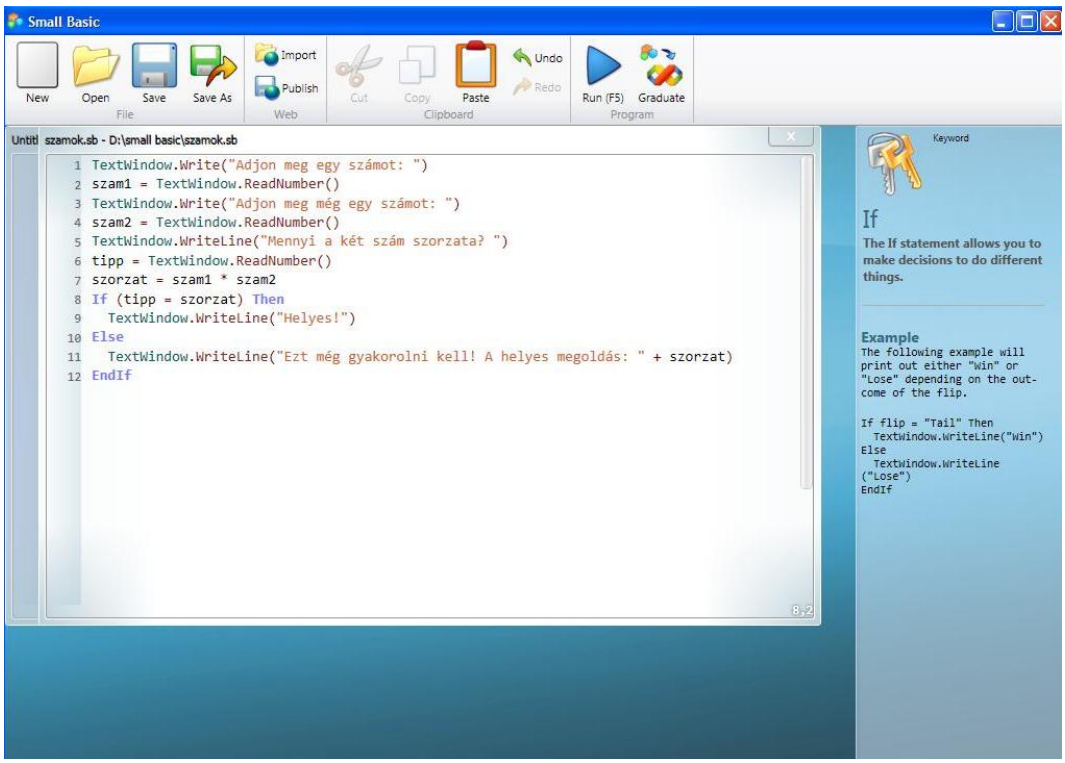
Talán már a 7. ábra rombuszába illesztett feltételből látszik, hogy a feltételes elágazás az alapján, hogy a megfogalmazott logikai feltétel teljesült vagy sem, két részre osztotta az algoritmus addig lineáris útját. Más érték került kiírásra, ha a feltétel teljesült, s más, ha nem. A feltételes elágazások segítségével a programok tehát nagyon sok „ágra bonthatók”.

Szorzóprogram feltételes elágazással

Írjuk be a Szerkesztőbe az alábbi programkódot:

```
TextWindow.Write("Adjon meg egy számot: ")
szam1 = TextWindow.ReadNumber()
TextWindow.Write("Adjon meg még egy számot: ")
szam2 = TextWindow.ReadNumber()
TextWindow.WriteLine("Mennyi a két szám szorzata? ")
tipp = TextWindow.ReadNumber()
szorzat = szam1 * szam2
If (tipp = szorzat) Then
    TextWindow.WriteLine("Helyes!")
Else
    TextWindow.WriteLine("Ezt még gyakorolni kell! A helyes megoldás: " + szorza
EndIf
```

A fenti programban a feltételes elágazást az *If, Then, Else* és *EndIf* kulcsszavak jelölik ki. Az If kulcsszó (az angol „ha” szóból) után zárójelben mindig valamilyen *logikai kifejezés* áll. (A zárójel használata nem kötelező.) A logikai kifejezések sokfélék lehetnek, de ha a lényegi vonásukat akarjuk megragadni, jó közelítéssel minden eldöntendő (igennel vagy nemmel megválaszolható kérdés) logikai kifejezésnek tekinthető.

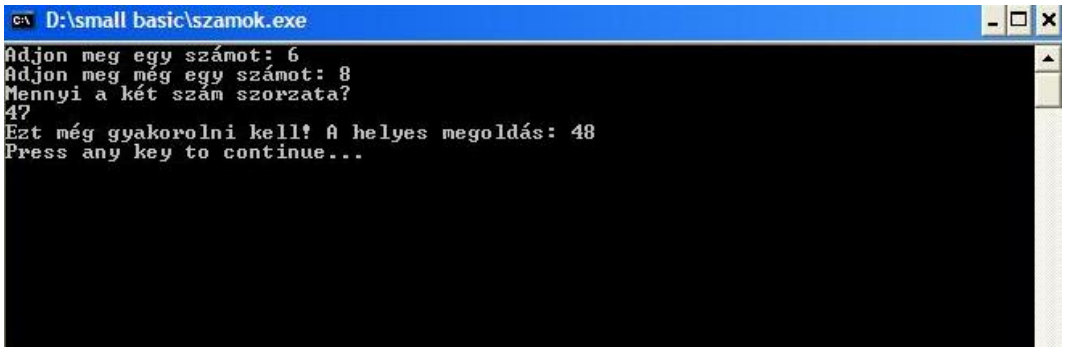


8. ábra: Szorzási példa a feltételes elágazásra

Logikai kifejezésekben gyakran valamilyen relációs (kisebb, nagyobb, egyenlő) jellel összehasonlítunk két elemet. (Későbbi programozói feladatokban bonyolultabb, triviális párhuzamokkal nehezebben

leírható logikai kifejezések is felmerülhetnek, azonban az összetettebb logikai definíciókat egyelőre – a jegyzet írójának megkönnyebbülésére is – nyugodtan elhagyhatjuk.)

9. ábra: A szorzási példa futási eredménye helyes válasz esetén



10. ábra: A szorzási példa futási eredménye helytelen válasz esetén

Ha az If után megadott logikai kifejezés teljesül (a *visszatérési érték* igaz), akkor a program a *Then* (az angol „akkor” szóból) után található sort vagy sorokat hajtja végre.

Ha a logikai kifejezés nem teljesül (a visszatérési érték hamis), akkor az értelmezőprogram figyelmen kívül hagyja a Then kulcsszó után írt utasítást vagy utasításokat és az *Else* (az angol „különben” szóból) kulcsszó utáni utasítást vagy utasításokat hajtja végre.

A feltételes szerkezetet az *EndIf* (az angol „vége” és „ha” szavakból) kulcsszó zárja le.

A feltételes elágazás kulcsszavai közül az Else – amennyiben a hamis visszatérési értéket külön nem akarjuk kezelni – és a „hamisrész” kapcsolódó utasításai nem kötelezőek.

6. fejezet - Ugróutasítás

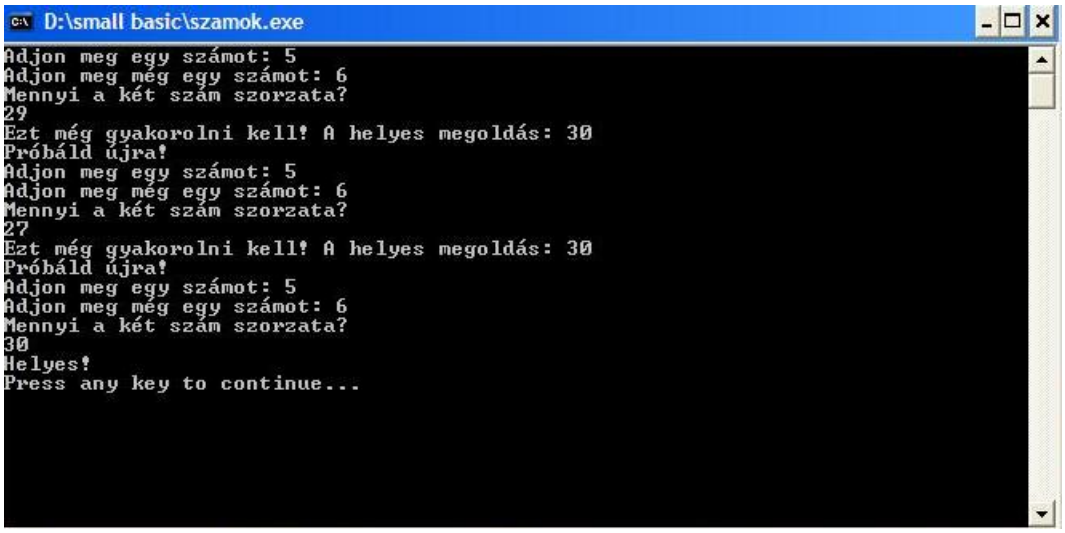
A ’80-as évek Basic programozói előszeretettel használták a *Goto* kulcsszóval (az angol „Menj ide:” kifejezésből) képzett utasításokat. A Goto a Small Basicnek is része, azonban az ilyen típusú ugróutasítás magasabb szinten inkább kerülendő. Számos programozási nyelv ma már nem is támogatja. Ellenzői szerint „pongyola” kódokhoz vezet. Tény, hogy rövid távú, kényelmi szempontok miatti használata jelentős részben rombolja egy programkód áttekinthetőségét, és emellett más programozástechnikai szabályokat is sérthet. Hibás használata pedig végtelen ciklusokhoz vezethet (a program ugyanazon utasításokon megy végig újra és újra, más szóval „lefagy”). Bár ez a szabványos ciklusutasítások helytelen használatkor is előfordulhat.

Tanulási fázisban azonban a Goto-t esetenként nyugodtan használhatjuk, (s átmenetileg a végtelen ciklusok tudatos használata is bocsánatos bűn).

Például a szorzási példánkat kiegészíthetjük úgy, hogy helytelen válasz esetén a felhasználónak újabb választ kelljen adnia.

```
start:
TextWindow.Write("Adjon meg egy számot: ")
szam1 = TextWindow.ReadNumber()
TextWindow.Write("Adjon meg még egy számot: ")
szam2 = TextWindow.ReadNumber()
TextWindow.WriteLine("Mennyi a két szám szorzata? ")
tipp = TextWindow.ReadNumber()
szorzat = szam1 * szam2
If (tipp = szorzat) Then
    TextWindow.WriteLine("Helyes!")
Else
    TextWindow.WriteLine("Ezt még gyakorolni kell! A helyes megoldás: " + szorza
TextWindow.WriteLine("Próbáld újra!")
Goto start>EndIf
```

A fenti programkódban két új elemre szeretnénk felhívni a figyelmet. A programba bárhová elhelyezhetünk ún. *címkéket*, amelyekre a későbbiekben vissza/átirányíthatjuk a programot. Esetünkben a „start:” címkéhez „Goto start” utasítással lépünk vissza, de csak akkor, ha a „Mennyi a két szám szorzata?” kérdésre a felhasználó rossz választ adott.



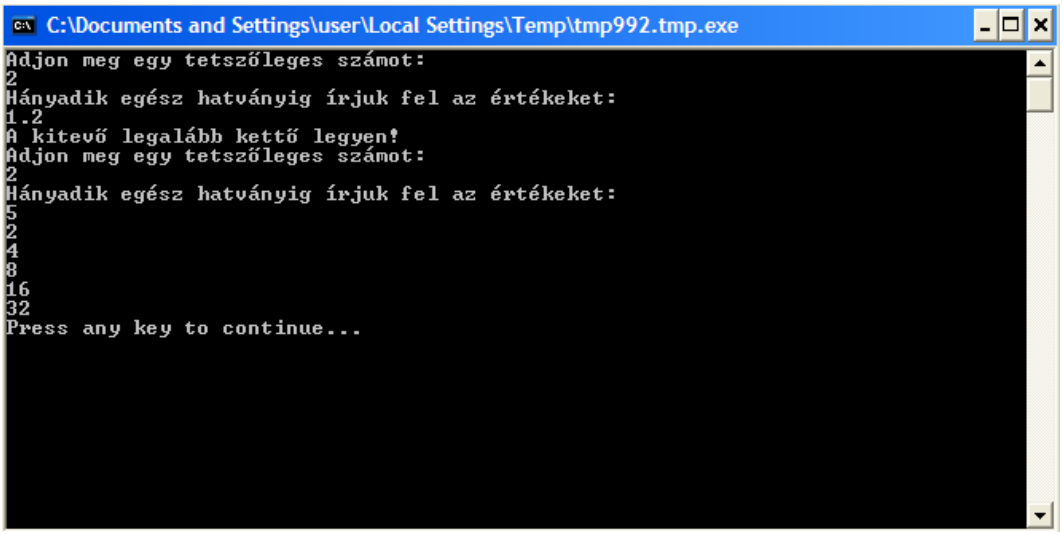
11. ábra: A Goto-val kiegészített szorzási példa futási eredménye több helytelen válasz esetén

Hatványprogram

Az alábbi program bekér a felhasználótól egy hatványalapot és egy hatványkitevőt, s az első hatványtól kezdve – kisebb (nem teljes) közbülső ellenőrzés után – a megadott hatványkitevőig kiírja a szám hatványait:

```
kezdes:
TextWindow.WriteLine("Adjon meg egy tetszőleges számot: ")
alap = TextWindow.ReadNumber()
TextWindow.WriteLine ("Hányadik egész hatványig írjuk fel az értékeket: ")
maxkitevo = TextWindow.ReadNumber()
korlat = maxkitevo
If (maxkitevo < 2) Then
    TextWindow.WriteLine("A kitevő legalább kettő legyen!")
    Goto kezdes
EndIf
kitevo = 0
kiiras:
kitevo = kitevo + 1
ertek = Math.Power(alap,kitevo)
TextWindow.WriteLine(ertek)
If (kitevo < korlat) Then
    Goto kiiras
EndIf
```

Az első, feltételes elágazásba ágyazott Goto kulcsszóval képzett utasítás a „kezdes” címkéhez lépteti vissza a programot, amennyiben a felhasználó által megadott kitevő kettőnél kisebb. (Nem lényeges feltétel, a Goto utasítás működésének szemléltetése végett került a programkódba.)



12. ábra: A hatványkiíró program egyik futási eredménye

A második Goto kulcsszóval képzett utasítás – miközben a „kitevo” változó értéke folyamatosan nő – egészen addig a „kiiras” címkéig lépteti vissza a programot, amíg a „kitevo” változó értéke meg nem haladja a „maxkitevo” változó értékét.

Felhívjuk még figyelmet a programban található *Math* (az angol „matematika” szóból) objektumra, amelyhez számos matematikai művelet tartozik. Esetünkben a Math objektum *Power(baseNumber, exponent)* – az angol „hatvány”, „hatványalap”, hatványkitevő” szavakból – műveletét hívtuk meg a megfelelő *argumentumok*/paraméterek helyén az általunk definiált változókat szerepeltetve.

7. fejezet - Ciklusok

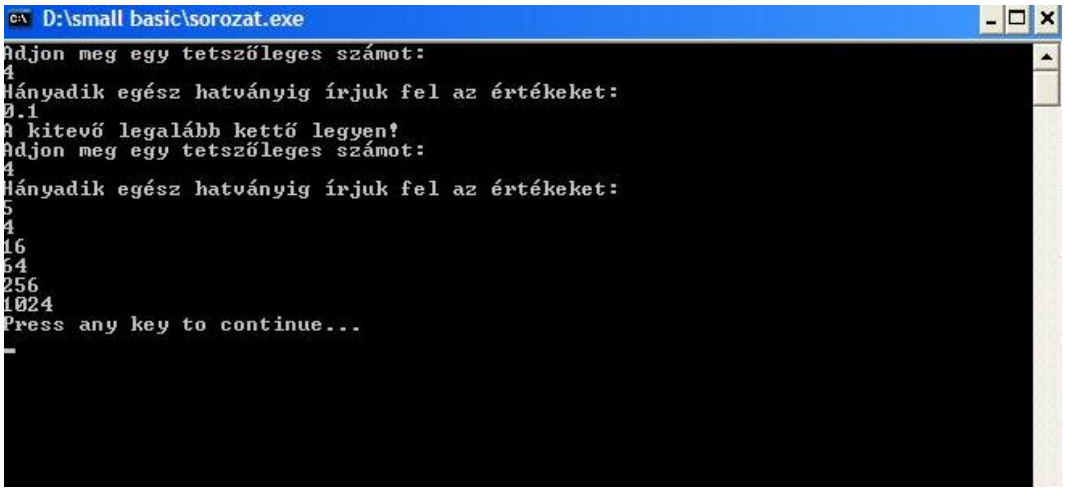
A ciklus vagy iteráció ismétlődő tevékenységek végrehajtására szolgál. Egy megadott műveletsort valamely darabszámszor (számlálós ciklus), vagy valamely feltétel teljesülése esetén, míg a feltétel igaz, újra és újra megismétlünk (elöl-, vagy hátultesztelő ciklus). Az elöltesztelő ciklus már a ciklus elején megvizsgálja az adott feltételt, és ha nem teljesül, nem hajtja végre a műveletsort. A hátultesztelő ciklus legalább egyszer lefut, a feltétel vizsgálata a műveletsor végrehajtása után történik meg.

Számlálós ciklus

Az előző fejezetben készített hatványkiíró program második Goto szerkezete „kiváltható” és egyszerűsíthető egy *ciklussal*:

```
kezdes:
TextWindow.WriteLine("Adjon meg egy tetszőleges számot: ")
alap = TextWindow.ReadNumber()
TextWindow.WriteLine("Hányadik egész hatványig írjuk fel az értékeket: ")
maxkitevo = TextWindow.ReadNumber()
If (maxkitevo < 2) Then
    TextWindow.WriteLine("A kitevő legalább kettő legyen!")
    Goto kezdes
EndIf
For kitevo = 1 To maxkitevo
    TextWindow.WriteLine(Math.Power(alap,kitevo))
EndFor
```

A fenti programkódban található ciklust *számlálós ciklusnak* hívjuk. A *For* (az angol „tól” -ból) kulcsszóval képzett utasítás után kezdőértéket adunk a *ciklusváltozónak* („kitevo = 1”).



13. ábra: A For ciklussal kiegészített hatványkiíró program egyik futási eredménye

A ciklus első futásakor a ciklusváltozó kezdőértéke mellett a program kiírja a megfelelő hatványt, azaz végrehajtja a *ciklusmagban* található utasításokat („TextWindow.WriteLine(Math.Power(alap,kitevo))”).

Ezután a *ciklusfej* első részében található ciklusváltozó értéke 1-gyel növeli az értékét egészen addig, amíg a To kulcsszó után megadott *végértéket* („maxkitevo”) el nem éri.

(Felhívjuk még a figyelmet a Math.Power művelet TextWindow.WriteLine műveletbe ágyazására.)

A kódot kiegészíthettük volna egy Step kulcsszóval képzett lépésközértékkel is:

```
For kitevo = 1 To maxkitevo Step 1
    TextWindow.WriteLine(Math.Power(alap,kitevo))
EndFor
```

Ha azonban a lépésköz egy, a Step elhagyható.

Elöltesztelő ciklus

A ciklusok (vagy iterációk) – mint már korábban említettük – a szekvenciával és a feltételes elágazással együtt képezik az általános programozási nyelvek „vázát”. E három alapelem segítségével – az adott programnyelv lehetőségeit kihasználva – lényegében „minden” programozási feladat megoldható.

A számlálós ciklus mellett még elöltesztelő és hátultesztelő ciklusokat különböztethetünk meg. A Small Basicben a számlálós ciklus mellett jellemzően elöltesztelő ciklusokat szoktunk használni. Nézzünk erre is egy példát!

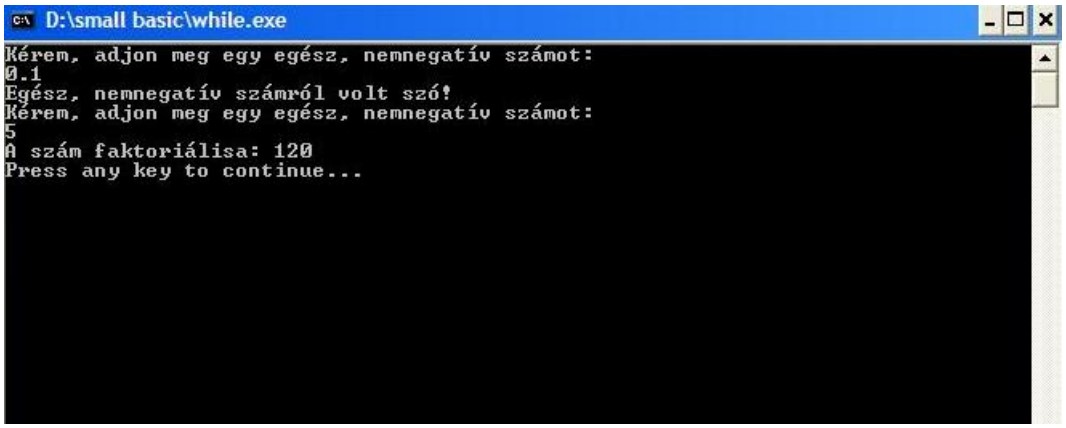
```
bekeres:
TextWindow.WriteLine("Kérem, adjon meg egy egész, nemnegatív számot:")
szam = TextWindow.ReadNumber()
If szam <> Math.Floor(szam) Or szam < 0 Then
    TextWindow.WriteLine("Egész, nemnegatív számról volt szó!")
    Goto bekeres
EndIf
szorzat = 1
While (szam > 1)
    szorzat = szorzat * szam
    szam = szam - 1
EndWhile
TextWindow.WriteLine("A szám faktoriálisa: " + szorzat)
```

A fenti program bekér a felhasználótól egy egész számot, majd kiszámolja annak faktoriálisát. (Az n nemnegatív, egész szám faktoriálisának az n-nél kisebb vagy egyenlő pozitív egész számok szorzatát nevezzük. Például 3! = 3*2*1 = 6, de 0! = 1). A program – mielőtt kiszámítaná a faktoriális értékét – megvizsgálja, hogy a felhasználó olyan számot adott-e meg, amelyből számolható faktoriális. Ha nem, akkor hibaüzenettel visszalép a szám bekéréséig.

A vizsgálat során felhasználtuk a Math objektum *Floor* (esetünkben: „egészrész”) műveletét. A „nem egyenlő” relációt pedig az „<>” *operátorral* fogalmaztuk meg.

Ha a szám megfelelő, a program 1-es kezdőértékkel definiálja a *szorzat* változót, majd továbblép az elöltesztelő ciklusunkra.

Az elöltesztelő ciklus végrehajtása során a program először megvizsgálja, hogy a ciklusfejben a *While* (az angol „amíg” szóból) után megfogalmazott feltétel („szam > 1”) teljesül-e. Ha igen, akkor a ciklusmagban lévő utasítások („szorzat = szorzat * szam illetve szam = szam – 1”) lefutnak, és a ciklus végrehajtása újból elkezdődik; ha nem, akkor a program a ciklus utáni sorban folytatódik, azaz a ciklusmagban lévő utasításokat az értelmezőprogram nem veszi figyelembe. Lehetséges, hogy az elöltesztelő ciklus egyszer sem fog lefutni, esetünkben ez n = 0 értéknél következik be. Tehát a program a nulla értéknél (melynek a faktoriálisa definíció szerint 1) is helyes eredménnyel fog visszatérni.)



14. ábra: A faktoriálist számító program egyik futási eredménye

8. fejezet - A grafikus ablak használata

A konzol – azaz a TextWindow ablak/objektum – használata meglehetősen korlátozott grafikai lehetőséget nyújt a felhasználónak, vizuálisan is esztétikus programok készítésére nem alkalmas.

A Small Basic azonban rendelkezik egy *GraphicsWindow* (az angol „grafika” és „ablak” szavakból) nevű grafikus ablakkal/objektummal is, amely már sokkal látványosabb alkalmazások írását teszi lehetővé.

Ha a lenti programkódot lefuttatjuk, a 15. ábrán található, üres, grafikus ablakhoz jutunk:

```
GraphicsWindow.Show()
```

Természetesen a grafikus ablak tulajdonságai megváltoztathatók.

```
GraphicsWindow.BackgroundColor = "Red"
GraphicsWindow.Title = "Grafikus ablakom"
GraphicsWindow.Width = 400
GraphicsWindow.Height = 200
GraphicsWindow.CanResize = "True"
GraphicsWindow.Show()
```

A fenti programkódban a *BackgroundColor, Title, Width, Height* és *CanResize* (rendre: „háttérszín”, „cím”, „szélesség”, „magasság”, „átméretezhetőség”) tulajdonságok értékeit állítottuk be, majd a Show() („mutat”) művelettel megjelenítettük az ablakot. A CanResize tulajdonság értékét „True”-ra („igaz”) állítottuk be, ezért a megjelenített ablak mérete megváltoztatható. Ha az érték „False” lenne, az ablakot nem lehetne a futás alatt árméretezni (a 16. ábrán ezért hiányoznak az átméretezést segítő ikonok).

15. ábra: A GraphicsWindow üres ablaka

16. ábra: Formázott grafikus ablak

Az egyes színek – esetünkben a háttérszín – beállításakor nemcsak a korábban felsorolt alapszínekből válogathatunk, hanem hexadecimális RGB-értékeket is felvehetünk. Az RGB-értékeknél, egyenként két-két hexadecimális (16-os számrendszerben írt) számmal határozzuk meg a Red („vörös”), a Green („zöld”) és a Blue („kék”) értékét.

A

```
GraphicsWindow.BackgroundColor = "Red"
```

utasítás helyett tehát a

```
GraphicsWindow.BackgroundColor = "#FF0000"
```

utasítást is használhattuk volna.

Grafikus elemek megjelenítése

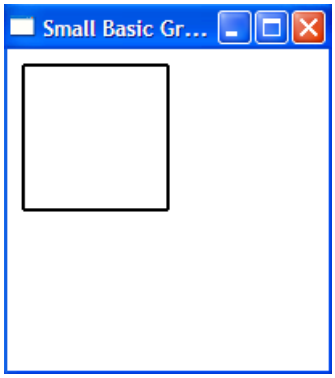
A grafikus ablak természetesen azért grafikus ablak, hogy a program futása közben szövegek mellett különböző grafikus elemeket is megjeleníthessünk.

A következő program erre ad egyszerű példát:

```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.DrawLine(10, 10, 10, 100)
GraphicsWindow.DrawLine(10, 10, 100, 10)
GraphicsWindow.DrawLine(10, 100, 100, 100)
GraphicsWindow.DrawLine(100, 10, 100, 100)
```

A program a GraphicsWindow objektum *DrawLine(x1, y1, x2, y2)* – „vonalhúzás” – művelete segítségével egy négyzetet rajzolt a grafikus ablakba. A négyzet négy vonal megrajzolásából állt össze. A DrawLine művelethez négy inputot kellett megadnunk: a vonal kiindulópontjának x és y koordinátáját, illetve végpontjának x és y koordinátáját. A koordinátarendszer origója azonban a matematikai tanulmányok során megszokottól eltérően a bal felső sarokban helyezkedik el! Tehát az x tengely

értékei az origóból kiindulva, „jobbra haladva” nőnek (ebben nincs eltérés a matematikához képest), viszont az y tengely értékei az origóból kiindulva, „lefelé haladva” növelik az értéküket.

