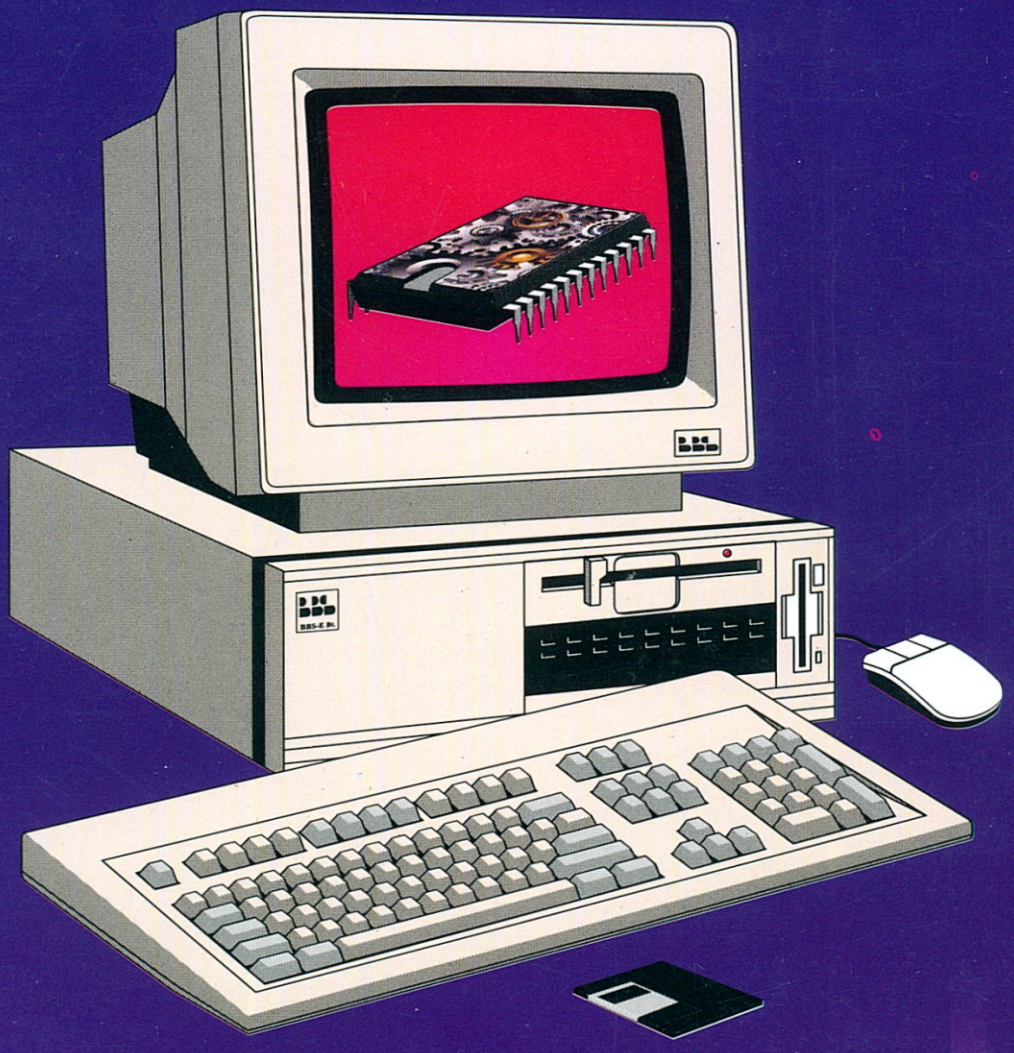
BBS-E Számítástechnikai és Könyvkiadó Betéti Társaság

Informatikai füzetek

Programozás elmélet, Quick basic

7. PROGRAMOZÁS



AJÁNLJUK:
ÖNÁLLÓ TANULÁSHOZ, OKJ-S KÉPZÉSEKHEZ (Számítógép-kezelő és Szoftverüzemeltető), SZAKKÖZÉPISKOLÁK, GIMNÁZIUMOK INFORMATIKA ÓRÁIRA, EGYÉB TANFOLYAMOKHOZ, STB.

VII.
Programozás

Szerzők: Sági Gábor
Molnár Csaba

Lektorálta: Vágó András, Székely Zsolt
Sorozatszerkesztő: Bártfai Barnabás

ISSN 1418-8791
ISBN 963 03 5286 9

Kiadja a BBS-E Betéti Társaság
1630 Budapest, Pf. 21.
Felelős kiadó: a BBS-E Betéti Társaság ügyvezetője

Nyomtatta és kötötte a Kaposvári Nyomda Kft. – 180624
Felelős vezető: Mike Ferenc

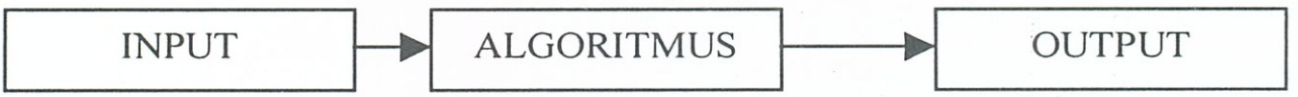
Tartalomjegyzék

1. Bevezetés a programozásba

1.1. Bevezetés

A számítástechnikában, hasonlóan az élet más területeihez, folyamatosan valamilyen problémát kell megoldanunk. A probléma megoldása során az általunk ismerttől - megfelelően összekapcsolt - műveletekkel jutunk el az ismeretlenig. Ezt a megoldási menetet nevezzük algoritmusnak.

Az algoritmus tehát nem más, mint műveletek megfelelő módon való összekapcsolása. Az ismert adatokat INPUT-nak, az ismeretlen adatok OUTPUT-nak nevezzük.



Gyakran előfordul, hogy egy feladat első ránézésre egyszerűnek tűnik, azonban egy kis gondolkodás után rájöhetünk, hogy a feladat nem is olyan egyszerű. Érdemes elgondolkodnunk azon, milyen bonyolult algoritmust kell végrehajtanunk ahhoz, hogy telefonáljunk. Első pillanatokban nem is gondolunk arra, hogy mit kell tennünk, ha foglalt a vonal vagy ha nem veszik fel a telefont stb. A mindennapok, a gyakran ismétlődő feladatmegoldás során ezek a felmerülő problémák számunkra már nem okoznak gondot, hiszen nagyon sokszor megoldottuk már, automatikusan cselekszünk ilyen helyzetekben. Másik példa, talán mindenki emlékszik arra, mikor először beült egy kocsiba vezetőként. Először komoly gondot okozhatott már a pedálok használata is, de később már a sebességváltás is könnyedén ment és mindez néhány kilométer vezetés után.

A számítástechnikában a programozó készíti el a probléma megoldásának eszközét. Ez az eszköz a program. A program egy adott feladat megoldására készített – számítógép által megértett – algoritmus alapján felépített utasítások összessége. Az adott programot fel kell készíteni arra, hogy az összes lehetőséget figyelembe vegye. Ne következzen be

az az eset – az előző példa alapján - , hogy a program ne tudjon mit csinálni akkor, ha netalántán az üzenetrögzítő kapcsol be. A programban nagyon részletesen meg kell adnunk, hogy a program mit is csináljon.

1.2. A program készítésének menete

Egy program elkészítése általában nem egy lépésben történik. Nézzük végig mi történik akkor, ha a főnök azt mondja a titkárnőjének, hogy hívja fel telefonon az egyik beosztottját. A titkárnő először megérti a problémát , majd eltervezi és végül végrehajtja a megoldási lépéssorozatot. Az algoritmus a következő fő lépésekből áll:

1. kinyitja a telefonregisztert
2. felemeli a telefonkagylót
3. tárcsáz, amíg a beosztott fel nem veszi a telefont
4. ha felvette a beosztott a telefont, bekapcsolja a főnökhöz
5. a telefonregisztert bezárja

1.2.1. A program elkészítésének lépései

Ahhoz, hogy egy programot megfelelő minőségben el tudjunk készíteni mindenképpen be kell tartani a program elkészítésének lépéseit, melyek a következők:

- elemzés
- tervezés
- végrehajtás
- ellenőrzés

1.2.2. Az algoritmus és szerkezete

A feladat megoldása során nagyon fontos a program algoritmusának elkészítése. A kérdés csak az, hogy ezt milyen részletességgel tegyük meg. Minden feladat további részfeladatokra bontható. Érdemes elgondolkodni azon, kell-e olyan problémákkal foglakozni, mint:

Hol van a telefonregiszter?

Melyik oldalon van a keresett személy?

Elérhető-e telefonon?

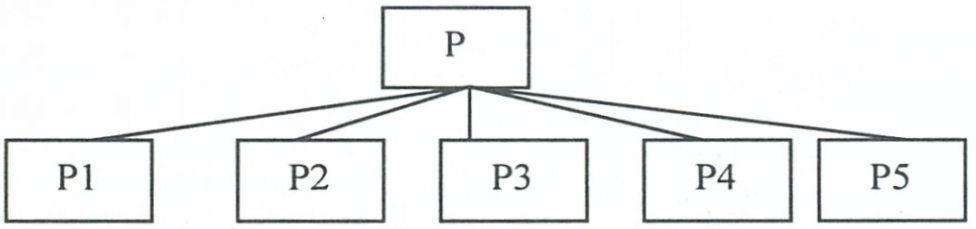
1.2.2.1. Szekvencia

Nézzük meg az előző feladat 5 lépését (a főnök a beosztottal szeretne beszélni):

P1: a titkárnő kinyitja a telefonregisztert
P2: a titkárnő felemeli a telefonkagylót
P3: tárcsáz, amíg a valaki fel nem veszi a telefont
P4: ha felvette valaki a telefont, kéri a beosztottat, és bekapcsol a főnökhöz
P5: a telefonregisztert bezárja

A probléma megoldása a részfeladatok egymás utáni megoldása.

A probléma megoldásának hierarchikus ábrája:



Ebben a feladatban az egyes részfeladatokat egymás után kell végrehajtani. A P1..P5 részprogramok szekvenciát (sorrendet) alkotnak.

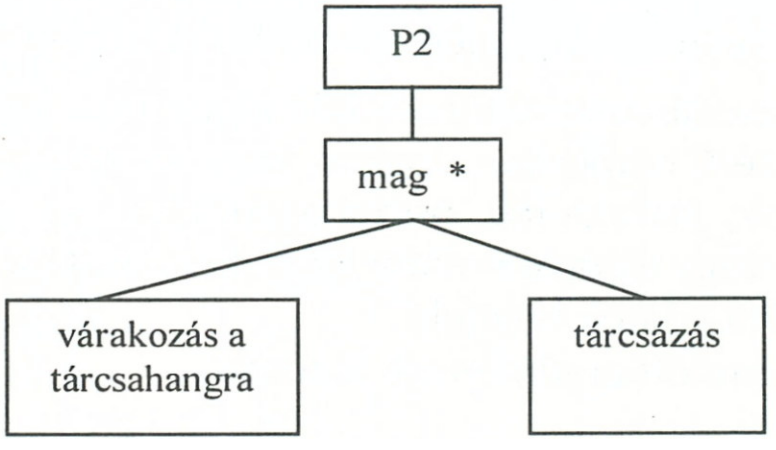
1.2.2.2. Iteráció

Gyakran előfordul, hogy bizonyos cselekménysorozatot többször kell végrehajtanunk.

Erre a példa a P3-as részfeladatot, ebben a feladatban egész addig kell tárcsáznunk, míg valaki fel nem veszi a telefont, ha senki sem veszi fel, később újra megpróbálhatjuk:

1. a kagyló felvétele után várjuk a tárcsahangot
2. ha megjött, tárcsázunk

Ezt a végrehajtási módot iterációnak nevezzük. Az iteráció magját az ismétlődő műveletek adják.

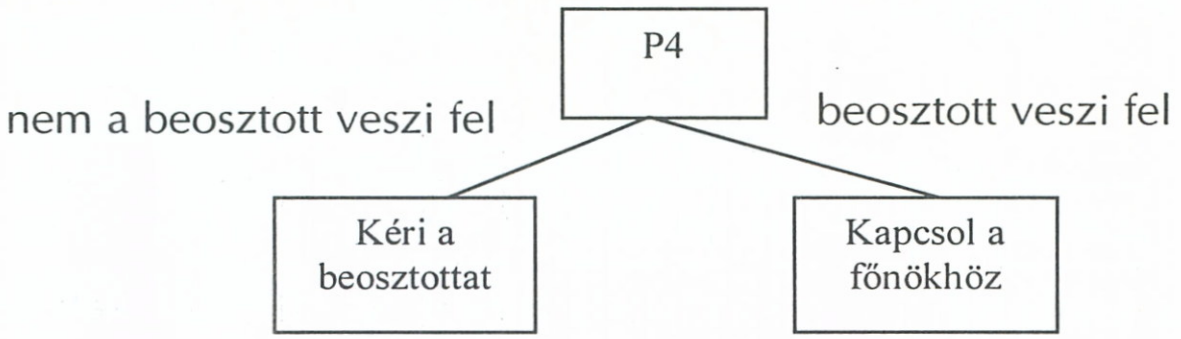


1.2.2.3. Szelekció

A mindennapi életben a döntés az egyik legfontosabb tevékenység. A döntés nem más, mint a dolgok közötti választás. A programozásban és az élet sok területén a választást szelekciónak nevezik.

Példaként nézzük meg a P4-es részfeladatot részletesen:
1. ha valaki felvette a telefont, meg kell kérdezni ki az,
2. ha a beosztott, kapcsol a főnökhöz
3. ha nem a beosztott, akkor kérni kell a beosztottat.

Az a folyamatot, mikor kiválasztjuk, hogy a beosztott vette fel a telefont vagy sem, döntésnek nevezzük.



Az eddig bemutatott ábrákat program-szerkezeti vagy hierarchia-ábrának nevezzük, mivel a program hierarchikus szerkezetét tükrözi.

1.2.3. Az algoritmussal szemben támasztott követelmények

Teljesség: minden információt tartalmaznia kell, ami a feladat megoldásához szükség.

Egyértelműség: az algoritmus leírása legyen egyértelmű, minden lépés csak egyféleképpen legyen értelmezhető.

Befejeződés: az algoritmusnak véges számú lépésben be kell fejeződni.

Az algoritmus minden egyes lépése egy műveletet valósít meg. Az emberi olvasásra szánt algoritmusok tartalmazhatnak meglehetősen összetett lépéseket is (pl.: lemegyek telefonálni), de a számítógépes algoritmusnak mindenképpen rendkívül részletesnek kell lennie.

A programozás során alapvetően 4 féle algoritmus lépést alkalmazunk:

számítás: egyszerű numerikus, logikai vagy karaktermű̈velet

döntés: számok, karakterek összehasonlítása, és ennek eredménye alapján egy vagy több alternatív lépés kiválasztása

input: számításra adatok bevitele

output: a kiszámított eredmények kiadása

1.2.4. Folyamatábrák

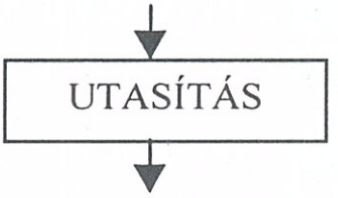
A hierarchia-ábra a probléma, illetve a megoldás szerkezetének ábrázolására alkalmas. Célszerű azonban egy másik ábrázolási módot alkalmazni, amely a folyamatot helyezi előtérbe és közérthető formájú, mégis félreérthetetlenül leírhatjuk vele az algoritmust. A végleges program-kód elkészítése előtt célszerű lehet egy korlátozott formájú magyar nyelvű leírást használni. Ezt a nyelvet pszeudonyelvnek nevezzük. A magyar nyelv sokszínűsége miatt ez a leírási mód nem biztos, hogy mindenki számára ugyanazt fogja jelenteni. Ezért az algoritmusok leírásához egy, a hétköznapi nyelvi formánál precízebb, ámde „emberi fogyasztásra alkalmas" formára is szükség van. Ezt valósíthatja meg az algoritmust kifejező, képi ábrázolás, a folyamatábra. Ez az ábrázolási módszer képi megjelenítési formája miatt áttekinthető és érthető.

Az algoritmus egy-egy lépésének leírásához a folyamatábrán egy-egy szimbólum felel meg. A lépések időben egymáshoz való viszonyát a szimbólumok síkbeli elhelyezkedésével, illetve összekapcsolásával ábrázolhatjuk.

Kezdő felhasználók számára mindenképpen célszerű a folyamatábra elkészítése és a folyamatábra alapján a program kódjának megírása, hiszen minden egyes szimbólum egy-egy utasításnak felel meg az adott programozási nyelven, így egy jól elkészített folyamatábra megadja a program kódját.

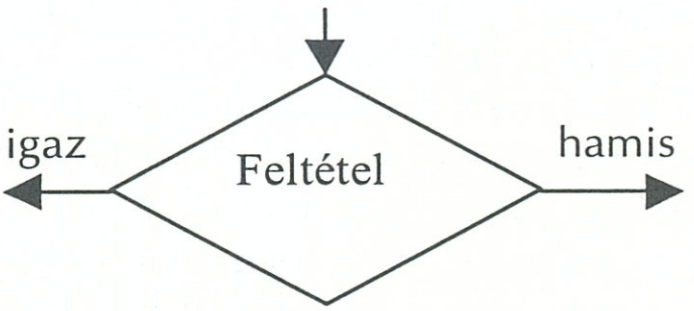
A folyamatábra szimbólumai:

Azt az algoritmus lépést, amelyet minden feltétel nélkül, egyszer hajtunk végre így jelöljük:



A téglalapba írjuk be az algoritmus lépést. A nyilak az algoritmusok időben lefolyását mutatják. Az ábrába bevezető nyíl előtt az időben korábban zajlódó folyamatok vannak, az ábrából kifele mutató nyíl pedig a következő utasításokat mutatja.

A választás lehetőségét a hegyére állított rombusszal ábrázoljuk. Ebben az esetben a választási lehetőségek száma kettő: igaz vagy hamis. Több egymás után következő rombusz esetében már bonyolultabb feltételrendszert is ki tudunk alakítani.



A rombusz belsejébe írhatjuk a választható feltételeket. Abban az esetben, ha az állítás hamis, akkor a „hamis nyíl" irányába fog az algoritmus folytatódni, ha igaz, értelemszerűen az „igaz nyíl" irányába.