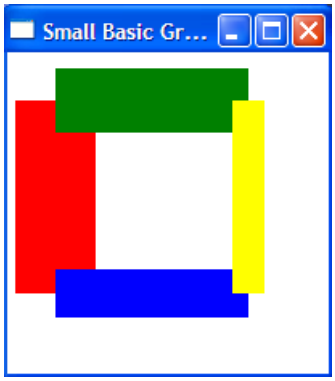


17. ábra: Grafikus ablak négyzettel

Színek, vonalszélesség, négyzetrajzolás

A következő program a *PenColor* („tintaszín”) és a *PenWidth* („vonalszélesség”) tulajdonságok megadásával és a DrawLine művelet inputjainak módosításával kissé más megjelenésű négyzetet rajzol a grafikus ablakba. (18. ábra)

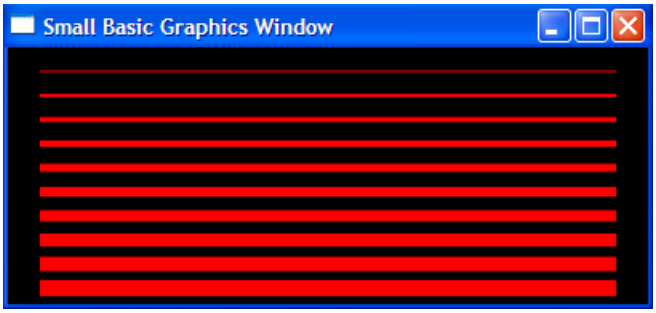
```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.PenColor = "Red"
GraphicsWindow.PenWidth = "50"
GraphicsWindow.DrawLine(30, 30, 30, 150)
GraphicsWindow.PenColor = "Green"
GraphicsWindow.PenWidth = "40"
GraphicsWindow.DrawLine(30, 30, 150, 30)
GraphicsWindow.PenColor = "Blue"
GraphicsWindow.PenWidth = "30"
GraphicsWindow.DrawLine(30, 150, 150, 150)
GraphicsWindow.PenColor = "Yellow"
GraphicsWindow.PenWidth = "20"
GraphicsWindow.DrawLine(150, 30, 150, 150)
```



18. ábra: A módosított négyzet

A következő program egy számlálós ciklus segítségével tölti fel növekvő vastagságú vonalakkal a grafikus ablakot:

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.Width = 400
GraphicsWindow.Height = 160
GraphicsWindow.PenColor = "Red"
For i = 1 To 10
    GraphicsWindow.PenWidth = i
    GraphicsWindow.DrawLine(20, i * 15, 380, i * 15)
EndFor
```



19. ábra: Vonalrajzolás számlálós ciklussal

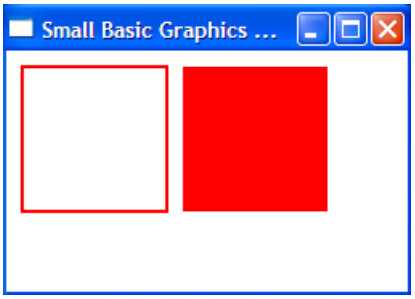
Kitöltés

A korábbi négyzetünk egyszerűbb módon is elkészíthető, s a felrajzolt síkidom színnel kitöltött változata is könnyen felrajzolható:

```
GraphicsWindow.Width = 250
GraphicsWindow.Height = 150
GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawRectangle(10, 10, 90, 90)
GraphicsWindow.BrushColor = "Red"
GraphicsWindow.FillRectangle(110, 10, 90, 90)
```

A kitöltés nélküli négyzetet a *DrawRectangle (x, y, width, height)* – „Rajzolj téglalapot!”, „szélesség”, „magasság” – művelettel készítettük el. (Természetesen négyzet helyett általános téglalapot is rajzolhattunk volna.) A DrawRectangle művelet inputjának első két attribútuma a téglalap (négyzet) bal felső sarkának (kezdőpontjának) x és y koordinátáját, a harmadik a szélességét, a negyedik a magasságát határozza meg.

A *FillRectangle(x, y, width, height)* – „Tölts ki színnel egy téglalapot!”, x, y, „szélesség”, „magasság” – művelet attribútumszerkezete azonos a DrawRectangle műveletével. A kitöltési színt a *BrushColor* tulajdonság határozza meg.

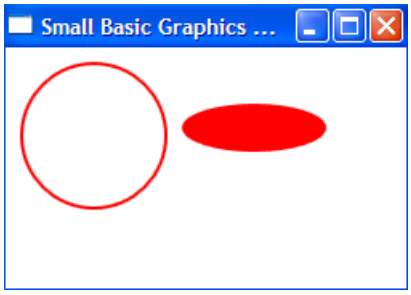


20. ábra: Négyzetrajzolás és -kitöltés

Az előzőekhez nagyon hasonló módon kört és ellipszist is rajzolhatunk, illetve kitölthetünk színnel:

```
GraphicsWindow.Width = 250
GraphicsWindow.Height = 150
GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawEllipse(10, 10, 90, 90)
GraphicsWindow.BrushColor = "Red"
GraphicsWindow.FillEllipse(110, 35, 90, 30)
```

A *DrawElipse* és a *FillEllipse* műveletek inputjának attribútumai rendre megfeleltethetők a DrawRectangle és a FillRectangle műveletekéivel .



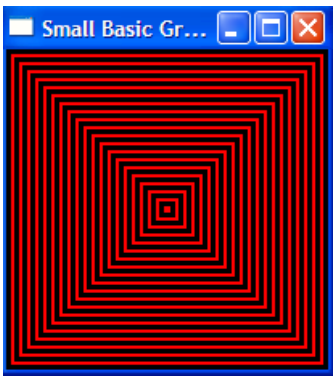
21. ábra: Ellipszisrajzolás és –kitöltés

Néhány grafikai alkalmazás

A következőkben néhány egyszerű elemekre épülő, de mégis látványos, tisztán grafikai elemekre épülő alkalmazás kódját közöljük.

```
'Op-art-négyzetek
'Az alábbi program négyzetekre épülő op-art-alkalmazást készít
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "Red"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
For i = 1 To 100 Step 5
    GraphicsWindow.DrawRectangle(100 - i, 100 - i, i * 2, i * 2)
EndFor
```

Az alkalmazás „lelke” egy számlálós ciklus, tanulmányozzuk a működését!



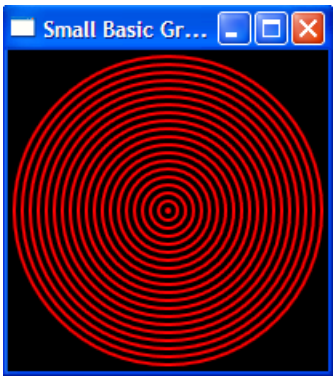
22. ábra: Op-art négyzetek

Az „Op-art négyzetek” alkalmazásban új elem, hogy a programkódba megjegyzéseket illesztettünk egy felső aposztróf (’) segítségével. Különösen összetettebb programkódjainkat érdemes mindig a megfelelő számú megjegyzéssel ellátni. A megjegyzések növelik a programkód olvashatóságát, ezzel későbbi munkánkat, a csoportmunkát, és programkód átadását is megkönnyítik. Kezdő programozók minél több megjegyzéssel lássák el a programkódot!

A felső aposztróffal bizonytalan, vagy egyelőre nem alkalmazandó kódrészleteket is megjegyzéssé alakíthatunk, ez programírás közben nagyon hasznos lehet.

Az „Op-art körök” alkalmazás felépítése nagyon hasonlít az előzőhöz.

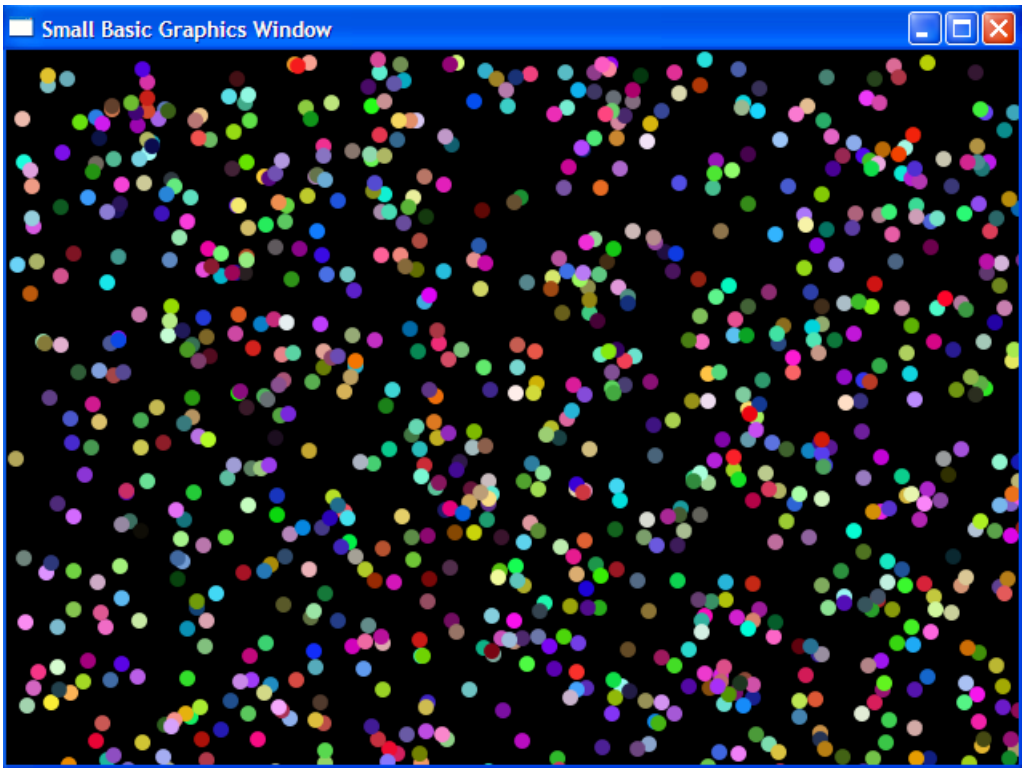
```
'Op-art körök
'Az alábbi program körökre épülő op-art alkalmazást készít
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "Red"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
For i = 1 To 100 Step 5
    GraphicsWindow.DrawEllipse(100 - i, 100 - i, i * 2, i * 2)
EndFor
```



23. ábra: Op-art körök

```
'Véletlenszerű körök
GraphicsWindow.BackgroundColor = "Black"
For i = 1 To 1000
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
    x = Math.GetRandomNumber(640)
    y = Math.GetRandomNumber(480)
    GraphicsWindow.FillEllipse(x, y, 10, 10)
EndFor
```

„A véletlenszerű körök” alkalmazás a GraphicsWindow *GetRandomColor()* műveletének segítségével a ciklus minden futása során véletlen színértéket rendel a GraphicsWindow BrushColor tulajdonságához. A Math objektum *GetRandomNumber(maxNumber)* – „véletlenszám” és „maximális értéke” – művelete segítségével előállított véletlenszámok maximális értékei igazodnak az ablak méreteihez. A ciklus utolsó futása után ezer véletlenszerűen elhelyezkedő és színű köröcske lesz látható a grafikus ablakban.



24. ábra: Véletlenszerű körök

Szövegmegjelenítés

A grafikus ablak természetesen szövegek megjelenítésére is alkalmas, így megfelelő algoritmusok és programozási eljárások segítségével valóban összetett alkalmazások készíthetők.

Egy egyszerű példa a szövegmegjelenítésre:

```
'Szöveg megjelenítése a grafikus ablakban
GraphicsWindow.Show()
GraphicsWindow.Width = 270
GraphicsWindow.Height = 150
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.FontSize = 36
GraphicsWindow.FontBold = "True"
GraphicsWindow.FontItalic = "True"
GraphicsWindow.BrushColor = "Red"
GraphicsWindow.DrawText(20, 40, "Üdvözöljük!")
```

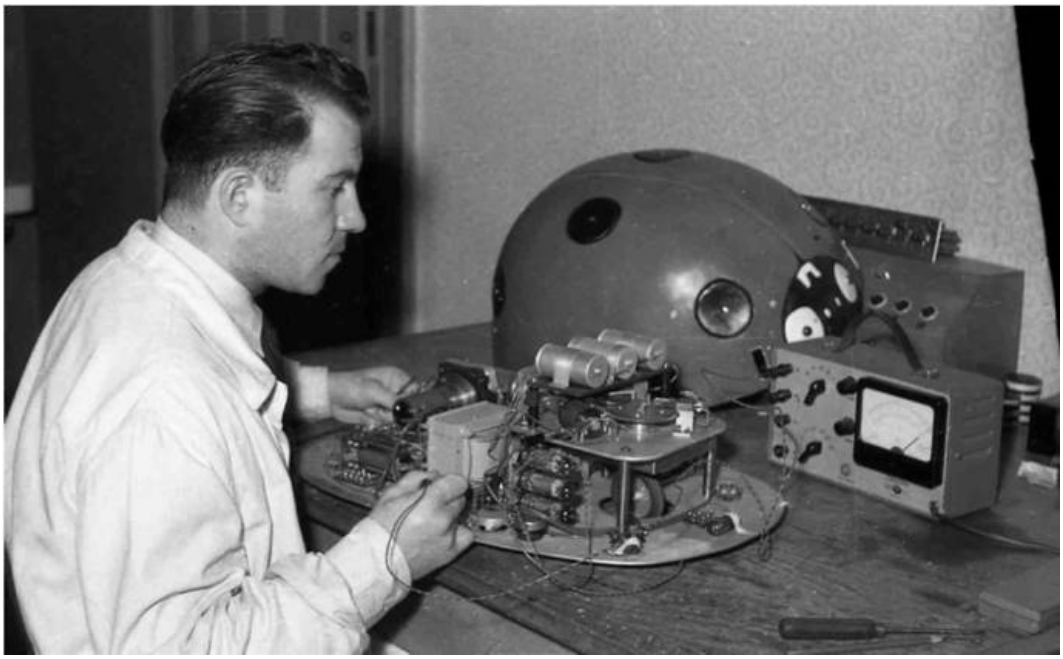
A GraphicsWindow objektum *FontSize* („betűméret”), *FontBold* („félkövér betű”) és *FontItalic* („dőlt betű”) – ez utóbbi kettő „True” vagy „False” értéket vehet fel –, illetve BrushColor tulajdonságaival beállítottuk a szöveg tulajdonságait, a *DrawText(x, y, text)* – „szöveg kiírása”, x, y, „szöveg” – művelettel kiírattuk a kívánt szöveget.

25. ábra: Szövegkiírás a grafikus ablakban

9. fejezet - Teknőcgrafika

A kezdetek

A ’70-es és ’80-as években – de lényegében ma is – a kezdő programozók körében különösen népszerű volt a LOGO programozási nyelv ún. *teknőc*e (*Turtle*), amely többé-kevésbé egy apró teknősre hasonlító objektumként parancsokra vagy utasításokra mozgott a képernyőn, és közben különböző műveleteket végzett. Egy magyar programozónak talán illik tudnia, hogy a teknőc „őse” és ihletője a Kalmár László felügyelete alatt Muszka Dániel által a szegedi egyetemen 1957-ben megépített kibernetikus katicabogár (egy mechanikus gép) volt.



26. ábra: Muszka Dániel a kibernetikus katicabogárral (forrás: Sulinet)

A teknőc használata

Bár az általános célú programozási nyelvek döntő többsége ilyen elemet nem használ, érdemes vele megismerkedni, mert a programozást kicsit közelebb hozza a kezdő programozóhoz. A teknőc egyszerű utasítással megjeleníthető a képernyő közepén:

```
Turtle.Show()
```

A megjelenő teknőc (27. ábra) számos művelet elvégzésére használható.

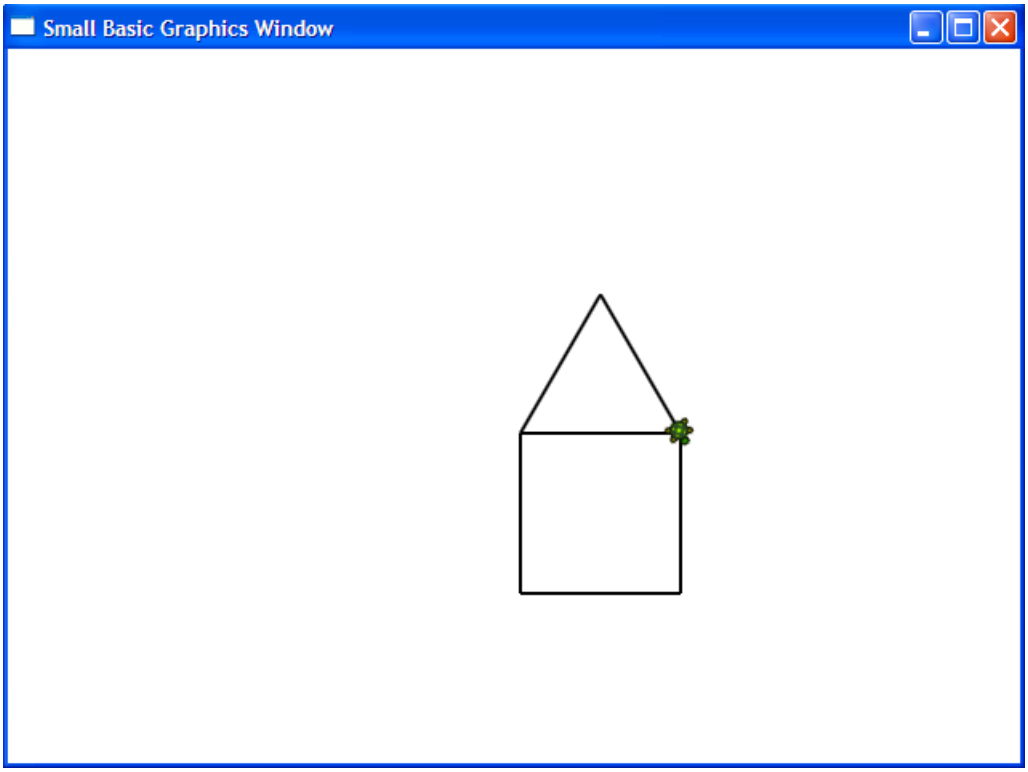
27. ábra: A teknőc

A teknőc használható egyszerűbb ábrák rajzolására:

```
'Házikó rajzolása
Turtle.Speed = 7
For i=1 To 4
    Turtle.TurnRight()
    Turtle.Move(100)
EndFor
Turtle.Turn(30)
Turtle.Move(100)
Turtle.Turn(120)
Turtle.Move(100)
```

A programkód első felében egy számlálós ciklus segítségével négyszer elvégezzük a következő műveletet: A Turtle objektum *TurnRight()* – „jobbra fordul” – műveletével a teknőcöt jobbra fordítjuk , majd a *Move(distance)* – „mozog”, „távolság”) – művelettel egy 100 képpont hosszú egyenest rajzoltatunk.

A „tető” megrajzolása során a Turtle objektum *Turn(angle)* – „fordul”, „szög” – műveletével a teknőcöt mozgás előtt 30, majd 120 fokkal fordítjuk el az épp aktuális helyzetéhez képest.



28. ábra: „Házikót” rajzoló teknőc

Összetettebb ábrák rajzolása teknőccel

Természetesen a teknőc összetettebb ábrák rajzolására is használható. A következő kód programozási szempontból is érdekes:

```
'A teknőc körei
lepesek = 50
hossz = 400 / lepesek
szog = 360 / lepesek
Turtle.Speed = 9
For j = 1 To 20
    For i = 1 To lepesek
        Turtle.Move(hossz)
        Turtle.Turn(szog)
    EndFor
    Turtle.Turn(18)
EndFor
```

A bevezetett változóinkat felhasználva a 29. ábrán látható alakzatot a teknőc *egymásba ágyazott ciklusok* segítségével rajzolta fel.

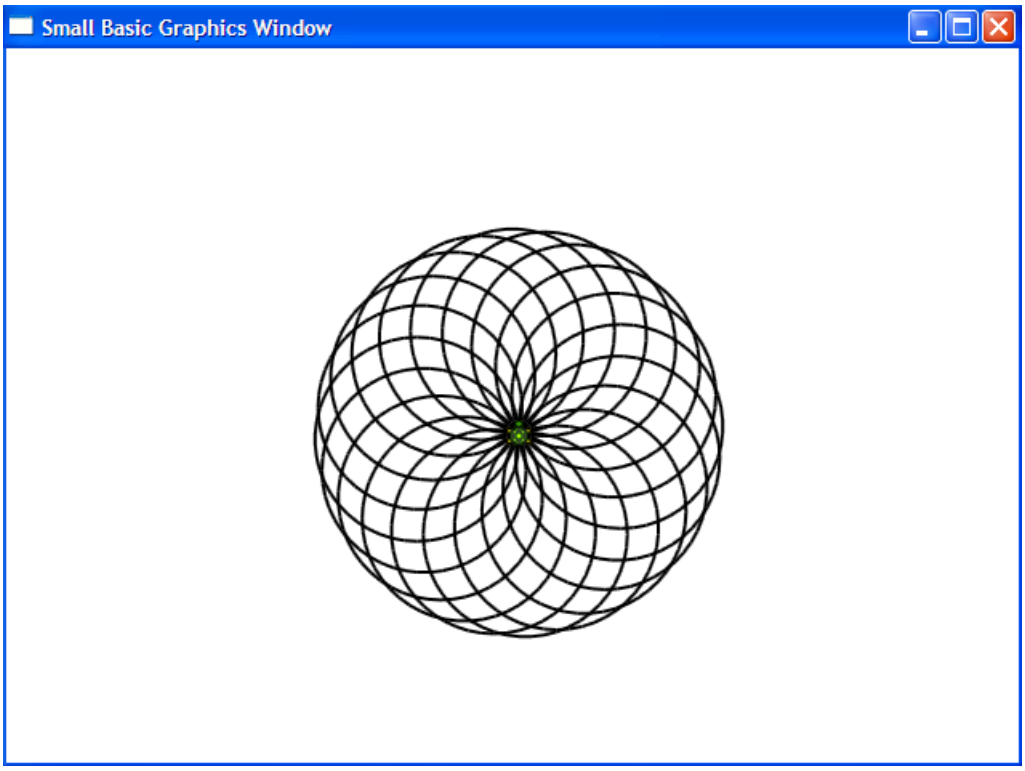
A kör ötven apró lépésre bontott – a teknőc minden lépés során egy apró fordul – felrajzolásáról a belső ciklus gondoskodik:

```
(külső ciklus feje)
For i = 1 To lepesek
    Turtle.Move(hossz)
```

```
    Turtle.Turn(szog)
EndFor
(külső ciklus magjának második fele és lezárása)
```

A külső ciklus összesen hússzor futtatja végig a belső ciklust úgy, hogy minden körfelrajzolás után 18 fokkal elfordítja a teknőcöt:

```
For j = 1 To 20
    (belső ciklus)
    Turtle.Turn(18)
EndFor
```



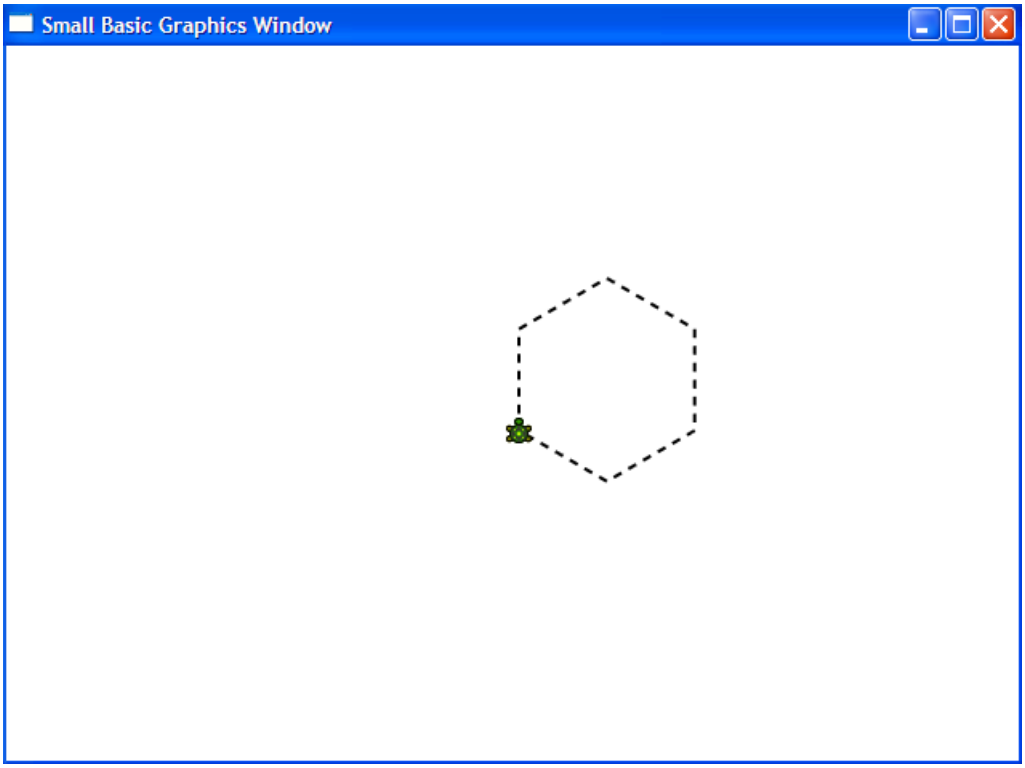
29. ábra: „A teknőc körei”

A teknőc természetesen úgy is tud mozogni, hogy eközben nem rajzol. A következő alkalmazás ezt használja fel:

```
'Hatszög szaggatott vonallal
lepesek = 6
hossz = 400 / lepesek
szog = 360 / lepesek
For i = 1 To lepesek
    For j = 1 To 6
        Turtle.Move(hossz / 12)
        Turtle.PenUp()
        Turtle.Move(szog / 12)
        Turtle.PenDown()
    EndFor
    Turtle.Turn(szog)
```

```
EndFor
```

A fenti program szerkezete sok hasonlóságot mutat az előzővel. Itt is ciklusok egymásba ágyazásával oldottuk meg a feladatot. A vonal szaggatott jellegét a Turtle objektum *PenUp()* („tollat felemel”) és *PenDown()* („tollat lenyom”) műveletével biztosítottuk.



30. ábra: Hatszög felrajzolása szaggatott vonallal

10. fejezet - Szubrutinok

Összetettebb programokat érdemesebb kisebb programrészekre, programszegmensekre, ún. *szubrutinokra* bontani. A szubrutinok használatával áttekinthetőbbé és rövidebbé tehetjük a programkódunkat. Az összetett feladatok részfeladatokra bontása általában a programírást is megkönnyíti. A szubrutin egy programban több helyről is behívható a főprogramba, használatával a felesleges ismétlődések kiküszöbölhetők.

A legtöbb professzionális célokra kialakított programozási nyelvtől eltérően a Small Basic szubrutinjai nem fogadnak paramétereket és nem adnak visszatérési értéket, tehát nem tekinthetők függvénynek.

Faktoriális kiszámítása szubrutinnal

A következő példa korábbi, faktoriálist kiszámító példánk átalakított változata:

```
bekeres:
TextWindow.WriteLine("Kérem, adjon meg egy egész, nemnegatív számot:")
szam = TextWindow.ReadNumber()
If szam <> Math.Floor(szam) Or szam < 0 Then
    TextWindow.WriteLine("Egész, nemnegatív számról volt szó!")
    Goto bekeres
EndIf
Fakt() 'Faktoriálist számító szubrutin behívása
TextWindow.WriteLine("A szám faktoriálisa: " + szorzat)

Sub Fakt 'Faktoriálist számító szubrutin
szorzat = 1
While (szam > 1)
    szorzat = szorzat * szam
    szam = szam - 1
EndWhile
EndSub
```

A *Fakt* szubrutint a *Sub* kulcsszó segítségével definiáltuk, a szubrutinhoz tartozó kódrészletet az *EndSub* kulcsszóval zártuk le. A szubrutint a programból a Fakt() utasítással (művelettel) hívtuk be. A program példát ad arra is, hogy a megjegyzéseket nemcsak önálló sorokba, hanem az utasításaink után is beírhatjuk.

11. fejezet - Tömbök

A számítástechnikában a tömb olyan adatszerkezet, amelyben minden egyes adatra azok sorszámával, indexével lehet hivatkozni. Az egydimenziós tömböket vektoroknak, a többdimenziós tömböket mátrixoknak is nevezhetjük.

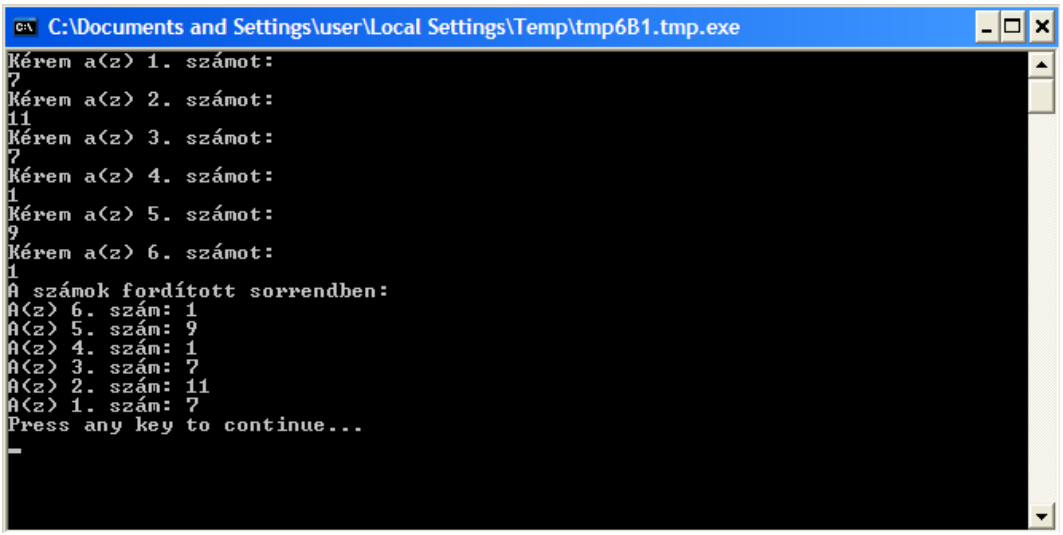
Fordított sorrend tömbbel

A tömbök használata nélkül nagyobb adatállományok, adatszerkezetek kezelése szinte lehetetlen. A következő, egyszerű példa bekér hat számot, majd fordított sorrendben kiírja azokat:

```
'Fordított sorrendbe állítás
For i = 1 To 6
    TextWindow.WriteLine("Kérem a(z) " + i + ". számot:")
    szam[i] = TextWindow.ReadNumber()
EndFor
TextWindow.WriteLine("A számok fordított sorrendben:")
For i = 6 To 1 Step -1
    TextWindow.WriteLine("A(z) " + i + ". szám: " + szam[i])
EndFor
```

A bekért öt darab számot a program a szam[i] tömbben tárolta. Tehát először a szam[1], majd a szam[2], szam[3], szam[4], szam[5] változóknak adtunk értéket. Mindezt az első számlálós ciklusunk biztosította.

A fordított sorrendben történő kiíráskor azt használtuk ki, hogy a bekért változóértékeket a program a tömbben sorszámmal ellátva (indexelve) tárolta, így a második ciklusunk segítségével a sorszámra hivatkozva a fordított sorrendben történő kiírás lehetővé vált.



31. ábra: Számok bekérése majd kiírása fordított sorrendben

Indexelés szöveggel

Az előző programban a tömböt számokkal indexeltük, azonban az indexelés nemcsak számokkal történhet:

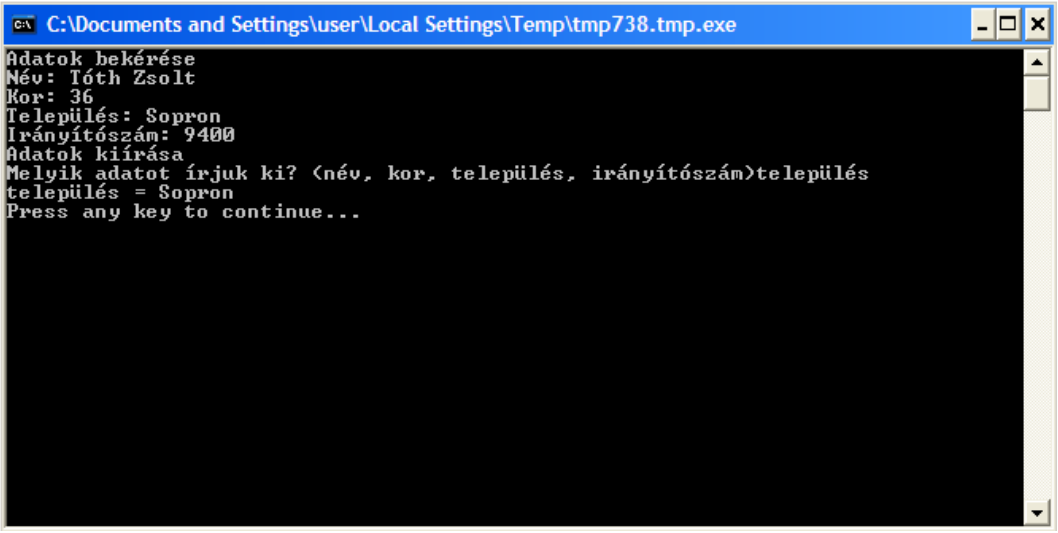
```
'Indexelés szöveggel
```

```
Adatbekérés()
Adatkiírás()

Sub Adatbekérés
TextWindow.WriteLine("Adatok bekérése")
TextWindow.Write("Név: ")
felh["név"] = TextWindow.Read()
TextWindow.Write("Kor: ")
felh["kor"] = TextWindow.ReadNumber()
TextWindow.Write("Település: ")
felh["település"] = TextWindow.Read()
TextWindow.Write("Irányítószám: ")
felh["irányítószám"] = TextWindow.Read()
EndSub

Sub Adatkiírás
TextWindow.WriteLine("Adatok kiírása")
TextWindow.Write("Melyik adatot írjuk ki? (név, kor, település, irányítószám)")
index = TextWindow.Read()
TextWindow.WriteLine(index + " = " + felh[index])
EndSub
```

A fenti program két szubrutinra oszlik, az egyik bekér a felhasználótól néhány adatot, és az indexelést már szövegek segítségével elvégezve, eltárolja azokat egy tömbben, a másik pedig a felhasználó választása alapján kiír egy értéket a tömbből. A programban a TextWindow objektum *Write(data)* („ír”, „adat”) műveletét is felhasználtuk. A Write a WriteLine-tól eltérően nem kezd új sort a kiírás után.

Az irányítószámot azért kértük be szövegként (Read()) és nem számként (ReadNumber()), mert valójában számjegyekből álló szövegről van szó, matematikai műveleteket irányítószámokkal nem végezhetünk.

A példaprogram során célzottan mindenhol – nemcsak a feliratoknál – magyar ékezetes betűket használtunk, hogy a hibás adatmegadás (a kiíratás során) ne vezessen felesleges hibához.



32. ábra: Szöveggel indexelt tömb példa futási eredménye

Többdimenziós tömbök

Az előző program viszonylag könnyen átalakítható több felhasználó adatait bekérő és kiíró programmá:

```
'Kéttdimenziós tömb
TextWindow.WriteLine("Felhasználók száma?")
szam = TextWindow.ReadNumber()

Adatbekérés()
Adatkiírás()

Sub Adatbekérés
For i=1 To szam
    TextWindow.WriteLine(i + ". felhasználó adatainak bekérése")
    TextWindow.Write("Név: ")
    felh[i]["név"] = TextWindow.Read()
    TextWindow.Write("Kor: ")
    felh[i]["kor"] = TextWindow.ReadNumber()
    TextWindow.Write("Település: ")
    felh[i]["település"] = TextWindow.Read()
    TextWindow.Write("Irányítószám: ")
    felh[i]["irányítószám"] = TextWindow.Read()
EndFor
EndSub

Sub Adatkiírás
TextWindow.WriteLine("Adatok kiírása")
TextWindow.Write("Melyik felhasználó adatát írjuk ki? (1, 2, ... n)")
index1 = TextWindow.Read()
TextWindow.Write("Melyik adatot írjuk ki? (név, kor, település, irányítószám)")
index2 = TextWindow.Read()
TextWindow.WriteLine(index1 + ". fehasználó " + index2 + " adata = " + felh[inde
EndSub
```

A fenti program a bekért adatokat egy kétdimenziós mátrixban tárolja, az alábbi séma szerint:

	Név	*Kor*	*Település*	*Irányítószám*
1. felh.	[1][név] = Pisti	[1][kor] = 25	[1][település] = Budapest	[1][irányítószám] = 1134
2. felh.	[2][név] = Juli	[2][kor] = 29	[2][település] = Sopron	[2][irányítószám] = 9400
…	…	…	…	…
n. felh.	[n][név] = Ági	[n][kor] = 18	[n][település] = Győr	[n][irányítószám] = 9025

1. táblázat: Kétdimenziós tömb adatszerkezete

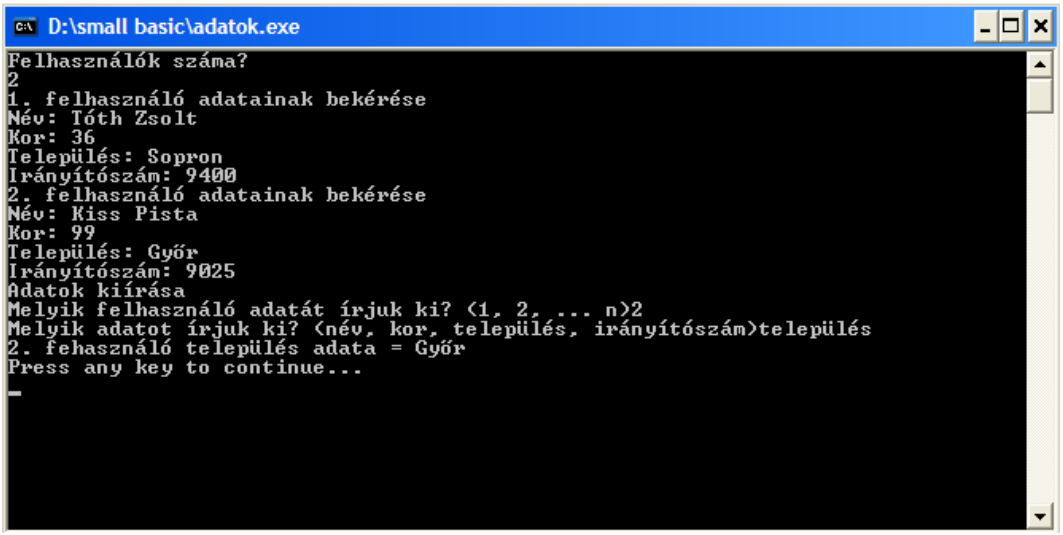
Kétdimenziós tömb adatszerkezete]

Talán érdemes megjegyezni, hogy a tömbök 3,4,5 vagy akár több dimenziósak is lehetnek, még ha három dimenziónál többet a valós térben nehezen is tudunk elképzelni.

A fenti program sok hibalehetőséget rejt magában (ez részben a korábbi programokra is igaz), hiszen rossz adatbevitellel a felhasználó helytelen eredményhez jut. Az ilyen problémákat valamilyen úton-módon professzionális szinten kezelni kell.

Az is jól látható korlát (ezernyi más probléma mellett), hogy az adatokat a program csak futási időben tárolja. „Élesben” ez nem lenne kielégítő, de az ilyen típusú problémák átvezetnek bennünket az adatbázis-kezelés témakörébe.

Talán ennél a példánál érdemes hangsúlyozni, hogy a jegyzetben lévő alkalmazások alapvetően apró példaprogramok, az egyes felmerülő problémák teljes körű tisztázása gyakran későbbi stúdiumokra marad.



33. ábra: A névbekérő-kiíró programunk egyik futási eredménye

Tömbre épülő grafikus alkalmazás

A tömbök alkalmazásának egy grafikus példája:

```
'Kiúszó körök
sorok = 6
oszlopok = 6
meret = 40
For r = 1 To sorok
    For c = 1 To oszlopok
        GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
        kor[r][c] = Shapes.AddEllipse(meret, meret)
        Shapes.Move(kor[r][c], c * meret, r * meret)
    EndFor
EndFor
For r = 1 To sorok
    For c = 1 To oszlopok
        Shapes.Animate(kor[r][c], 0, 0, 1000)
        Program.Delay(200)
    EndFor
EndFor
```

Érdemes a fenti programot az eddigiek alapján önállóan kielemezni. Találunk benne a köröket 6 sorba és 6 oszlopba felíró, majd onnan egyenként a bal felső sarokba „visszamozgató”, egymásba ágyazott ciklusokat, s számos új elemet.

A *Shapes* („alakzatok”) objektum egyes műveletei segítségével különböző alakzatokat tudunk megjeleníteni, mozgatni, elforgatni.

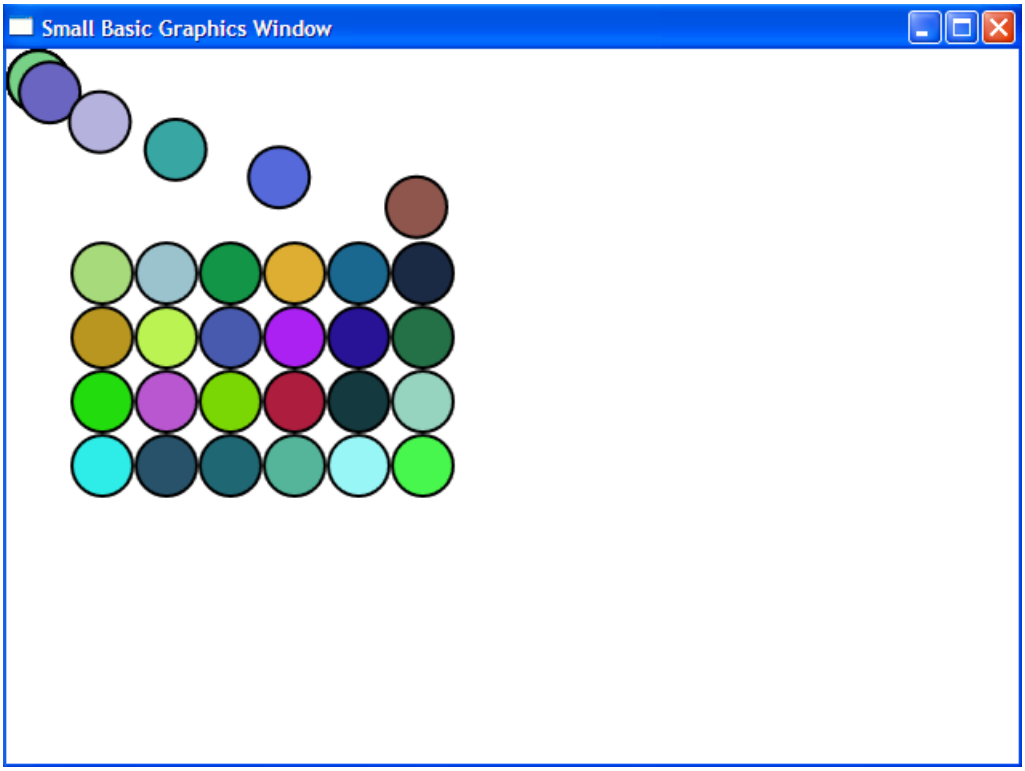
Esetünkben a *Shapes.AddEllipse(width, height)* – „ellipszis (kör) hozzáadása”, „szélesség”, „magasság” – művelet segítségével 48 darab kis színes körrel töltjük fel a „kor[r][c]” kétdimenziós tömböt. A Shapes.AddEllipse(meret, meret) művelet a ciklusok lefutása során a bal felső sarokban el is helyezi

a 48 darab kört, amelyek egymás „fölött” természetesen csak egynek látszanának a további utasítások nélkül.

Az alakzatok pozícióba rendezéséről a *Move(shapeName, x, y)* művelet – „mozgat”, „alakzat neve”, az új x és y koordináták – gondoskodik. A köröket hatszor hatos táblába rendező Shapes.Move(nkor[r][c], c * meret, r * meret) műveletnél érdemes tudatosítani, hogy a tömbben tárolt grafikus elemek kerülnek az adott pozícióba.

A *Shapes.Animate(shapeName, x, y, duration)* – „animált mozgatás” , „alakzat”, új x és y koordináta, „időtartam” (milliszekundumban) – adott időtartam alatt új pozícióba mozgat egy elemet. Esetünkben a „Shapes.Animate(negyzet[r][c], 0, 0, 1000)” ciklusok segítségével, egymás után a bal felső sarokba mozgatja vissza a köröket, egyenként 1000 milliszekundum alatt.

A Program objektum *Delay(milliSeconds)* – „késleltetés”, „milliszekundum” – művelete a megadott milliszekundumokkal késlelteti a következő sorra ugrást, azaz a program futását. Esetünkben a 200 milliszekundumnyi késleltetés nélkül a körök vizuálisan szinte egyszerre (egymás után, de szemmel követhetetlen időeltéréssel) úsznának ki a bal felső sarokba.



34. ábra: Pillanatfelvétel a „Kiúszó körök” program futásáról

12. fejezet - Események és vezérlők

Az alábbi „Teknőcrajzoló” program az *események* és a *vezérlők* használatára ad példát:

```
'Teknőcrajzoló
GraphicsWindow.Title = "Teknőcrajzoló"
GraphicsWindow.ShowMessage("Szia! Kellemes rajzolást!", "Köszöntés")

gomb[1] = Controls.AddButton("Balra", 10, 230)
gomb[2] = Controls.AddButton("Jobbra", 565, 230)
gomb[3] = Controls.AddButton("Menj!", 290, 10)
gomb[4] = Controls.AddButton("Rajzol", 10, 10)
gomb[5] = Controls.AddButton("Nem rajzol", 540, 10)
Turtle.Show()
Controls.ButtonClicked = Kattintas

Sub Kattintas
    n = Controls.LastClickedButton
    If n = gomb [1] Then
        Turtle.Turn(-10)
    EndIf
    If n = gomb [2] Then
        Turtle.Turn(10)
    EndIf
    If n = gomb [3] Then
        Turtle.Move(5)
    EndIf
    If n = gomb [4] Then
        Turtle.PenDown()
    EndIf
    If n = gomb [5] Then
        Turtle.PenUp()
    EndIf
EndSub
```

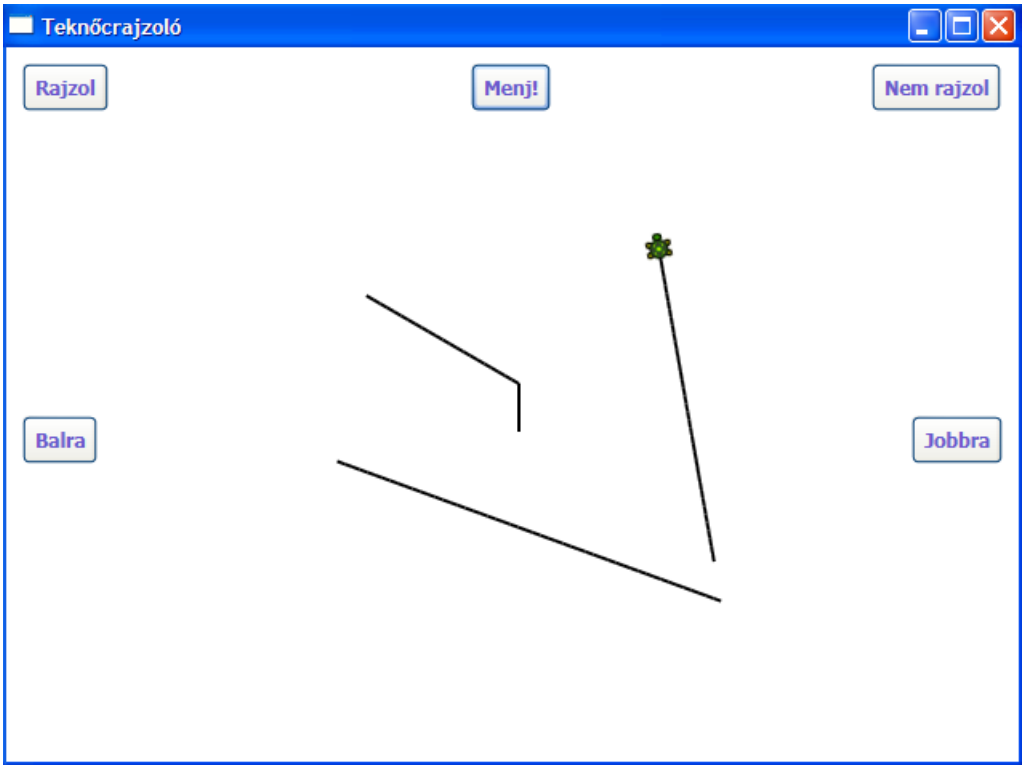
A program az első fázisban beállítja a grafikus ablak fejlécében lévő címet (GraphicsWindow.Title = "Teknőcrajzoló"), majd a GraphicsWindow.ShowMessage("Szia! Kellemes rajzolást!", "Köszöntés") sorral köszönti a felhasználót.

A GraphicsWindow *ShowMessage(text, title)* – „Üzenet megjelenítése”, „szöveg”, „cím” – művelet segítségével egy *szövegdobozt* jelenítünk meg.

A szövegdobozban megjelenő „OK” gombra kattintás után a Controls („vezérlők”) objektum *AddButton(caption, left, top)* – „gombot megjelenít”, „cím”, „bal” (x koordináta), „felső” (y koordináta) – eljárása segítségével megjelenítünk öt nyomógombot (vezérlőt), s egyúttal feltöltjük velük a gomb tömböt.

A vezérlők segítségével némi interaktivitást vihetünk a programunkba. A vezérlők különböző felhasználói tevékenységeket érzékelnek futási időben (kattintás, szövegbeírás), amelyek lényegében inputként szolgálhatnak a program számára. A felhasználó (de tágabb értelemben, más fejlesztőkörnyezetben nemcsak a felhasználó) vezérlőkön és más elemeken végzett tevékenységeit eseményeknek nevezzük, s természetesen léteznek felhasználói akarat nélkül bekövetkező, és kezelhető események is. A különböző események kezelésére épülő programozási technikát *eseményvezérelt programozásnak* hívjuk.

Kifinomult programfejlesztői környezetek gyakran nagyon gazdag vezérlőkészletet biztosítanak a fejlesztéshez. A Small Basic csak néhány vezérlő használatát teszi lehetővé, de a használatuk érdekesebbé és rugalmasabbá teheti a programjainkat. A Small Basic az eseményvezérelt programozás lehetőségeiben sem közelíti meg a nagyobb „testvéreit” (Microsoft Visual Basic, Visual Basic .NET, Java, Python stb.), de az eseményvezérelt programozás logikájának elsajátítására nagyon is alkalmas.



35. ábra: A „Teknőcrajzoló” program egy lehetséges futása

A „Teknőcrajzoló” program egy darab eseményt kezel. Ha a felhasználó a grafikus ablakban található gombok egyikére kattint, akkor a *Controls* objektum *ButtonClicked* („gombra kattintottak”) *eseménykezelője* meghívja az általunk definiált „Kattintas” szubrutint.

Formailag a Controls.ButtonClicked = Kattintas utasítás kísértetiesen emlékeztet egy változó értékadására, ne keverjük össze az értékadást a szubrutin behívásával!

A szubrutinunk minden gombra kattintás (bármelyik gombra kattinthatunk) után az n változóba tölti a Controls objektum *LastClickedButton* („az utoljára kattintott gomb”) értékét (az érték esetünkben button1, button2 … button5 lehet), majd öt If… Then… szerkezettel megvizsgálja, hogy az érték a korábban feltöltött „gomb[…]” tömbünk melyik értékével azonos. Ha a vizsgált feltétel teljesül, a program végrehajtja a szükséges teknőcműveleteket.

Több esemény

A program futása közben természetesen több esemény is kezelhető.

```
GraphicsWindow.Show()

GraphicsWindow.MouseDown = Egerle
GraphicsWindow.MouseMove = Egermozog
GraphicsWindow.KeyDown = Gomble

Sub Egerle
    If Mouse.IsLeftButtonDown Then
```

```
        GraphicsWindow.Title = "A baloldali egérgombra kattintottál."
    ElseIf Mouse.IsRightButtonDown Then
        GraphicsWindow.Title = "A jobboldali egérgombra kattintottál."
    EndIf
EndSub

Sub Egermozog
    GraphicsWindow.Title = "Egérpozíció: " + Mouse.MouseX + ", " + Mouse.MouseY
EndSub

Sub Gomble
    If GraphicsWindow.LastKey = "I" Then
        Mouse.ShowCursor()
        GraphicsWindow.Title = "A kurzor látható. Nyomd le az N betűt, ha el aka
    ElseIf GraphicsWindow.LastKey = "N" Then
        Mouse.HideCursor()
        GraphicsWindow.Title = "A kurzor rejtett. Nyomd le az I betűt, ha meg al
    EndIf
EndSub
```

A fenti program három különböző eseményt kezel. Bármelyik egérgomb lenyomása, az egér mozgatása és a billentyűzeten egy gomb lenyomása meghív egy-egy eseménykezelő szubrutint.

A GraphicsWindow *MouseDown* („egérgomb lenyomása”), *MouseMove* („egér mozgatása”) és *KeyDown* („billentyű lenyomása”) eseménykezelői hívják meg az egyes eseménykezelő szubrutinokat. A szubrutinokon belül eddig nem tárgyalt elemeket hívunk segítségül a programozási feladat megoldására.

Az „Egerle” szubrutinban található feltételes elágazások a Mouse objektum *IsLeftButtonDown* („lenyomták az egér baloldali kapcsolóját”) és az *IsRightButtonDown* („lenyomták az egér jobboldali kapcsolóját”) tulajdonságaira épülnek. Ha valamelyik tulajdonság értéke igaz, akkor a grafikus ablak fejlécében lévő felirat megváltozik.

A korábbiakban az *ElseIf* – „különben-ha” – kulcsszóval nem találkoztunk, működése az If kulcsszóval formált utasításéval azonos, azonban az ElseIfben található feltételt a program csak akkor vizsgálja meg, ha a külső If… Then… szerkezetünk hamis (Else, pontosabban: ElseIf) ágára kerülünk.

Az „Egermozog” szubrutin a Mouse objektum *MouseX* („egér x pozíciója”) és *MouseY* („egér y pozíciója”) tulajdonságainak értékeit használja fel.

A Gomble szubrutin az utolsó lenyomott billentyű értékét vizsgálja meg a GraphicsWindow *LastKey* („utolsó lenyomott gomb”) tulajdonsága segítségével. Az egérmutató megjelenítése és eltüntetése (az I, illetve N gombok lenyomása esetén) a Mouse objektum *ShowCursor()* és *HideCursor()* műveleteivel történik.

Az eseménykezelő szubrutinok meghívásakor a többi szubrutintól eltérően nem használunk zárójelet.

13. fejezet - Flickr

A Small Basicbe több olyan elemet is beépítettek, amelyeknek gyakorlati haszna ugyan kevés, de segítségükkel a programozás közelebb hozható a fiatal korosztályokhoz. Az alábbi alkalmazás a népszerű *Flickr* képmegosztóról olvas be egy képet háttérként, az ablak méreteihez igazodva:

```
GraphicsWindow.Show()
kep = Flickr.GetRandomPicture("kutya")
szelesseg = GraphicsWindow.Width
magassag = GraphicsWindow.Height
GraphicsWindow.DrawResizedImage(kep, 0, 0, szelesseg, magassag)
```

A véletlen kép kiválasztása a Flickr objektum *GetRandomPicture(tag)* – „véletlen kép betöltése", „címke" – műveletével történik. A kép megjelenítéséhez a *DrawResizedImage(imageName, x, y, width, height)* – „átméretezett képet rajzol", x és y koordináta, „szélesség", „magasság" – műveletet hívjuk segítségül. A Flickrbe feltöltött képekből a keresési algoritmus olyan képeket válogat ki, amelyek címkéi között szerepel az általunk megadott címke.

36. ábra: A képbetöltő program egyik futási eredménye

14. fejezet - Játékprogramok Small Basic programozási nyelven

Az interneten számos program érhető el, amelyek fejlesztői nagyon kevés utasítással komoly programokat készítettek. Közülük a kezdő programozók számára talán a játékprogramok a legérdekesebbek. Small Basic programozási nyelven nem készíthetünk olyan kifinomult játékprogramokat – és általában véve bármilyen alkalmazást – mint a „professzionális” programozási nyelveken, de sokkal előbb sikerélményhez jutunk egy-egy program megírásával.

„Ütögetős” program

Az alábbi játékprogram talán a lehető legegyszerűbb. Elemezzük a programkódot önállóan! (A programkód részletes magyarázatát elhagyjuk.)

```
GraphicsWindow.BackgroundColor = "White"
GraphicsWindow.BrushColor = "Red"
uto = Shapes.AddRectangle(120, 12)
GraphicsWindow.BrushColor = "Orange"
labda = Shapes.AddEllipse(16, 16)
GraphicsWindow.MouseMove = OnMouseMove
x = 0
y = 0
dx = 1
dy = 1
Ismetlodes:
x = x + dx
y = y + dy
szelesseg = GraphicsWindow.Width
hossz = GraphicsWindow.Height
If (x >= szelesseg - 16 or x <= 0) Then
    dx = -dx
EndIf
If (y <= 0) Then
    dy = -dy
EndIf
hx = Shapes.GetLeft (uto)
If (y = hossz - 28 and x >= hx and x <= hx + 120) Then
    dy = -dy
EndIf
Shapes.Move(labda, x, y)
Program.Delay(5)
If (y < hossz) Then
    Goto Ismetlodes
EndIf
GraphicsWindow.ShowMessage("Sajnos,vesztettél.", "Játék vége")

Sub OnMouseMove
    paddleX = GraphicsWindow.MouseX
    Shapes.Move(uto, paddleX - 60, GraphicsWindow.Height - 12)
EndSub
```

37. ábra: „Ütögetős” játékprogramunk

A fenti program – mint általában az összes program – elsősorban akkor válhat érdekessé, ha egyéni „igényeket” kielégítve továbbfejlesztik. A jegyzet végén még felhívjuk a figyelmet néhány játékprogramra.

15. fejezet - Kiegészítések a Small Basichez

A Small Basic eredeti formájában elsősorban a játékos tanulást szolgálja, azonban a nyelv támogatja, hogy a fejlesztők olyan kiegészítéseket írjanak, amelyek segítségével az alapcsomag korlátozott lehetőségein túllépve professzionális alkalmazások készítésére nyílik lehetőség. A kiegészítések elemeihez külön magyarázatot általában nem adunk.

LitDev kiegészítés

Az internetről letölthető az ingyenes *LitDev* extension (jelenleg pl. a http://litdev.hostoi.com/ oldalról, de ez a későbbiekben változhat), amely valóban professzionális elemeket visz az alapverzióba.

A fejlesztők számára érdekes *LitDev_Source.zip* fájlra egyelőre nincs szükségünk, ezért töltsük le az oldalról az alapvetően minden szükséges *.dll* és *.xml* fájlt tartalmazó *LitDev_v1.0.zip* fájlt.

Kicsomagolás után a *LitDev_v1.0* könyvtárban lévő a *LitDev.xml* és a *LitDev.dll* fájlokat – operációs rendszertől függően – másoljuk át a *C:\Program Files\Microsoft\Small Basic\lib* vagy a *C:\Program Files (x86)\Microsoft\Small Basic\lib* könyvtárba.

A LitDev fejlesztői a Small Basic 20 alapobjektumát 22 további objektummal egészítették ki a szükséges műveletekkel, tulajdonságokkal és eseményekkel együtt. Érdemes kipróbálni és elemezni a kicsomagolt könyvtár alkönyvtáraiban található alkalmazások mindegyikét, ötletesek és látványosak. A példaalkalmazások azonban korántsem merítik ki a kiegészítésben rejlő lehetőségeket, ráadásul a kiegészítés nyújtotta lehetőségeket folyamatosan bővítik.

A kiegészítések hozzáadása után a kiegészítés programnyelvi elemeinek leírása éppúgy látható, mint az alapváltozaté. Használatukhoz ennek ellenére mindig érdemes áttanulmányozni a fejlesztői dokumentációt.

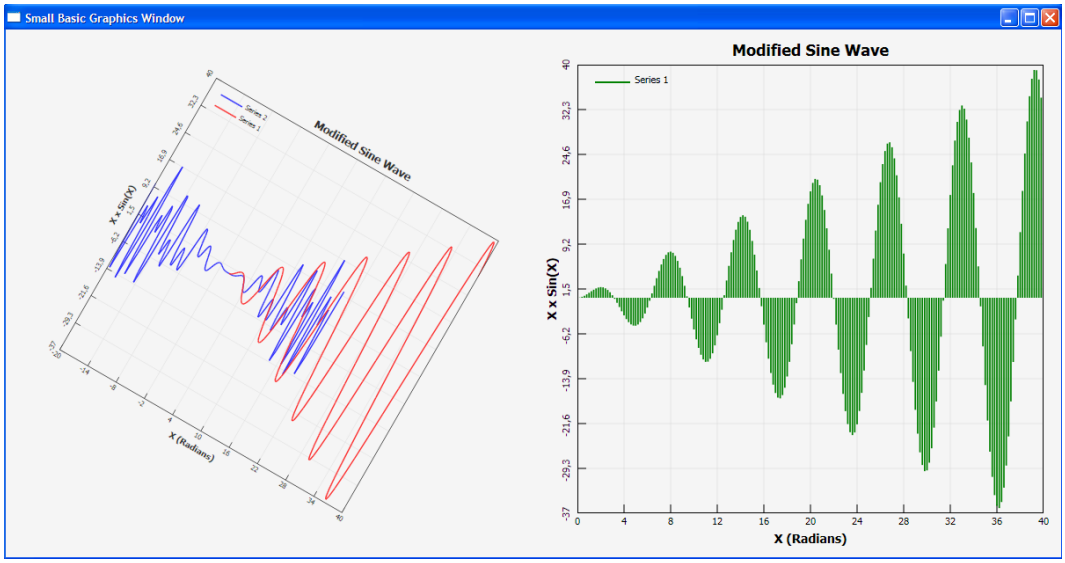
Statisztikai alkalmazás LitDevvel

Az alábbi *graph-shapes.sb* alkalmazás pl. bepillantást enged a Small Basic LitDev kiegészítésében rejlő statisztikai-fizikai-modellezési lehetőségekbe:

```
gw = 1200
gh = 600
GraphicsWindow.Width = gw
GraphicsWindow.Height = gh
GraphicsWindow.BackgroundColor = LDColours.WhiteSmoke
LDGraph.BorderColour = "#00000000"
LDGraph.InteriorColour = "#00000000"
For i = 0 To 200
    x = i/5
    y = x*Math.Sin(x)
    data1[x] = y
EndFor
For i = -100 To 100
    x = i/5
    y = x*Math.Sin(x)*Math.Sin(x*x/7)
    data2[x] = y
EndFor
LDGraph.ExportCSV(data1,Program.Directory+"\Graphing.csv")
```

```
graph1 = LDGraph.AddGraph(10,10,gw/2-20,gh-20,"Modified Sine Wave","X (Radians)
LDGraph.AddSeriesLine(graph1,"Series 1",data1,"Red")
LDGraph.AddSeriesLine(graph1,"Series 2",data2,"Blue")
LDGraph.DeleteSeries(graph1,"Series 1")
LDGraph.AddSeriesLine(graph1,"Series 1",data1,"Red")
Shapes.Zoom(graph1,0.7,0.7)
Shapes.Rotate(graph1,30)
Shapes.SetOpacity(graph1,80)
LDShapes.ZIndex(graph1,1)
graph2 = LDGraph.AddGraph(610,10,gw/2-20,gh-20,"Modified Sine Wave","X (Radians
LDGraph.AddSeriesHitogram(graph2,"Series 1",data1,"Green")
LDGraph.StopEvents()
```

A program elsősorban az LitDev kiegészítés LDGraph, LDShapes és LDEvents objektumainak lehetőségeit használja ki.



38. ábra: Szinuszhullámok felrajzolása a LitDev kiegészítés segítségével

Teaching kiegészítés

A Small Basic népszerű kiegészítéseit tartalmazó csomag a *Teaching extensions*. A http://extendsmallbasic.codeplex.com/downloads/get/261276 címről (a link változhat) letölthető csomagban lévő, gyerekeknek és kezdő programozóknak szánt kiegészítést a LitDev kiegészítésnél megismert módon adhatjuk hozzá a fejlesztőkörnyezethez. A letöltött *SmallBasicFun.zip* állomány kicsomagolása után a *SmallBasicFun.dll* és a *SmallBasicFun.xml* állományokat kell átmásolnunk a lib könyvtárba.

Telepítés után a http://www.teachingkidsprogramming.com/ oldalon lévő egyszerű alkalmazásokat is tanulmányozhatjuk és tesztelhetjük a Small Basic fejlesztőkörnyezet segítségével.

Data kiegészítés

Hasonlóan népszerű kiegészítés – számos új objektummal, tulajdonsággal, művelettel, eseménnyel – a *Data extension*, amely jelenleg a http://social.msdn.microsoft.com/Forums/en-US/smallbasic/thread/c8a5d124-b02d-44dd-982e-abf18d60f4b7 és a http://social.msdn.microsoft.com/Forums/en-US/smallbasic/thread/94015515-5579-4463-b885-9d77ec11b2cb threadről tölthető le. A letöltött .exe fájl futtatása után nem kell manuálisan a megfelelő helyre másolnunk a kiegészítés eléréséhez szükséges .dll és .xml állományt.

A Data kiegészítés az egyik legizgalmasabb kiterjesztés, de egyes hibái miatt a letölthetősége nem folyamatos, e sorok írásakor is éppen fejlesztik.

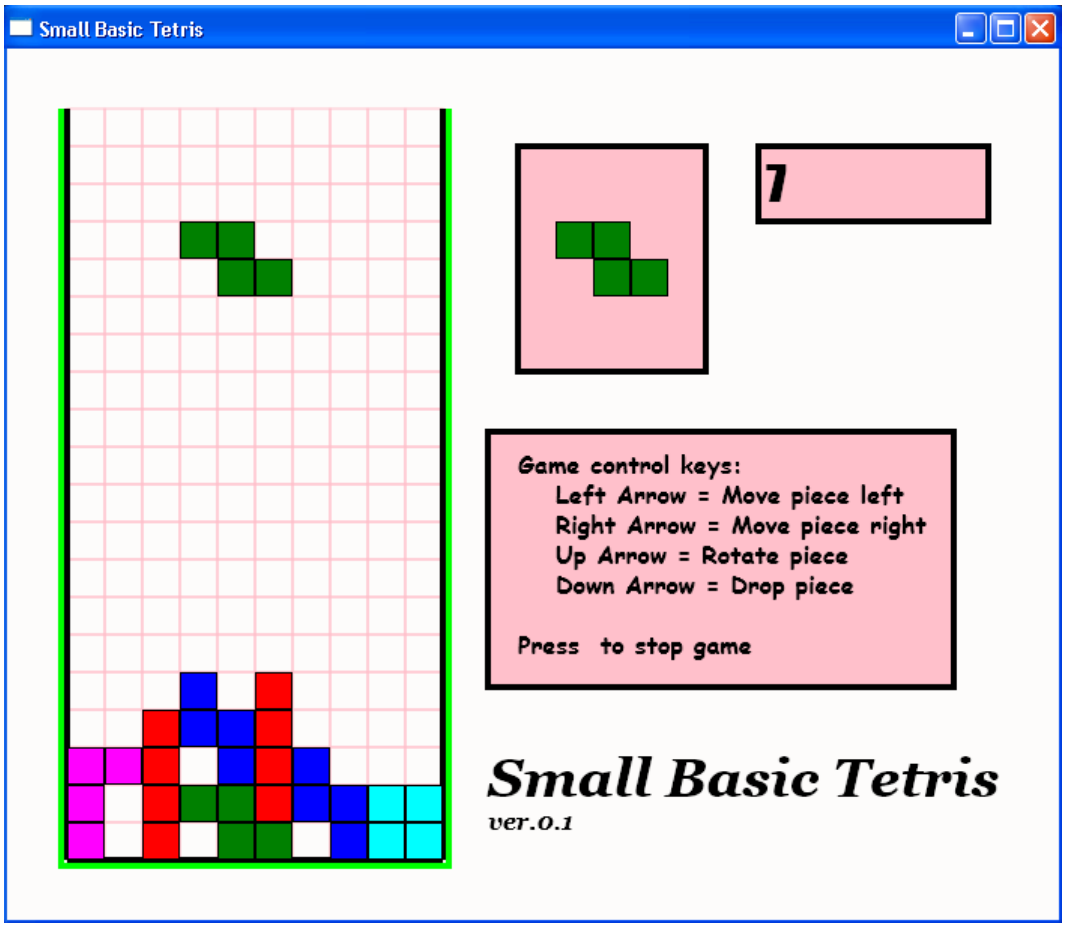
Természetesen a kiegészítők száma sokkal magasabb, az egyes kiegészítésekben rejlő lehetőségek megismerése további – akár önálló – tanulást igényel. Érdemes folyamatosan böngészni a különböző Small Basic-fórumokat – pl. a http://social.msdn.microsoft.com/Forums/en-US/smallbasic/ fórumot –, programozási tanácsok mellett a kiegészítésekben rejlő lehetőségekről is hasznos információkhoz juthatunk.

Program importálása

Számos alkalmazás programkódját a fejlesztőkörnyezet *Import* gombjára kattintva az ID megadása után importálhatjuk.

Különböző fejlesztői fórumokon a programozók gyakran csak a program azonosítóját (*ID*-jét) adják meg, sőt, hosszabb programkódoknál a http://smallbasic.com/ oldal is inkább csak az ID-ét közli.

Pl. a TETRIS ID-vel a népszerű, azonos nevű játékprogramot hívhatjuk be a fejlesztőkörnyezet szerkesztőjébe.



40. ábra: Small Basic Tetris

16. fejezet - Adatbázis-kezelés a Small Basicben

Bár nem zárható ki, hogy a jövőben a fejlesztők *relációs adatbázisok* kezelésére alkalmas kiegészítést írnak a Small Basichez, jelenleg ilyen, általánosan használható kiegészítés nincs, bár van néhány biztató próbálkozás. Az adatbázis-kapcsolatos alkalmazások írása amúgy is mélyebb programozási és adatbázis-kezelési (SQL) ismereteket igényel, így egy alapvetően tanulást és szórakozást szolgáló programozási nyelvben talán ez szükségtelen.

Ennek ellenére egyszerűbb adatszerkezetek tartós – a futási időn túli tárolására – esetenként szükségünk lehet.

Adatok szövegfájlban

A Small Basic lehetővé teszi az egyes adatok fájlokba való kiírását és beolvasását.

```
A["a"] = "Golden"
A["b"] = "alma"
A["c"] = "zöld"
A["d"] = "elfogyott"
File.WriteContents("D:\small basic\adatbazis\db.txt", A)
```

A fenti program például - amennyiben a megadott meghajtó, könyvtár és alkönyvtár létezik, a „db.text” fájlba menti az A tömb értékeit. A program a *File* objektum *WriteContents(filePath, contents)* – „tartalmak kiírása”, „elérési út”, „tartalmak” – műveletére épül.

Adatok Excel-táblában

Szövegfájlokkal azonban egy kicsit nehézkes az adatkezelés, ezért használjuk inkább a jóval könnyebben kezelhető Microsoft Excelt az adataink tárolására! (Természetesen az Excel sem adatbázis-kezelő, de egyszerűbb adatstruktúrák kezelésére kiválóan alkalmas.)

Ehhez először is telepítenünk kell az *Excel Library for Small Basic* kiegészítést a http://excel4smallbasic.codeplex.com/ oldalról. A telepítő automatikusan a lib alkönyvtárba telepíti a szükséges fájlokat. (A kiegészítés a .NET 3.5 vagy magasabb verziószámú keretrendszer mellett legalább Excel 2007-es verziójú táblázatkezelőt is igényel.)

Hozzunk létre egy *elso. xlsx* nevű fájlt egy tetszőleges könyvtárban! (Mi ezt a saját gépünkön a *D:\small basic\adatbazis* könyvtárban tesszük.)

Az alábbi kód egyszerű adatfeltöltést tesz lehetővé:

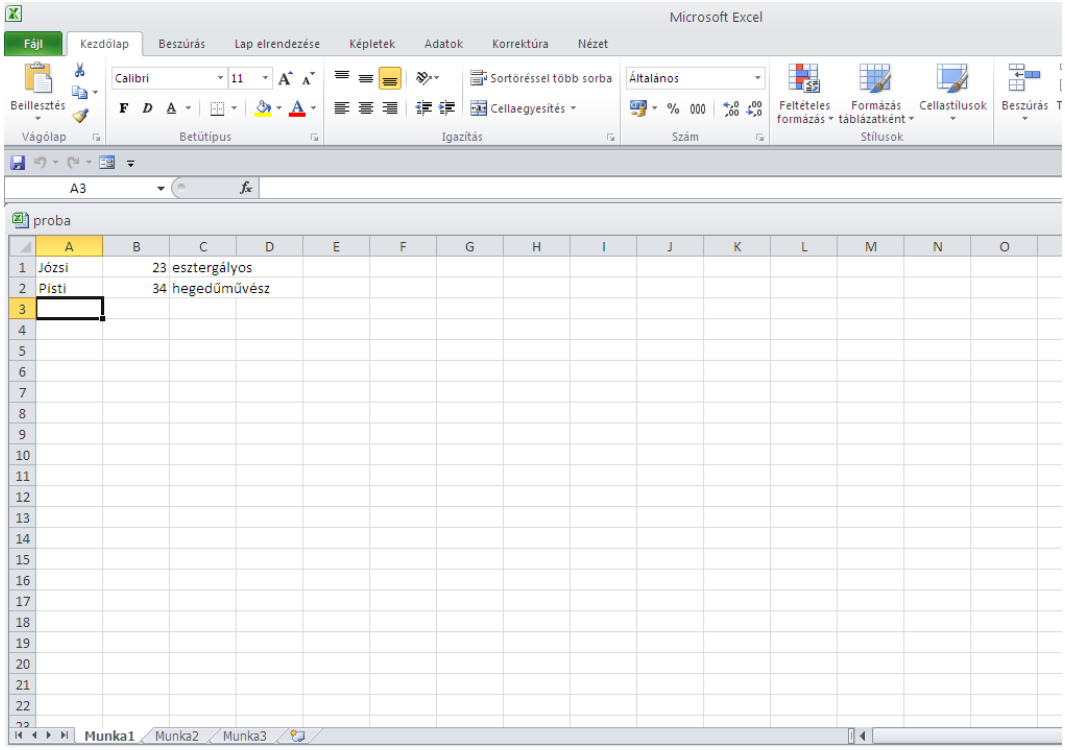
```
n = 0
ertek = "kezdőérték"
While ertek <> ""
    n = n + 1
    ertek = Excel.ReadCell("D:\small basic\adatbazis\proba.xlsx","Munka1", n, 1
    EndWhile
Excel.CloseBook("D:\small basic\adatbazis\proba.xlsx")
TextWindow.Write("Név?")
Adat[1] = TextWindow.Read()
TextWindow.Write("Kor?")
```

```
Adat[2] = TextWindow.ReadNumber()
TextWindow.Write("Foglalkozás?")
Adat[3] = TextWindow.Read()
For i = 1 To 3
    Excel.WriteCell("D:\small basic\adatbazis\proba.xlsx","Munka1", n, i, Adat[
EndFor
Excel.CloseBook("D:\small basic\adatbazis\proba.xlsx")
TextWindow.WriteLine("Adatfelvétel megtörtént!")
```

Először egyszerű algoritmussal megvizsgáljuk, hogy eddig hány sorba írtunk, majd a következő sorba feltöltjük a bekért értékeket.

Az Excel-fájl megnyitása és az adott érték beolvasása az *Excel.ReadCell(filePath, sheetName, rowIndex, columnIndex)* – „cellát beolvas”, „elérési út”, „munkalapnév”, „sorindex”, „oszlopindex” – művelet történik. Egy adott cellába az *Excel.WriteCell(filePath, sheetName, rowIndex, columnIndex, newValue)* – „cellába ír”, „elérési út”, „munkalapnév”, „sorindex”, „oszlopindex”, „új érték” – művelet segítségével írhatunk. Az Excel-táblát az *Excel.CloseBook(filePathOrName)* – „bezár”, „elérési út vagy fájlnév” – művelettel zárhatjuk be és menthetjük el. E műveletekkel a korábban elmentett adatok változókba történő beolvasása is megoldható.

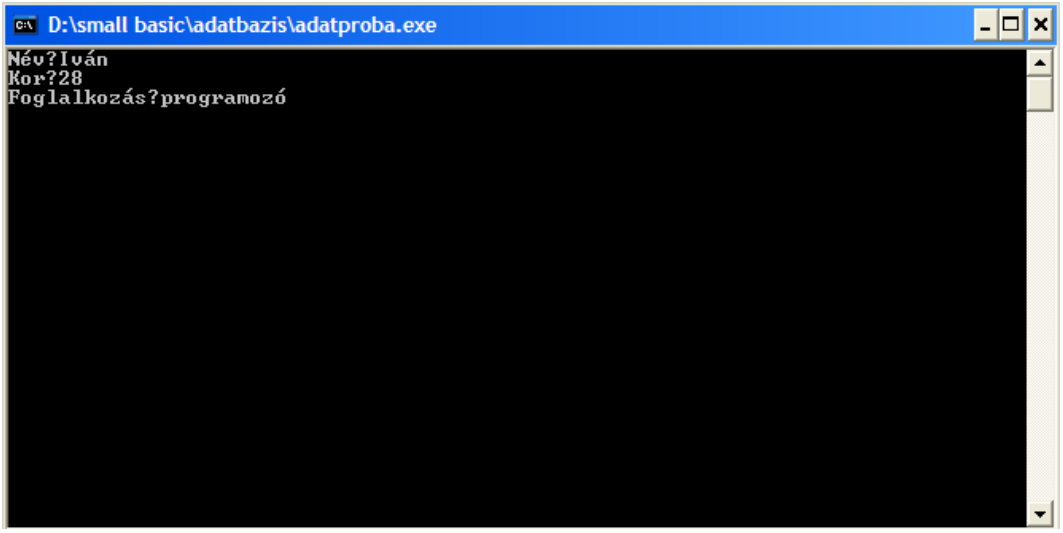
Az ertek változó kezdőértékének megadására azért volt szükség, mert enélkül a ciklus magjában lévő utasítások egyszer sem futottak volna le.



41. ábra: Adatkezelő programunk Excel-fájlja két adatfeltöltés után

Érdemes megjegyezni, hogy az Excel sok más Microsoft-szoftverhez hasonlóan – de a szabad szoftverek világában is találunk példákat – az Excel lehetőségeit sokkal jobban kihasználó VBA programozási nyelvvel (pl. a 2010-es verziónál Visual Basic for Applications in Excel 2010) programozható. A VBA sok elemében nagyon hasonlít a Small Basichez, bár inkább a nagyobb testvér, a Visual Basic egyik „klónjának” tekinthető.

A fenti program természetesen csak apró, korántsem tökéletes példa, de az adatok beolvasása és a futási időben keletkező adatok tartós tárolása számos programozási feladatnál felmerülhet.



42. ábra: Adatkezelő programunk futási időben

17. fejezet - Small Basic-példák

Az alábbi programokhoz a kódokon belüli megjegyzéseken túl csak minimális mértékben fűzünk további megjegyzéseket. Kizárólag az „alapnyelvi” elemekre épülnek, kiegészítéseket nem használnak. Leginkább azt hivatottak szemléltetni, hogy némi ötletességgel néhány soros programok is lehetnek szórakoztatók. Természetesen sokkal összetettebb és látványosabb programok is írhatók, ezek a különböző online programkönyvtárakban elérhetők.

Helyesírás

Az alábbi program bevezetést ad a szövegek kezelésébe:

```
'Helyesírás-ellenőrző program
'Angol helyesírás-ellenőrző
'A Dictionary.GetDefinition(word) műveletre épül.
'Internet-hozzáférés szükséges hozzá.
'
szoveg = "abcdefghijklmnopqrstuvwxyz'ABCDEFGHIJKLMNOPQRSTUVWXYZ"
While (bevittszoveg <> "  ") 'két szóköz
    TextWindow.WriteLine(" ")
    TextWindow.WriteLine("Írja be az angol nyelvű mondatot: ")
    bevittszoveg = Text.Append(TextWindow.Read(),"  ")
    For i = 1 To Text.GetLength(bevittszoveg)
        If (Text.GetIndexOf(szoveg,Text.GetSubText (bevittszoveg,i,1)) = 0) Then
           If (szo <> "") Then
                If (Dictionary.GetDefinition(szo) = "") Then
                    TextWindow.BackgroundColor="white"
                    TextWindow.ForegroundColor="black"
                    TextWindow.Write(szo)
                    TextWindow.BackgroundColor="black"
                    TextWindow.ForegroundColor="white"
                Else
                    TextWindow.Write(szo)
                EndIf
                    szo = ""
            EndIf
                TextWindow.Write(Text.GetSubText(bevittszoveg,i,1))
            Else
                szo = Text.Append(szo,Text.GetSubText(bevittszoveg,i,1))
        EndIf
    EndFor
EndWhile
```

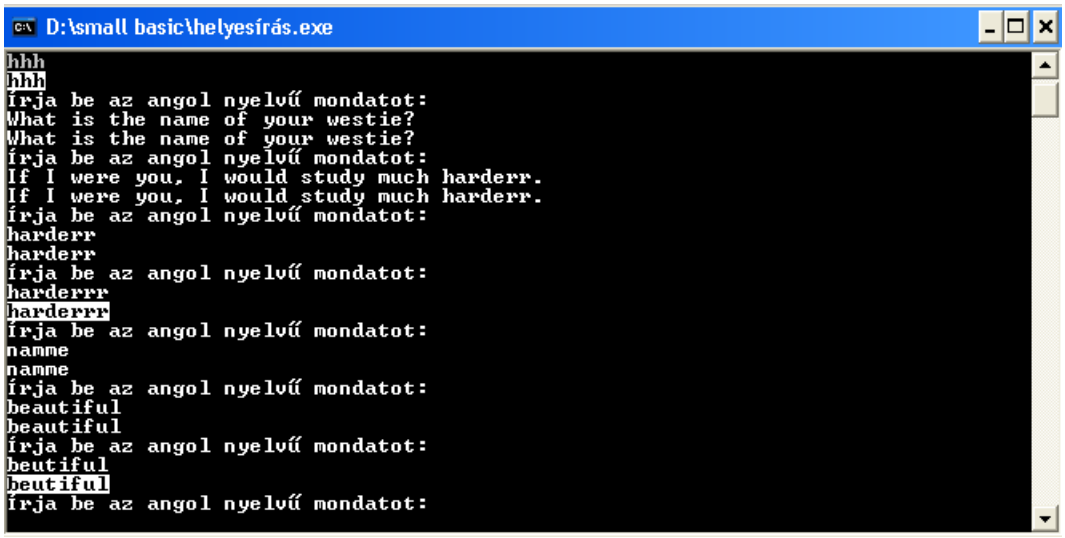
A program új elemei:

- A *Text(„szöveg”)* objektum szövegek kezelésére alkalmas műveleteket tartalmaz.
  - A Text objektum *Append(text1, text2)* – „összefűz”, „első szöveg”, „második szöveg” – művelete két szövegdarabot összefűzve egy szöveget ad eredményül.
  - A Text objektum *GetLength(text)* – „hosszúság meghatározása”, „szöveg” – művelete megadja az adott szöveg karakterekben mért hosszát.
  - A Text objektum *GetIndexOf(text, subText)* – „index meghatározása”, „szöveg”, „részszöveg” (keresett szöveg) – művelete megadja azt a pozíciót, ahol a keresett részszöveg a szövegben

megjelenik. Ha a keresett szöveget a szöveg nem tartalmazza, a visszatérési érték nulla. (A keresett szöveg természetesen egyetlen karakter is lehet.)

- A Text objektum *GetSubText(text, start, length)* – „részszöveg előállítása”, „szöveg”, „kezdőpozíció”, „hosszúság” – művelete a szöveg egy adott kezdőpozíciójából adott számú karakter hosszú szöveget állít elő.

- A *Dictionary.GetDefinition(word)* - „szótár”, „definíció meghatározása”, „szó” – művelet a Small Basic online elérhető szótárából a keresett szó definícióját adja eredményül.



43. ábra: A helyesírás-ellenőrző program egyik futási eredménye

„Jóslat”

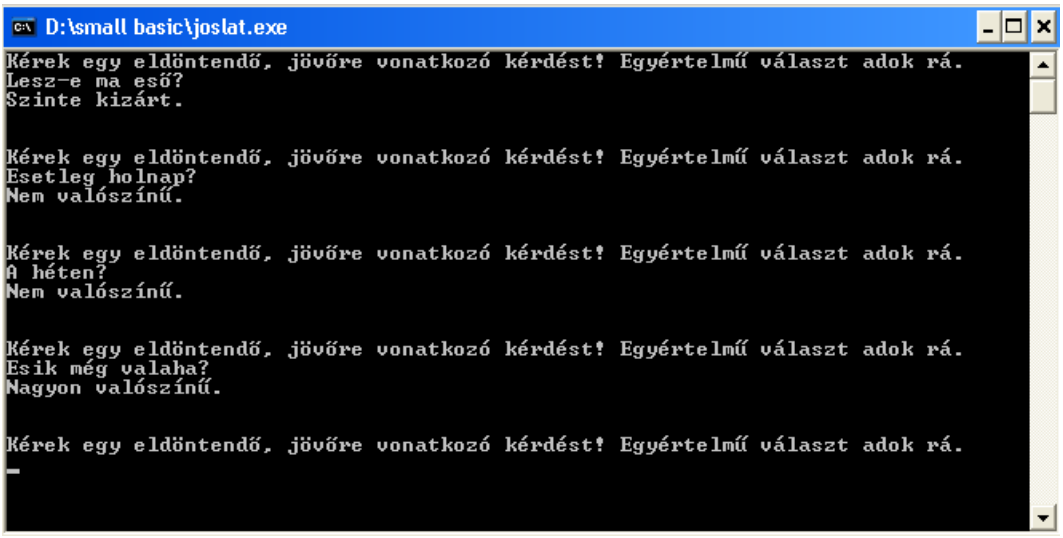
Egyszerű program a tömbök gyakorlására:

```
Kezdet:
joslat[1] = "Biztosan."
joslat[2] = "Nagyon valószínű."
joslat[3] = "Valószínű."
joslat[4] = "Előfordulhat."
joslat[5] = "Talán."
joslat[6] = "Nem valószínű."
joslat[7] = "Aligha."
joslat[8] = "Szinte kizárt."
joslat[9] = "Biztos, hogy nem."
joslat_tombhossz=Array.GetItemCount(joslat)
kovetkezo=Text.GetCharacter(13)+Text.GetCharacter(10)