1.2.5. A strukturált problémamegoldás

Egy adott feladatra a problémát megoldó algoritmust általában nem kapjuk meg készen, ezt nekünk kell elkészíteni, ki kell fejlesztenünk azokat. A probléma megoldása sikeresen elvégezhető strukturált módszerekkel.

A strukturált problémamegoldás három részből áll:

1. a probléma megfogalmazása és megértése
2. az algoritmus megtervezése
3. a program elkészítése

1. A probléma megfogalmazása és megértése: ez a rész segít megérteni az adott problémát, félreértéseket tisztázni és a nyitott kérdésekre válaszolni. Ebben a részben pontosan le kell írnunk:

- a modell elemeit
- az inputok és az outputok tartalmát és formáját
- a feldolgozás fő elemeit, az adatok közötti összefüggéseket

2. Az algoritmus megtervezése: az első pontban leírtak alapján fejlesztjük ki az algoritmust, definiáljuk a modell változóit, leírjuk az adatok szerkezetét és kapcsolatát. A változók közötti kapcsolatok meghatározása alapján összeállítjuk a megoldás elemeit és azokat logikus rendbe fűzzük.

Az algoritmus elkészítése lépésenként finomítva történik: először általában valamilyen pszeudonyelven írjuk le az algoritmust, utalva az egyes algoritmus részek kapcsolatára., majd a feladatot részekre bontjuk. Ezt követei a részfeladatok pontosítása addig, míg az egyes elemek egyértelművé nem válnak, vagyis az adott programnyelven közvetlenül leírhatók nem lesznek.

3. A második lépés elkészülte után következik a program elkészítése, vagyis a kódolás. Ez nem más, mint az adott algoritmust megvalósító program elkészítése.

A strukturált problémamegoldás előnye, hogy az algoritmusok részekre bontásával bármilyen bonyolultságú probléma megoldható, ha megfelelő sorrendben hajtjuk végre a lépéseket. Előnye még, hogy az első és második lépés megfelelő minőségű elvégzése után a programozónak már csak a kódolással kell foglalkoznia és nem kell menetközben a program kódján módosítani. Ezen előnyök jelentősen csökkentik

a hibalehetőségeket, leegyszerűsítik a programozást, valamint egy esetleges későbbi programmódosítás során a javítások végrehajtását.

Nézzünk egy feladatot a strukturált feladat megoldási módszerre!

Két általunk megadott numerikus érték (szám) közül azt a számot kell kiírni a képernyőre amelyik nagyobb. Tervezzük meg azt az algoritmust amelyik bekéri tőlünk a két számot és eldönti, hogy az első szám vagy a második szám a nagyobb!

Jelöljük a két számot így: SZAM1, SZAM2. A nagyobbat tegyük a NAGYOBB változóba.

Amelyik szám nagyobb, írjuk ki a képernyőre.

Az algoritmus első közelítésben az alábbi lépésekből fog állni:

- SZAM1 és SZAM2 bekérése
- a nagyobb szám kiválasztása
- az eredmény kinyomtatása.

Az első és a harmadik lépés egyszerű, de a második már bonyolultabb, ezért ez további finomításra szorul:

ha SZAM1 > SZAM2, akkor NAGYOBB = SZAM1

ha SZAM1< SZAM2, akkor NAGYOBB = SZAM2

Ezek után már bármilyen számunkra ismert programnyelven kódolhatjuk a feladatot.

1.2.6. A program készítésének folyamata

A probléma megfogalmazása, az algoritmus elkészítése és a kódolás után még további munkafázisok vannak:

- formális ellenőrzés
- program tesztelés
- éles használat

A formális ellenőrzés során az alkalmazott fordítóprogram segítségével az kódolás szintaktikai (nyelvhelyességi) ellenőrzése történik meg. Ha a kód szintaktikailag helyes, akkor a fordítóprogram a gép számára használható kódot készít. Abban az esetben, ha szintaktikai hibát talál a fordítóprogram, jelzi azt és a kódot nem hagyja gépi nyelvre lefordítani. A hiba javítása után újbóli ellenőrzéskor - ha minden hibát kijavítottunk - a fordítóprogram gépi nyelvre fordítja a programot.

A következő lépés a tesztelés. Ebben a fázisban olyan próbaadatokkal töltjük fel a gépet, amelyek lehetőséget biztosítanak a hibák feltárására (pl.: az előző feladat esetén a program nem fog helyesen működni, ha a két szám azonos). Itt még célszerű kevés adattal dolgozni, hiszen

az itt végzett munka jó program esetén hiábavaló. Ettől függetlenül ezt a lépést nem célszerű kihagyni, mert egy esetlegesen rosszul működő program hibáira fényt derít, mielőtt még élesben használnánk a programot. Ha ebben a fázisban hibásan működik a program a probléma megfogalmazásától újra kell kezdeni a munkát. Ez első ránézésre időrablónak tűnhet. Általában nem az egész feladatmegoldás rossz, hanem annak egy kis hányada. A hibák felderítése érdekében mindenképpen célszerű a korábban készített dokumentáció áttekintése és javítása.

Ha a teszteléssel végeztünk, jöhet az éles használat, amikor valódi adatokkal töltjük fel a programot. Ha a tesztelésünk lelkiismeretes volt és jó eredménnyel zárult, az esetek többségében az éles üzemeltetés sem fog gondot okozni. Bonyolultabb programok esetében azonban előfordulhat, hogy a tesztelés során van olyan programrész, amit nem teszteltük le és éles futtatásnál derül csak ki a hiba. Ekkor is vissza kell menni a feladat megoldásának elejéhez és ki kell javítani a programot.

A program írója, illetve tesztelője, ha mód van rá, két különböző személy legyen.

1.3. Programozási alapfogalmak

A programokat karakterenként írjuk. A programnyelvek általában rögzített karaktereket használhatnak.
Ezek a jelek:

az angol ABC betűi és az aláhúzásjel: A...Z, a...z, _

a számok: 0...9

speciális karakterek: +, -, *, /, = , <, >, (,), [,], ., ,: ,; ,, ', #, $

A programok legkisebb egységei az illető nyelv szimbólumai. Ezek az elemek, melyekből egy programot fel lehet építeni:

- a programozási nyelv által előírt kulcsszavak
- a programozó által képzett nyelvi elemek (változók, azonosítók, konstansok)
- különböző műveleti és elhatároló jelek.

E szimbólumokból meghatározott szabályok szerint utasításokat lehet összerakni, amelyek valamilyen műveletet végeznek el a gépen. Az utasítások sorozata adja a megoldandó programot.

A fejezetben a példaprogramok TURBO PASCAL-ban és BASIC-ben íródtak, bemutatva ezáltal, mennyire mások és egyben mennyire hasonlóak a különböző programnyelvek. A példaprogramok szemléltetik az adott probléma megoldását a különböző programnyelveken.

1.3.1. A programírás konvenciói

A nyelv szintaxisát azok a szabályok alkotják, amelyek a programozási nyelv elemeinek helyes használatát biztosítják. A megengedett nyelvi szerkezetek jelentéstartama a szemantika.

A programozási nyelvekben vannak bizonyos „kulcsszavak", amelyeknek speciális jelentése van, ilyen szavak például az utasítások, függvények is. Ezek az úgynevezett fenntartott szavak, melyek csak igen korlátozott módon használhatók bizonyos feladatokra.

1.3.2. Az azonosítók

Programozás során gyakran szükséges olyan karaktersorozatot használni, amit a programozó definiál változónévként vagy egyéb azonosítóként. Programnyelvektől függetlenül ezek nem kezdődhetnek számjeggyel, nem tartalmazhatnak speciális karaktereket, valamint szóközt. Az azonosító maximális hossza programnyelvtől függő, de általában 8 karaktert használnak.

1.3.3. A konstans

Konstans olyan adat, amely a műveletek során nem változik meg, értéke állandó marad. A legtöbb programnyelvben többféle típusú konstans lehet:

- Egész konstans: pozitív, negatív számok, valamint a nulla: (pl.: 1, -134, 0)
- Valós konstans: minden olyan szám, amelyben tizedespont szerepel (pl.: 1,2 vagy –0,13)
- Karakter konstans: egyetlen karakter, ami lehet szám, betű, valamint speciális karakter
- Szó konstans: több karakterből álló karaktersorozat. Néhány karakter kivételével minden karakter használható.
- Boole-féle konstans: két lehetséges értéke van: igaz vagy hamis

1.3.4. A változók

A változó olyan objektum, amelynek értéke a program futása során változik. A változókra azonosítókkal hivatkozunk. A változó típusai hasonlóan programnyelvtől függően változhatnak, de megegyeznek a konstansok típusaival. Egy adott típusú változó csak a neki megfelelő értékeket tartalmazhatja (nem lehet egy egész változóba egy szót tárolni), illetve művelet csak azonos típusú változók között lehetséges.

Ha nem ilyen műveletet próbálunk végrehajtani, az típushibát fog okozni.

Minden programozási nyelvben vannak olyan függvények, amelyek a változók közötti típusváltást elvégzik, abban az esetben, ha ez lehetséges, ilyen átalakítás lehet például:

- számból karakter
- karakterből szám, ha a karakter csak számokat tartalmaz

Ezek programnyelvtől függenek.

1.3.5. kifejezések

A kifejezések konstansokból, változókból, műveleti jelekből illetve zárójelekből és függvényekből állhatnak. A zárójelek a műveletek végrehajtásának sorrendjét határozzák meg.
Példa kifejezésre:

```
(a+2)-(cos(30°)/b)*d
```

a leggyakrabban használt műveleti jelek:

+	összeadás
–	kivonás
*	szorzás
/	osztás

1.3.6. A műveletek végrehajtásának sorrendje (prioritása)

A műveleteket mindig a matematika szabályai szerint kell végrehajtani.

Először mindig a zárójelben lévő műveletet kell végrehajtani.

A szorzás és az osztás magasabb prioritású művelet, mint az összeadás és a kivonás, tehát azokat előbb kell végrehajtani. A szorzás és az osztás, valamint az összeadás és a kivonás azonos prioritású. Azonos prioritású műveletek esetén a végrehajtási sorrend balról-jobbra történik. Az aritmetikai kifejezésekben a függvények egy számított értékkel térnek vissza, ezt az értéket fogja a kifejezésbe behelyettesíteni (a függvényekről később részletesebben lesz szó).

1.3.7. Boole-féle algebra

A programozás során nagyon gyakori feladat, hogy valamilyen adatok összehasonlításának eredményétől függően kell valamilyen utasítássorozatot végrehajtani. Egy összehasonlítás eredménye vagy igaz vagy hamis lehet, több választási lehetőség nincs. Sokszor akad olyan

feladati is, hogy több adatot kell valamilyen logikai rendszer szerint összekapcsolni, ehhez nyújt segítséget a Boole-féle algebra ismerete.

A Boole-féle algebra három elemet tartalmazhat az összehasonlítások kombinálására:

AND (és): Akkor és csak akkor igaz, ha A és B is igaz,
OR (vagy): Akkor igaz, ha A vagy B igaz,
NOT (nem): Akkor igaz, ha A hamis,
A és B logikai értékek. Értékük vagy igaz (true) vagy hamis (false)

A logikai értékek ábrázolására igazságtáblázatot szoktak használni, amely tartalmazza a lehetséges alternatívákat.

Az AND (és) művelet igazságtáblája:

A	B	A és B
0	0	0
1	0	0
0	1	0
1	1	1

Az OR (vagy) művelet igazságtáblája:

A	B	A vagy B
0	0	0
1	0	1
0	1	1
1	1	1

A NOT (nem) művelet igazságtáblája:

A	Nem A
1	0
0	1

A műveletek prioritása:
Zárójelben lévő műveletek
NOT művelet
AND, OR művelet

Ahhoz, hogy két vagy több adatot össze tudjunk hasonlítani, meg kell határozni, milyen összehasonlító (reláció) műveletet alkalmazzunk:

= egyenlő
> nagyobb, mint
< kisebb, mint
>= nagyobb vagy egyenlő
<= kisebb vagy egyenlő
<> nem egyenlő

A reláció jelekkel aritmetikai kifejezéseket hasonlíthatunk össze, az összehasonlítás általános alakja:

```
"kifejezés1" "relációs jel"  "kifejezés2"
```

pl.: a következő aritmetikai művelet csak akkor igaz, ha A nagyobb, mint B, ellenkező esetben hamis.

```
A > B
```

Az egyes kifejezésekben használhatók aritmetikai műveletek, ezeket célszerű zárójelbe tenni.

Pl.: `(A-5) > (B*3)`

Logikai műveleti jeleket csak logikai kifejezésekre lehet alkalmazni. Ha A, B és C numerikus változók, és azt akarjuk hogy a logikai művelet csak abban az esetben legyen igaz, ha A nagyobb B-nél és C-nél is azt így kell felírnunk:

```
(A > B) AND (A > C)
```

Nem így :

```
A > (B AND C),
```

mivel B és C aritmetikai értékek és nem logikai értékek.

1.3.8. Az értékadó utasítás

A számítások végrehajtásához a számítógépnek meg kell adni milyen műveletet hajtson végre, milyen adatokkal, és melyik változóba kerüljön. Az értékadás formája a következő:

Változó = kifejezés,

Pl.: `A =12 vagy`
`C = 3*12*(R+39)*sin(30°)`

Ahol az egyenlet bal oldalán lévő változóba kerül az egyenlet jobb oldalán lévő kifejezés értéke. Látható, hogy a kifejezésben szerepelhet szám, függvény, változó és konstans is. Természetesen az értékadáskor a változó típusától függően megadhatunk karaktereket, szavakat, illetve logikai értékeket is.

```
BETU = 'P'
SZO = 'De jó!'
LOGIKA = true (igaz)
```

Az értékadás művelete programnyelvenként eltérő lehet.

1.3.9. Input és output

A fejezet elején már volt szó arról, hogy a probléma megoldása során az ismert (input) adatoktól jutunk el az ismeretlenig (output), megfelelő algoritmussal. Az input/output utasítások a környezettel való kapcsolattartást biztosítják. A kapcsolattartás billentyűzeten, mágneslemez egységen, mágnesszalag egységen, illetve a képernyőn keresztül történhet.

1.3.9.1. Az input

Az input célja, hogy a program számára adatokat vigyünk be a gép memóriájába, amit a program futása során feldolgoz.

1.3.9.2. Az output

A program az input feldolgozása után valamilyen eredményre jut, ezt az eredményt nevezzük outputnak. Az output megjelenése a képernyőn vagy a nyomtatón, gyakran mágneses tárolóegységen történhet. Az output megjelenítése az esetek többségében formázottan történik, ez azt jelenti, hogy meg kell adni a megjelenített adat megjelenítési környezetét, milyen hosszú lehet a kiírandó karaktersorozat, milyen legyen a színe stb.

1.3.10. A soros programozás

Minden programnyelvben megvannak azok a szabályok, amelyek meghatározzák a program szerkezetét. Ilyen szabályok az utasítások szerkezete, megadásának sorrendje. A soros programozás nem biztosít lehetőséget döntések (szelekció), illetve ciklusok (iteráció) kezelésére.

Ettől függetlenül fontos ismernünk a soros programozás lehetőségét, hiszen az utasítások nagy része sorosan követi egymást, például egy cikluson belül is.

Az utasítok sorrendjére a programozás során nagyon oda kell figyelnünk, ha két utasítást összecserélünk, az az egész program eredményére hatással lehet, nagyon valószínű, hogy a program nem azt fogja csinálni amit elvárunk tőle.

1.3.11. A programok szerkezete

A program írásának első lépése, hogy deklaráljuk a program számára a számítógépbe bekerülő és a végrehajtás során használandó változók helyét. Ez a deklaráció néhány nyelvben nem kötelező, de a programnyelvek többségében elkerülhetetlen (BASIC-ben nem szükséges). A deklarációs rész tartalmazza a konstansokat, valamint a program futási környezetének beállításait. Ezt a részt a program deklarációs részének nevezzük. Ezek az utasítások a nem végrehajtandó utasítások körébe tartozik. Magasszintű programozási nyelveknél a fordítóprogram dönti el, hogy az adott változóhoz mekkora tárterületet foglal le, a változó típusa alapján. A tárolási helyekre az azonosítókkal hivatkozhatunk.

A deklarációs részt követik a végrehajtandó utasítások. Az utasításokon kívül itt célszerű elhelyezni a megjegyzéseket (commenteket), melyek a program működésének leírására szolgálnak.

Példák soros programozásra:
1, Egy általunk megadott szöveg kiíratása a képernyő középső sorba.

A probléma megoldásának menete:

1.Változó deklarálása
2.Képernyőtörlés
3.Adat bekérése
4.Pozicionálás középső sorba
5.Az adat kiíratása

A feladat megoldása BASIC nyelven (KIIR.BAS):

```
CLS
INPUT  "Kérem a megadott szöveget:", szoveg$
CLS
LOCATE 12,1
PRINT szoveg$
```

A feladat megoldása PASCAL nyelven (KIIR.PAS):

```
PROGRAM P1;
USES CRT;
VAR
Szoveg: string[80]
BEGIN
CLRSCR;
READ(szoveg);
CLSSCR;
WRITELN(szoveg);
END.
```

2, Másodfokú egyenlet megoldása. A program megoldása során nem foglalkozunk azzal, hogy az egyenletnek esetleg nincs megoldása.

A megoldáshoz szükséges képlet:

$$GYOK_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Megoldás:

A, B, C bekérése
GYOK1 és GYOK2 kiszámítása
GYOK1 és GYOK2 kiíratása

Megoldás BASIC-ben (MASOD.BAS):

```
CLS
INPUT "Kérem, adja meg az A-t:"; A
INPUT "Kérem, adja meg az B-t:"; B
INPUT "Kérem, adja meg az C-t:"; C
CLS
D = SQR ( B * B - 4*A*C)
GYOK1 = ( - B + D ) / ( 2 * A )
GYOK2 = ( - B - D ) / ( 2 * A )
LOCATE 5, 5
PRINT „ Az első gyök:", GYOK1
LOCATE 7, 5
PRINT "A második gyök:", GYOK2
```

Megoldás PASCAL-ban (MASOD.PAS):

```
ROGRAM masodfoku;

{ A program nem foglalkozik azzal, ha az egyen-
letnek nincs megoldása.
Nem megfelelő értékek esetén a program hibaüze-
nettel leáll! }
USES crt;
VAR             {Változók deklarálása}
a, b, c : integer;
d, gyok1, gyok2: real;
BEGIN
CLRSCR;
WRITE('Kérem, adja meg az A értékét:');
READLN(a);
WRITE('Kérem, adja meg a B értékét:');
```

```
READLN(b);
WRITE('Kérem, adja meg a C értékét:');
READLN(c);

d := sqrt( b * b - 4 * a * c );
gyok1 :=( - b + d ) / ( 2 * a );
gyok2 :=( - b - d ) / ( 2 * a );

WRITELN('Az egyenlet első gyöke: ', gyok1);
WRITELN('Az egyenlet második gyöke: ', gyok2);
READLN;
END.
```

1.4. Alternatívák és ciklusok kezelése a programban

A mindennapi életben az egyik leggyakoribb művelet a döntés. A cselekvések döntő többsége függ valamilyen környezeti elemtől. Pl.: Milyen az idő, mennyi ismeretünk van a döntéshez stb. Hasonló módon a programozás során is használni kell döntési mechanizmusokat. Ezek használata bizonyos probléma megoldásoknál elkerülhetetlen. A programozás során egy vagy több változó összehasonlításából jön ki egy logikai eredmény (igaz vagy hamis), a logikai eredmény alapján fut az A vagy a B algoritmus felé a program.