Ciklus:
TextWindow.Write(("Kérek egy eldöntendő, jövőre vonatkozó kérdést! Egyértelmű v
TextWindow.Read()
TextWindow.WriteLine((joslat[Math.GetRandomNumber(joslat_tombhossz)] + kovetkez
Goto Ciklus
```

A program új elemei:

- A tömbműveletek kezelését szolgáló *Array* („tömb”) objektum *GetItemCount(array)* – „elemek összeszámolása”, „tömb” – művelete megszámolja, hogy egy tömb hány elemből áll.

- A Text objektum *GetCharacter(characterCode)* – „karakter kiírása”, „karakterkód” – művelet a karakterek Unicode-táblában lévő száma alapján visszaadja az adott karaktert. A 13-as és 10-es karakterkód kombinációja a sortörést jelenti (13 = CR – carriage return , (írógép)kocsi/kurzor vissza, , 10 = LF –line feed, soremelés/következő sor).



44. ábra: A „Jóslat” program működés közben

A programkódban használt végtelen ciklus általában nem szerencsés, komolyabb programozási feladatoknál kerülendő. Az egyszerűség kedvéért a későbbiekben is több helyen használjuk ezt a módszert, de érdemes megjegyezni, hogy a standard programozási módszertanban ez lényegében tiltott eszköz. Kezdő programozók azonban minden további nélkül élhetnek vele.

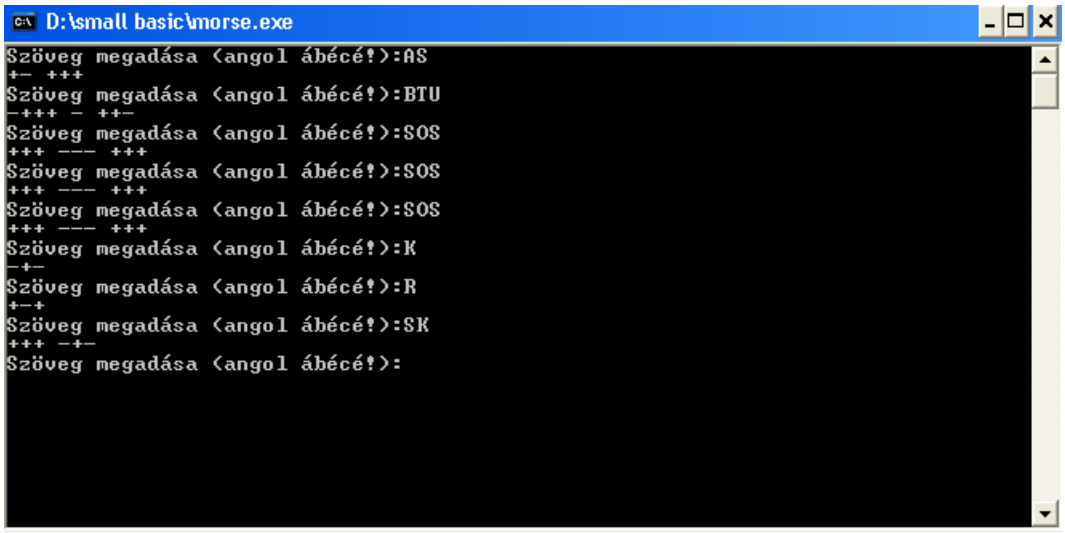
Morzeábécé

Egyszerű Morzeábécé -alkalmazás:

```
Morse="a=+-;b=-+++;c=-+-+;d=-++;e=+;f=++-+;g=--+;h=++++;i=++;j=+---;k=-+-;l=+-+
Ciklus:
TextWindow.Write("Szöveg megadása (angol ábécé!):")
t=TextWindow.Read()
For i=1 To Text.GetLength(t)
    TextWindow.Write(
    (Morse[Text.ConvertToLowerCase(Text.GetSubText(t,i,1))])+" ")
EndFor
TextWindow.WriteLine(" ")
Goto Ciklus
```

A program új eleme:

- A Text objektum *ConvertToLowerCase(text)* – „kisbetűsre átkonvertál”, „szöveg” – művelete kisbetűsre alakítja át a nagybetűs szövegeket.

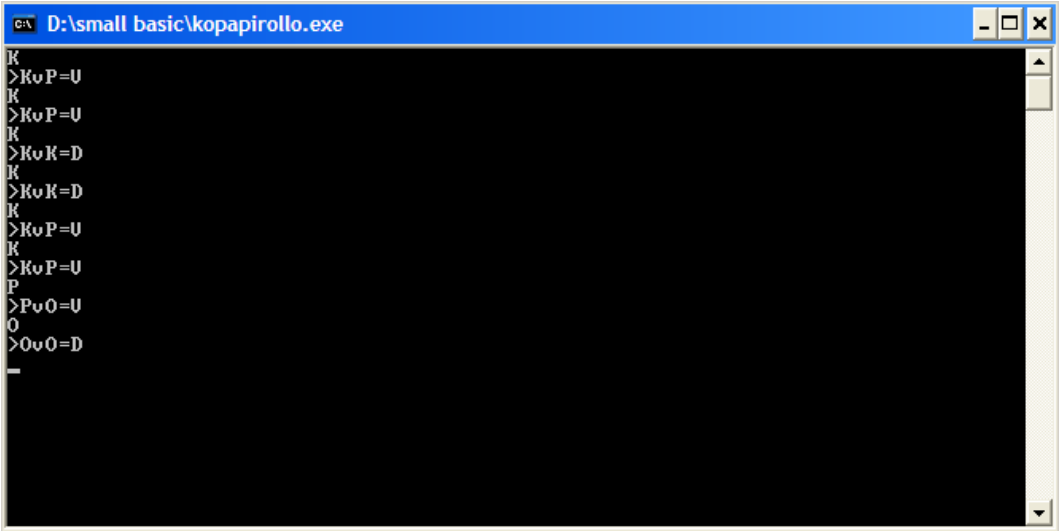


45. ábra: Morzeábécé-alkalmazás

Kő-papír-olló játék egysoros programmal

Az alábbi játék lényegi része egyetlen programsor. Élesben nem szabad vagy legalábbis nem érdemes ilyen szintű „kódsűrítésre” törekedni, de érdekességéként érdemes tanulmányozni az alábbi – törvényszerűen „fapados”, továbbfejlesztésre szoruló – játékprogramot:

```
'Kő-papír-olló - példaprogram az egysoros játékprogramra
'
'A kő erősebb ollónál
'Az olló erősebb a papírnál
'A papír erősebb a kőnél
'
'A játék során csak egy nagybetűs karaktert kell megadnunk.
'(K)ő,(P)apír,(O)olló
'A program is kiválaszt egy véletlen játékot a számítógépnek, mint ellenfélnek,
'
'">K" jelentése: "Követ választottál."
'">P" jelentése: "Papírt választottál."
'">O" jelentése "Ollót választottál"
'"vK" jelentése: "A számítógép követ választott."
'"vP" jelentése: "A számítógép papírt választott."
'"vO" jelentése: "A számítógép ollót választott."
'"=G" jelentése: "Győztél!"
'"=V" jelentése: "Vesztettél!"
'"=D" jelentése: "Döntetlen."
'Tehát ">KvO=G" jelentése:
'"Te követ választottál, a számítógép pedig ollót. Nyertél!"
'A PROGRAM TOVÁBBFEJLESZTENDŐ! :-)

Játék:

TextWindow.WriteLine
    ((Text.GetSubText(Text.GetSubText(">KvK=D>KvP=V>KvO=G>PvK=G>PvP=D>PvO=V>OvK=
Goto Játék
```

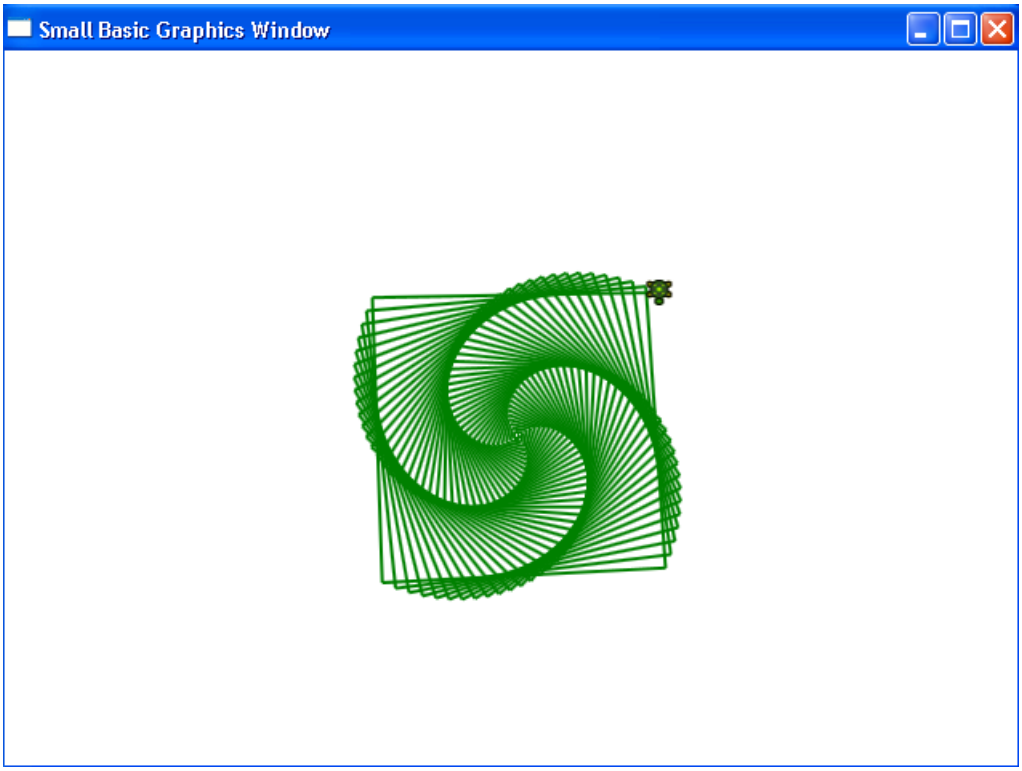


46. ábra: A jegyzet írójának pechszériája a kő-papír-olló játékban

A „Zöld hullám” alkalmazás

Az alábbi, nagyon egyszerű kódra épülő program, a kód összetettségéhez képest látványos eredményt ad. A program a teknőcgrafikára épül:

```
GraphicsWindow.Show()
GraphicsWindow.PenColor = "Green"
Turtle.Speed = 9
szog = 91
Rajzolas()

Sub Rajzolas
    n = 0
    For i = 1 to 180
        Turtle.Move(n)
        Turtle.Turn(szog)
        n = n + 1
    EndFor
EndSub
```



47. ábra: A „Zöld hullám” alkalmazás

Koch-görbék

A programozók tömör algoritmusa és a programkód rövidségéhez mérten látványos futási eredménye miatt kedvelik a *Koch-görbe* vagy *Koch-hópehely* felrajzolását. A Koch-görbe a matematikatörténet *Helge von Koch* svéd matematikus által 1904-ben leírt első fraktálja.

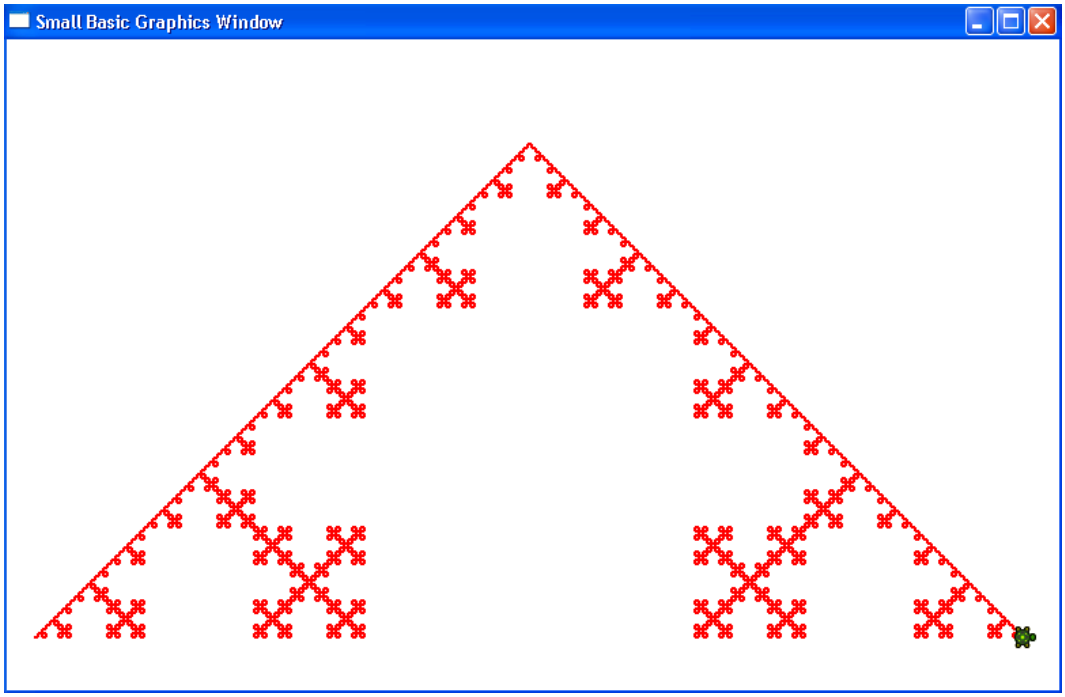
A görbe alapváltozatát úgy állíthatjuk elő, hogy a szabályos háromszög oldalait elharmadoljuk, majd a középső harmadára ismét egy szabályos háromszöget rajzolunk. A háromszögek oldalait szintén harmadoljuk, és háromszöget rajzolunk rájuk. Ezt a „végtelenségig” folytathatjuk.

A Koch-görbének számos altípusa ismert, mi ezek közül mutatunk be két alapváltozatot. Mindkét program a teknőcgrafikára épül.

Az első Koch-görbénk egy ún. első típusú *kvadratikus* Koch-görbe:

```
szog = 90
hossz = 3
iteracio = 5
GraphicsWindow.BackgroundColor = "White"
GraphicsWindow.PenColor = "Red"

Turtle.Speed = 9
Turtle.Turn(-90)
Turtle.PenUp()
Turtle.Move(300)
Turtle.Turn(-90)
Turtle.Move(200)
Turtle.Turn(-90)
Turtle.PenDown()
```

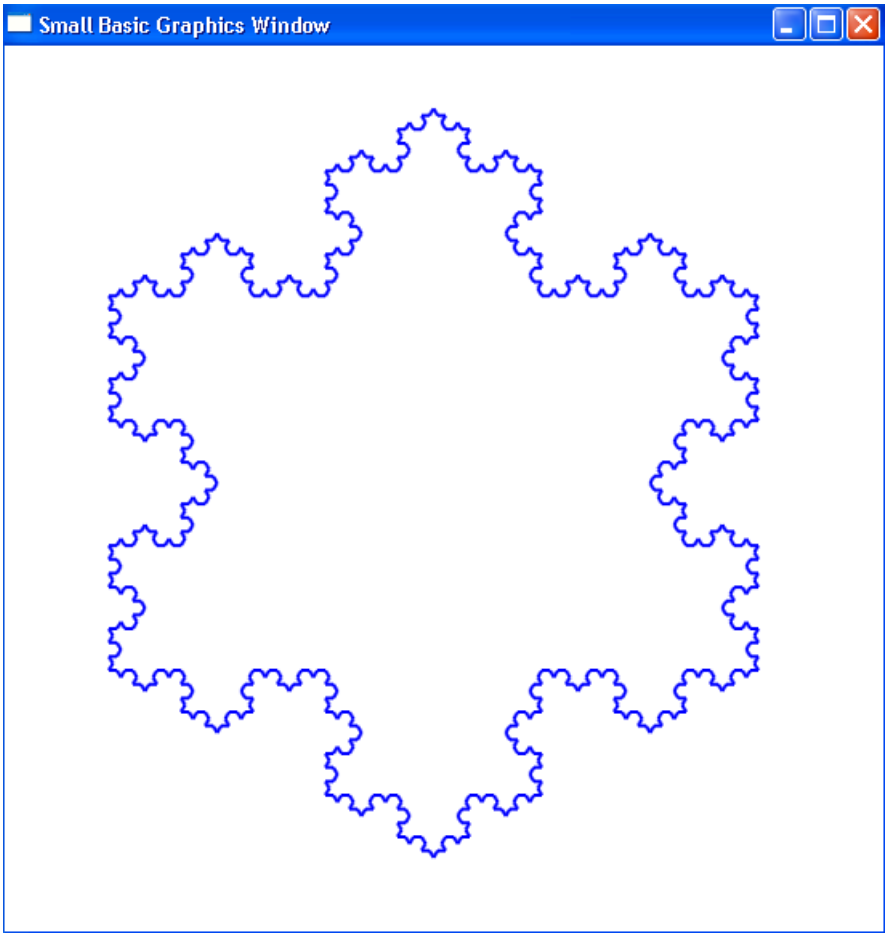
```
Rajzolas_szabaly()

Sub Rajzolas_szabaly
    iteracio = iteracio - 1
        If (iteracio = 0 ) Then
            Turtle.Move(hossz)
            Turtle.Turn(-szog)
            Turtle.Move(hossz)
            Turtle.Turn(szog)
            Turtle.Move(hossz)
            Turtle.Turn(szog)
            Turtle.Move(hossz)
            Turtle.Turn(-szog)
            Turtle.Move(hossz)
        Else
            Rajzolas_szabaly()
            Turtle.Turn(-szog)
            Rajzolas_szabaly()
            Turtle.Turn(szog)
            Rajzolas_szabaly()
            Turtle.Turn(szog)
            Rajzolas_szabaly()
            Turtle.Turn(-szog)
            Rajzolas_szabaly()
        EndIf
    iteracio = iteracio + 1
EndSub
```



48. ábra: Kvadratikus Koch-görbe (1. típus)

Második típusú kvadratikus Koch-görbe:

```
szog = 60
hossz = 5
iteracio = 4

GraphicsWindow.BackgroundColor = "White"
GraphicsWindow.PenColor = "Blue"

Turtle.Speed = 9
Turtle.PenUp()
Turtle.Turn(90)
Turtle.Move(150)
Turtle.Turn(90)
Turtle.Move(150)
Turtle.Turn(90)
Turtle.PenDown()

Rajzol_kezdet()

Turtle.Hide()

Sub Rajzol_szabaly
    iteracio = iteracio - 1
    If (iteracio = 0) Then
        Turtle.Move(hossz)
        Turtle.Turn(-szog)
        Turtle.Move(hossz)
        Turtle.Turn(szog)
        Turtle.Turn(szog)
        Turtle.Move(hossz)
        Turtle.Turn(-szog)
        Turtle.Move(hossz)
    Else
        Rajzol_szabaly()
        Turtle.Turn(-szog)
        Rajzol_szabaly()
        Turtle.Turn(szog)
        Turtle.Turn(szog)
        Rajzol_szabaly()
        Turtle.Turn(-szog)
        Rajzol_szabaly()
    EndIf
    iteracio = iteracio + 1
EndSub

SubRajzol_kezdet
    Rajzol_szabaly()
    Turtle.Turn(szog)
    Turtle.Turn(szog)
    Rajzol_szabaly()
    Turtle.Turn(szog)
    Turtle.Turn(szog)
    Rajzol_szabaly()
EndSub
```



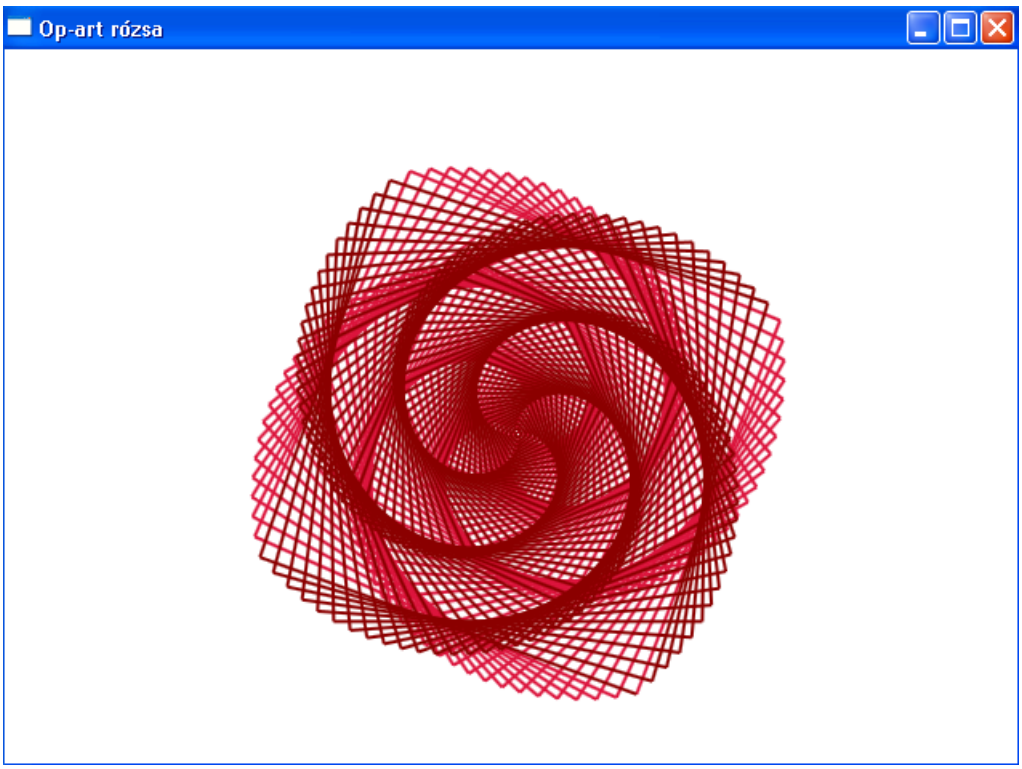
49. ábra: Kvadratikus Koch-görbe (2. típus)

Op-art rózsa

Egyszerű, teknőcgrafikára épülő alkalmazás „op-art” stílust követő rózsa rajzolására:

```
GraphicsWindow.Title = "Op-art rózsa"
DrawRose()

Sub DrawRose
    GraphicsWindow.BrushColor = "White"
    GraphicsWindow.FillRectangle(0,0,GraphicsWindow.Width,GraphicsWindow.Height
    GraphicsWindow.PenWidth = 2
    GraphicsWindow.PenColor = "Crimson"
    Turtle.Speed = 10
    Turtle.Show()
    For elhossz = 1 To 250
        Turtle.Move(elhossz)
        Turtle.Turn(89)
    EndFor
    GraphicsWindow.PenColor = "DarkRed"
    For elhossz =0 To 249
        Turtle.Move(250-elhossz)
        Turtle.Turn(89)
    EndFor
```

```
    Turtle.Hide()
EndSub
```



50. ábra: „Op-art rózsa”

Landolás

Mint már korábban említettük az interneten számos, látványos és érdekes játékprogram elérhető Small Basic nyelven. Az alábbi programot is egy ilyen programötlet ihlette. Csak néhány soros, de jó kiindulási alap egy komolyabb alkalmazáshoz:

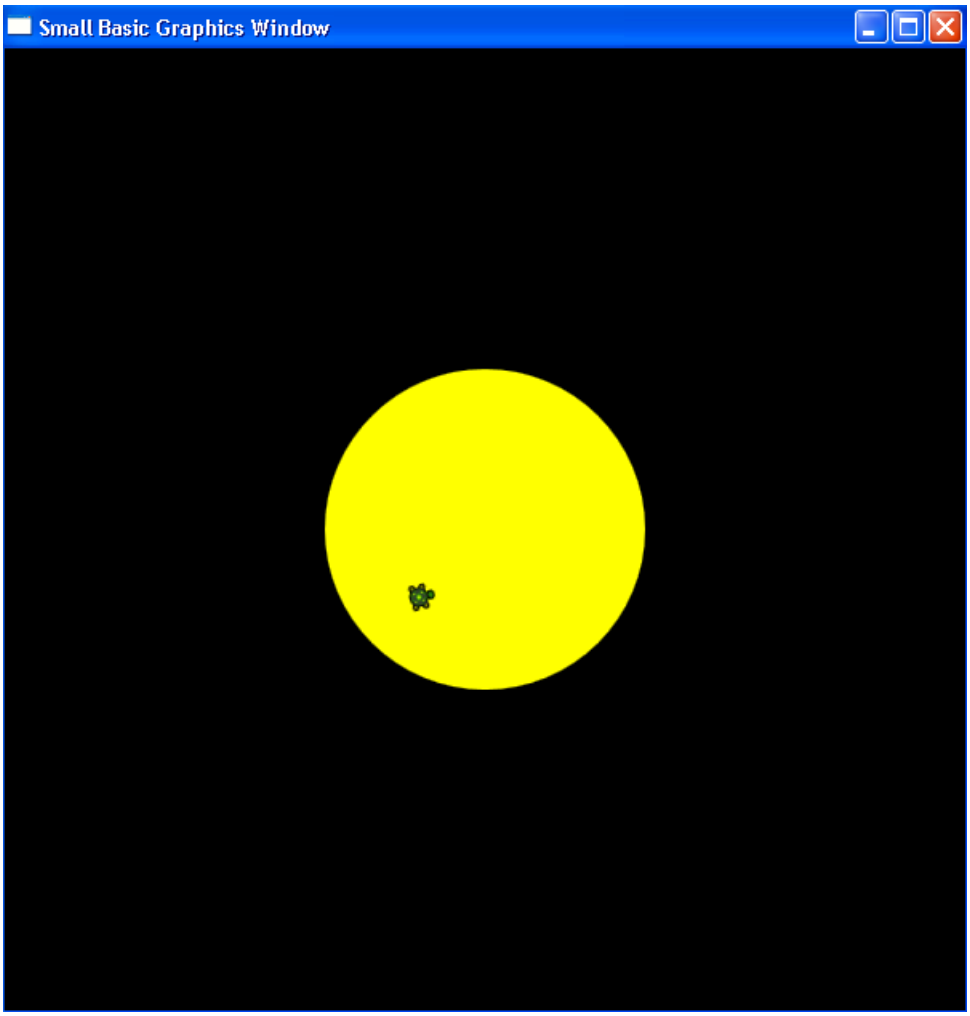
```
' A kezdeti sebesség beállítása x és y irányba
sebx = 0.03
seby = 0

gravitacio = 0.001 ' A gravitáció beállítása

GraphicsWindow.BackgroundColor = "Black" ' A világűr fekete
GraphicsWindow.BrushColor = "Yellow" ' A bolygó sárga

' A grafikus ablak mérete
GraphicsWindow.Width = 600
GraphicsWindow.Height = 600

GraphicsWindow.FillEllipse(200,200,200,200) ' A bolygó megrajzolása

'Az űrutazó teknőc kezdeti pozíciói
pozx = 0
pozy = 130
```

```
irany = 0 ' A teknőc kezdő utazási irányának beállítása

Turtle.Show()

Foprogram:

irany = Mouse.MouseX ' Egérmozgatással navigálunk.

If Mouse.IsLeftButtonDown Then
    ' "Gázt adunk, ha a a bal egérkapcsolót lenyomjuk.
    sebx = sebx + 0.0001 * Math.Sin(Math.GetRadians(irany))
    seby = seby + 0.0001 * Math.Cos(Math.GetRadians(irany)) '
EndIf

R2 = pozx * pozx + pozy * pozy ' Innentől minden tiszta fizika
gravero = gravitacio / R2
R = Math.SquareRoot(R2)
gravhanyados = gravero / R
sebx = sebx - gravhanyados * pozx
seby = seby - gravhanyados * pozy

pozx = pozx + sebx
pozy = pozy + seby

Turtle.X = 300 + pozx
Turtle.Y = 300 - pozy
Turtle.Angle = irany

If GraphicsWindow.LastKey <> "Escape" Then ' ESC-vel lassítunk
    Goto Foprogram
EndIf
```

A játék során egérmozgatással állítjuk be az űrutazó teknőc irányát, bal egérkapcsolóval gyorsítunk, az ESC billentyű lenyomásával „landolunk”.

A program új elemei:

- A Math objektum *Sin(angle)* – „szinusz”, „fok” – művelete egy szög szinuszát, Cos(angle) – „koszinusz”, „fok” – művelete egy szög koszinuszát számolja ki.
- A Math objektum *SquareRoot(number)* – „négyzetgyök”, „szám” – művelete kiszámolja egy szám négyzetgyökét.



51. ábra: Sikeres landolás

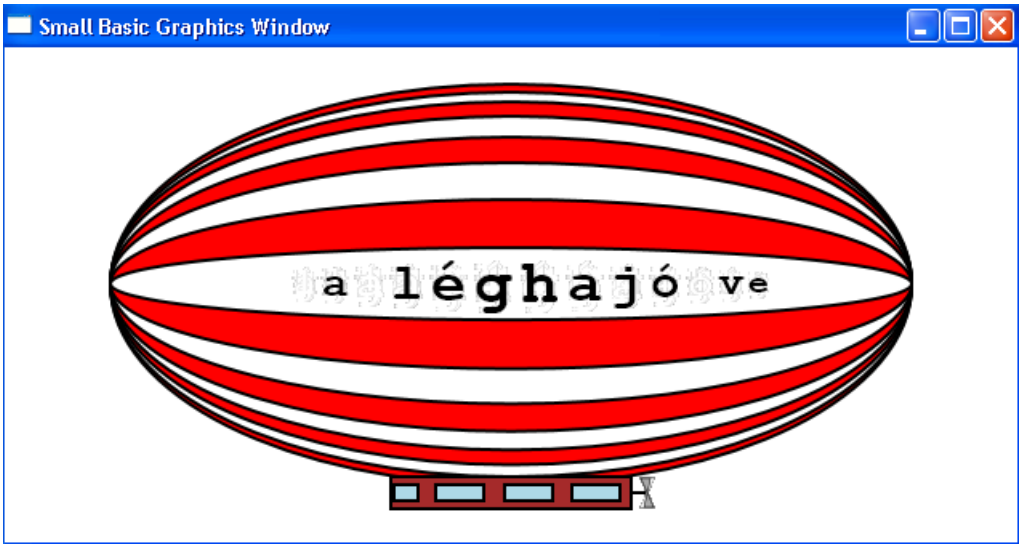
„Retroléghajó”

Az oldalán fényreklámot hordozó léghajó az egyik legősibb Basic-program, ezért már csak hagyománytiszteletből is érdemes megismerkednünk Small Basic-kódjával. Figyeljük meg a változókezelést és a GraphicsWindow alakzatainak használatát:

```
'Kezdőértékek
hossz = 75
szoveghossz = 175
lszel = 500
lmag = 250
lepes = 8
res = 10
kozepx = GraphicsWindow.Width / 2
kozepy = GraphicsWindow.Height / 2
hatter = "White"
uzenet = "Száll a léghajó velünk"

'A légcsavar megrajzolása
GraphicsWindow.BackgroundColor = hatter
GraphicsWindow.BrushColor = "red"
```

```
GraphicsWindow.PenColor = "black"
eszel = lszel
emag = lmag
For i = 1 To lepes
    GraphicsWindow.FillEllipse(kozepx - (eszel/2),kozepy - (emag/2) - hossz,esze
    GraphicsWindow.DrawEllipse(kozepx - (eszel/2),kozepy - (emag/2) - hossz,esze
    hg = emag
    emag = emag - res
    res = res + (res * 0.35)
    If Math.Remainder(i,2) = 0 Then
        GraphicsWindow.BrushColor = "Red"
        GraphicsWindow.PenColor = "Black"
    Else
        GraphicsWindow.BrushColor = "White"
        GraphicsWindow.PenColor = "Black"
    EndIf
EndFor

'Léghajó megrajzolása
GraphicsWindow.BrushColor = "brown"
GraphicsWindow.FillRectangle(kozepx - 75,kozepy + (lmag/2) - hossz - 5,150,20)
GraphicsWindow.DrawRectangle(kozepx - 75,kozepy + (lmag/2) - hossz - 5,150,20)
GraphicsWindow.DrawLine(kozepx + 75, kozepy + (lmag/2) - hossz + 5, kozepx + 85

GraphicsWindow.BrushColor = "lightblue"
GraphicsWindow.FillRectangle(kozepx - 73,kozepy + (lmag/2) - hossz - 0,15,10)
GraphicsWindow.DrawRectangle(kozepx - 73,kozepy + (lmag/2) - hossz - 0,15,10)

GraphicsWindow.FillRectangle(kozepx - 47,kozepy + (lmag/2) - hossz - 0,30,10)
GraphicsWindow.DrawRectangle(kozepx - 47,kozepy + (lmag/2) - hossz - 0,30,10)

GraphicsWindow.FillRectangle(kozepx - 4,kozepy + (lmag/2) - hossz - 0,30,10)
GraphicsWindow.DrawRectangle(kozepx - 4,kozepy + (lmag/2) - hossz - 0,30,10)

GraphicsWindow.FillRectangle(kozepx + 38,kozepy + (lmag/2) - hossz - 0,30,10)
GraphicsWindow.DrawRectangle(kozepx + 38,kozepy + (lmag/2) - hossz - 0,30,10)

GraphicsWindow.FontName = "Courier New"
GraphicsWindow.FontSize = 36
pontok = "............."
megj = pontok + uzenet + pontok
i = 1

'Főciklus
While "true"
    i = i + 1
    If i >= Text.GetLength(megj) - Text.GetLength(pontok) Then
        i = 1
    EndIf
    prt = Text.GetSubText(megj,i,text.GetLength(pontok))
    textcolor = "black"
    MegjText()
    Program.Delay(100)
    textcolor = "white"
    MegjText()
    Porog()
EndWhile
```

```
'Fényreklám
Sub MegjText
    GraphicsWindow.BrushColor = textcolor
    lpoz = 24
    For j=0 To 6
        kar = Text.GetSubText(prt,j,1)
        betumeret = 21 + (j * 3)
        GraphicsWindow.FontSize = betumeret
        GraphicsWindow.DrawText(kozepx - szoveghossz+(lpoz),kozepy - hossz - ((
        lpoz = lpoz + (betumeret * 0.75)
    EndFor
    For j=7 To 1 Step -1
        kar = Text.GetSubText(prt,(Text.GetLength(prt)-j),1)
        betumeret = 21 + ((j - 1) * 3)
        GraphicsWindow.FontSize = betumeret
        GraphicsWindow.DrawText(kozepx - szoveghossz+(lpoz),kozepy - hossz - ((
        lpoz = lpoz + (betumeret * 0.75)
    EndFor
EndSub

'Propeller forgatása
Sub Porog
    GraphicsWindow.PenWidth = 0.5
    If PorogSwitch = 0 Then
        PorogSwitch = 1
        GraphicsWindow.BrushColor = hatter
        GraphicsWindow.PenColor = hatter
        GraphicsWindow.FillTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +
        GraphicsWindow.DrawTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +
        GraphicsWindow.FillTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +
        GraphicsWindow.DrawTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +

        GraphicsWindow.BrushColor = "darkgray"
        GraphicsWindow.PenColor = "black"
        GraphicsWindow.FillTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +
        GraphicsWindow.DrawTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +
        GraphicsWindow.FillTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +
        GraphicsWindow.DrawTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +
    Else
        PorogSwitch = 0
        GraphicsWindow.BrushColor = hatter
        GraphicsWindow.PenColor = hatter
        GraphicsWindow.FillTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +
        GraphicsWindow.DrawTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +
        GraphicsWindow.FillTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +
        GraphicsWindow.DrawTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +

        GraphicsWindow.BrushColor = "darkgray"
        GraphicsWindow.PenColor = "black"
        GraphicsWindow.FillTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +
        GraphicsWindow.DrawTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +
        GraphicsWindow.FillTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +
        GraphicsWindow.DrawTriangle((kozepx + 85),(kozepy + (lmag/2) - hossz +
    EndIf

EndSub
```



52. ábra: „Retroléghajó”

Természetesen a kapott eredményt nem lehet egy *Flash*-animációhoz hasonlítani – reményeink szerint a kurzus után a Flash *Action Scriptje* is egy-kettőre elsajátítható –, de érdemes megpróbálkozni a továbbfejlesztésével!

18. fejezet - Néhány általános programozási feladat

Érdemes megismerkednünk a különböző programozási feladatok megoldása során gyakran felmerülő, általános algoritmusok Small Basic-kódjaival, a későbbiekben ezek jó szolgálatot tehetnek.

Keresés

Minimális és maximális elem kiválasztása

Gyakori feladat, hogy egy halmazból az elemeket egyedi tulajdonságaik alapján kell kiválasztanunk. Egész számokat tartalmazó tömbből a legkisebb és a legnagyobb kiválasztása az alábbihoz hasonló szerkezetű program segítségével végezhető el:

```
'Maximális és minimális érték kiszámítása egy véletlen számokkal feltöltött 10 e

TextWindow.WriteLine("A számok:")

For i=1 To 10
    szamok[i]=Math.GetRandomNumber(10)
    TextWindow.WriteLine(szamok[i])
EndFor

min = szamok [1]
max = szamok [1]

For n = 1 To 10
    If min > szamok [n] Then
        min = szamok [n]
    EndIf
    If max < szamok [n] Then
        max = szamok [n]
    EndIf
EndFor

TextWindow.WriteLine("Maximális érték:" + max)
TextWindow.WriteLine("Minimális érték:" + min)
```

Lineáris és bináris keresés

Definiáljunk egy *x* keresendő értéket! Az alábbi algoritmusokkal arra szeretnénk választ találni, hogy x megtalálható-e az adott tömbben, s ha igen hányadik elemként. Az egyszerűség kedvéért megint csak véletlen számokkal dolgozunk.

Lineáris keresés tömbben

A lineáris keresés meglehetősen egyszerű módszer, rendezetlen (nem sorba rendezett) elemeket tartalmazó tömböknél is használható. Lényegében végigmegyünk a tömb elemein, s egyenként megvizsgáljuk az esetleges egyezést. Egy lehetséges megoldás:

```
'Véletlen szám lineáris keresése egy véletlen számokkal feltöltött tömbben

TextWindow.WriteLine("Ezek között a számok között keresünk:")

For i = 1 To 10
    szamok[i] = Math.GetRandomNumber(10)
    TextWindow.WriteLine(szamok[i])
EndFor

szam = Math.GetRandomNumber(10)
TextWindow.WriteLine("A keresett szám:" + szam)

elofordulas = 0

For n = 1 To 10
    If szamok[n] = szam Then
        TextWindow.WriteLine("A szám a(z) " + n + ". helyen szerepel.")
        elofordulas = elofordulas + 1
    EndIf
EndFor

If elofordulas = 0 Then
    TextWindow.WriteLine("A szám nem szerepel a tömbben.")
EndIf
```

53. ábra: Lineáris keresés tömbben

Rendezett tömböknél a lineáris keresés esetenként felesleges iterációkat jelent, hiszen ha a tömb rendezett, akkor elég csak addig folytatni a vizsgálatot, amíg a keresett elemnél nagyobb elemre találunk.

Bináris keresés

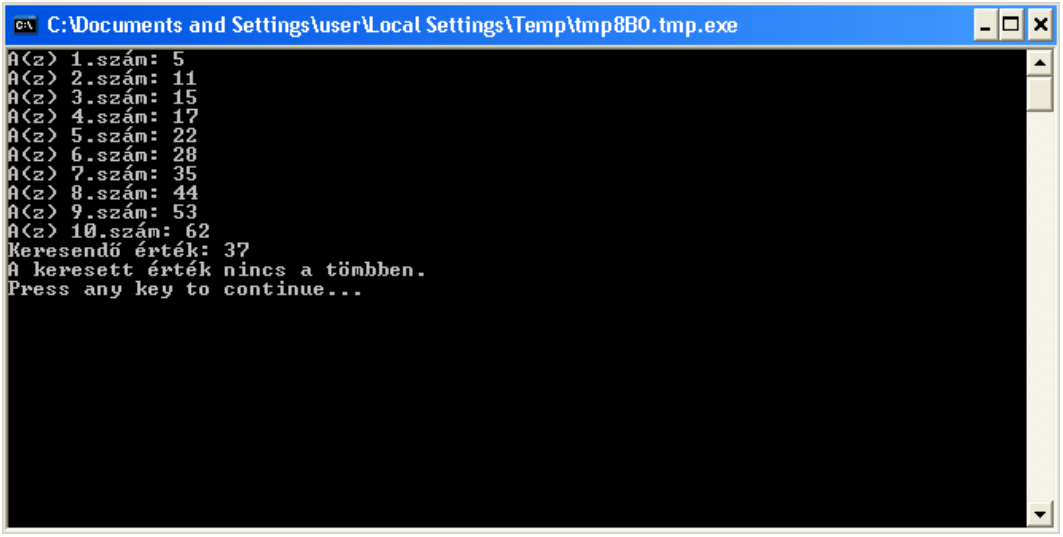
Bináris keresést rendezett tömbökben végezhetünk. Az egyszerűség kedvéért csak a növekvő sorrendbe rendezett tömbben keresünk, az eljárás a csökkenő sorrendbe rendezett tömbnél is hasonló.

```
'Bináris keresés rendezett tömbben
```

```
'Rendezett tömb előállítása véletlenszámokkal (1-100),
'a rendezettséget egy egyszerű hozzáadással biztosítjuk
kulonbseg = 0
For i = 1 To 10
    szamok[i] = Math.GetRandomNumber(10) + kulonbseg
    kulonbseg = szamok[i]
    TextWindow.WriteLine("A(z) " + i + ".szám: " + szamok[i])
EndFor

'Keresési érték
keresendo = Math.GetRandomNumber(szamok[10])
TextWindow.WriteLine("Keresendő érték: " + keresendo)

'A bináris keresés
also = 1
felso = 10
darab = 0
While also < felso + 1
    kozepso = Math.Floor((also + felso) / 2)
    If keresendo = szamok[kozepso] Then
        darab = darab + 1
        TextWindow.WriteLine("A szám a(z) " + kozepso + ". helyen szerepel.")
    EndIf
    If keresendo > szamok[kozepso] Then
        also = kozepso + 1
    Else
        felso = kozepso - 1
    EndIf
EndWhile

If darab = 0 Then
    TextWindow.WriteLine("A keresett érték nincs a tömbben.")
EndIf
```

A fenti, véletlen számokkal dolgozó programban a keresés során a keresési értéket először a tömb közepén elhelyezkedő értékkel hasonlítjuk össze. Ha a két érték egyenlő, a keresést befejezzük. Ha a keresett elem kisebb, mint a középső elem, akkor a több alsó felében keresünk tovább, ha magasabb, akkor a felső felében. A ciklus által megismételt eljárás előbb-utóbb – ha van egyező érték – elvezet a keresett elemhez.

A bináris keresés – nagy adathalmazoknál – elvileg gyorsabb, mint a lineáris keresés, ezért gyakori, ismétlődő keresésnél érdemes az adatainkat sorba rendezni. Természetesen a sorba rendezés is időt igényel, ezért a módszerek közötti választás gyakran nem egyértelmű.



54. ábra: Bináris keresés tömbben

Rendezés

Az adatok rendezésére többféle módszer ismert, érdemes többel megismerkedni, hiszen különböző helyzetekben az egyes algoritmusok használhatósága eltérhet egymástól. (A jegyzetben bemutatott rendezési módszereknél több módszer terjedt el.)

Beszúrásos rendezés

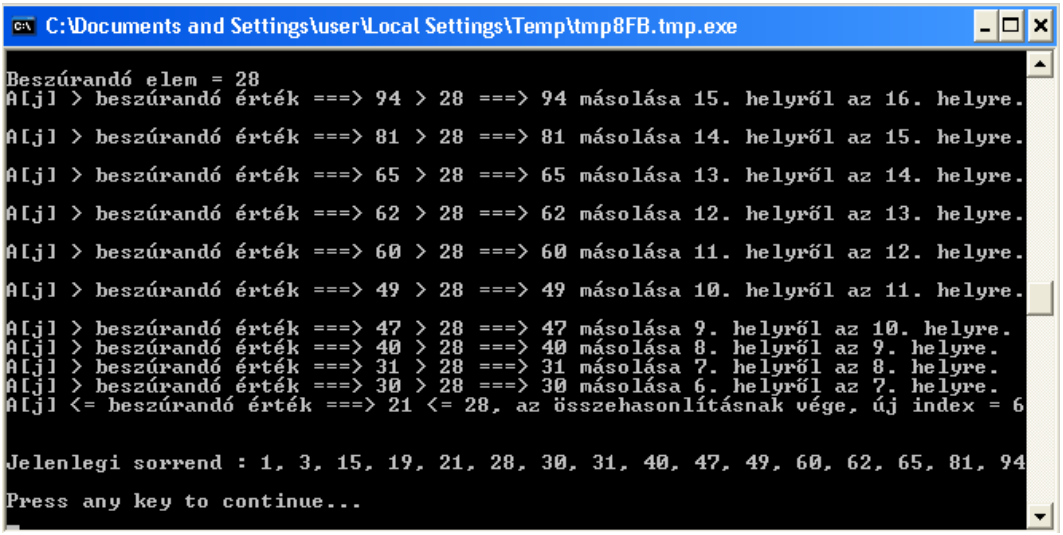
A beszúrásos rendezés során a rendezendő tömbünk első elemét egyelemű tömbnek tekintjük, amely nyilvánvalóan rendezett. Ezután felveszünk egy újabb elemet, s rendezzük a kételemű tömböt. Újabb és újabb elemekkel addig egészítjük ki és rendezzük a tömböt, amíg az eredeti tömbünk rendezett nem lesz. Amikor a rendezett résztömbhöz egy újabb elemet adunk, a hozzáadott elem miatt a tömbrészlet általában nem lesz rendezett.

Az új elem helyét ilyenkor az eredetileg rendezett résztömbben a korábban vázolt lineáris vagy bináris kereséssel kereshetjük meg, majd a keresett helyre beszúrhatjuk az elemet. A beszúrás során a rendezett tömbrészletben a beszúrásra váró elem „felett” elhelyezkedő elemeket eggyel feljebb toljuk, a „lejjebb” elhelyezkedőket eggyel lejjebb.

Példaprogram a beszúrásos rendezésre, lépésekre bontva:

```
Sub PrintList
     TextWindow.WriteLine("")
     TextWindow.Write("Jelenlegi sorrend : ")
     For l = 1 To Array.GetItemCount(A) - 1
         TextWindow.Write(A[l] + ", ")
     EndFor
     TextWindow.WriteLine(A[l])
EndSub

'Véletlenszámok
For d = 1 To Math.GetRandomNumber(10)+10 'Számok 10 és 20 között
     A[d] = Math.GetRandomNumber(100)
EndFor
PrintList()

For i = 2 To Array.GetItemCount(A) 'Kezdés a második a pozíciótól
```

```
        ertek = A[i]
        TextWindow.WriteLine("Beszúrandó elem = " + ertek)
        j = i - 1
        kesz = "False"
           While (kesz = "False")
           If (A[j] > ertek) Then
                TextWindow.WriteLine("A[j] > beszúrandó érték ===> " + A[j] + " >
               A[j + 1] = A[j]
               j = j - 1
               If (j <= 0) Then
                   TextWindow.WriteLine("Az érték az első pozícióban, az összehaso
                   kesz = "True"
               EndIf
           Else
               TextWindow.WriteLine("A[j] <= beszúrandó érték ===> " + A[j] + " <=
               kesz = "True"
           EndIf
       EndWhile
       A[j + 1] = ertek
       PrintList()
   EndFor
```



55. ábra: Beszúrásos rendezés

Buborékrendezés és ládarendezés

A buborékrendezés során a tömb első eleméről a vége felé haladva a szomszédos elemeket összehasonlítjuk, s ha a tömb elejéhez közelebb lévő elem nagyobb, mint az utána következő, akkor a két elem sorrendjét felcseréljük. Ha elértük a tömb végét, akkor a sorozat legnagyobb eleme a helyére, a tömb utolsó pozíciójába került. Ezután folyamatosan az utolsó elemmel rövidült, tehát eggyel kisebb elemszámmal rendelkező tömbön hajtjuk végre a buborékrendezést, amíg minden elem a helyére nem kerül (az utolsó végighaladáskor már nem történik csere).

A ládarendezés (valójában az elnevezés félrefordítás, a „leszámláló” rendezés angol neve alapján inkább fiókrendezés lenne helyes) során a tömbünk értékeivel indexelünk egy másik tömböt, a tömb minden egyes eleme egy-egy fiók lesz. Az új tömbünkben azt tároljuk, hogy az egyes értékekből hány darab szerepel az eredeti tömbünkben. A rendezés során végigmegyünk az új tömbön, és annyi elemet írunk a rendezett tömbünkbe, amennyi az egyes indexeknél darabszámként szerepel.

Összehasonlító program a beszúrásos, buborék- és ládarendezésre:

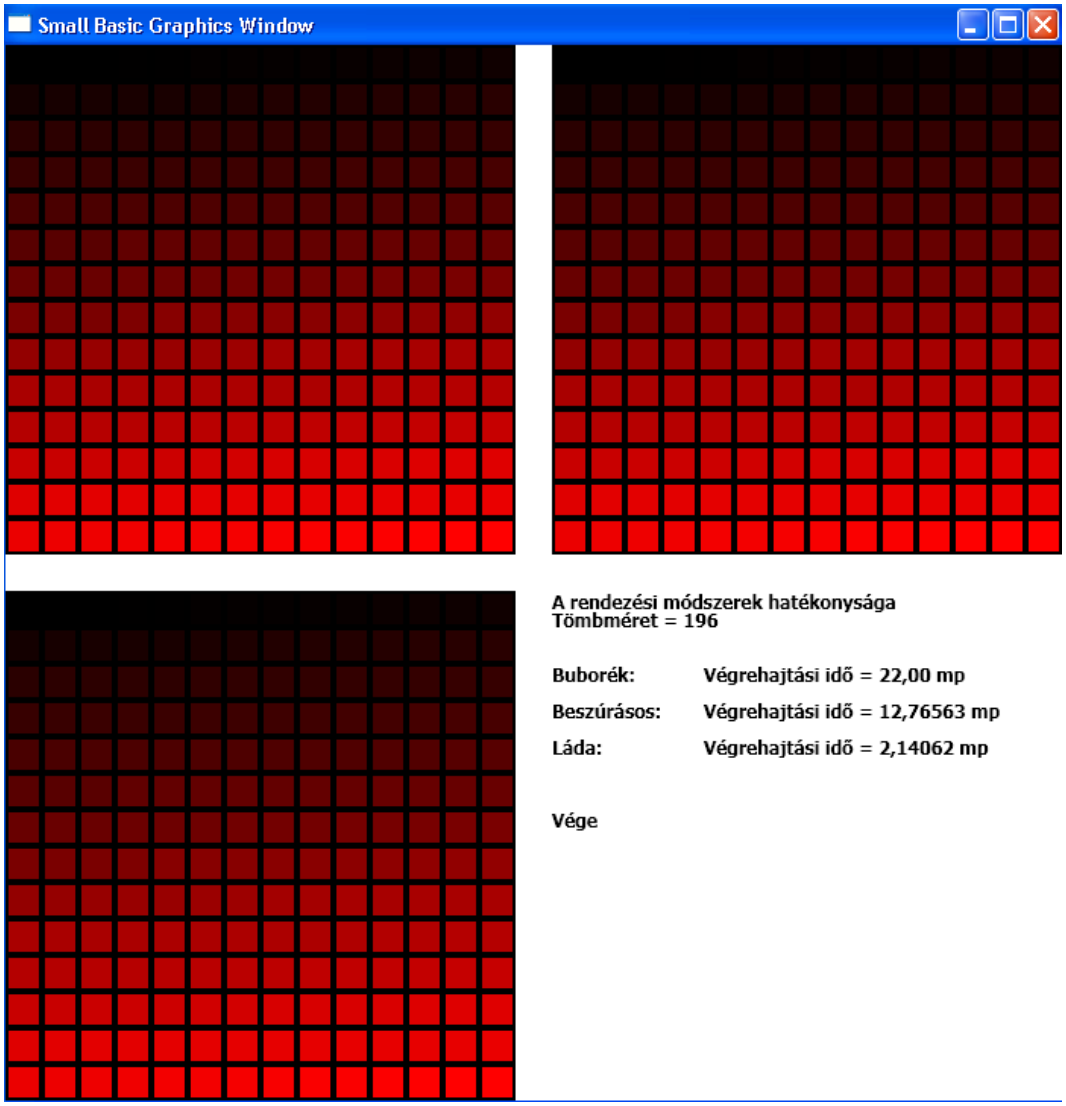
```
sorok = 14
oszlopok = 14
meret = 24
tavx = (oszlopok + 1) * meret
tavy = (sorok + 1) * meret
GraphicsWindow.Height = sorok * meret * 2 + meret
GraphicsWindow.Width = oszlopok * meret * 2 + meret
GraphicsWindow.Show()
GraphicsWindow.BackgroundColor = "White"

'Véletlen tömb kialakítása
For r = 0 To sorok - 1
    For c = 0 To oszlopok - 1
        voros = Math.GetRandomNumber(256) - 1
        zold = 0
        kek = 0
        GraphicsWindow.BrushColor = GraphicsWindow.GetColorFromRGB(voros,zold,k
        i = r * oszlopok + c
        negyzetek[i]["obj"] = Shapes.AddRectangle(meret,meret)
        negyzetek2[i]["obj"] = Shapes.AddRectangle(meret,meret)
        negyzetek3[i]["obj"] = Shapes.AddRectangle(meret,meret)
        Shapes.Move(negyzetek[i]["obj"], c * meret, r * meret)
        Shapes.Move(negyzetek2[i]["obj"],c * meret + tavx, r * meret)
        Shapes.Move(negyzetek3[i]["obj"],c * meret, r * meret + tavy)
        negyzetek[i]["val"] = kek + zold + voros
        negyzetek2[i]["val"] = kek + zold + voros
        negyzetek3[i]["val"] = kek + zold + voros
    EndFor
EndFor

tombhossz = i

GraphicsWindow.BrushColor = "black"
cim = "A rendezési módszerek hatékonysága"
GraphicsWindow.DrawText(tavx, tavy, cim)
tombmeretszoveg = "Tömbméret = " + (tombhossz + 1)
GraphicsWindow.DrawText(tavx, tavy + 0.5 * meret, tombmeretszoveg)

GraphicsWindow.DrawText(tavx, tavy + 2 * meret, "Buborék:")
kezdet = Clock.ElapsedMilliseconds
buborek()
bubido = Clock.ElapsedMilliseconds - kezdet
bubszoveg = "Végrehajtási idő = " + bubido/1000 + " mp"
GraphicsWindow.DrawText(tavx + 100, tavy + 2 * meret, bubszoveg)

GraphicsWindow.DrawText(tavx, tavy + 3*meret, "Beszúrásos:")
kezdet = Clock.ElapsedMilliseconds
beszuras()
beszido = Clock.ElapsedMilliseconds - kezdet
beszszoveg = "Végrehajtási idő = " + beszido / 1000 + " mp"
GraphicsWindow.DrawText(tavx+100, tavy + 3 * meret, beszszoveg)

GraphicsWindow.DrawText(tavx, tavy + 4 * meret, "Láda:")
kezdet = Clock.ElapsedMilliseconds
lada()
```

```
ladaido = Clock.ElapsedMilliseconds - kezdet
ladaszoveg = "Végrehajtási idő = " + ladaido / 1000 + " mp"
GraphicsWindow.DrawText(tavx + 100, tavy + 4 * meret, ladaszoveg)

GraphicsWindow.DrawText(tavx, tavy + 6 * meret, "Vége")
Sub lada
    inc = Math.Round(tombhossz/2)
    While inc > 0
        For i = inc To tombhossz
            atm = negyzetek3[i]["val"]
            atmdoboz = negyzetek3[i]["obj"]
            j = i
            while (j >= inc) and (negyzetek3[j-inc]["val"] > atm)
                negyzetek3[j]["val"] = negyzetek3[j-inc]["val"]
                negyzetek3[j]["obj"] = negyzetek3[j-inc]["obj"]
                r = Math.Floor((j) / oszlopok)
                c = Math.Remainder(j,oszlopok)
                Shapes.Move(negyzetek3[j]["obj"],c * meret, r * meret + tavy)
                j = j - inc
            EndWhile
            negyzetek3[j]["val"] = atm
            negyzetek3[j]["obj"] = atmdoboz
            r = Math.Floor((j)/oszlopok)
            c = Math.Remainder(j,oszlopok)
            Shapes.Move(negyzetek3[j]["obj"],c * meret r * meret + tavy)
        EndFor
        inc = Math.Round(inc / 2.2)
    Endwhile
Endsub

Sub beszuras
    For i = 1 to tombhossz
        ertek = negyzetek2[i]["val"]
        atmdoboz = negyzetek2[i]["obj"]
        j = i - 1
        kesz = "false"
        While kesz = "false"
            If negyzetek2[j]["val"] > ertek then
                negyzetek2[j + 1]["val"] = negyzetek2[j]["val"]
                negyzetek2[j+1]["obj"] = negyzetek2[j]["obj"]
                r = Math.Floor((j+1)/oszlopok)
                c = Math.Remainder(j+1,oszlopok)
                Shapes.Move(negyzetek2[j+1]["obj"],c * meret + tavx, r * meret)
                j = j - 1
                If j < 0 then
                    kesz = "true"
                EndIf
            Else
                kesz = "true"
            EndIf
        EndWhile
        negyzetek2[j+1]["val"] = ertek
        negyzetek2[j+1]["obj"] = atmdoboz
        r = Math.Floor((j + 1) / oszlopok)
        c = Math.Remainder(j + 1,oszlopok)
        Shapes.Move(negyzetek2[j+1]["obj"],c * meret + tavx, r * meret)
    EndFor
Endsub
```

```
Sub buborek
    imax = tombhossz
    cserelt = "true"
    While cserelt = "true"
        cserelt = "false"
        For bubi = 1 To imax
            If negyzetek[bubi]["val"] < negyzetek[bubi-1]["val"] Then
                csere()
            EndIf
        Endfor
    imax = imax - 1
    Endwhile
EndSub

Sub csere
    atmdoboz = negyzetek[bubi-1]["obj"]
    dobertatm = negyzetek[bubi-1]["val"]
    negyzetek[bubi-1]["obj"]=negyzetek[bubi]["obj"]
    negyzetek[bubi-1]["val"]=negyzetek[bubi]["val"]
    negyzetek[bubi]["obj"]=atmdoboz
    negyzetek[bubi]["val"]=dobertatm
    r = Math.Floor((bubi-1)/oszlopok)
    c = Math.Remainder(bubi-1,oszlopok)
    Shapes.Move(negyzetek[bubi-1]["obj"],c * meret, r * meret)
    r = Math.Floor((bubi)/oszlopok)
    c = Math.Remainder(bubi,oszlopok)
    Shapes.Move(negyzetek[bubi]["obj"],c * meret, r * meret)
    cserelt = "true"
EndSub
```

A program új elemei:

- A GraphicsWindow objektum *GetColorFromRGB(red, green, blue)* – „szín előállítása RGB-ből”, „vörös”, „zöld”, „kék” – művelete a vörös, a zöld és a kék szín erősségének (0-255) beállításával kikever egy színt.
- A rendszeridő-tulajdonságokat tartalmazó *Clock* objektum *ElapsedMilliseconds* – „eltelt milliszekundumok” – tulajdonsága megmutatja az 1900 óta eltelt milliszekundumok számát.
- A Math objektum *Round(number)* – „kerekít”, „szám” – művelete egészre kerekít egy számot.
- A Math objektum *Remainder(dividend, divisor)* – „maradék” „osztandó”, „osztó” – művelete visszaadja két szám hányadosának osztási maradékát.
- Az algoritmus különböző árnyalatú vörös kockákat generál véletlenszerűen, ezeket háromféleképpen sorba rendezi, az eredmény grafikusan jelenik meg.



56. ábra: Rendezési módszerek összehasonlítása

Euler-problémák

Leonhard Euler svájci matematikus tiszteletére a http://projecteuler.net/index.php?section=problems oldalon több száz olyan matematikai alapfeladat található, amelyeket programozási eszközökkel lehet megoldani. Az oldalon található feladatok minden – kezdő és haladó – programozó számára kiváló „agytornát” jelentenek.

Ismerkedjünk meg néhány feladat megoldásával!

1. probléma

Feladat: Írjuk fel 3 és 5 összes ezer alatti többszörösének összegét!

Program a probléma megoldására:

```
maxertek = 1000
szum = 0
```

```
For i = 1 To maxertek - 1 ' maxertek alatt
    If Math.Remainder(i, 3) = 0 Or Math.Remainder(i, 5) = 0 Then
        szum = szum + i
    EndIf
EndFor
TextWindow.WriteLine("3 és 5 összes 1000 alatti többszörösének összege: " + szu
```

2. probléma

Feladat: Írassuk ki a Fibonacci-sorozat páros értékeinek összegét egészen addig, amíg a legnagyobb érték meg nem haladja a négymilliót!

A Fibonacci-számok a matematikában az egyik legismertebb sorozat elemei. A sorozat első két eleme 0 és 1, a további elemeket az előző kettő összegeként kapjuk: 0, 1, 1, 2, 3, 5, 8, 13 …

(A természetben számos folyamat – pl. a pozsgások leveleinek osztódása – a Fibonacci-sorozat segítségével írható le.)

Program a probléma megoldására:

```
a = 1
b = 2
szum = 0
maxertek = 4000000

While a < maxertek
    If (Math.Remainder(a, 2) = 0) Then
        szum = szum + a
    EndIf
    tarolo = a
    a = b
    b = b + tarolo
EndWhile
TextWindow.WriteLine("A Fibonacci-sorozat négymilliónál nem nagyobb tagjainak ö
```

3. probléma

Feladat: Számoljuk ki egy összetett szám legnagyobb prímosztóját!

Az alábbi program egy összetett szám legnagyobb osztóját is kiírja a prímosztó mellett:

```
osszetett_szam = 263168 'Más természetes szám is választható
TextWindow.WriteLine("Természetes számunk = " + osszetett_szam)
maxoszto = Math.Floor(Math.SquareRoot(osszetett_szam))
TextWindow.WriteLine("Legnagyobb osztó = " + maxoszto)
TextWindow.WriteLine("A legnagyobb prímosztó számítása...")