A logikai eredmény aszerint igaz vagy hamis, hogy a két vagy több érték közötti összehasonlítás során milyen eredményre jutunk.

Pl.:

ha A=3 és B=6, akkor

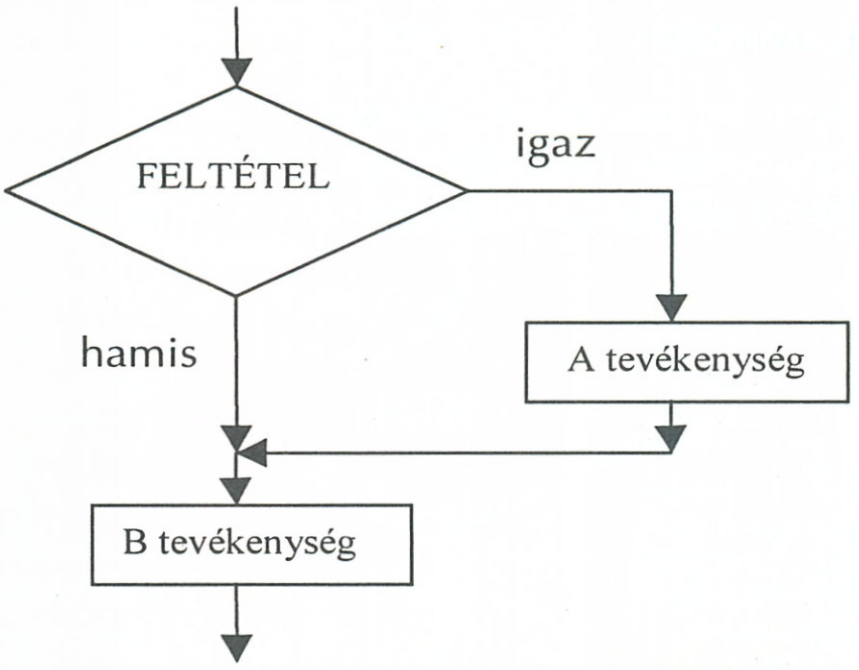
A > B eredménye HAMIS

A < B eredménye IGAZ

Az összehasonlítások között a Boole-féle algebra elemei használhatók.

1.4.1. Egyágú döntések

Az döntések lehetnek egyágúak abban az esetben, ha azt akarjuk, hogy az A algoritmus csak akkor hajtódjon végre, ha a feltétel teljesül, abban az esetben, ha a feltétel nem teljesül, fusson tovább a program, de az A algoritmust ne végezze el.



Feladat: Kérjen be a program egy számot, és csak akkor írja ki az üzenetet („A szám nagyobb tíznél"), ha a szám nagyobb, mint tíz.

A program megoldása BASIC-ben (NAGYOBB1.BAS).

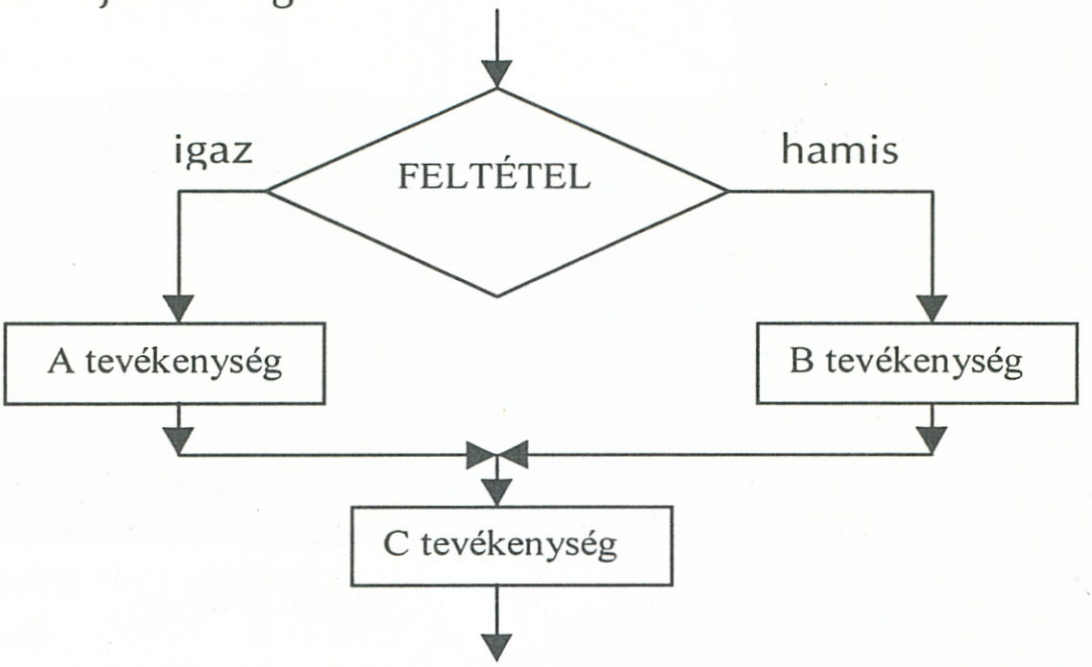
```
CLS
INPUT "Kérem, adjon meg egy számot:", szam
IF szam > 10 THEN
   PRINT "A szám nagyobb tíznél"
END IF
```

A program megoldása PASCAL-ban (NAGYOBB.PAS).

```
PROGRAM nagyobb10;
{ Ha 10-nél nagyobb számot adunk meg a program
kiírja azt }
USES crt;

VAR
szam : integer;

BEGIN
CLRSCR;
WRITE('Kérem, adjon meg egy 10-n,l nagyobb szá-
mot:');
READLN(szam);
IF szam > 10 THEN
WRITELN('A szám nagyobb, mint 10!');
READLN;
END.
```

1.4.2. Kétágú döntés

Abban az esetben beszélünk kétágú döntésről, ha a feltétel IGAZ voltakor az A algoritmus hajtódik vége, a feltétel HAMIS voltakor a B algoritmus hajtódik végre.



Feladat: Kérjük be egy személy életkorát. Ha több mint 18 éves, írjuk ki, hogy a személy nagykorú, ha nincs még 18 éves, írjuk ki, hogy nem nagykorú.

A program megoldása BASIC-ben (NAGYOBB1.BAS).

```
CLS
INPUT "Kérem, adjon meg egy számot:", szam
IF szam > 10 THEN
   PRINT "A szám nagyobb tíznél"
END IF
```

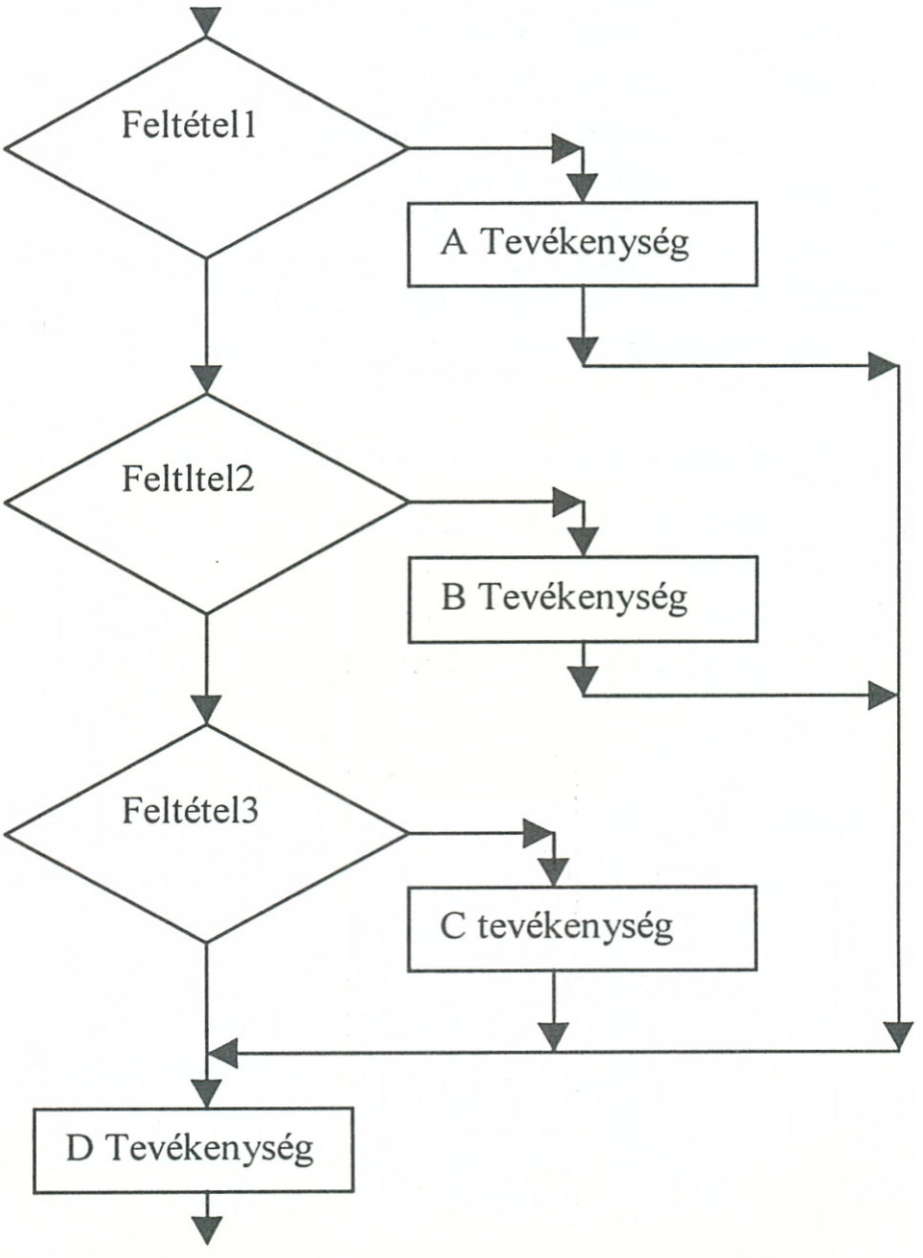
A program megoldása PASCAL-ban (NAGYOBB.PAS).

```
PROGRAM nagyobb10;
{ Ha 10-nél nagyobb számot adunk meg a program
kiírja azt }
USES crt;

VAR
szam : integer;

BEGIN
CLRSCR;
WRITE('Kérem, adjon meg egy 10-n,l nagyobb szá-
mot:');
READLN(szam);
IF szam > 10 THEN
```

```
    WRITELN('A szám nagyobb, mint 10!');
    READLN;
    END.
```

1.4.3. Több lehetséges érték közötti választás

A választások lehetősége sok esetben azonban nem egy vagy kettő, hanem esetenként akár több száz is lehet. Ezt többágú szelekciónak nevezzük. Erről akkor beszélünk, ha a feltétel1 teljesülése esetén az A algoritmus, feltétel2 teljesítése esetén a B algoritmus, feltétel3 teljesítése esetén a C algoritmus hajtódik végre.



A különböző programnyelvek ennek megvalósítására az úgynevezett CASE szerkezetet használják. Ez a szerkezet jelentősen meggyorsítja és egyszerűsíti a programozási munkát.

Feladat: A program kérje be tőlünk az osztályzatokat és írja ki azokat betűvel.

1 – elégtelen
2 – elégséges
3 – közepes
4 – jó
5 – jeles

A feladat megoldása BASIC-ben (OSZTALYZ.BAS):

```
CLS
INPUT "Kérem, adja meg az osztályzatot:", osztalyzat
CLS
SELECT CASE osztalyzat
      CASE 1
        PRINT "Elégtelen"
      CASE 2
        PRINT "Elégséges"
      CASE 3
        PRINT "Közepes"
      CASE 4
        PRINT "Jó"
      CASE 5
        PRINT "Jeles"
 END SELECT
```

A feladat megoldása PASCAL-ban (OSZTALYZ.PAS):

```
PROGRAM osztalyzat;
USES crt;
VAR
osztalyzat: integer;

BEGIN
CLRSCR;

WRITE('Kérem, adja meg az osztályzatot:');
READ(osztalyzat);
CASE osztalyzat OF
     1: WRITE('Elégtelen');
     2: WRITE('Elégséges');
     3: WRITE('Közepes');
     4: WRITE('Jó');
     5: WRITE('Jeles');
END;
READLN;
END.
```

1.4.4. Feltétel nélküli vezérlés átadás

Ez a programozási megoldás nem túl szerencsés, ha lehet a használatát kerülni kell. A lényege, hogy egy ugró utasításban megadjuk honnan folytassa a program a futását. A vezérlés átadás címke használatával történi.

1.4.5. A ciklikus tevékenység (iteráció) kezelése a programban

A számítógépet eleinte adatfeldolgozásra használták, a feldolgozása nagy mennyiségű adattal, de általában azonos művelettel történik. Mindennapi példa egy gyár dolgozóinak fizetésemelése. Ha a gyárban 1000-en dolgoznak nem szükséges a programban 1000-szer egymásután kiadni azt a parancsot, hogy a dolgozó fizetését emelje fel, hanem egyszer kell ezt megtenni, de a végrehajtást 1000-szer megismételni. Ezt nevezzük ciklusnak (iterációnak). Az hogy ezt a műveletet hányszor kell végrehajtani a dolgozók létszáma határozza meg. A ciklus egészen addig tart míg az úgynevezett ciklusváltozó el nem éri a dolgozók számát. A ciklusváltozó tehát, nem más mint egy érték, amelytől függ a ciklus végrehajtásának száma. Előkészítés során ezt a ciklusváltozót is be kell állítani.

A ciklus felépítése:

- előkészítés
- ciklusmag
- ciklusfeltétel vizsgálat

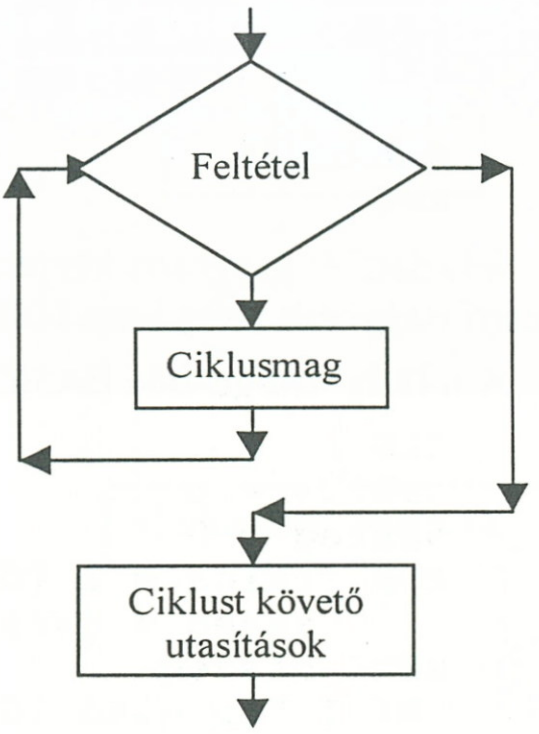
A ciklusmag magában foglalja a végrehajtandó utasításokat, valamint a ciklusváltozó értékének módosítását.

A ciklusokat működési módjuk szerint három csoportba sorolhatjuk:

- előtesztelő
- hátultesztelő
- növekményes

1.4.5.1. 4.5.1 Előltesztelő ciklus

A nevében is benne van, csak akkor fogja a ciklusmagban lévő utasításokat végrehajtani, ha a ciklusba a belépési feltétel teljesül. Ez azt jelenti, hogy ciklus addig fog végrehajtódni, míg a belépési feltétel igaz. Abban az esetben, ha a belépési feltétel már nem igaz, a vezérlés átadódik a ciklus utáni első utasításra.



1. feladat: A program addig kérjen be számokat, míg a számok összege nem lesz nagyobb 1000-nél.

A feladat megoldás BASIC-ben (1000IG.BAS).

```
CLS
szam = 0
osszeg = 0
DO WHILE a < 1000
        CLS
        INPUT "Adjon meg egy számot:", szam
        osszeg = osszeg + szam
LOOP
PRINT "A számok összege:", osszeg
END
```

A feladat megoldása PASCAL-ban (1000IG.PAS).

```
PROGRAM _1000ig;
{ A program addig kér be számokat, míg azok ösz-
szege nem lesz nagyobb 1000-nél }
USES crt;
VAR
szam, osszeg : integer;

BEGIN
CLRSCR;
osszeg := 0;
szam   := 0;
WHILE osszeg < 1000 do
   BEGIN
   CLRSCR;
   WRITE('Kérem, adjon meg egy számot:');
     READLN(szam);
     osszeg := osszeg + szam;
   END;
WRITE('A számok összege:', osszeg);
READLN;
END.
```

2. feladat: A program kérjen be tőlünk számokat addig, amíg a bekért szám nagyobb nem lesz 100-nál.

A feladat megoldás BASIC-ben (TOBB100.BAS).