If Math.Remainder(maxoszto, 2) = 0 Then
    maxoszto = maxoszto + 1
EndIf

For i = maxoszto To 3 Step -1
    If Math.Remainder(osszetett_szam, i) = 0 Then
        Primellenorzes()
```

```
        If prim = "True" Then
            Goto Befejezes
        EndIf
    EndIf
EndFor

Befejezes:
TextWindow.WriteLine("Legnagyobb prímosztó = " + i)

Sub Primellenorzes
    prim = "True"
    For j = 2 To Math.SquareRoot(i)
        If Math.Remainder(i, j) = 0 Then
            prim = "False"
            Goto Ciklusbolki
        EndIf
    EndFor
Ciklusbolki:
EndSub
```

4. probléma

Feladat: Keresse meg azt a legnagyobb palindromszámot, amely két darab kétjegyű szám szorzatával előállítható!

A palindromszám olyan természetes számot jelent, amely számjegyeit fordított sorrendben felírva az eredeti számot kapjuk.

Program a probléma megoldására:

```
maxpalindrom = 0

For i = 990 To 100 Step -11 ' A palindromszámok mindig 11 szorzatai
    For j = 999 To 100 Step -1
        szorzat = i * j
        Palindrom_ellenorzes()
        If vajonpalindrom = "True" Then
            TextWindow.WriteLine("Vizsgált palindromszám: " + szorzat + "...")
            If szorzat > maxpalindrom Then
                maxpalindrom = szorzat
                Goto Ciklusbolki
            EndIf
        EndIf
    EndFor
Ciklusbolki:
EndFor

TextWindow.WriteLine("Két háromjegyű szám szorzatából előálló legnagyobb palind

Sub Palindrom_ellenorzes
    vajonpalindrom = "False"
    szorzatforditva = ""
    hossz = Text.GetLength(szorzat)
    For k = 1 To hossz
        kerszoveg = Text.GetSubText(szorzat, k, 1)
        szorzatforditva = Text.Append(kerszoveg, szorzatforditva)
```

```
    EndFor
    If szorzatforditva = szorzat Then
        vajonpalindrom = "True"
    EndIf
EndSub
```

5. probléma

Feladat: Keresse meg azt a legkisebb számot, amely az összes 1 és 20 közötti számmal maradék nélkül osztható!

Program a probléma megoldására:

```
a = 1

For i = 1 To 6000000000 Step a
    j = 0
    l = 0
    For j = 20 To 10 Step -1 'Az 1 és 9 közötti számokat nem kell bevonni a  vi
        k = Math.Remainder(i,j)
        If k = 0 Then
            l = l + 1
            If l > 0 And l > b then
                b = l
                a = i
            EndIf
            If l = 10 Then
                Goto Befejezes
            EndIf
        Else
            Goto Ciklusbolki
        EndIf
    EndFor
    Ciklusbolki:
EndFor

Befejezes:
If l = 10 Then
    TextWindow.WriteLine(i + " az összes 1 és 20 közötti számmal maradék nélkül
EndIf
```

6. probléma

Feladat: Mennyi a különbség az első száz természetes szám négyzetének összege és az első száz természetes szám összegének négyzete között?

Az 12 + 22 + 32 + … + 1002 és az (1 + 2 + 3 + … + 100)2 összegek közötti különbségre vagyunk kíváncsiak.

Program a probléma megoldására:

```
maxertek = 100
szum = 0
negyzetek_osszege = 0
```

```
For i = 1 To Maxertek
    szum = szum + i
    negyzetek_osszege = negyzetek_osszege + i * i
EndFor

osszegek_negyzete = szum * szum
kulonbseg = osszegek_negyzete - negyzetek_osszege
TextWindow.WriteLine("Különbség = " + kulonbseg)
```

7. probléma

Feladat: Melyik a 10001. prímszám?

Program a probléma megoldására:

```
keresett_prim = 10001

For i = 2 To 200000
    Prim_ellenorzes()
    If prim_e = "True" Then
        k = k + 1
        If k = keresett_prim Then
            Goto Program_vege
        ElseIf Math.Remainder(K,10) = 0 Then
            TextWindow.WriteLine(i + " iteráció után eddig " + k + " prímet tal
        EndIf
    EndIf
EndFor

Sub Prim_ellenorzes
    prim_e = "True"
    For j = 2 To Math.SquareRoot(i)
        If Math.Remainder(i, j) = 0 Then
            prim_e = "False"
            Goto Ciklusbol_ki
        EndIf
    EndFor
    Ciklusbol_ki:
EndSub

Program_vege:
If k = keresett_prim Then
    TextWindow.WriteLine("A " + keresett_prim + ". prím: " + i)
    Else
        TextWindow.WriteLine("A program futása idő
        előtt véget ért, a keresett szám 200000 iteráció után sem
        számítható ki.")
EndIf
```

Természetesen a többi „Euler-probléma” megoldása is érdekes programozói feladat, de azt már az olvasóra bízzuk.

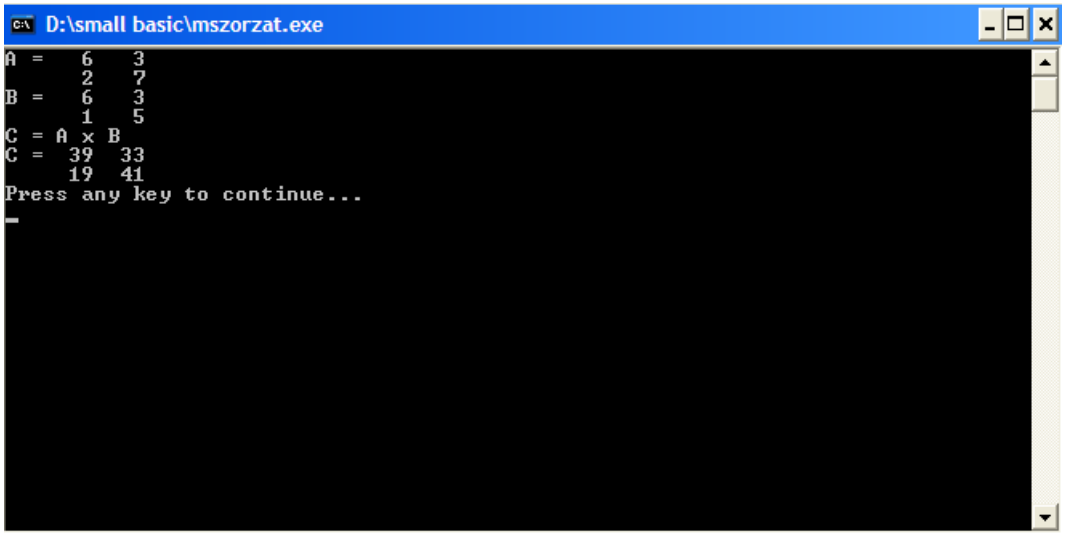
Mátrixalgebra

Mátrixalgebrai műveleteket a kiegészítések nélküli Small Basic-kel is végezhetünk, bár bonyolultabb műveleteknél összetett programozói fogások alkalmazására lehet szükség.

A következő egyszerű program két, két sorból és oszlopból álló, véletlenszámokkal feltöltött mátrixot szoroz össze:

```
max_szamjegy = 3
szoveg = " "
n = 2

Veletlen_AB_matrix_letrehozasa()
celmatrix = "A"
Matrix_felirasa()
celmatrix = "B"
Matrix_felirasa()

a = "A"
b = "B"
TextWindow.WriteLine("C = A x B")
c = "C"
Szorzat()
celmatrix = "C"
Matrix_felirasa()

Sub Veletlen_AB_matrix_letrehozasa
    For i = 1 To n
        For j = 1 To n
            M["A"+Text.Append(i, j)] = Math.GetRandomNumber(9)
            M["B"+Text.Append(i, j)] = Math.GetRandomNumber(9)
        EndFor
    EndFor
EndSub

Sub Matrix_felirasa
    For i = 1 To n
        If i = 1 Then
            TextWindow.Write(celmatrix + " = ")
        Else
            TextWindow.Write(" ")
        EndIf
        For j = 1 To n
            mij = M[celmatrix + Text.Append(i, j)]
            l = Text.GetLength(Mij)
            hossz = max_szamjegy - l
            TextWindow.Write(Text.GetSubText(szoveg, 1, hossz) + mij + " ")
        EndFor
        TextWindow.WriteLine("")
    EndFor
EndSub

Sub Szorzat
    For i = 1 To n
        For j = 1 To n
        cij = 0
```

```
            For k = 1 To n
                aik = M[a + Text.Append(i, k)]
                bkj = M[b + Text.Append(k, j)]
                cij = cij + aik * bkj
            EndFor
            M[c + Text.Append(i, j)] = cij
        EndFor
    EndFor
EndSub
```



57. ábra: Mátrixszorzat

A mátrixok méretének és az elvégzendő mátrixműveletek számának növekedésével párhuzamosan azonban az alapváltozat által biztosított lehetőségek egyre szűkösebbnek tűnhetnek.

Ha gyakran dolgozunk mátrixokkal, érdemes a korábban már bemutatott LitDev kiegészítést telepíteni. A LitDev *LDMatrix* objektumának műveletei ugyanis tartalmazzák a legfontosabb mátrixműveleteket, s ezzel megkönnyítik a munkánkat.

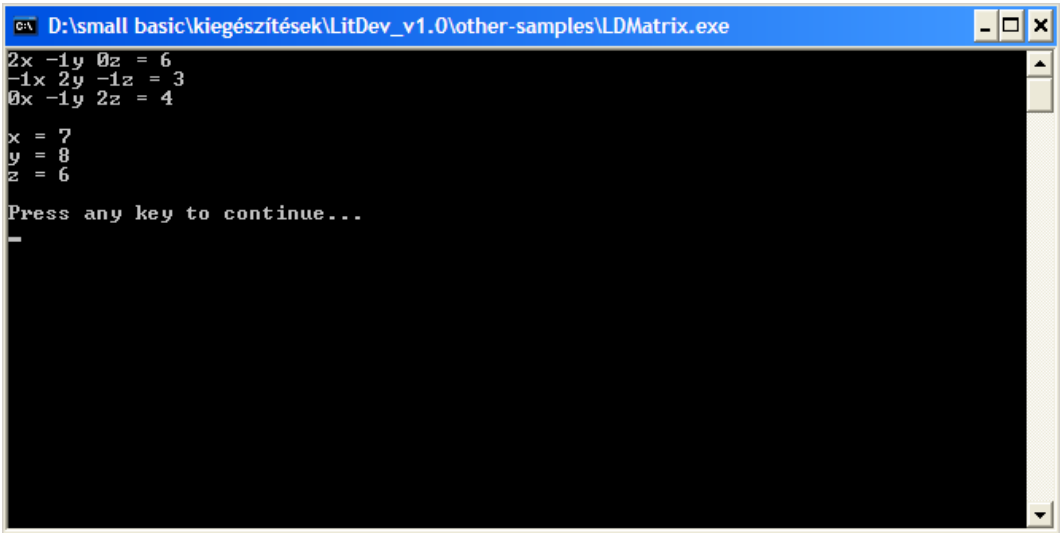
A következő, LitDev kiegészítéssel készített program kiszámolja egy háromváltozós, három egyenletből álló, elsőfokú egyenletrendszer gyökeit:

```
matrix = LDMatrix.CreateMatrix(3,3)
inverz = LDMatrix.CreateMatrix(3,3)
baloldali_matrix = LDMatrix.CreateMatrix(3,1)
jobboldali_matrix = LDMatrix.CreateMatrix(3,1)

LDMatrix.SetValue(matrix, 1, 1, 2)
LDMatrix.SetValue(matrix, 1, 2, -1)
LDMatrix.SetValue(matrix, 1, 3, 0)
LDMatrix.SetValue(matrix, 2, 1, -1)
LDMatrix.SetValue(matrix, 2, 2, 2)
LDMatrix.SetValue(matrix, 2, 3, -1)
LDMatrix.SetValue(matrix, 3, 1, 0)
LDMatrix.SetValue(matrix, 3, 2, -1)
LDMatrix.SetValue(matrix, 3, 3, 2)

LDMatrix.SetValue(jobboldali_matrix, 1, 1, 6)
LDMatrix.SetValue(jobboldali_matrix, 2, 1, 3)
```

```
LDMatrix.SetValue(jobboldali_matrix, 3, 1, 4)

LDMatrix.Inverse(matrix,inverz)
LDMatrix.Multiply(inverz,jobboldali_matrix,baloldali_matrix)

var[1] = "x"
var[2] = "y"
var[3] = "z"
For i = 1 To 3
    For j = 1 To 3
        TextWindow.Write(LDMatrix.GetValue(matrix, i, j)+var[j]+" ")
    EndFor
    TextWindow.Write("= "+LDMatrix.GetValue(jobboldali_matrix, i, 1))
    TextWindow.WriteLine("")
EndFor
TextWindow.WriteLine("")
For i = 1 To 3
    TextWindow.WriteLine(var[i]+" = "+LDMatrix.GetValue(baloldali_matrix, i, 1)
EndFor
TextWindow.WriteLine("")
```

Az LDMatrix objektum felhasznált műveletei:

- A *CreateMatrix(rows, cols)* – „mátrix létrehozása”, „sorok”, „oszlopok” – művelet létrehoz egy mátrixot. A megfelelő paramétert egynek véve sor- és oszlopvektor is definiálható.
- A *SetValue(matrixName, row, col, value)* – „érték megadása”, „mátrix neve”, „sor”, „oszlop”, „érték” – művelet segítségével a mátrix egyes elemeit tölthetjük fel értékekkel.
- Az *Inverse(matrix, inverse)* – „inverz”, „mátrix”, „inverz” – művelet megadja egy mátrix inverzét.
- A *Multiply(matrix1, matrix2, result)* – „mátrixszorzat”, „első mátrix”, „második mátrix”, „eredménymátrix” – művelet mátrixokat szoroz össze.
- *A GetValue(matrixName, row, col)* – „értéket visszaad”, „mátrix neve”, „sor”, „oszlop” – visszaad egy mátrix adott sorában és oszlopában lévő értéket.



58. ábra: Elsőfokú egyenletrendszer megoldása

Statisztikai számítások

A Small Basic alaprendszerével megfelelő programozói tudással felvértezve lényegében minden statisztikai művelet megoldható. Azonban – az előző esethez hasonlóan – ha elsősorban a statisztikai problémákra koncentrálunk, s a programozást inkább csak eszközként kezeljük, érdemes a Small Basic statisztikai elemeket is tartalmazó kiegészítéseit használni.

A LitDev kiegészítés egyszerűbb statisztikai műveletekre kiváló választás.

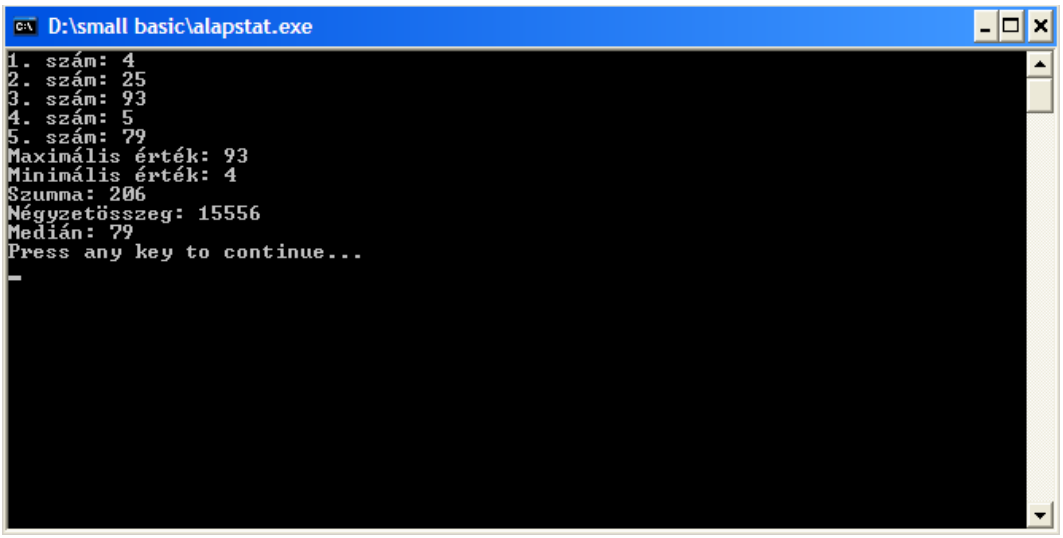
Az alábbi példaprogram néhány statisztikai mutatót számít ki:

```
For i = 1 To 5
    A[i] = Math.GetRandomNumber(100)
    TextWindow.WriteLine(i + ". szám: " + A[i])
EndFor

LDStatistics.SetArray(A)

max = LDStatistics.Max
TextWindow.WriteLine("Maximális érték: " + max)
min = LDStatistics.Min
TextWindow.WriteLine("Minimális érték: " + min)
szum = LDStatistics.Sum
TextWindow.WriteLine("Szumma: " + szum)
szum2 = LDStatistics.Sum2
TextWindow.WriteLine("Négyzetösszeg: " + szum2)
median = LDStatistics.Median
TextWindow.WriteLine("Medián: " + median)
```

A program új elemei:

- Az *LDStatistics* objektum *SetArray(array)* – „tömb beállítása”, „tömb” –művelete meghatározza, hogy melyik tömb az éppen aktuálistömb, amelyre a statisztikai műveletek végrehajthatók, illetve amelynek a tulajdonságai lekérdezhetők.
- Az LDStatistics objektum *Min, Max, Sum, Sum2, Median* tulajdonságai az aktuális tömb minimum, maximum, számtani összeg, négyzetösszeg és medián értékeit tartalmazzák.

Az LDStatistics objektum műveletei és tulajdonságai a felsoroltaknál jóval összetettebb statisztikai mutatók számítását is lehetővé teszi.



59. ábra: Egyszerű statisztikai mutatók számítása

Animációs alapproblémák

Nem tekinthetünk el attól az igénytől, hogy a hallgatók grafikailag látványosabb, animált elemeket tartalmazó programokat szeretnének írni. Ehhez természetesen esztétikai érzékre és sok-sok kreativitásra van szükség, de talán néhány alapvető eljárás bemutatása is hasznára lehet a leendő fejlesztőknek.

Bár a Small Basic alapváltozata is rengeteg lehetőséget nyújt pl. játékprogramok írásához (nézünk erre még majd példákat), a következő alfejezetekben bemutatott néhány alapvető, egyszerű animációs eljárásnál a LitDev kiegészítés nyújtotta előnyökkel élünk.

Pattogó labda

Az alábbi, LitDev-műveleteket használó program egy labda pattogását szimulálja:

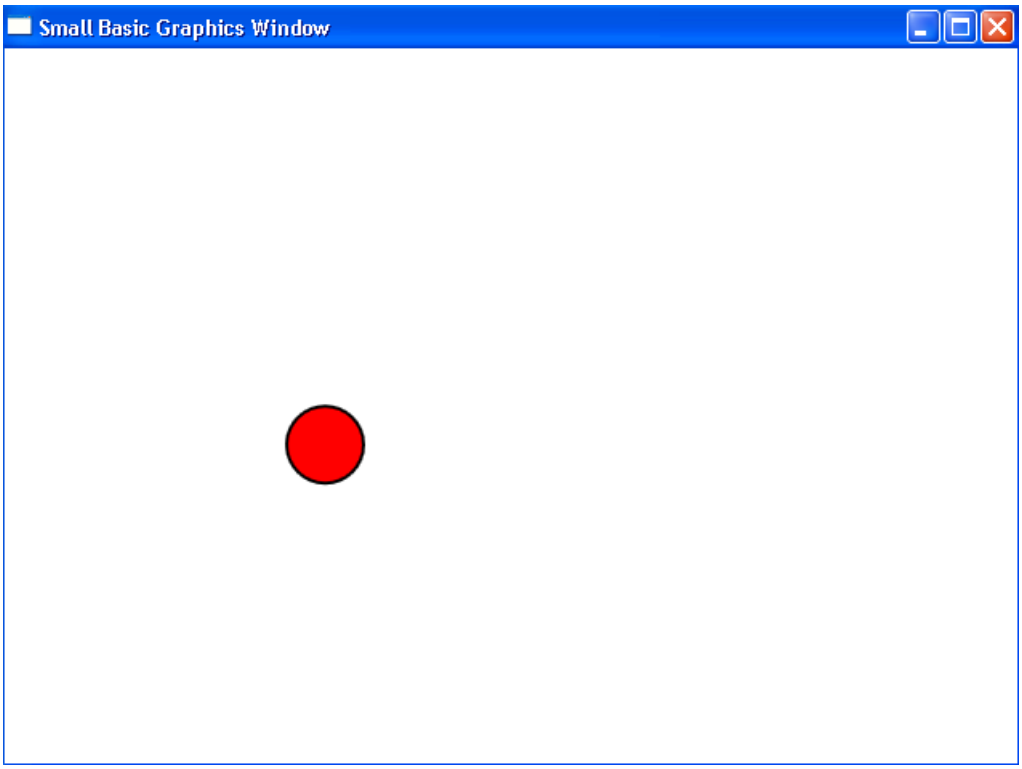
```
GraphicsWindow.BrushColor = "Red"
labda = Shapes.AddEllipse(50,50)
LDPhysics.AddMovingShape(labda,0,1,1)
LDPhysics.SetPosition(labda,200,100,0)
GraphicsWindow.ShowMessage("A befejezéshez nyomja le a V billentyűt", "Üzenet")

While "True"
    LDPhysics.DoTimestep()
    Program.Delay(20)
    If GraphicsWindow.LastKey = "V" Then
        Program.End()
        EndIf
EndWhile
```

A program új elemei:

- Az *LDPhysics* objektum *AddMovingShape(shapeName, friction, restitution, density)* – „mozgó alakzat hozzáadása”, „alakzat neve”, „súrlódás”, „visszapattanás”, „tömörség” – művelet egy mozgó alakzatot definiál. A labda mozgását esetünkben leginkább a restitution (visszapattanás) értéke befolyásolja. Ha a legnagyobb (1) érték helyett 0-t választanánk, a labda nem pattanna vissza a földről. 0.9-es érték mellett pedig fokozatosan kisebbeket pattogva állna meg a talajon. A friction

(súrlódás) szintén 0 és 1 között változó értéke a másik testhez történő súrlódáskor bekövetkező lassulás értékét állítja be.

- A LDPhysics objektum *SetPosition(shapeName, posX, posY, angle)* – „pozíció beállítása”, „x koordináta”, „y koordináta”, „elforgatás szöge” – művelete az alakzat pozícióját állítja be.
- Miután az egyes objektumokat – esetünkben egyet – definiáltuk, az animációért lényegében *DoTimeStep()* művelet lesz felelős. A művelet meghívása után egy „időegység” eltelik, s az objektumok frissítik a pozíciójukat. A folyamatos frissítést mi egy végtelen ciklussal értük el, de emellett egyszerű példát adtunk a ciklus (és a program) megszakítására is.



60. ábra: Pillanatfelvétel a pattogó labdáról

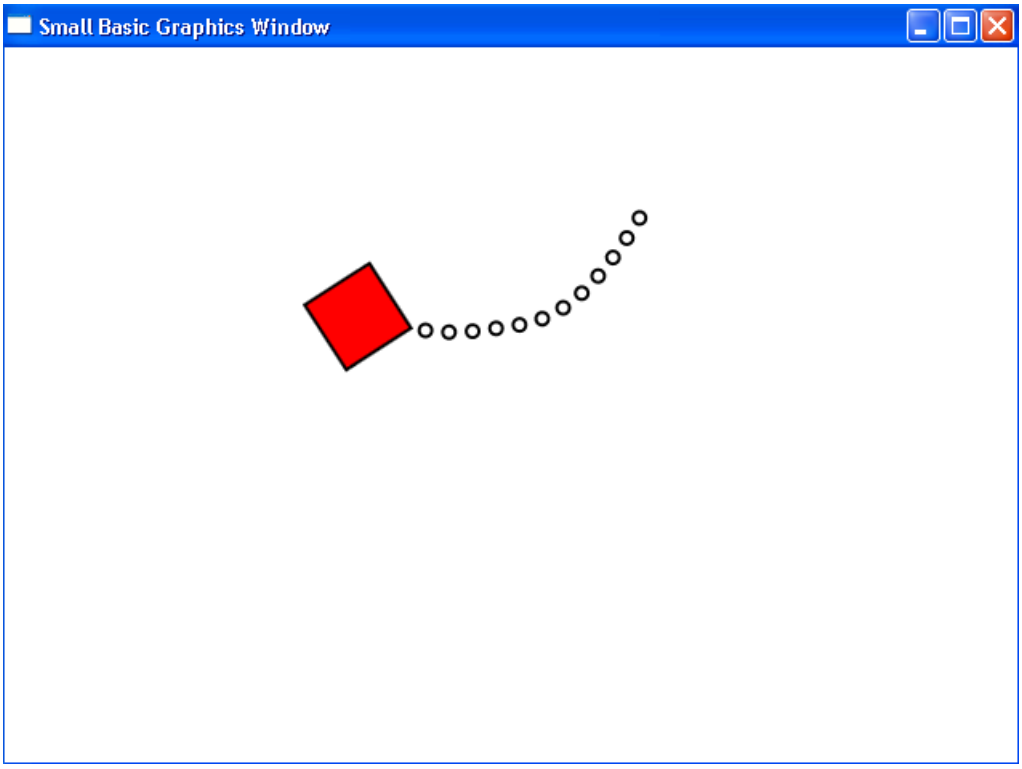
Felfüggesztett doboz

Az alábbi program egy láncra felfüggesztett doboz mozgását szimulálja:

```
GraphicsWindow.BrushColor = "Red"
doboz = Shapes.AddRectangle(50,50)
LDPhysics.AddMovingShape(doboz,0,1,1)
LDPhysics.SetPosition(doboz,200,100,0)

forgoresz1 = LDPhysics.AddMovingAnchor(225,125)
LDPhysics.AttachShapes(forgoresz1,doboz)
forgoresz2 = LDPhysics.AddFixedAnchor(400,100)
LDPhysics.AddChain(forgoresz1,forgoresz2)