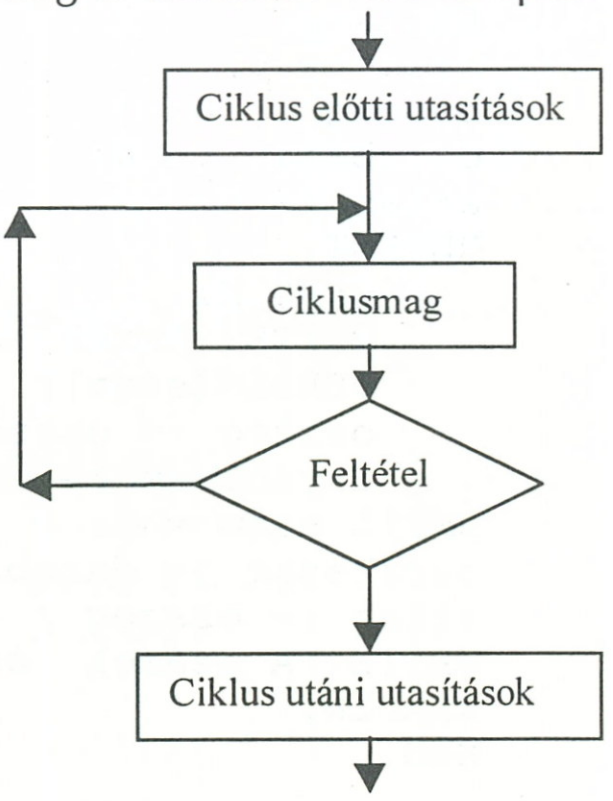
```
CLS
ciklus = 0
osszeg = 0
FOR ciklus = 1 TO 100
    osszeg = osszeg + ciklus
NEXT osszeg
PRINT "Az első 100 szám összege:", osszeg
END
```

A feladat megoldása PASCAL-ban (TOBB100.PAS).

```
PROGRAM nagyobb100;
{ Ha 100-nál nagyobb számot adunk meg, a program
befejezi a futását }
USES crt;

VAR
szam : integer;

BEGIN
CLRSCR;
szam a: = 0;
WHILE szam < 100 DO
      BEGIN
        CLRSCR;
        WRITELN('Ha 100-nál nagyobb számot adunk
meg a programnak befejeződik a futása!');
        WRITE('Kérem, adjon meg egy számot:');
        READLN(szam);
      END;
END.
```

1.4.5.2. Hátultesztelő ciklus

A hátultesztelő ciklus abban különbözik az előtesztelőstől, hogy a ciklus egészen addig végre fog hajtódni míg a ciklusból való kilépési feltétel nem teljesül. A ciklus során a ciklusmag egyszer legalább le fog futni, hiszen csak a ciklusmag lefutása után fog a kilépési feltétel vizsgálata megtörténni, ellentétben az előtesztelős ciklusnál, ahol a ciklus csak akkor futhat le egyszer, ha a belépési feltétel teljesül.



Feladat: Készítsünk átlagszámító programot! A program addig kérje be tőlünk a számokat, míg nullát nem adunk meg. Ha nullát adtunk meg, számolja ki és írja ki a képernyőre az átlagot.

A feladat megoldás BASIC-ben (ATLAG.BAS).

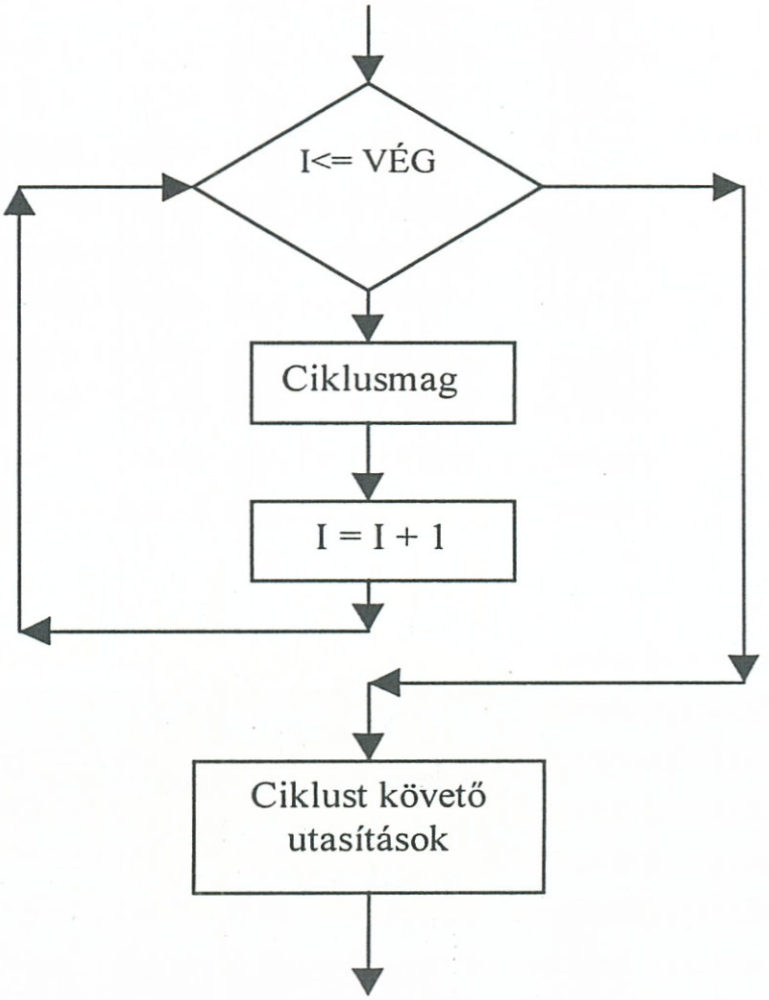
```
CLS
szam = 0
darabszam = 0
osszeg = 0
DO
 INPUT "Kérem, adja meg a következő számot:", szam
 darabszam = darabszam + 1
 osszeg = osszeg + szam
LOP UNTIL szam = 0
darabszam = darabszam -1
atlag = osszeg / darabszam
PRINT "A beírt számok  átlaga:", atlag
```

A feladat megoldása PASCAL-ban (ATLAG.PAS).

```
PROGRAM atlag;
{ A program addig kér be számokat, míg 0-t nem
adunk meg, majd a bekért
számoknak kiszámolja az  átlag t }
USES crt;

VAR
szam, osszeg, darabszam: integer;
atlag: real;

BEGIN
szam := 0;
osszeg := 0;
darabszam := 0;
atlag := 0;
REPEAT
    CLRSCR;
    WRITE('Kérem adja meg a következő számot:');
    READLN(szam);
    osszeg := osszeg + szam;
    darabszam := darabszam + 1;
UNTIL szam = 0;
darabszam := darabszam - 1;
atlag := osszeg / darabszam;
WRITE('A számok  átlaga:', atlag);
READLN;
END.
```

1.4.5.3. Növekményes ciklus

A ciklusmagban van egy olyan változó, amely a ciklus végrehajtása után változik (növekszik vagy csökken). Ebben a cikluskezelő formában meghatározhatjuk, hányszor fusson le a ciklusmag. A program a ciklusmagot mindaddig végrehajtja, míg a feltétel igaz, tehát a ciklusváltozó meg nem haladja a ciklus végértékét. Ezt a cikluskezelést FOR-NEXT féle ciklusnak nevezzük.



Feladat: 1 és 100 között adjuk össze a számokat és az eredményt írjuk ki a képernyőre.

A feladat megoldás BASIC-ben (100OSSZ.BAS).

```
CLS
ciklus = 0
osszeg = 0
FOR ciklus = 1 TO 100
    osszeg = osszeg + ciklus
NEXT osszeg
PRINT "Az első 100 szám összege:", osszeg
END
```

A feladat megoldása PASCAL-ban (100OSSZ.PAS).

```
PROGRAM osszegzes;
{ Az első 100 szám összegét írja ki. }
USES crt;

VAR
szam, osszeg: integer;

BEGIN
CLRSCR;
osszeg :=0;
ciklus := 0;
```

```
FOR ciklus:= 1 TO 100 DO
    BEGIN
      osszeg := osszeg + ciklus;
    END;
WRITE('Az első 100 szám összege:', osszeg);
READLN;
END.
```

1.5. Tömbök

A számítástechnikai gyakorlatban, csakúgy mint a mindennapi életben, gyakran használunk olyan adatokat, amelyek összetartozóak. A hétköznapi életben ezek számsorok, táblázatok, névsorok alakjában jelenik meg. Ilyen lehet egy osztálynévsor, egy busz megállói stb., az ilyen táblázatokat egynemű (egy dimenziós) táblázatnak nevezzük. Gyakran találkozhatunk olyan táblázatokkal, amelyek több bemenete van. Ezeket több bemenetű (több dimenziós) táblázatnak nevezzük. Ilyen táblázat lehet az autóstérképek végén lévő táblázat, amelyből a városok távolságát kereshetjük ki egymástól. A táblázat adott sorának és oszlopának találkozásából olvashatjuk ki a két város távolságát. A példa egy kétbemenetű táblázat. Az ilyen szerkezetű adattárolás tömböknek nevezzük.

1.5.1. A tömbök használata

A tömböt használata előtt deklarálnunk kell, a tömb típusát (szám, karakter, szöveg, logikai stb.), és a tömb nagyságát (hány sorból álljon).

A tömb elemeit indexek segítségével azonosítjuk. Az indexek 0-tól vagy 1-től kezdődnek, programnyelvtől függően. Az indexek segítségével érhetjük el a tömb adatait.

Nézzünk egy példát:

	1.	2.
Index	Név	Szakma
1.	Mekk Elek	Ezermester
2.	Vihar Viktor	Meteorológus
3.	Csöp Csaba	Vízvezeték szerelő
4.	Bú Béla	Humorista

Az első sorban az oszlopok száma van megadva.

1.5.2. Tömb elemeire való hivatkozás

A tömbök használata során elkerülhetetlen, hogy a tömb elemeivel valamilyen műveletet végezzünk, ehhez az szükséges, hogy az tömb elemeit el tudjuk érni. Ezt úgy tehetjük meg hogy a tömb neve után megadjuk az elérni kívánt elem sorszámát és oszlopszámát. A sor- és oszlopszám kereszteződésében találhatjuk meg az elemet.

Ha a tömb 3. Sorára és annak 2. Oszlopára akarunk hivatkozni ("Vízvezeték szerelő") azt a következő módon tehetjük meg:

```
Tömb(3,2)
```

1.5.3. Tömb feltöltése

Ahhoz, hogy egy tömböt használni tudjuk, a tömböt fel kell tölteni elemekkel. Ehhez szükség van az elem tömbbeli helyére és a feltöltendő adatra.

A tömb elemeinek értékadása a következőképpen történhet:

```
Tömb(3,2) = "esőjós"
```

1.5.4. Tömb elemeinek kiíratása

A tömb értékei ki is írathatjuk, de itt is meg kell adni, melyik elemet akarjuk kiíratni:

```
print(tömb(1,2)),
```

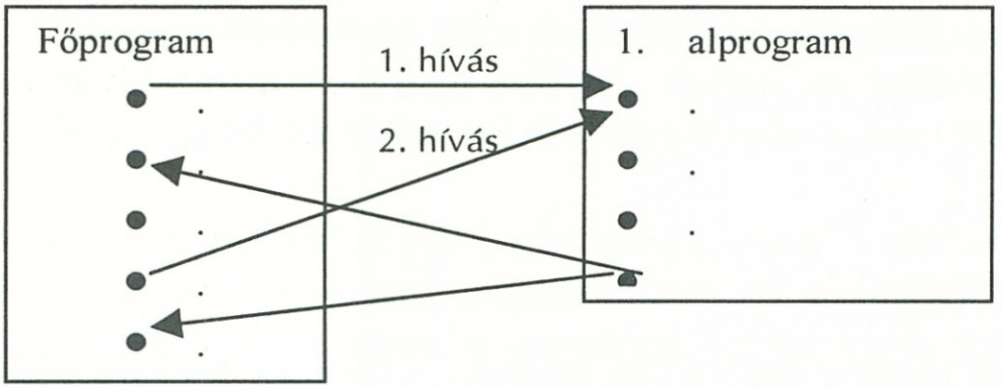
ekkor az utasítás a "Ezermester"-t fogja megjeleníteni.

1.6. Alprogramok

Az eddig bemutatottak már elviekben bármilyen bonyolultsású program végrehajtását lehetővé teszik, ez azonban gyakorlatban nem teljesen igaz. Vannak olyan problémák, melyek az eddig tanultak alapján csak nehezen és bonyolultan valósíthatók meg. Gondoljunk egy korábban már említett példára a telefonálásra. Ha egész nap telefonálnánk, akkor ezt ciklikusan megtehetnénk – tegyük fel, hogy mindig ugyanazt a számot hívjuk -, de mi a teendő, ha a telefonálások közben valami mást is kell csinálni, például le kell gépelni egy levelet. Ha ezt lefordítjuk a számítógép nyelvére, arra az eredményre jutunk, hogy egy állandóan ismétlődő feladat többször is szerepelni fog a program kódjába. Ez nemcsak megnöveli a program hosszát, hanem a tesztelések során sem lesz könnyű egy esetleges hiba felderítése. E probléma meg-

oldására találták ki az alprogramokat. Az alprogramot a program során többször egymástól függetlenül végrehajtandó feladatok végrehajtására kell használni.

Ha egy alprogramot használni akarunk, akkor az alprogramot meg kell hívnunk.



Az alprogramoknak általában maximum 7 további szinten al-alprogramja lehet. Ezeket az al-alprogramokat az alprogramból kell meghívni.

Azt, hogy az alprogram, milyen adatokkal, illetve változókkal dolgozik, az eljárás paramétereivel lehet megadni.

Az alprogramoknak két csoportjuk van:

- függvények
- eljárások

1.6.1. Függvények

A függvények olyan alprogramok, melyek egy vagy több megadott argomentum megadásával számolnak ki egy eredményt. A függvények használatának ez a korlátozása bizonyos feladatok alkalmazására lehetetlenné teszi. A függvény használatának van azonban egy nagy előnye, hogy a függvény beilleszthető bármilyen kifejezésbe. A függvényben zárójelben lévő értékek (x,y,z) a függvény argumentumai:

```
C = fuggveny(x,y,z)*3
```

Ez adott esetben jelentős könnyítést eredményezhet.
A függvényt, mint egyfajta alprogramot használatakor meg kell hívni.

1.6.2. Eljárás

Az eljárás és a függvény közötti legnagyobb különbség az, hogy az eljárás több eredményt is visszaadhat, viszont az eljárás nem használható kifejezésekben.

1.6.3. A főprogram és az alprogram kapcsolata

A főprogram és az alprogram együtt alkotják a programot. A kettő kapcsolatát az adott programnyelv határozza meg. Főprogram nélkül nem lehet használható alprogramot készíteni.

1.6.4. Az azonosítók hatásköre

Az alprogramokban, hasonlóan a programokhoz, definiálhatunk azonosítókat. Az ilyen azonosítót lokális változónak nevezzük, csak az adott alprogramban, illetve az alprogram al-alprogramjaiban használhatók. Ha egy főprogramban definiáltunk egy azonosítót, és az alprogramban is definiáltunk egyet ugyanolyan névvel, akkor az alprogramban definiált változó lesz az érvényes, mihelyt kilépünk az alprogramból a főprogram által definiált változó lesz ismét érvényes. A főprogramban definiált azonosítókat globálisnak nevezzük, és ezeket a program futása során bármikor elérhetjük, kivéve az előbb említett esetet.

2. Programozás Quick Basic nyelven

2.1. Bevezetés

Tankönyvünk gyakorlati részében a Quick Basic programozási nyelvet fogjuk megismerni. Bár a programozás – elvileg legalábbis – független a programozási nyelvtől, de fejben vagy papíron nem lehet megtanulni a programok írását; ugyanúgy, ahogy az autóvezetést sem lehet autó nélkül elsajátítani.

Feltehetik a kérdést, hogy miért pont a Quick Basic nyelvet fogjuk használni. A Basic programozási nyelv, ahogy már a neve* is mutatja, a magas szintű programozási nyelvek közül a legegyszerűbb. Különösebb előkészületek nélkül már működő programokat írhatunk benne, míg egy fejlettebb nyelven – pl. C vagy Pascal – a programnak kötött szerkezete van, a legegyszerűbb programocska kedvéért is sok sort kell begépelnünk. A Basic mindenképpen a kezdők programozási nyelve.

Sokszor feltették nekem azt a kérdést, hogy manapság, amikor szinte minden problémára létezik szoftver, kell-e programozási ismeret azoknak, akik nem profi programozók. A válaszom az, hogy nem kell azoknak, akik a számítógépet csak munkaeszköznek tekintik, de akit már egy kicsit is érdekel a masinája, annak melegen ajánlható. A számítógépes ismeretekhez, a számítógép kultúrájához mindenképpen hozzátartozik a programozás, és segít abban, hogy egy kicsit megismerhessük és megérthessük gépünk „lelkivilágát". Azonfelül a programok írása megtanít bennünket sok olyasmire – logikus gondolkodásra, pontos tervezésre, a kis részletek el nem hanyagolására, stb. –, amit az élet egyéb területein is alkalmazhatunk. Főleg gyerekeknél nagyon haszno-

* A BASIC egy betűszó, az angol *Beginner's All-purpose Symbolic Instruction Code* kifejezésből ered. Kifejlesztői, John Kemény és Thomas Kurtz, egy kezdők által is könnyen elsajátítható, tehát egyszerű, viszonylag kis utasításkészletű, általános célú programozási nyelvnek szánták. A betűszó körülbelül ezt fejezi ki.

sak ezek a képességek; nem is kell magyarázni, hogy mekkora hasznát vehetik ezeknek az iskolában például a matematika (vagy bármely más) órán. Jobb is, ha a figyelmüket ezzel kötjük le: ha már számítógépezik egész nap, mindenképpen jobb, ha programokat ír, mintha harci játékokkal játszana. A programozás van olyan hasznos és kreatív, mint bármelyik építőjáték (egy kicsit az is: építőköveink az egyes utasítások, ezekből kell valami olyat alkotni, ami „megáll a saját lábán").

Természetesen a programírás nem jelenti azt, hogy nosza, essünk neki aztán majd csak lesz belöle valami! Gondos előkészületeket, tervezést igényel. A könyv első részében már megismert algoritmus-leíró eszközök közül a szöveges leírást fogjuk alkalmazni. A rövidebb, egyszerűbb programoknál – amelyek csak egyetlen vezérlési szerkezetet tartalmaznak – a terv akár el is maradhat, hiszen a program átlátható. A hosszabb, bonyolultabb programoknál a tervezés már kötelező.

Az első fejezetben megismerkedünk a Quick Basic fejlesztői környezetével, s ha ezen túl vagyunk, akkor kezdődhet a programírás. Minden fejezetben példákon keresztül mutatjuk be a Quick Basic nyelvet, és a fejezetek végén feladatokat is lehet találni, amelyekkel gyakorolható az adott témakör. A feladatok egy vagy több megoldása a lemezmellékleten is megtalálható. Ha más megoldással, de ugyanazt sikerült elérnie, akkor a feladat ugyanúgy jó, mintha pontosan az általam elképzelt változatot írta volna meg. Minden feladatnak több megoldása lehetséges, bár ezek eltérhetnek egymástól hosszban, hatékonyságban, bonyolultságban ...stb.

Igyekeztem a könyvet úgy elkészíteni, hogy csoportos munkánál tanári kézikönyvként is szolgálhasson, de az otthoni, egyéni tanulás se okozzon gondot a segítségével. Talán mindenki talál benne érdekes és hasznos feladatokat. Célom volt továbbá, hogy a kezdőkkel megszerettessem a programírást és ne legyen elriasztó sem a fogalmazás, sem a feladatok. A programozás nagyon jó és kreatív játék!

Még egy megjegyzés: a tankönyv használatához megkívántatik némi gépkezelői alapismeret. Programok indítása, kész anyagok mentése, betöltése, ...stb. nem szerepel a könyvben.

Jó programozást és jó szórakozást kíván:

a szerző.

2.2. A Quick Basic programozási környezete

2.2.1. A programról általában

A Quick Basic nyelv 1.1-es változatával ismerkedünk meg; ez a program az MS-DOS rendszer részeként a legtöbb IBM-kompatibilis személyi számítógépen megtalálható. A Quick Basic-nek ez a változata nem tud futtatható állományt készíteni, tehát a program végrehajtásához szükség van a Quick Basic környezetre is. Ez azzal magyarázható, hogy a fordítóprogram (compiler) hiányzik a rendszerből, csak sorfordító, ún. interpreter áll a rendelkezésünkre. Ennek az a tulajdonsága, hogy az elkészített programon soronként végigmegy, és minden sorról eldönti, hogy végrehajtható-e; ha igen, akkor végrehajtja, és továbblép a következőre; ha nem, akkor hibaüzenetet ad, és leáll.

A Quick Basic nyelv utasításokból és függvényekből áll. Az utasítások önállóan kiadhatók egy programsorban, a függvények értéket adnak eredményül, ezért általában egy értékadás jobb oldalán szerepelnek.

2.2.2. A Quick Basic indítása

A Quick Basic programot a prompt után begépelt `qbasic` parancscsal indíthatjuk (a `qbasic.exe` állomány rendszerint a `DOS` könyvtárban található meg). Elindítás után egy ablak jelenik meg a monitoron, amelyben az alábbi két lehetőség közül választhatunk:

- Press Enter to see the Survival Guide (Nyomj ENTER-t a Súgó megtekintéséhez)
- Press ESC to clear this dialog box (Nyomj ESC-t a párbeszéd-ablak eltüntetéséhez)

A Súgó használatához angol nyelvtudás, vagy legalábbis a gépkezelés közben felvett némi nyelvismeret szükséges. (A Súgóból való kilépéshez is ESC-t kell nyomni.) Ha már írnánk a programot, akkor is nyomjuk az ESC-t.

2.2.3. A fejlesztői környezet

2.2.3.1. Menüsor

A képernyő legfelső sora a menüsor. A megszokott legördülő menüket találjuk itt. Az egyes menüket vagy egérrel való rákattintással, vagy ALT+kezdőbetű billentyűkombinációval nyithatjuk meg. A menüpontok közül az ENTER-rel választhatunk.

2.2.3.2. Információs sor

A képernyő legalsó sorában található az információs sor. Itt láthatjuk a leggyakrabban használt billentyűkombinációkat, valamint a jobb oldalon a kurzor pozíciójának sor- és oszlop koordinátáit.

2.2.3.3. Programablak

A munkaterület két részre osztott. A felső, nagyobb ablak a programablak. Itt írhatjuk meg a programot. Az ablak fejlécében látható az éppen szerkesztett állomány neve (amíg még nem adtunk neki nevet, addig az *Untitled* felirat olvasható).

Az ablak alján és jobb szélén gördítősáv található, amely a megszokott módon kezelhető egérrel.

2.2.3.4. Parancsablak

A munkaterület alsó, jóval kisebb ablaka az *Inmediate* feliratú parancsablak. Itt a begépelt parancs az ENTER lenyomására azonnal végrehajtódik. Akkor használhatjuk, ha egy utasítást ki akarunk próbálni anélkül, hogy programot kelljen köré írni, vagy ha egy parancsot akarunk közvetlenül kiadni (pl. le akarjuk törölni a képernyőt). A programablak és a parancsablak között az F6 billentyűvel tudunk váltani.

2.2.4. A program szerkesztése

A program utasítások sorozata, amelyet a gép jól meghatározott sorrendben hajt végre. A végrehajtás sorrendje – hacsak valamilyen vezérlési szerkezettel meg nem változtatjuk – az ún. szekvenciális végrehajtás, vagyis az egyes sorokat egymás után, sorban hajtja végre az értelmező program. A programot úgy célszerű elkészíteni, hogy egy sorban egy utasítást szerepeltetünk. Ha egy sorba több utasítást szeretnénk írni, akkor az egyes utasításokat kettősponttal (:) kell egymástól elválasztani.

Ha egy sort begépeltünk, új sort az ENTER billentyűvel kezdhetünk. Ennek hatására a kurzor a következő sorba kerül, a Quick Basic rendszer értelmezőprogramja pedig ellenőrzi, hogy a begépelt utasítás szintaktikailag megfelelő-e. Ha nem, akkor hibaüzenetet kapunk, és megmutatja azt a helyet, ahol szerinte a hiba van. Amennyiben elfogadta a sort, akkor azt látjuk, hogy beformázza az utasításokat: nagybetűssé változtatja, szóközökkel tagolja a programsort, ...stb. Ezért a gépelés során mindegy, hogy kis- avagy nagybetűket használunk, a Quick Basic a végén úgyis olyanra alakítja, amilyennek látni szeretné.

2.2.5. A kész program futtatása

Ha a programunk készen van, és szeretnénk megnézni, hogy mit is csinál, ahhoz le kell futtatnunk: rá kell venni a gépet, hogy hajtsa végre az utasításainkat. A futtatáshoz a SHIFT-F5 billentyűkombinációt kell alkalmazni. A program futásának végeztével a képernyő alsó sorában megjelenik a „Press any key to continue..." (nyomj egy billentyűt a folytatáshoz) üzenet. Egy tetszőleges billentyű lenyomására visszatérünk a programunk szerkesztéséhez.

Sokszor előfordul – főleg a képernyőkép tervezésénél –, hogy jó lenne látni az előző programfutás eredményét. Nyomjuk le az F4 funkcióbillentyűt, ennek a segítségével a kék ablak „mögé" nézhetünk. Tetszőleges billentyű lenyomására visszatérhetünk a programunk szerkesztéséhez.

A program futása megszakítható a CTRL-C vagy CTRL-BREAK billentyűk lenyomásával.

2.2.6. A képernyő felépítése

A program írása során különösen sokat foglalkozunk azzal, hogy a képernyőre íratunk valamilyen értéket vagy szöveget. Ahhoz, hogy esztétikus feliratokat tudjunk készíteni, pontosan ismernünk kell a képernyő adta lehetőségeket, elsősorban a méretét.

A képernyő alapállapotban **25 sort és 80 oszlopot** tartalmaz. (Egy sor és egy oszlop által meghatározott téglalapba egyetlen karakter kerülhet.) A sorok számozása 1-től 25-ig, az oszlopoké 1-től 80-ig terjed, a kiindulási pont a bal felső sarok. A Quick Basic az alsó két sort (24. és 25.) magának tartja fent, ezért ha valamit ide írunk, akkor furcsa dolgok történhetnek. Ajánlatos tehát megelégedni az első 23 sor használatával, ez a feladatok nagy többségénél tökéletesen elég is.

A képernyőn mindig tudnunk kell, hogy éppen hol járunk, mindig van egy **aktuális hely.** Ezt jelzi számunkra a **kurzor**. A kurzor pozícióját meg tudjuk változtatni. Ez azért fontos, mert a képernyőre íratás mindig a kurzor helyén kezdődik. Alapállapotban a kurzor a képernyő bal felső sarkában van (1. sor, 1. oszlop).

A monitoron egy sor-oszlop koordinátákkal megadott helyen egy karakter lehet. Minden karakterhez hozzárendelhetünk színeket is, nevezetesen egy **karakterszínt,** ami a karakter vonalainak és pontjainak színét jelenti; illetve egy **háttérszínt**, ami a karakter mögötti képernyőrész (az a bizonyos kis téglalap) színét jelenti.

2.3. Az első lépések: Képernyőkezelés

A programok a legtöbb esetben úgy működnek, hogy valamilyen információt várnak a felhasználótól (bemenő adat), és ebből aztán valamilyen kimenő adatot gyártanak, amit a felhasználóval közölni illik. Tehát a számítógép folyamatos kommunikációban áll a kezelőjével, és a kommunikáció színtere a monitor. Azaz kérdéseket és válaszokat kell a képernyőre kiírni, pontosabban kiíratni.

A számítógép-felhasználók általában igényesek. Ezért általában törekedjünk arra, hogy a képernyőre írt kérdéseink és adataink esztétikusan nézzenek ki. tehát nem elég csak úgy kiíratni; tudnunk kell a képernyő egy adott pontjára írni, tudnunk kell színeket használni, ...stb.

A képernyőkezelés legfontosabb utasításai a következők (használatukat a következő fejezet mutatja be):

PRINT: kiíratás a képernyőre
CLS: a képernyő letörlése
LOCATE: a kurzor helyének megadása

2.3.1. Egyszerű kiíratások

1. példa: Írassuk ki a nevünket a képernyőre!

Megoldás: Ez a világ legrövidebb programja, egyetlen utasításból áll. *

```
PRINT "Zseni Alfonz"
```

A programot **A kész program futtatása** *c. fejezetben leírtak alapján a SHIFT-F5 billentyűkombinációval tudjuk végrehajtatni. Figyeljük meg, hogy az* ***a szöveg****, amit ki szeretnénk íratni,* ***macskakörmök között szerepel****!*

A megoldással az a gond, hogy alig vesszük észre a képernyőn a *„Zseni Alfonz"* feliratot, hiszen az tele van egyéb, már előzőleg is ott lévő szövegekkel. A megoldás: képernyőtörlés.

2. példa: Írassuk ki a nevünket a tiszta, letörölt képernyőre!

Megoldás: Az előző programunkat egyetlen utasítással kell kiegészítenünk.

* A programokban a PRINT utasítás általában nagyon sokszor szerepel, ezért – hogy a gépelés könnyebb legyen – a PRINT helyett elég egy ? (kérdőjel) leírása.

```
CLS
PRINT "Zseni Alfonz"
```

Az előző példákban láthattuk, hogy a szöveg a képernyő bal felső sarkába került. Miért? A PRINT utasítás mindig oda ír, „ahol éppen tartunk" a monitoron. ez kezdetben a képernyő bal felső sarka. De minden esetben van a képernyőn egy aktuális hely, és ezt jelzi a *kurzor*. **Tehát a PRINT oda írja ki a szöveget, ahol a kurzor van.** Ahhoz, hogy tetszőleges helyre írjunk, tehát a kurzor pozícióját kell megadnunk. Erre szolgál a LOCATE utasítás. Használata:

LOCATE sor, oszlop

ahol a sor és az oszlop a képernyő egy pontjának koordinátáit jelenti. Természetesen nem kötelező mindkét paramétert megadni.

3. példa: Írassuk ki a nevünket a képernyő közepére!

Megoldás: A kiíratás előtt meg kell adnunk azt, hogy hová írjunk. A képernyő közepe kb. a 12. sor 40. oszlopa. (l. ***A Képernyő méretei*** c. fejezetben)

```
CLS
LOCATE 12, 34
PRINT "Zseni Alfonz"
```

Most már a képernyő tetszőleges helyére tudunk tetszőleges szöveget kiíratni. Most nézzük meg, hogyan tudunk több szöveget kiíratni!

4. példa: Írassuk ki a képernyőre a nevünket, és alá a telefonszámunkat!

Megoldás: Egyetlen kiíró utasítással egy szöveget írathatunk ki (majd később láthatjuk, hogy ez nem feltétlenül van így), tehát két PRINT utasításra van szükség.

```
CLS
PRINT "Zseni Alfonz"
PRINT "123-45-67"
```

A fentiekben azt mondtuk, hogy a PRINT utasítás oda írja ki a szöveget, ahol a kurzor van. Ebből a példából kitűnik, hogy a kurzor az első kiíratás után (PRINT „Zseni Alfonz") **automatikusan a következő sor elejére kerül,** *hiszen a második kiíratás* (PRINT „123-45-67") *eredménye ott jelent meg.*

5. példa: Írassuk ki a képernyő közepére a nevünket és a telefonszámunkat egymás alá!

Megoldás: Az előző programhoz képest csak az a különbség, hogy most a képernyő közepére kell kiíratni. Ehhez az előzőekben megismert LOCATE utasítást kell alkalmaznunk.

```
CLS
LOCATE 12, 34
PRINT "Zseni Alfonz"
LOCATE ,34
PRINT "123-45-67"
```

Ebben a példában van egy nagyon furcsa sor: `LOCATE ,34`. *Mit is jelent ez? Próbáljuk ki, hogy mit csinál a program, ha kihagyjuk ezt a sort! A „123-45-67" szöveg a 13. sor elejére kerül, hiszen a kurzor az előző kiíratás után ide került. Nekünk a telefonszámot ennek a sornak 34. oszlopába kell elhelyeznünk. Megadhatnánk ezt úgy is, hogy kiírjuk:* `LOCATE 13, 34`. *De mivel a sor koordinátáján nem akarunk változtatni, ezért ezt üresen hagyhatjuk. A vessző azért kell, hogy a program tudja: a 34 a második paraméter, tehát az oszlop koordinátája.*

Egyetlen PRINT utasítással nem csak egyetlen szöveg kiíratása lehetséges, hanem többé is. Ekkor a kiíratandó szövegeket a PRINT után fel kell sorolni vesszővel, vagy pontosvesszővel elválasztva. A pontosvessző hatására a kurzor nem megy a következő sor elejére, hanem helyben marad, tehát az egyes szövegek egymás mellé kerülnek. A vessző hatása hasonló, de a kurzor nem helyben marad, hanem az aktuális sorban néhány karakterrel odébb ugrik. (A képernyőn 12 karakterenként található egy-egy képzeletbeli függőleges vonal. A vessző hatására a kurzor a következő ilyen pozícióra ugrik.)

6. példa: Változtassuk meg az előző példaprogramot úgy, hogy a szövegeket egymás mellé írjuk!

Megoldás: Használjunk vesszőt az egyes szövegek elválasztására, és csak egyetlen PRINT utasítást alkalmazzunk.

```
CLS
LOCATE 12, 25
PRINT "Zseni Alfonz", "123-45-67"
```

Próbáljuk ki, mi történik, ha a vesszőt pontosvesszőre cseréljük!

7. példa: Írassuk ki a nevünket és a telefonszámunkat a képernyő közepére egymás alá, de írjuk eléjük a kategóriát! (Név: Zseni Alfonz, telefonszám: 123-45-67)

Megoldás:

```
CLS
LOCATE 12, 30
PRINT "Név: "; "Zseni Alfonz"
LOCATE ,30
PRINT "Telefonszám: "; "123-45-67"
```

Feladatok:

1. Írassa ki a 20. sorba, a képernyő jobb szélére a nevét!
2. Írassa ki a 12. sor 30. oszlopába: "Helló!!!"!
3. Írassa ki a 17. sorba egymás mellé: alma, körte, barack!
4. Írasson ki a képernyő 7. sorának nagyjából a közepére egy tetszőleges szöveget!
5. Írassa ki a képernyőre a *János vitéz* első négy sorát, nagyjából középre!

   „Tüzesen süt le a nyári nap sugára
   Az ég tetejéről a juhászbojtárra.
   Felesleges dolog sütnie oly nagyon
   A juhásznak úgyis nagy melege vagyon."
6. Készítsen el a képernyőre egy kis névjegyet (tartalmazza a név, foglalkozás, cím, telefon adatokat)!
7. Készítsen a képernyőre egy kis meghívót, amelyben üzletfelét hívja meg egy tárgyalásra!
8. A billentyűzeten nem meglévő karaktereket az ASCII kódjuk segítségével, az ALT+kód leütésével lehet bevinni. Rajzoljon a képernyőre keretet! (A kódokat a tankönyv végén megtalálja.)

2.3.2. Használjunk színeket!

A képernyő sokkal látványosabbá és izgalmasabbá tehető, ha mondanivalónkat színek alkalmazásával közöljük.

A képernyő minden karakter-helyéhez tartozik egy karakterszín és egy háttérszín. Azt, hogy milyen színeket használunk a kiíratás során, az aktuális karakterszín, illetve aktuális háttérszín adja meg. Az aktuális színek kezdetben: háttérszín: fekete, karakterszín: fehér. Az aktuális színeket tudjuk megváltoztatni a COLOR utasítással. Az utasítás használata:

COLOR karakterszín, háttérszín

ahol mind a karakterszín, mind a háttérszín a következő kódokkal adható meg (zárójelben a színek angol elnevezései):

fekete (black)	0
sötétkék (blue)	1
zöld (green)	2
cián (cyan)	3
vörös (red)	4
bíbor (magenta)	5
barna (brown)	6
halványszürke (gray)	7
sötétszürke (darkgray)	8
világoskék (lightblue)	9
világoszöld (lightgreen)	10
világoscián (lightcyan)	11
világosvörös (lightred)	12
világosbíbor (lightmagenta)	13
sárga (yellow)	14
fehér (white)	15

Egy COLOR utasítás kiadása után mindaddig a megadott színnel írunk a képernyőre, amíg egy újabb COLOR utasítással meg nem változtatjuk azt!

A világos színek általában sötét megfelelőik intenzív változatai.

Háttérszínnek csak a 0-7 kódú színek adhatók, ellenkező esetben a program a kódból 8-at kivon (így lesz pl. a sárga háttérszínből barna).

A szöveget villogtathatjuk, ha a karakterszín kódjához 16-ot hozzáadunk (pl. villogó piros szöveget kapunk, ha a vörös kódjához (4) 16-ot hozzáadva a `COLOR 20` utasítást adjuk ki.