While "True"
    LDPhysics.DoTimestep()
    Program.Delay(20)
EndWhile
```

A program új elemei:

- Az LDPhysics objektum *AddMovingAnchor(posX, posY)* – „mozgó forgórész hozzáadása”, „x koordináta”, „y koordináta” – művelet egy nem csak forgásra, hanem többirányú mozgásra is képes elemet ad hozzá a grafikus ablakhoz.
- Az LDPhysics objektum *AddFixedAnchor(posX, posY)* – „mozgó forgórész hozzáadása”, „x koordináta”, „y koordináta” – művelet egy csak forgásra képes elemet ad hozzá a grafikus ablakhoz.
- Az LDPhysics objektum *AttachShapes(shape1, shape2)* – „alakzatok összekapcsolása”, „első alakzat”, „második alakzat” – művelete összekapcsol két alakzatot, amelyek az összekapcsolás után együtt mozognak.
- Az LDPhysics objektum *AddChain(shape1, shape2)* – „lánc hozzáadása”, „első alakzat”, „második alakzat” – művelete rugalmas lánccal kapcsol össze két alakzatot.



61. ábra: Pillanatfelvétel a lendületet vevő „dobozról”

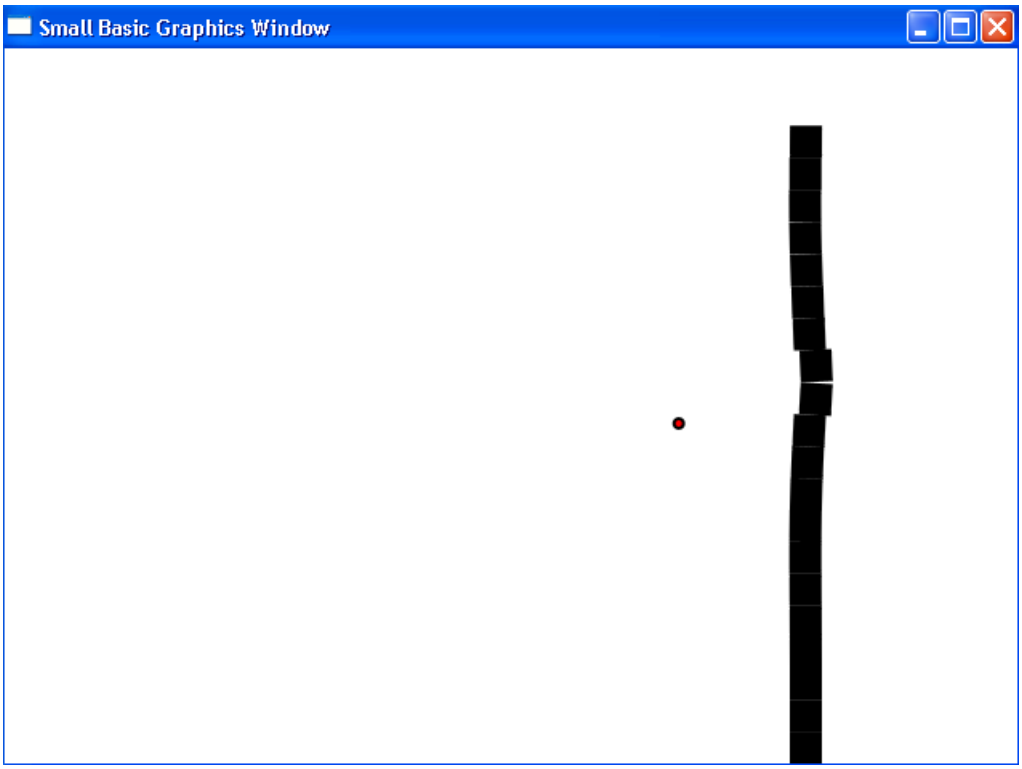
Több alakzat mozgása-ütközése

Az alábbi programban egy nagy sebességgel mozgó, rugalmas golyó összedönt egy dobozokból felépített alakzatot:

```
For i = 1 To 20
    GraphicsWindow.BrushColor = "Black"
    doboz[i] = Shapes.AddRectangle(20,20)
    LDPhysics.AddMovingShape(doboz[i],0.3,0.8,1)
    LDPhysics.SetPosition(doboz[i],500,GraphicsWindow.Height-20*i+10,0)
EndFor

GraphicsWindow.BrushColor = "Red"
golyo = Shapes.AddEllipse(8,8)
```

```
LDPhysics.AddMovingShape(golyo,0,1,1)
LDPhysics.SetPosition(golyo,50,200,0)
LDPhysics.SetVelocity(golyo,1000,0)

While ("True")
    LDPhysics.DoTimestep()
    Program.Delay(20)
EndWhile
```

A program új eleme a *LDPhysics objektum SetVelocity(shapeName, velX, velY)* – „sebesség beállítása”, „alakzat neve”, „sebesség az x tengely mentén”, „sebesség y tengely mentén” – művelet, amely egy objektum sebességét határozza meg.



62. ábra: Az első ütközés után

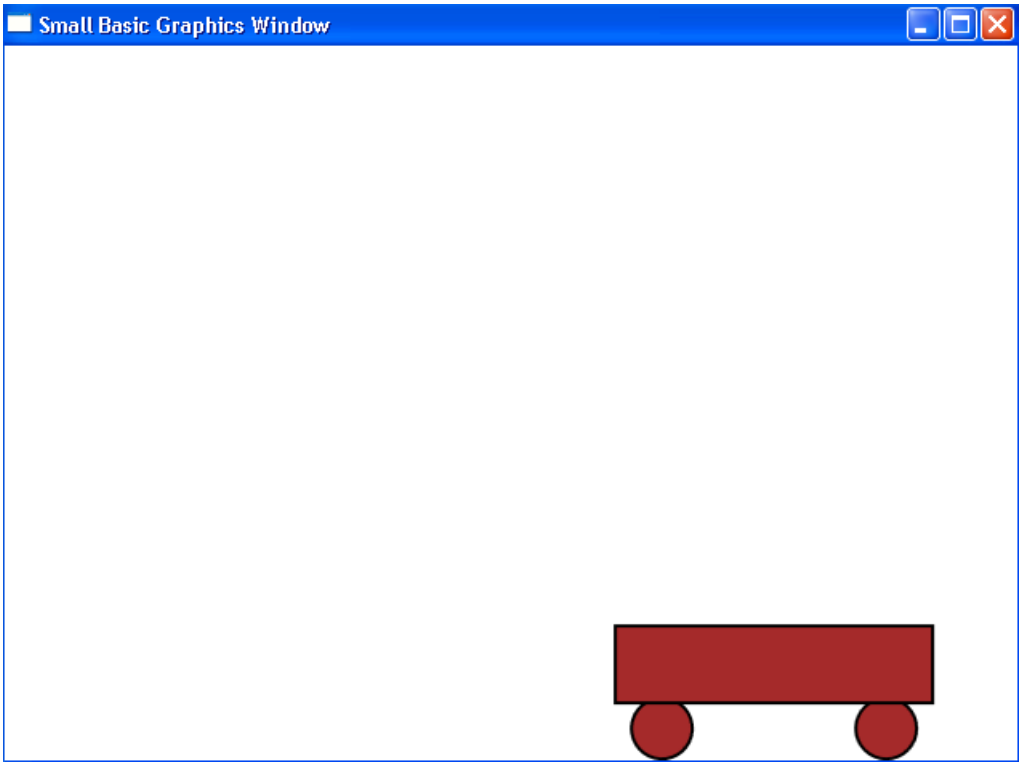
Mozgás és ütközés sík felületen

Az alábbi programban egy sematizált kocsi rugalmasan visszapattan a falról, majd lassulva leáll:

```
GraphicsWindow.BrushColor = "Brown"

kerek1 = Shapes.AddEllipse(40,40)
kerek2 = Shapes.AddEllipse(40,40)
kocsi = Shapes.AddRectangle(200,50)

LDPhysics.AddMovingShape(kerek1,1,0,1)
LDPhysics.AddMovingShape(kerek2,1,0,1)
LDPhysics.SetPosition(kerek1,180,400,0)
LDPhysics.SetPosition(kerek2,320,400,0)
LDPhysics.AddMovingShape(kocsi,0.3,0.5,1)
```

```
LDPhysics.SetPosition(kocsi,250,360,0)
LDPhysics.AttachShapesWithRotation(kerek1,kocsi)
LDPhysics.AttachShapesWithRotation(kerek2,kocsi)
LDPhysics.SetVelocity(kocsi,100,0)

While ("True")
    LDPhysics.DoTimestep()
    Program.Delay(20)
EndWhile
```

A program új eleme az LDPhysics objektum *AttachShapesWithRotation(shape1, shape2)* – „alakzatok összekapcsolása elfordulással”, „első alakzat”, „második alakzat” –művelete, amely nemcsak összekapcsol két alakzatot, hanem az egymás körüli elfordulást is lehetővé teszi.



63. ábra: Pillanatfelvétel a sík felületen mozgó „kocsiról”

Mozgó ballonra felfüggesztett mozgó tárgy

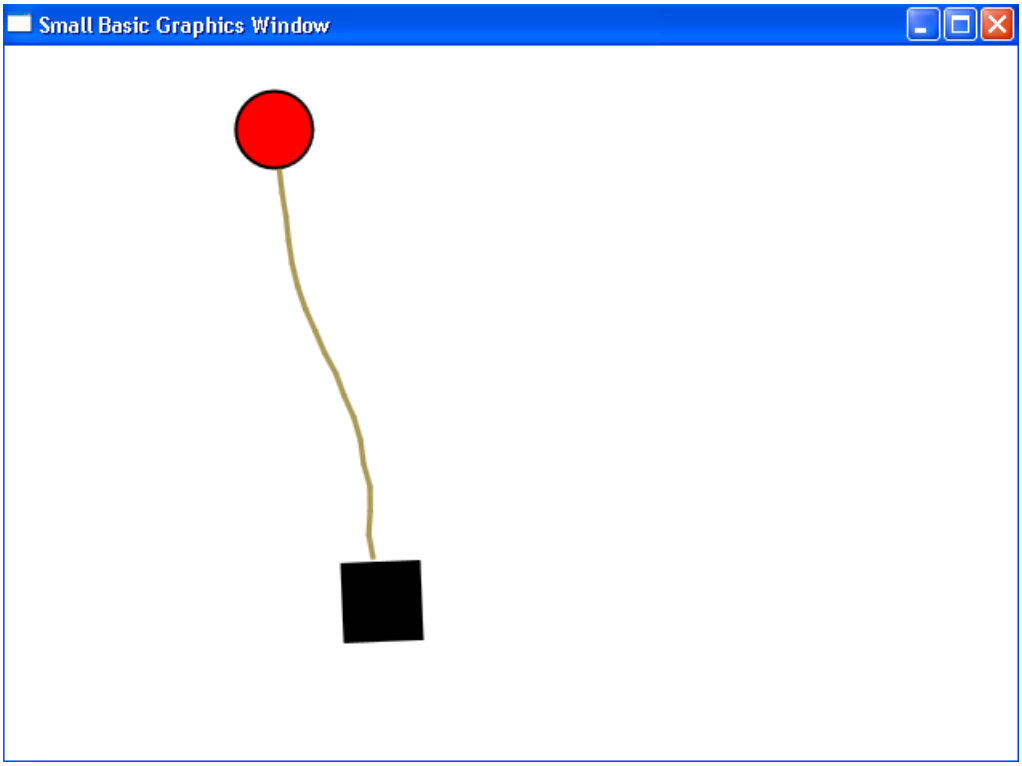
Az alábbi program egy léggömb és a léggömbre felfüggesztett doboz együttes mozgását szimulálja:

```
GraphicsWindow.BrushColor = "Red"
leggomb = Shapes.AddEllipse(50,50)
GraphicsWindow.BrushColor = "Black"
doboz = Shapes.AddRectangle(50,50)

LDPhysics.AddMovingShape(leggomb,0.3,0.5,1)
LDPhysics.SetPosition(leggomb,100,100,0)
LDPhysics.AddMovingShape(doboz,0.3,0.5,1)
LDPhysics.SetPosition(doboz,300,300,0)

leggomb_forgoresz = LDPhysics.AddMovingAnchor(100,125)
```

```
LDPhysics.AttachShapesWithRotation(leggomb,leggomb_forgoresz)
doboz_forgoresz = LDPhysics.AddMovingAnchor(300,275)
LDPhysics.AttachShapesWithRotation(doboz,doboz_forgoresz)

kotel = LDPhysics.AddRope(leggomb_forgoresz,doboz_forgoresz)
doboz_tomeg = LDPhysics.GetMass(doboz)
leggomb_tomeg = LDPhysics.GetMass(leggomb)
leggomb_forgoresz_tomeg = LDPhysics.GetMass(leggomb_forgoresz)
doboz_forgoresz_tomeg = LDPhysics.GetMass(doboz_forgoresz)
kotel_tomeg = LDPhysics.GetMass(kotel)

ossztomeg = leggomb_tomeg + doboz_tomeg + kotel_tomeg + leggomb_forgoresz_tomeg
While ("True")
    LDPhysics.SetForce(leggomb,0,-100*ossztomeg)
    LDPhysics.DoTimestep()
    Program.Delay(20)
EndWhile
```

A program új elemei:

- Az LDPhysics objektum *AddRope(shape1, shape2)* – „kötelet hozzáad”, „első alakzat”, „második alakzat” – művelete két objektumot kapcsol össze egy „kötéllel”.
- Az LDPhysics objektum *GetMass(shapeName)* – „tömeget meghatároz”, „alakzat neve” – művelete „elméleti” kilogrammban meghatározza egy alakzat „tömegét”.
- Az LDPhysics objektum *SetForce(shapeName, forceX, forceY)* – „gyorsulási erő meghatározása”, „gyorsulás x tengely mentén”, „gyorsulás y tengely mentén” – művelete meghatározza egy alakzat gyorsulási erejét.



64. ábra: Pillanatfelvétel a „léggömb” és a vele összekapcsolt alakzat mozgásáról

A határ a „csillagos ég”

A könyvben bemutatott alapvető programozási ismeretek és technikák sok gyakorlással, különböző programozási feladatok önálló megoldásával, a későbbiekben esetleg új programozási nyelvek megtanulásával véleményünk szerint bárkit elindíthatnak a profi programozóvá válás útján. Természetesen nem kell mindenkinek erre törekednie, de az algoritmusépítés alapjai – mint ahogy azt a bevezető részben már kifejtettük – számos területen hasznosak lehetnek nemcsak a professzionális fejlesztők számára.

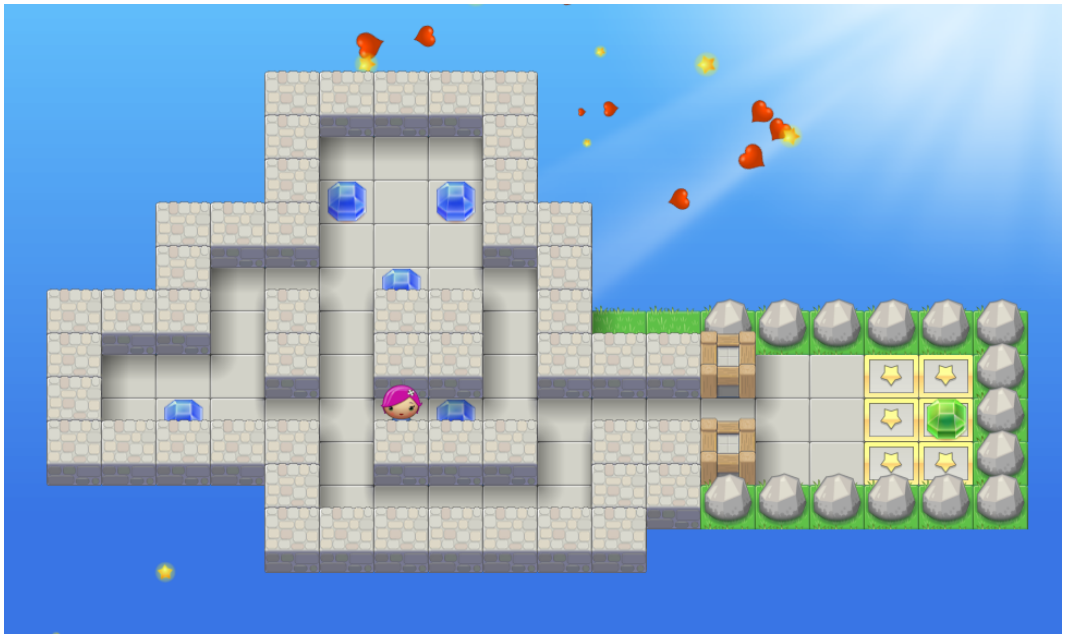
Akik kedvet kaptak a programozáshoz, az első időszakban próbálják kihasználni a Small Basic alapváltozata által nyújtott lehetőségeket. Természetesen különböző kiegészítésekkel sok esetben könnyebb megoldani az egyes feladatokat, azonban a kezdetben több nehézséggel járó út követése – az alapváltozat használata minden problémára – hosszútávon gyümölcsöző lehet. Korlátozott programnyelvi elemekkel a programozó ugyanis sokkal kreatívabb munkára kényszerül, s az így elsajátított algoritmusépítési készségek más programnyelvi környezetben is megkönnyíthetik a fejlesztést.

A Small Basic alaprendszerének használata – a pedagógiai megfontolásokon túl – sok sikerélményt tartogat a fejlesztők számára. Ezt bizonyítandó, érdemes – a megadott kód segítségével – importálni, kipróbálni és továbbfejleszteni vagy átalakítani (akár csak magyar nyelvűvé) az alábbi, alaprendszerrel készített programokat. Ne feledjük, a felsorolt programokat nem professzionális programozók vagy cégek készítették, a programkód nyilvános és a kívánt mértékben átalakítható! Ne a kereskedelmi szoftverek szintjéhez mérjük őket, hanem a kezdő programozók tudásához!

A hallgatók figyelmébe ajánlott programok:

- Ébresztőóra: MDW647,
- Animált labdák: ZLJ620,
- Képgaléria: PPG822 (a képeket érdemes saját képekre cserélni),
- Életjáték: RVR167 (az egyik legrégebbi programozási feladat),
- Második világháborús légi csata: ZZD394 (a saját gépünket egérrel irányítjuk, s az egérgombokkal tüzelünk),
- Harc az aszteroidákkal: ASTEROIDS (az űrhajót a kurzorbillentyűkkel irányítjuk, és a Space billentyű lenyomásával tüzelünk),
- Bűnügyi nyomozás: DWK528 (szöveges játék, csak angolul tudóknak ajánljuk),
- Gorillák: NLQ667-3 (két gorilla banánokkal dobálja egymást nagyvárosi háztetőkön, dobás előtt a dobás szögét és a sebességet kell megadnunk),
- Aknakereső: DBM434 (nagyon hasonlít a Windows operációs rendszerekhez mellékelt játékhoz),
- Számkitalálós logikai játék: CJC272 (szöveges játék, de kevés angol nyelvtudást igényel),
- Labdagyűjtés: KXW612,
- Puzzle: LJH960,
- Kő-papír-olló: CMJ212 (az általunk korábban bemutatottnál kifinomultabb),
- Menekülés a kígyó elől: SNKBITE,
- „Soko”: SOKO-92 (kurzorral mozogva megfelelő helyre kell tolni a gyémántokat),
- „Stargates”: STARGATES (kurzorral mozogva Space-szel tüzelve játszhatunk a nyolcvanas évek népszerű videójátékához hasonló programmal),

- „Ütögetés” négy ütővel: MDJ923 (a jegyzetben található program továbbfejlesztett verziója)
- Hanoi tornyai: HANOI (igazi agytorna, a játékszabályokról számos helyen tájékozódhatunk).



65. ábra: A SOKO-92 alkalmazás

Természetesen az egyes programok folyamatosan változnak, s előfordulhat, hogy már új címeken érhetők el.

19. fejezet - Zárszó

Reméljük, hogy a jegyzettel sikerült a programozás „misztikus” jellegét szertefoszlatni, s a hallgatók sikeresen elsajátították azokat a programozói fogásokat, amelyek segítségével képesek lehetnek egyszerűbb programokat írni, vagy – megfelelő érdeklődés esetén, újabb programozási nyelveket elsajátítva – akár professzionális programozóvá válni.

Irodalom

Albert, István. *A .NET Framework és programozása*. Szak Kiadó Kft.. 2004.

Bailey, Allen. *Microsoft Small Basic - Graphics Window http://www.slideshare.net/allenbailey/21-graphics-window (Hozzáférés: 2012. június 22.)*. 2010.

Bártfai, Barnabás. *Makróhasználat Excelben*. BBS-INFO könyvek. 2012.

Barzee, Rex A.. *The Handbook for Beginning Programmers with Examples in Visual Basic*. Amazon Digital Services, Inc.. 2012.

Beck, Kent. *Implementációs minták*. Panem Kft.. 2008.

Boehm, Anna. *Murach's Visual Basic 2010*. Mike Murach & Associates. 2010.

Conrod, Philip és Lou, Tylee. *Beginning Microsoft Small Basic*. Kidware Software, LLC. 2010.

Cornell, Paul. *Downloadable Code Examples http://blogs.msdn.com/b/paulcornell/archive/2010/01/04/small-basic-16-downloadable-code-examples-so-far.aspx (Hozzáférés: 2012. július 3.)*. 2010.

Csőke, Lajos és Garamhegyi, Gábor. *A számítógép-programozás logikai alapjai – Algoritmusok és elemi adatszerkezetek*. Nemzeti Tankönyvkiadó Zrt.. 2004.

Dusza, Árpád. *Algoritmusok Pascal nyelven – Az alapoktól az emeltszintű érettségiig*. Dusza Bt.. 2006.

Farkas, Csaba. *A programozás alapjai Visual Basic .NET-ben*. Jedlik Oktatási Stúdió Kft.. 2009.

Farkas, Csaba és Szabó, Marcell. *A programozás alapjai Visual Basicben*. Jedlik Oktatási Stúdió Kft.. 2005.

Farkas, Károly. *Játékos teknőcgeometria*. Szak Kiadó Kft.. 2011.

Fóthi, Ákos. *Bevezetés a programozáshoz*. ELTE Eötvös Kiadó Kft.. 2005.

Foxall, James. *Sams Teach Yourself Visual Basic 2010 in 24 Hours Complete Starter Kit (Sams Teach Yourself - Hours)*. Sams. 2010.

Gamma, Erich és Helm, Richard. *Programtervezési minták*. Kiskapu Kiadó. 2004.

Halvorson, Michael. *Microsoft Visual Basic 2010 Step by Step (Step by Step (Microsoft))*. Microsoft Press. 2010.

Harris, Simon és Ross, James. *Kezdőkönyv az algoritmusokról*. Szak Kiadó Kft.. 2006.

Ivanyos, Gábor, Rónyai, Lajos, és Szabó, Réka. *Algoritmusok*. Typotex Eleketronikus Kiadó Kft.. 2008. Iványi, Antal. *Informatikai algoritmusok 2*. ELTE Eötvös Kiadó Kft.. 2005.

Juhász, Tibor és Kiss, Zsolt. *Programozási ismeretek*. Műszaki Könyvkiadó Kft.. 2011.

Kernighan, Brian W. és Ritchie, Dennis M.. *A C programozási nyelv – Az ANSI szerint szabványosított változat*. Műszaki Könyvkiadó. 1994.

Keys, Kenny L.. *VBA Codes in Word Processors Are Fun, Simple, and Easy to Learn in One Hour or Less: VBA for Students, Parents, and Professionals (Second Edition)*. Amazon Digital Services, Inc.. 2012.

Kingsley-Hughes, Adrian, Kingsley-Hughes, Kathie, és Korol, Julitta. *Kezdőkönyv a programozásról*. Szak Kiadó Kft.. 2006.

Kotsis, Domokos, Légrádi, Gábor, Nagy, Gergely, és Szénási, Sándor. *Többnyelvű programozástechnika – Object Pascal, C++, C#, Java*. Panem Kft.. 2007.

Kuzmina, Jekatyerina, Tamás, Péter, és Tóth, Bertalan. *Programozzunk Visual Basic rendszerben!*. ComputerBooks. 2006.

Lengyel, László és Tóth, Bálint. *Programozás, algoritmusok*. Typotex Kiadó. 2002.

Mackey, Alex. *A NET 4.0 és a Visual Studio 2010*. Szak Kiadó Kft.. 2010.

Martin, Robert C.. *Tiszta kód – Az agilis szoftverfejlesztés kézikönyve*. Kiskapu Kiadó. 2010.

Martin, Robert C.. *Túlélőkönyv programozóknak*. Kiskapu Kiadó. 2011.

Provincien, Zeven. *Sorting algorithm demo http://smallbasic.com/program/?SORTVIZ (Hozzáférés: 2012. június 1.)*. 2010.

Sakamoto, J.. *Excel Library for Small Basic http://excel4smallbasic.codeplex.com/ (Hozzáférés: 2012. június 7.)*. 2011.

Schmidt, Stephan. *Learn Programming with Microsoft Small Basic http://codemonkeyism.com/learn-programming-microsoft-small-basic/ (Hozzáférés: 2012. június 22.)*. 2010.

„Simpleton Geek". *Anything goes Simpleton Geek http://ramstrong.blogspot.hu (Hozzáférés: 2012. Július 11.)*. 2012.

Smith, Charles. *Charles SmithVisual Basic Database Management : Essentials+(Professionals) KIT*. Shiyam. 2012.

Snell, Mike. *Code Camp 2009 - Small Basic Presentation. Visual Studio Unleashed – Mike Snell on .NET Development http://visualstudiounleashed.com/mikesnell/category/Small-Basic.aspx (Hozzáférés: 2012. június 12.)*. 2009.

Végh, László. *Programozás http://prog.ide.sk/ (Hozzáférés: 2012. június 22.)*. 2011.

(szerző nélkül). *Data Extension http://wiki.smallbasic.com/Data%20Extension.ashx (Hozzáférés: 2012. június 15.)*. 2011.

(szerző nélkül). *Extending Small Basic http://blogs.msdn.com/b/smallbasic/archive/2008/10/27/extending-small-basic.aspx (Hozzáférés: 2012. június 9.)*. 2008.

(szerző nékül). *Extend SmallBasic http://extendsmallbasic.codeplex.com/ (Hozzáférés: 2012. július 5.)*. 2011.

(szerző nélkül). *Koch-görbe. Wikipédia http://hu.wikipedia.org/wiki/Koch-g%C3%B6rbe (Hozzáférés: 2012. május 26.)*. 2010.

(szerző nélkül). *Post your sample source code here and get featured on our blogs! http://social.msdn.microsoft.com/Forums/en-US/smallbasic/thread/eaacf826-9d12-4e3c-aa10-1b2788f160a6/ (Hozzáférés: 2012. május 30.)*. 2012.

(szerző nélkül). *Project Euler Solutions http://social.msdn.microsoft.com/Forums/en/smallbasic/thread/c209f91a-278c-43b3-ad03-7ecb4c8b6de1 (Hozzáférés: 2012. június 8.)*. 2012.

(szerző nélkül). *Project Euler solutions using Small Basic http://wiki.smallbasic.com/ProjectEuler.ashx (Hozzáférés: 2012. május 9.)*. 2011.

(szerző nélkül). *Small Basic, MSDN http://social.msdn.microsoft.com/Forums/en-US/smallbasic/ (Hozzáférés: 2012. május 4.)*. 2012.

(szerző nélkül). *Small Basic Program List http://wiki.smallbasic.com/Small%20Basic%20Program%20List.ashx?HL=game (Hozzáférés: 2012. május 27.)*. 2012.

(szerző és évszám nékül). *Microsoft Small Basic – An introduction to Programming* http://download.microsoft.com/download/9/0/6/90616372-C4BF-4628-BC82-BD709635220D/Introducing%20Small%20Basic.pdf *(Hozzáférés: 2012. május 2.)*.

(szerző és évszám nékül). *Project Euler* http://projecteuler.net/ *(Hozzáférés: 2012. június 16.)*.

(szerző és évszám nékül). *Small Basic Enthusiasts – a Small Basic Facebook-csoportja* http://smallbasic.com/doc.aspx *(Hozzáférés: 2012. május 7.)*.

(szerző és évszám nékül). *Microsoft Small Basic reference documentation* http://smallbasic.com/doc.aspx *(Hozzáférés: 2012. május 5.)*.

(szerző és évszám nékül). *Small Basic – Spread the joy of programming* http://litdev.hostoi.com/ *(Hozzáférés: 2012. július 5.)*.

(szerző és évszám nékül). *Small Basic Recipes* http://teachingkidsprogramming.org/blog/kids-learning-programming/recipes-for-small-basic/ *(Hozzáférés: 2012. június 26.)*.

(szerző és évszám nékül). *Small Basic-wiki* http://wiki.smallbasic.com/ *(Hozzáférés: 2012. május 8.)*.