8. példa: Írassuk ki a nevünket a képernyő közepére világoszöld színnel!

Megoldás: A világoszöld szín kódja a 10, tehát a COLOR utasításban ezt adjuk meg. Fontos, hogy a COLOR utasítás mindig előzze meg azt a PRINT utasítást, amelyre vonatkoztatni akarjuk!

```
CLS
LOCATE 12, 34
COLOR 10
PRINT "Zseni Alfonz"
```

9. példa: Írassuk ki a nevünket és telefonszámunkat egymás alá, különböző karakter- és háttérszíneket használva!

Megoldás:

```
CLS
COLOR 7, 4
PRINT "Zseni Alfonz"
COLOR 14, 1
PRINT "123-45-67"
```

Szükségünk lehet arra is, hogy a képernyő teljes területének hátterét megváltoztassuk. Ehhez egy kicsit térjünk vissza a már megismert CLS utasításra. A CLS feladata a képernyőtörlés. Igen ám, de hogyan? Úgy, hogy a képernyő minden egyes karakterpozícióját az aktuális háttérszínűre színezi! Ez tehát alkalmas lehet arra, hogy a teljes képernyő hátterét megváltoztassuk, csak az kell hozzá, hogy a CLS előtt kell egy másik háttérszínt megadni a COLOR utasítással.

10. példa: Színezzük a képernyőt zöldre, és erre fehérrel írjuk ki a nevünket!

Megoldás:

```
COLOR 15, 2
CLS
PRINT "Zseni Alfonz"
```

Az első sorban megadott COLOR utasítás beállítja az aktuális karakterszínt fehérre (15), az aktuális háttérszínt pedig zöldre (2). A második sorban a CLS háttérszínűre, azaz zöldre színezi a képernyőt, majd a harmadik sor kiíratása fehérrel (az aktuális karakterszínnel) történik.

Feladatok:

9. Vörös képernyőre villogó fehér színnel írassa ki: "Ez nem olvasható!!!"
10. Írjon ki a képernyő öt különböző helyére öt különböző színnel öt keresztnevet!
11. Az előző fejezet végén található feladatokat(1-7) alakítsa át színesre!
12. Készítsen egy színes plakátot a képernyőre, amely a cég hétvégi küldöttgyűlésére hívja fel a figyelmet!

2.3.3. Egyszerű számolások

Az előzőekben megismertük a szövegek képernyőre íratásának lehetőségeit. Most próbáljunk számolni egy kicsit! használjuk a számítógépet úgy, mint egy számológépet. Figyeljük meg a következő programot:

11. példa:

```
CLS
PRINT "6*8"
PRINT 6*8
```

*A programot lefuttatva a következőket látjuk: az első sorban a "6*8"* ***szöveget****, a második sorban a 6*8* ***számítás eredményét****, a 48-at. Amennyiben egy karaktersorozatot macskakörmök közé teszünk, akkor azt a Quick Basic szövegnek veszi, akármilyen karakterekből álljon is. ha nem tesszük idézőjelek közé, akkor a program felismeri, hogy számokról és egy műveleti jelről van szó, és el tudja végezni a számítást, majd a kiíró utasítás hatására az eredmény fog megjelenni a monitoron.*

12. példa: A 4 és 7 számokkal végezzük el az alapműveleteket, és írassuk ki az eredményt!

Megoldás:

```
CLS
PRINT "4+7=";
PRINT 4+7
PRINT "7-4=";
PRINT 7-4
PRINT "4*7=";
PRINT 4*7
PRINT "7/4=";
PRINT 7/4
```

Figyeljük meg a fenti programot! Egy nagyon fontos íratlan szabályt alkalmazunk: ***nem írunk ki a képernyőre adatokat anélkül, hogy meg ne magyaráznánk azokat.*** *A programokat legtöbb esetben nem a saját magunk számára írjuk, és bár mi tudjuk, hogy mi az a négy szám, ami a monitoron megjelenik, de más már nem fogja tudni. Ezért soha ne sajnáljuk az időt arra, hogy a programot ellássuk az efféle, a program működésére hatással nem lévő, de nagyon fontos kiíratásokra! Ezzel e programunkat „intelligenssé" tehetjük. Figyeljük meg a magyarázó sorok végére kitett pontosvesszőt. Ennek szerepét már ismertettük: ott tartja a*

kurzort a kiírt szöveg végénél, és a következő kiírás ide fog kerülni. Természetesen megtehetjük, hogy az egymás melletti szövegeket egyetlen PRINT utasításba írjuk át a következő módon:

```
PRINT "4+7="; 4+7
```

de ez első látásra talán nehezen értelmezhető (most még, de később használjuk ezt az alakot), ezért választottuk a példában a hosszabb, de egyszerűbb írásmódot.

*A példából kitűnik, hogy az alapműveletek jelei a következők: + (összeadás), - (kivonás), * (szorzás) és / (osztás). A hatványozás jele a ^ (kalap), de az egyszerűbb hatványozást (négyzetre vagy köbre emelés) kiválthatjuk szorzással.*

Feladatok:

13. Számítsa ki a 4+7*3 összeg és a (4+7)*3 szorzat értékét! Mit figyel meg a műveletek sorrendjéről?
14. Írasson ki két számot (25 és 90) egymás mellé, majd végezze el az alapműveleteket velük, és az eredményt írja a képernyő 4-7-ik soraiba, a 20. oszlopban kezdve. Az eredményeket lássa el szöveges magyarázatokkal (pl. Összeg: 115), és használjon színeket!

2.3.4. A képernyő felbontásának megváltoztatása

Időnként szükség lehet arra, hogy a képernyőre több adatot írjunk ki, mint amennyi ott elfér. Ekkor meg kell változtatnunk a képernyő sorainak és oszlopainak számát. erre szolgál a WIDTH utasítás. Használata:

WIDTH oszlop, sor

ahol az oszlop 40 vagy 80 lehet, a sor pedig 25, 43 vagy 50. (Nem tévedés, itt előbb az oszlopok számát kell megadni, ellentétben a LOCATE utasítással!)

A WIDTH utasítást a program elején célszerű elhelyezni, de használata csak akkor javasolt, ha a meglévő képernyő kicsi az adataink számára. Értelemszerűen az oszlopok számét 40-re csökkenteni 80-ról itt nem érdemes, csak a sorok számát növelni 25-ről 43-ra vagy 50-re.

13. példa: Méretezzük át a képernyőt 50 sorosra, és írassuk ki a nevünket az 5. sor 10. oszlopába!

Megoldás:

```
WIDTH 80, 50
CLS
LOCATE 5, 10
PRINT "Zseni Alfonz"
```

Próbáljuk meg változtatni a WIDTH paramétereit, és figyeljük meg a kiírt szöveg változásait! Próbáljunk ki minden lehetséges párosítást (40–25, 40–43, 40–50, 80–25, 80–43, 80–50)!

2.4. Változók

A változók a program adattárolói. Minden változó egy adatot képes tárolni. Az adatok különfélék lehetnek, pl. numerikus (szám jellegű) vagy szöveges, de mindegyiket változókban tárolhatjuk. A változóknak kell adni egy **azonosítót** (nevet), amellyel hivatkozni tudunk rá, ha szükséges valamely művelet elvégzéséhez; illetve minden változónak van **értéke**, amellyel dolgozunk. A változó neve vagy azonosítója tetszőleges lehet, de a következő szabályokat Ezért be kell tartanunk:

- az azonosító tartalmazhatja az angol ABC betűit (kis és nagybetűk között a Quick Basic nem tesz különbséget), és számokat, de csak betűvel kezdődhet,
- az azonosító nem tartalmazhat jeleket, valamint szóközt sem, illetve
- az azonosítónak tükröznie kell, hogy milyen típusú az adott változó (numerikus vagy szöveges), ezért a szöveges változók nevének utolsó karaktere a $ (dollárjel) kell hogy legyen!

Tehát érvényes numerikus változónevek lehetnek: a, b, abc, kiskutya, dollar, forint, stb.
Érvényes szöveges változónevek lehetnek: a$, kutya$, nev$, cim$, stb.
Érvénytelen változónevek: név$, cím$, szám, stb. (mert ékezetes betűket tartalmaznak), tel szám, anya neve, stb. (mert szóközt tartalmaznak), stb.

A program futása alatt a változók értéke változhat (innen a név is). Kezdetben a numerikus változók értéke (kezdőérték) 0 (nulla), a szöveges változóké "" (üres szöveg, vagy üres sztring) (vigyázat, a két idézőjel között nem szóköz van, hanem semmi!).

2.4.1. Értékadás

A változóknak a LET utasítással adhatunk értéket. A LET jelentése „legyen", formája:

LET változónév = érték (olvasd: legyen a változó egyenlő „érték"-kel)

példák:
LET a = 12 (legyen az **a** változó értéke 12)
LET kutya = 35 (legyen a **kutya** változó értéke 35)
LET nev$ = "Zseni Alfonz" (Legyen a **nev$** változó értéke "Zseni Alfonz").

Figyeljük meg, hogy a szöveges változók értékét (tehát magát a szöveget) itt is idézőjelek között kell szerepeltetni, akárcsak a PRINT utasításnál.

A értékadásnál a LET szócska elhagyható.

2.4.2. Változók értékének kiíratása

Az előbbiekben már megismerkedtünk a kiíratással, azaz a PRINT utasítással. Egy változó értékének kiíratásakor a PRINT után meg kell adni annak a változónak az azonosítóját, amelyet ki szeretnénk íratni.

14. példa: Adjunk értéket az **x** változónak, és írassuk ki a képernyő közepére!

Megoldás:

```
CLS
LET x = 25
LOCATE 12, 40
PRINT x
```

A `PRINT x` *programsor hatására nem az "x" karakter, hanem az* ***x*** *változó értéke (25) kerül kiíratásra. A változó nevét (azonosítóját) soha nem szabad idézőjelek közé tenni!*

15. példa: Adjunk értéket a **szam** változónak, írassuk ki, majd módosítsuk az értékét, és azt is írassuk ki!

Megoldás:

```
CLS
LET szam = 25
PRINT szam
LET szam = 100
PRINT szam
```

A program a 25, majd alatta a 100 számokat adja eredményül.

16. példa: Adjunk értéket a **nev$** szöveges változónak, és írassuk ki a képernyő közepére!

Megoldás:

```
CLS
LET nev$ = "Halihó!!"
LOCATE 12, 36
PRINT nev$
```

A program eredménye a "Halihó!!" szöveg a képernyő közepén.

2.4.3. A változók egyszerű alkalmazásai

17. példa: Változók használatával számítsuk ki, hogy 1200 forint hány dollárnak felel meg, és az eredményt jelenítsük meg a képernyőn (1$ a könyv írásakor kb. 210 Ft-ot ér)!

Megoldás: Egy-egy változót használunk a forint, illetve a dollár értékének tárolásához. Legyenek ezek **forint** és **dollar**. (Célszerű mindig ún. beszédes változónevet használni, ami megmondja, hogy milyen jellegű értéket tárolunk benne.)
Fentebb említettük, hogy értékadásnál a LET szócska elmaradhat, innentől kezdve nem is írjuk ki.

```
CLS
forint = 1200
dollar = forint / 210
PRINT "1200 Ft"; dollar; "$-nak felel meg"
```

Figyeljük meg, hogy a **dollar** *változó egy számítás eredményét (Ft/210) kapta értékül az értékadás során, a forint változó viszont egy konkrét számot (1200). Az utolsó sorban az eredmény közlésénél alkalmazkodtunk ahhoz a szabályhoz, hogy az eredményünket magyarázzuk. A PRINT utasításban három értéket íratunk ki egymás mellé (erről gondoskodik a pontosvessző); két szöveget, ami a tulajdonképpeni magyarázat, illetve egy változó eredményét, ami tulajdonképpen a számítás végeredménye.*

Feladatok: (a feladatokat változók alkalmazásával oldja meg!)

15. Számítsa ki, hogy 15$ hány forintnak felel meg!
16. Számítsa ki, hogy 120MB hány KB-nak felel meg!

17. Számítsa ki, hogy egy 6 láb 4 hüvelyk magas ember hány centiméter (láb=30,48 cm; hüvelyk: 2,54 cm)!
18. Számítsa ki, hogy délután 17 óra 20 perckor hány perc telt el a napból!
19. Számítsa ki, hogy 120 000 Ft mennyit kamatozik egy év alatt 15%-os kamatláb mellett!
20. Számítsa ki, hogy mennyibe kerül egy utazás autóval Hatvanba (60 km), ha a benzin literje 150 Ft és a kocsi 6 litert fogyaszt 100 km-en!
21. Az **a** változó értéke legyen 150, a **b** változó értéke pedig 43. Cserélje meg a két változó értékét!

2.4.4. Értékadás a program futása közben

Az előző fejezet feladatai jók, de nem általánosak. Nem tudjuk kiszámítani, hogy valahány dollár mennyi forintnak felel meg, csak azt, hogy 15$ mennyit ér. Az ilyen program öncélú, gyakorlatilag semmit sem ér. A helyes működés az lenne, ha a programocskánk megkérdezné: hány dollárt akarsz váltani? – és a kérdésre adott válasz alapján számítaná és írná ki az eredményt. Ez annyit jelent, hogy a program megírásakor nem ismerjük még a változó értékét, csak futás közben derül ki, tehát nem is adhatjuk meg előre az értékadó utasítással. Rá kell venni a programot, hogy **kérdezzen, várja meg a választ és a kapott választ tárolja el egy változóban**. Ennek az utasítása az INPUT. Használata:

INPUT változónév

ahol a változónév annak a változónak az azonosítója, melyikben a bekért értéket eltároljuk.

Az utasítás használatát a következő példa mutatja meg.

18. példa: Írjuk át a dollár-forint átváltásos programunkat INPUT alkalmazásával!

Megoldás:

```
CLS
INPUT forint
dollar = forint / 210
PRINT "1200 Ft"; dollar; "$-nak felel meg"
```

Ha lefuttatjuk a programot, akkor azt látjuk, hogy a képernyő bal felső sarkában megjelent egy kérdőjel. Most ért a program az INPUT*-hoz, tehát most várakozik a válaszra. Adjuk meg az 1200-at, és a gépelést zárjuk le az ENTER-rel! Most megint meg-*

kapjuk az előző eredményt! Most futtassuk újra a programot, de most 5800-at adjunk meg! Erre a számítás eredménye is más lett. Próbálja ki, hogy mi történik akkor, ha nem számot válaszol a kérdésre! Futtassuk újra a programot, és adjuk azt a választ, hogy "ezer" (szöveggel)! A program jelez, hogy ez nem megfelelő (hiszen a ***forint*** *nevű változónk numerikus típusú), kiírja a képernyőre a „Redo from start" (újra elölről) üzenetet, és ismét felteszi a kérdést (megjelenik a kérdőjel). Addig nem fogadja el a begépelt adatot, ameddig nem numerikus típusú (tehát szám) adatot adunk neki.*

Az INPUT utasítás segítségével elérhetjük tehát, hogy kommunikáljunk a programunkkal. Igen ám, de micsoda beszélgetés ez? A programunk nem tesz fel kérdést, csak várja a választ. Az, hogy a programunk feltesz egy kérdést azt jelenti, hogy kiír a képernyőre egy kérdést, ezt pedig már jól ismerjük: PRINT.

19. példa: Írjuk át a fenti programot úgy, hogy kérdezzen!

Megoldás: Egy új sort kell beírni az INPUT utasítás elé

```
CLS
PRINT "Hány forintod van? Megmondom mennyi $-t ér!"
INPUT forint
dollar = forint / 210
PRINT "1200 Ft"; dollar; "$-nak felel meg"
```

A kérdés feltevésénél nagyon ügyeljünk arra, hogy az egyszerű, könnyen érthető és egyértelmű legyen. Pontosan magyarázza meg, hogy mit is akarunk a felhasználótól, de ne legyen terjengős. Nem szabad sajnálni a fáradtságot egy-egy jó kiírás megfogalmazására, mert bár ez nem befolyásolja a program működését, de nagyon fontos. Ha a program saját magunknak készül, és tudjuk, hogy mi a kérdés, akkor is ki kell íratni, hiszen két hónap múlva nem biztos, hogy még mindig emlékszünk rá. Ha másnak készítjük, akkor felesleges is, hogy kihangsúlyozzuk a fontosságát, ez magától értetődik.

Nem feltétlenül kell a kérdés feltevéséhez belekeverni a dologba a PRINT utasítást, az INPUT képes a kérdés feltételére is. Ekkor az alakja a következő:

INPUT "Kérdés szövege"; változónév vagy

INPUT "Kérdés szövege", változónév

A két alak az elválasztójelben (vessző, illetve pontosvessző) tér el egymástól. A különbség az, hogy a vessző alkalmazásával az INPUT nem jeleníti meg a kérdőjelet.

20. példa: Írjuk át az előző programot az új ismereteink segítségével!

Megoldás:

```
CLS
INPUT "Hány forintod van? Megmondom mennyi $-t
ér!", forint
dollar = forint / 210
PRINT "1200 Ft"; dollar; "$-nak felel meg"
```

Az INPUT *sorban a vesszőt alkalmazzuk, mert nincs értelme kitenni a kérdőjelet.*

21. példa Készítsünk programot, amelyben bekérünk két számot és kiíratjuk az összegüket!

Megoldás: Most nem egy, hanem két számot kell bekérnünk, ehhez pedig két különböző változóra van szükség.

```
CLS
INPUT "Kérem az első számot: ";, a
INPUT "Kérem a második számot: ";, b
osszeg = a+b
PRINT "A két szám összege: "; osszeg
```

Figyeljünk arra, hogy a különböző változók különböző azonosítóval rendelkezzenek! A program megoldható az ***osszeg*** *változó használata nélkül is, ekkor a kiíratásnál közvetlenül az* ***a/b*** *értéket kell kiíratni.*

Feladatok:

22. Írja át az előző fejezet feladatait(13-20) az INPUT utasítás használatával, vagyis az adatokat ne adja meg előre, hanem mindegyikre kérdezzen rá!
23. Kérjen be egy szöveget (ügyeljen a változó nevére!), például egy nevet, és írja ki a képernyő közepére!
24. Kérjen be egy szöveget (pl. egy nevet) és egy színt! A bekért szöveget a bekért színnel jelenítse meg a képernyő közepén!
25. Kérjen be két számot, majd írassa ki az összegüket, különbségüket, szorzatukat és hányadosukat!
26. Módosítsa a névjegy-készítő programot úgy, hogy ne a saját névjegyét készítse el a képernyőn, hanem azét, aki a gép elé ül!

(Kérdezze meg a nevét, címét, stb., és ezeket a szövegeket írassa ki a képernyőre. A névjegy legyen esztétikus, alkalmazzon színeket is!)

2.5. Számlálós ciklusok

A programírás során sokszor kerülünk olyan helyzetbe, hogy egy utasítást vagy utasítássorozatot sokszor kell végrehajtani. Ilyenkor megoldás lehet az, ha sokszor egymás után leírjuk az ismétlendő részt, de ez unalmas is, felesleges is, és ha a végrehajtások száma elég sok, akkor bizony elég sokáig is tart. Egyszerűbb megoldás az, ha azt mondom: itt van ez az utasítássor, ezt hajtsd végre valahányszor. Ezt a vezérlési szerkezetet nevezzük ciklusnak.

A feladatok egy részénél tudjuk előre, hogy a ciklusunknak hányszor kell végrehajtani azt a bizonyos utasítássorozatot, más részüknél azonban nem – például ha a ciklusnak egy feltétel teljesülése esetén kell leállnia. Az előbbi ciklust számlálós ciklusnak, az utóbbit tesztelős ciklusnak nevezzük. Ebben a fejezetben csak a számlálós ciklusokkal foglalkozunk, a tesztelős ciklusokat egy kicsit későbbre hagyjuk.

A számlálós ciklusok szerkezete a következő lenne a leírtak alapján:

```
Ciklus valahányszor (ez a ciklusfej)
    Ciklusmag (ez az az utasítás vagy utasítás-sorozat, amelyet többször
kell végrehajtani)
Ciklus vége
```

Ezzel szemben a programozás során nem ezt alkalmazzuk, hanem definiálunk egy ún. ciklusváltozót, amely számlálja a végrehajtásokat (innen az elnevezés) egy adott kezdőértéktől elkezdve egészen a végértékig.

```
Ciklus ciklusváltozó=kezdőérték-től végérték-ig (ez a ciklusfej)
    Ciklusmag
Ciklus vége
```

Ilyenkor a végrehajtások számát úgy kapjuk meg, hogy a végértékből kivonjuk a kezdőértéket és hozzáadunk egyet (hiszen a kezdőértéket magát is bele kell számítanunk).

A Quick Basic a számlálós ciklust a FOR-NEXT kulcsszópárral szervezi. Használata:

FOR ciklusváltozó=kezdőérték **TO** végérték
ciklusmag (ezek azok az utasítások, amelyeket ismételni kell)
NEXT ciklusváltozó

22. példa: Írjuk tele a képernyő első sorát # jelekkel!

Megoldás: az ismételni kívánt utasítás egy # jel kiíratása, tehát a ciklusmagban egy PRINT „#" sor lesz. Ezt 80-szor kell végrehajtani, hiszen a képernyő oszlopainak száma 80. A ciklusváltozó 1-től 80-ig fog elszámolni.

```
CLS
FOR i = 1 TO 80
    PRINT "#";
NEXT i
```

A program a következőképpen működik: az első FOR *hatására az* ***i*** *változó felveszi a kezdőértéket (1). kiírja a # jelet, majd következik a* NEXT*. Ennek hatására a vezérlés visszatér a* FOR *utasításra (2. sor), itt* ***az i változó értéke 1-gyel nő.*** *Most a program megvizsgálja ezt az értéket: ha már túlhaladta a végértéket (80), akkor a vezérlés a* NEXT *utáni első sorba kerül (illetve annak hiányában a program véget ér), különben újra végrehajtja a ciklusmagban szereplő utasítás(oka)t. Teszi ezt egészen addig, amíg a ciklusváltozó értéke meg nem haladja a végértéket. A ciklusváltozó természetesen bármilyen azonosítót kaphat, csak numerikus legyen. Itt is célszerű a „beszédes" nevek használata. Ha csak számolunk vele – mint példánkban is – akkor a neve nem érdekes, ilyenkor az i, j, k, stb. betűket szoktuk alkalmazni (ez persze csak konvenció, el lehet térni tőle). Egy programban sok ciklus szerepelhet, a ciklusok egymásba ágyazhatók, ezért van arra szükség, hogy a* NEXT *sorban megadjuk a ciklusváltozót, ezzel azonosítva azt a ciklust, amelyiket le kell zárni. (A Quick Basic-ben ez elhagyható, amennyiben egyértelmű, hogy melyik ciklust zárjuk le, azaz csak egyetlen ciklusunk fut.) A* FOR *és* NEXT *utasítások között szereplő utasításokat (tehát a ciklusmagot) egy kicsivel beljebb írtam. Ez azt a célt szolgálja, hogy a program áttekinthetőbbé váljon, hiszen első ránézésre lehet látni a ciklus kezdetét* (FOR) *és végét* (NEXT)*. A Basic persze enélkül is pontosan fogja a programot végrehajtani, ez csak a saját munkánk megkönnyítése miatt célszerű. Úgy mondjuk ezt, hogy a* ***programot struktúráljuk****. A későbbiekben az írásmódot következetesen viszem tovább, és az olvasónak is melegen ajánlom a betartását.*

Figyeljünk rá, hogy a programban a PRINT *sor végére tegyük ki a pontosvesszőt, mert e nélkül a # jelek egymás alá kerülnek!*

23. példa: Írassuk ki a nevünket tízszer egymás alá az ötödik sorban kezdve!

Megoldás:

```
CLS
LOCATE 5
FOR i = 1 TO 10
    PRINT "Zseni Alfonz"
NEXT i
```

A LOCATE *utasítással érjük el, hogy az első kiíratás az ötödik sorba kerüljön. Figyeljük meg, hogy a* ***ciklus előtt*** *szerepel, ezáltal csak a ciklusmag első végrehajtására lesz hatással.*

24. példa: Írassuk ki a nevünket tízszer egymás alá az ötödik sorban kezdve, de ne a sor elejére, hanem a 30. oszlopba!

Megoldás:

```
CLS
LOCATE 5
FOR i = 1 TO 10
    LOCATE ,30
    PRINT "Zseni Alfonz"
NEXT i
```

A program egy LOCATE *utasítással bővült, amely ezúttal a* ***ciklusmagban*** *kapott helyet. Feladata az adott sorban a 30. oszlopra pozícionálás. Azért került a ciklusmagba, mert mindegyik kiíratásra hatással kell legyen, nem csak az elsőre.*

Feladatok:

27. Írja tele a képernyő 10. sorát "@" jelekkel!
28. Kérjen be egy számot 1 és 20 között, és a bekért számnak megfelelő sort írja tele "@" jelekkel!
29. Írja tele a képernyő 70. oszlopát "@" jelekkel!
30. Kérjen be egy számot 1 és 80 között, és a bekért számnak megfelelő oszlopot írja tele "@" jelekkel!
31. Írjon ki a képernyő 12. sorában a 30. oszlopnál kezdve 20 db "@" jelet!
32. Írjon a képernyő 40. oszlopába a 10. és 20. sorok közé "@" jeleket!
33. Írja ki a keresztnevét a 12. sorba egymás mellé annyiszor, ahányszor kifér!

2.5.1. A ciklusváltozó felhasználása

Az eddigi példáinkban és feladatainkban nem láttuk hasznát annak, hogy a ciklusváltozó számlálja, hogy a ciklusmagot hányszor hajtotta már végre, csupán arra volt szükségünk, hogy összesen 80-szor, vagy 20-szor, vagy valahányszor ismételje meg. A ciklusváltozó azonban a ciklusmagban felhasználható, és ezzel nagyon hasznossá tudja magát tenni. Minden további szöveg helyett nézzünk egy példát:

25. példa: Írassuk ki egymás alá a számokat növekvő sorrendben 1-től 20-ig!

Megoldás: A ciklusmagban most egy szám kiíratása szerepel, de ez a szám folyamatosan változik. A ciklusváltozó szintén folyamatosan változik, az értéke mindig azt mutatja, hogy most éppen hányadszor hajtotta végre a ciklusmag utasításait. A feladatunk az, hogy először (i=1) az 1-es számot, másodszor (i=2) a 2-es számot, ... 20-adszorra (i=20) pedig a 20-as számot kell kiíratnunk. Figyeljük meg, hogy az **i** változónak minden esetben pontosan azonos az értéke a kiíratandó számmal. A megoldás tehát: írassuk ki az **i** változó értékét!

```
CLS
FOR i = 1 TO 20
    PRINT i
NEXT i
```

26. példa Módosítsuk az előző példát úgy, hogy a számokat visszafelé írjuk ki (tehát 20-tól 1-ig)!

Ez még nem a megoldás: Az első ötlet az lehetne, hogy a ciklusfejben a számlálás ne 1-től 20-ig, hanem 20-tól 1-ig menjen. Próbáljuk ki!

```
CLS
FOR i = 20 TO 1
    PRINT i
NEXT i
```

Látható, hogy a programunk nem csinál semmit. Miért? Mert a ciklusváltozó kezdőértéke már kezdetben meghaladja a végértéket, és ez esetben a ciklus egyszer sem hajtódik végre (lásd az előző fejezetben, ahol a ciklus működését ismertettem). Később

látni fogjuk, hogy ezen lehet segíteni, de most még valami más megoldást kell keresni.

Megoldás: Ha a ciklusfejhez nem nyúlhatunk, akkor csak a kiíratásban lehet valamit változtatni. Az **i** értékét valahogyan át kell alakítani. Pontosan a kívánt eredményt adja a **21-i** képlet. (Ha az **i**=1, akkor a **21-i**=10; ha az **i**=2, akkor a **21-i**=9, stb.)

```
CLS
FOR i = 1 TO 20
    PRINT 21-i
NEXT i
```

Feladatok:

34. Írja ki a számokat 1-től 20-ig egymás alá, de a 40. oszlopba!
35. Módosítsa az előző feladatot úgy, hogy a számokat visszafelé írja ki!
36. Írja ki a számokat 50-től 60-ig egymás alá a 40. oszlopba!
37. Módosítsa az előző feladatot úgy, hogy a kiíratás az 5. sorban kezdődjön!

27. példa Írassuk ki a képernyő 20. oszlopába a számokat 1-től 20-ig egymás alá, és minden szám mellé írassuk ki a kétszeresét is!

Megoldás: A feladatnak két értelmes megoldása lehetséges, attól függően, hogy az oszlopokat vagy a sorokat íratjuk-e ki először.

1. megoldás: Először írassuk ki a számokat, majd a második lépésben a kétszeresüket! Egy oszlop megjelenítéséhez egy ciklus kell, két oszlophoz tehát két ciklus egymás után.

```
CLS
FOR i = 1 TO 20
    LOCATE ,20
    PRINT i
NEXT i
LOCATE 1
FOR j = 1 TO 20
    LOCATE ,30
    PRINT j*2
NEXT j
```

Az első ciklus íratja ki a számokat egymás alá a 20. oszlopba. A LOCATE *utasításban a sort nem kell megadni, hiszen a kurzor minden kiírás után automatikusan a következő sorba kerül. A második ciklus végzi a számok kétszeresének kiíratását a 30. oszlopba. Ez előtt viszont szükséges az első sorba való visszapozícionálás.*

A második ciklusban a ***j*** *ciklusváltozót használtuk. Általában célszerű a különböző ciklusokat más-más változóval azonosítani (később látjuk majd, hogy ciklusok egymásba ágyazásánál kötelező is), bár az első ciklus lezárása után az* ***i*** *ciklusváltozó újra szabadon felhasználható lett volna.*

2. megoldás: Egy lépésben írassuk ki a számot, és mellé a kétszeresét.

```
CLS
FOR i = 1 TO 20
    LOCATE ,20
    PRINT i, i*2
NEXT i
```

Látható, hogy ez a megoldás tömörebb, egyszerűbb, de ugyanazt a megoldást adja, mint az előző. A feladatok nagy része többféleképpen is megoldható, és amennyiben a jó eredményt kapjuk, akkor a megoldás jónak tekinthető. Azonban mégis vannak különbségek az egyes programok között végrehajtási időben, bonyolultságban, stb. Egy kész program esetén lehet gondolkodni egy esetleges jobb, hatékonyabb megoldáson.

28. példa: Írassunk ki a képernyőre 20 db "#" karaktert a bal felső sarokból indulva átlósan (tehát az első # jel az 1. sor 1. oszlopában, a második # jel a 2. sor második oszlopában legyen, és így tovább)!

Megoldás: Most a ciklusváltozó értékét a kiíratás helyének meghatározására használjuk fel, hiszen minden alkalommal a sor és az oszlop sorszáma megegyezik a ciklusváltozó értékével.

```
CLS
FOR i = 1 TO 20
    LOCATE i, i
    PRINT "#"
NEXT i
```

29. példa: Változtassuk meg az előző programot úgy, hogy a képernyő jobb felső sarkából induljon a kiíratás!

Megoldás: Most is a ciklusváltozó értékét vesszük alapul, de egy kicsit át kell alakítani. Csak az oszlop megadása rossz, minden más változatlanul hagyható. Vegyük észre, hogy az oszlop a **81-i** képlettel adható meg!

```
CLS
FOR i = 1 TO 20
    LOCATE i, 81-i
    PRINT "#"
NEXT i
```

Feladatok:

38. Módosítsa a fenti példát úgy, hogy a bal alsó sarokból induljon a kiíratás, majd írja meg a jobb alsó sarokból indulva is!
39. Módosítsa a feladatot úgy, hogy írassa ki 1-től 20-ig a számokat a bal felső sarokból indulva átlósan! Majd sorban a többi sarokból is!
40. Írassa ki 20-tól 30-ig a számokat, majd melléjük a számok kétszeresét, valamint a náluk 5-tel kisebb számot (pl. 25, 50, 20)!
41. Írassa ki 10-től 30-ig a számok 3-szorosát!
42. Írassa ki 1-tól 20-ig a számok kétszeresénél 1-gyel kisebb számot! Milyen számsorozatot kapott?
43. Készítsen programot, amely kiírja az 5-ös számra vonatkozó szorzótábla-részletet 20-ig a következő formában:
    1 * 5 = 5
    2 * 5 = 10
    ...
    20 * 5 = 100!

30. példa: Adjuk össze a számokat egytől tízig!

Megoldás: A feladat az 1+2+3+4+5+6+7+8+9+10 összeg kiszámítása. Ehhez szükség lesz egy változóra, ami az összeget tárolja (legyen ez az **osszeg**). A feladatot felbontjuk elemi lépésekre:
osszeg = **1**
osszeg = 1+**2** = 3
osszeg = 1+2+**3** = 3+**3** = 6
osszeg = 1+2+3+**4** = 6+**4** = 10 ...stb.
Egy lépésben csak egy összeadást végzünk el. A ciklusunk 1-től 10-ig fog menni, tehát a ciklusváltozó érté-

ke pontosan azokon a számokon megy végig, amelyeket az összeghez hozzá kell adni.

```
CLS
osszeg = 0
FOR i = 1 TO 10
    osszeg = osszeg + i
NEXT i
PRINT "1-10-ig a számok összege:"; osszeg
```

A program második sorában az ***osszeg*** *változó kezdőértéket kap (ha ez elmarad, a BASIC automatikusan a 0 (nulla) kezdőértéket adja a változónak). A számítást az első látásra kissé nehezen emészthető* `osszeg = osszeg + i` *sor végzi el. A sor jelentése a következő: az* ***osszeg*** *változó előző értékéhez adjuk hozzá az* ***i*** *változó értékét, és ezt az új értéket tároljuk mostantól az* ***osszeg*** *változóban. Természetesen nem feledkezünk el az eredmény kiíratásáról sem (utolsó sor), ez a ciklus után helyezkedik el, hiszen csak a végösszegre vagyunk kíváncsiak, a részeredményekre nem.*

31. példa Adjuk össze a számokat 50-től 150-ig!

Megoldás: A ciklusnak itt 100 összeadást (50-től 150-ig) kell végeznie, de érdemes a ciklust nem 1-től 100-ig, hanem 50-től 150-ig számlálni.

```
CLS
osszeg = 0
FOR i = 50 TO 150
    osszeg = osszeg + i
NEXT i
PRINT "50-150-ig a számok összege:"; osszeg
```

Feladatok:

44. Készítsen programot, amely összeszorozza 1-től 10-ig a számokat! Vigyázzon a kezdőértékre!
45. Adja össze az első N természetes számot (az N értékét kérje be)!
46. Készítsen programot, amelyben kérjen be egy tetszőleges számot, és ettől a számtól kezdve adjon össze tíz számot!

2.5.2. A FOR–NEXT–STEP szerkezet

32. példa Írassuk ki a képernyőre 10-től 100-ig a számokat 10-esével egymás alá!

Megoldás:

```
CLS
FOR i = 1 TO 10
    PRINT i*10
NEXT i
```

33. példa Írassuk ki a számokat 1-től 20-ig egymás alá, de visszafelé!

Megoldás: A példát már megoldottuk az előző fejezetben:

```
CLS
FOR i = 1 TO 20
    PRINT 21-i
NEXT i
```

A fenti két példát az előzőek alapján meg tudtuk oldani, de a megoldás nem kényelmes, hiszen a kiíratás során egyszerűbb vagy bonyolultabb számításokat kellett végeznünk. Mennyivel könnyebb lenne, ha a ciklusunkat tudnánk úgy szervezni, hogy

Ciklus i = 10-től 100-ig 10-esével
  vagy
Ciklus i = 20-tól 1-ig -1-esével
  stb.

Természetesen megtehetjük, és a Quick Basic lehetőséget is ad rá. Az alkalmazott szerkezet:

FOR i = 10 **TO** 100 **STEP** 10
  vagy
FOR i = 20 **TO** 1 **STEP** -1
  illetve általánosan:
FOR ciklusváltozó = kezdőérték **TO** végérték **STEP** lépésköz

Ennek a két ciklusfejnek pontosan az a hatása, mint amit fentebb megfogalmaztunk. Persze most újra át kell gondolni mindazt, ami a ciklusok működésével kapcsolatban az előző fejezetben elhangzott. A pontos működés a következő: először a **ciklusváltozó** felveszi a **kezdőértéket**. A NEXT elérésekor a vezérlés visszakerül a FOR sorra. Itt a **ciklusváltozó** értéke **lépésközzel** változik (az előző két példát figyelembe véve 10-zel nő, illetve 1-gyel csökken). Amennyiben már meg-

haladta a végértéket, akkor a ciklus véget ér, ha még nem, akkor a ciklusmag hajtódik végre újra.

Vigyázni kell, hogy pozitív lépésköz esetén a kezdőérték ne haladja meg a végértéket, negatív lépésköz esetén viszont a kezdőértéknek kell nagyobbnak lennie!

34- 35. példa Írjuk át a fenti két példát a FOR–NEXT–STEP szerkezet alkalmazásával!

34. példa megoldása:

```
CLS
FOR i = 10 TO 100 STEP 10
    PRINT i
NEXT i
```

35. példa megoldása:

```
CLS
FOR i = 20 TO 1 STEP -1
    PRINT i
NEXT i
```

Feladatok:

47. Készítsen programot, amely az első sorban minden második oszlopba kiír egy „@" karaktert!
48. Készítsen programot, amely a képernyőre minden harmadik sorba kiírja a „Hello, te zseni!" szöveget!
49. Írassa ki a 100, 200, 300, stb. számokat egymás alá 2000-ig a 60. oszlopba!
50. Írassa ki a 25, 50, 75 stb. számokat egymás alá a 20. oszlopba (500-ig), és minden szám mellett írassa ki a kétszeresénél 1-gyel nagyobb számot is!
51. Írassa ki 66-tól csökkenő sorrendben az összes pozitív, 3-mal osztható számot!
52. Írassa ki 100-tól 50-ig visszafelé az összes 5-tel osztható szám kétszeresét!

2.5.3. Egymásba ágyazott ciklusok

36. példa Írjuk tele a képernyő első 20 sorát „#" karakterekkel!

Megoldás: Egy sor teleírását már megoldottuk egy ciklus segítségével. Most ezt kell megismételni 20-szor, akkor tehát ez is egy ciklus.

```
CLS
FOR i = 1 TO 20
    LOCATE i, 1
    FOR j = 1 TO 80
        PRINT "#";
    NEXT j
NEXT i
```

*Az **j** változóval azonosított ciklust (röviden: j ciklus) belső ciklusnak, az i ciklust külső ciklusnak nevezzük. Az i cikluson belül a* LOCATE *utasítás a kurzort az i. sor elejére helyezi, majd a már ismert módon a belső ciklus segítségével elvégezzük a sor teleírását. Ezt a tevékenységet ismételjük 20-szor a külső ciklus segítségével. (Az **i** változó azonosítja a sorokat, a **j** változó az oszlopokat.)*

Nagyon fontos szabály az egymásba ágyazott ciklusoknál, hogy előbb a belső ciklust kell lezárni a NEXT utasítással, és csak azután a külsőt! (Ez a példán is jól megfigyelhető.)

37. példa Töltsük fel a képernyőn a 20. és 60. oszlopok, illetve 5. és 15. sorok által határolt téglalapot „#" karakterekkel!

Megoldás: A feladat nagyon hasonlít az előzőhöz, de itt a pozícionálást (LOCATE) egy kicsit másképp oldjuk meg...

```
CLS
FOR i = 5 TO 15
    FOR j = 20 TO 60
        LOCATE i, j
        PRINT "#"
    NEXT j
NEXT i
```

A feladat megoldásában ezúttal minden egyes # karakter helyét megadtuk, hiszen a belső ciklusba került a LOCATE *utasítás, míg az előző példánál csak minden sor kezdetét adtuk meg, és a soron belül a ; (pontosvessző) gondoskodott a kurzor pozícionálásáról. Az előző feladat megoldása talán jobb, de ez a megoldás univerzálisabb.*

Feladatok:

53. Írja át a fenti programot úgy, hogy a kiíratás ne sorfolytonosan, hanem oszlopfolytonosan történjen (tehát először az 1. oszlopot, azután a másodikat, stb. írassa ki)!

54. Készítsen programot, amely a képernyő minden második sorát teleírja "@" karakterekkel!
55. Módosítsa az előző feladatot úgy, hogy az egyes sorokban csak minden harmadik "@" kerüljön kiíratásra!

38. példa Készítsünk a képernyő közepére egy működő digitális órát, ami 5 percig jár!

Megoldás: Ez a program nagyon látványos, és mindamellett elég egyszerű. Lássuk csak, hogyan is kell működnie: végig kell számolnunk 0-tól 59-ig (másodpercek), ez egy perc. Az órának 5 percig kell járnia, tehát ezt a számlálást 5-ször kell végrehajtani. A belső ciklus tehát a másodperceket számlálja, a külső a perceket. Az időket természetesen másodpercenként ki kell írni, mindig a képernyő ugyanazon helyére. Azt, hogy a másodpercek tényleg egy másodpercenként kövessék egymást, arról a SLEEP(mp) utasítás segítségével tudunk gondoskodni. Ehhez a sorhoz érve a program annyi másodpercet várakozik, amennyit a zárójelek közt megadunk.

```
CLS
COLOR 12, 7
FOR perc = 0 TO 4
    FOR mp = 0 TO 59
        LOCATE 12, 36
        PRINT perc; ":"; mp
        SLEEP(1)
    NEXT mp
NEXT perc
```

Figyeljük meg, hogy a ciklusváltozók ezúttal a ***perc*** *és* ***mp*** *neveket viselik, a semmitmondó* ***i*** *és* ***j*** *helyett.*

39. példa Módosítsuk az előző programot úgy, hogy órákat is számláljon!

Megoldás: Ha két ciklus egymásba ágyazható, akkor három is. Szükségünk van tehát egy harmadik, legkülső ciklusra, amely az órákat számlálja. A perc ciklust módosítanunk kell, hogy végigmenjen egy teljes órán (0-tól 59-ig). Sajnos a program futását nem tudjuk kivárni, ezért kivesszük a SLEEP utasítást, de így a program túl

gyors lesz. A várakoztatást tehát másképp kell megoldani...

```
CLS
COLOR 12, 7
FOR ora = 0 TO 23
    FOR perc = 0 TO 59
        FOR mp = 0 TO 59
            LOCATE 12, 34
            PRINT ora; ":"; perc; ":"; mp
            FOR i = 1 TO 500 : NEXT i
        NEXT mp
    NEXT perc
NEXT ora
```

A program most már végighalad egy egész napon (24 órán, pontosabban 0:0:0-tól 23:59:59-ig). A várakoztatást egy üres ciklussal oldhatjuk meg (üres ciklusnak nevezzük azt a ciklust, amelyben a ciklusmag hiányzik). Az üres ciklus végrehajtása is valamennyi – bár elég csekély – időbe kerül, és ha elég sokszor hajtjuk végre, akkor az már érezhető időtartamot jelent, ezért várakoztatásra felhasználható. Az, hogy a ciklus hányszor fusson le, géptípusonként eltérő, ki kell kísérletezni azt az értéket, amelyiket a legjobbnak ítéljük.

Feladatok:

56. Készítsen számjegyes számlálót a képernyőre (amilyen a villanyórákban vagy kilométerórákban is van)! A számláló először két számjegyes legyen (tehát 00-tól 99-ig számlál), majd bővítse fokozatosan négy számjegyesig!
57. Készítsen szorzótáblát a képernyőre (1-től 10-ig)!

2.6. Elágazások

40. példa Készítsünk programot, amelyben bekérünk két számot és kiíratjuk a hányadosukat!

Megoldás: A programhoz nagyon hasonlót már a változók kapcsán készítettünk, csak ott a két szám összegét kellett számítani...

```
CLS
INPUT "Kérem az első számot: ";, a
INPUT "Kérem a második számot: ";, b
PRINT "A két szám hányadosa: "; a/b
```

A program futtatásánál próbáljunk ki különböző értékeket! Ezt hívjuk a program tesztelésének. Próbáljuk ki a 12 és 0 (nulla) számokkal is! A program futási hibával leállt, hiszen nullával akartunk osztani.

Mi lehetne a fenti problémára a megoldás? Az osztás előtt meg kellene vizsgálni, hogy az elvégezhető-e. Próbáljuk meg megfogalmazni a vizsgálatot: Ha a második szám nem nulla, akkor osztunk, különben nem. Ezt a szerkezetet **elágazásnak** nevezzük, legegyszerűbb formája a következő:

HA feltétel AKKOR utasítás(ok)

Vagyis, ha a feltétel teljesül (igaz), akkor és csak akkor hajtsuk végre az utasítás(oka)t. A Quick Basic-ben ez a szerkezet a következőképpen néz ki (egyelőre feltételezve, hogy az AKKOR után csak egyetlen utasítást kell végrehajtani):

IF feltétel **THEN** utasítás

A feltételben valamilyen logikai kifejezésnek kell szerepelnie, olyannak, amelynek az eredménye IGAZ vagy HAMIS (az eldöntendő kérdésre hasonlít a feltétel). A feltételben használható relációk a következők: = (egyenlő), <> (nem egyenlő), < (kisebb), > (nagyobb), <= (kisebb vagy egyenlő), és >= (nagyobb vagy egyenlő). Összetettebb feltételek megfogalmazására használhatjuk a NEM (NOT), ÉS (AND), VAGY (OR) és XOR (kizáró vagy) logikai műveleteket.

41. példa Írjuk át az előző példaprogramot úgy, hogy az osztás csak akkor hajtódjon végre, ha a második szám nem nulla!

Megoldás:

```
CLS
INPUT "Kérem az első számot: ";, a
INPUT "Kérem a második számot: ";, b
IF b<>0 THEN PRINT "A két szám hányadosa: "; a/b
```

Ez a kis program már nem fog „kiakadni", ha a második szám nulla, de nem is csinál semmit! Elfelejtettük kezelni azt az esetet, amikor a második szám nulla! Mit is kellene ilyenkor csinálni? Osztani semmiképpen, tehát jelenítsünk meg egy szöveget, amelyben közöljük, hogy ezt nem végezzük el. Egy második vizsgálattal, vagyis még egy IF–THEN szerkezettel ez könnyen megoldható. Ezt az új vizsgálatot az előző elé tesszük be, mert

általános szabály, hogy ***először a rossz eseteket dolgozzuk fel,*** *tehát amelyek a program további útjára nézve zsákutcának bizonyulhatnak, nem a program tényleges feladatával foglalkoznak.*

```
CLS
INPUT "Kérem az első számot: ";, a
INPUT "Kérem a második számot: ";, b
IF b=0 THEN PRINT "Nullával való osztás értelmet-
len!!!"
IF b<>0 THEN PRINT "A két szám hányadosa: "; a/b
```

A fenti példából kitűnik, hogy a program írása során az elágazásokat úgy kell kezelni, hogy **minden lehetséges esetet** – minden „utat", ami felé elágaztathatunk –, **figyelembe kell venni.**

A fenti szerkezetet a következőképpen fogalmaztuk meg:

Ha b=0 akkor Ki: "Nullával való osztás értelmetlen!!!"
Ha b≠0 akkor Ki: a/b

Ebben a második vizsgálat (Ha b≠0) felesleges, a szerkezet átfogalmazható a következőképpen:

Ha b=0 akkor Ki: "Nullával való osztás értelmetlen!!!"
különben Ki: a/b

Ennek a szerkezetnek (HA–AKKOR–KÜLÖNBEN) a Quick Basic megfelelője a következő (még mindig feltételezve, hogy mind az AKKOR, mind a KÜLÖNBEN után csak egy-egy utasítást kell végrehajtani):

IF feltétel **THEN** utasítás **ELSE** utasítás

Fontos, hogy az egész szerkezet egyetlen programsorban legyen!

42. példa Írjuk át a fenti programot az ELSE alkalmazásával!

Megoldás:

```
CLS
INPUT "Kérem az első számot: ", a
INPUT "Kérem a második számot: ", b
IF b=0 THEN PRINT "Nullával..." ELSE PRINT
"Hányados:"; a/b
```

A kiírásokat csak azért egyszerűsítettem, hogy itt is kiférjen egy sorba a teljes elágazás.

Az eddigiek során feltételeztük, hogy a THEN és az ELSE után egyaránt csak egy-egy utasítást kell végrehajtani. Ha már többet, akkor a szerkezetet egy kicsit át kell fogalmaznunk a következőképpen:

IF feltétel **THEN**
    utasítás1
    utasítás2
    ...
END IF

vagy az ELSE használatával:

IF feltétel **THEN**
    utasítás1
    utasítás2
    ...
ELSE
    utasítás1
    utasítás2
    ...
END IF

A THEN-ágon lévő utasítások akkor hajtódnak végre, ha a feltétel igaz, az ELSE-ág utasításai pedig akkor, amikor a feltétel hamis. Itt is érdemes a programlistát strukturálva gépelni: lássuk hol kezdődik és hol végződik a szerkezet. Amennyiben az IF–THEN–ENDIF vagy IF–THEN–ELSE–ENDIF szerkezeteket alkalmazzuk, akkor a THEN és az ELSE után (vele azonos sorba) már nem írhatunk utasítást!

43. példa Módosítsuk előző programunkat úgy, hogy az osztás eredménye zölddel, a hibaüzenet pedig villogó vörössel jelenjen meg!

Megoldás: Alkalmazzuk az IF–THEN–ELSE–ENDIF szerkezetet

```
CLS
INPUT "Kérem az első számot: ", a
INPUT "Kérem a második számot: ", b
IF b=0 THEN
    COLOR 12+16
    PRINT "Nullával való osztás értelmetlen!!!"
ELSE
    COLOR 10
    PRINT "A két szám hányadosa: "; a/b
END IF
```

44. példa Kérjünk be egy számot, és döntsük el róla, hogy 1 és 10 között van-e!

Megoldás: Már nem elég egy egyszerű feltételt megfogalmazni, összetett feltételre van szükség

```
CLS
INPUT "Mondj egy számot!", a
IF a>1 AND a<10 THEN
    PRINT "A szám 1 és 10 között van"
ELSE
    PRINT "A szám nem egy és 10 között van!"
END IF
```

Az `IF-THEN-ELSE-ENDIF` *szerkezet helyett természetesen alkalmazhattuk volna az egyszerűbb* `IF-THEN-ELSE` *szerkezetet is, ebben az esetben az egész szerkezet egy sorba kell hogy kerüljön.*

Feladatok:

58. Kérjen be egy számot, és írja ki, hogy a szám pozitív-e (pozitív, nem pozitív)!
59. Kérjen be egy számot, és döntse el, hogy kisebb-e 10-nél (kisebb, nem kisebb)!
60. Kérjen be egy számot, és írja ki, hogy kétjegyű-e!
61. Kérjen be egy számot, és döntse el, hogy lehet-e ez egy szín kódja (0-15)!
62. Kérjen be két számot, és írassa ki a nagyobbikat!
63. Kérjen be két számot, és írja ki, hogy egyenlők-e!

2.6.1. Elágazások három- vagy többfelé

45. példa Kérjünk be két számot, és írassuk ki, hogy melyik a nagyobb!

Megoldás: Három lehetséges eredmény van: 1. az első kisebb, 2. az első nagyobb, 3. egyenlőek. Tehát háromfelé kell elágaztatni. Eddigi ismereteinkkel ez természetesen lehetséges, hiszen az IF kezdetű sorokat akárhányszor ismételhetjük.

```
CLS
INPUT "Az első szám:", a
INPUT "A második szám:", b
IF a<b THEN PRINT "Az első szám kisebb"
IF a>b THEN PRINT "Az első szám nagyobb"
IF a=b THEN PRINT "A két szám egyenlő"
```

A megoldás jó, de van egy komoly "szépséghibája": ha már az első feltétel (a<b) igaz, akkor a másik kettőről eleve tudjuk, hogy az hamis, felesleges megvizsgálni. A hiba kiküszöbölhető az ELSE alkalmazásával. Persze az ELSE-ágon egy újabb vizsgálatot kell végezni, erre szolgál az ELSEIF utasítás. Használata:

```
IF feltétel1 THEN
    utasítás(ok)
ELSEIF feltétel2 THEN
    utasítás(ok)
ELSE
    utasítás(ok)
END IF
```

Ez a szerkezet háromfelé ágaztatást valósít meg. Ha többfelé kellene elágaztatni a programot, akkor több ELSEIF beiktatásával a feladat megoldható.

46. példa Írjuk át a fenti példát az ELSEIF alkalmazásával!

Megoldás:

```
CLS
INPUT "Az első szám:", a
INPUT "A második szám:", b
IF a<b THEN
    PRINT "Az első szám kisebb"
ELSEIF a>b THEN
    PRINT "Az első szám nagyobb"
ELSE PRINT "A két szám egyenlő"
END IF
```

A megoldás nagy hátránya, hogy a programot bonyolulttá, nehezen áttekinthetővé teszi. Ezért nem is ajánlom az alkalmazását (legalábbis kezdetben), helyette az előző példa háromszoros IF–THEN-jét érdemes használni, amely a program hatékonysága szempontjából ugyan rosszabb (hiszen felesleges vizsgálatokat kell végezni), de egyszerűbb és áttekinthetőbb.

Amennyiben egy változó értékét kell vizsgálnunk, akkor létezik egy egyszerűbb eszköz: a SELECT CASE. Lássuk hogyan is működik:

SELECT CASE vizsgálandó_kifejezés
CASE érték(ek)
utasítás(ok)
CASE érték(ek)
utasítás(ok)
...
CASE ELSE
utasítás(ok)
END SELECT

A SELECT CASE alkalmazására lássunk egy példát:

47. példa Kérjünk be egy osztályzatot (pl. matekjegyet), és a kapott eredményt értékeljük szövegesen!

Megoldás: A feladatot először IF–THEN szerkezetekkel, majd a SELECT CASE szerkezettel adjuk meg.

```
CLS
INPUT "Hányasod volt matekból év végén? ", jegy
IF jegy < 1 THEN PRINT "Ennyire nem lehettél
rossz!"
IF jegy = 1 THEN PRINT "Csak nem megbuktál?"
IF jegy>=2 AND jegy<=4 THEN PRINT "Matekból bár-
milyen jegy jó..."
IF jegy = 5 THEN PRINT "Te egy kis zseni vagy!"
IF jegy > 5 THEN PRINT "Érdekesen osztályoz a ta-
národ..."
```

2. megoldás:

```
CLS
INPUT "Hányasod volt matekból év végén? ", jegy
SELECT CASE jegy
    CASE IS < 1
        PRINT "Ennyire nem lehettél rossz!"
    CASE 1
        PRINT "Csak nem megbuktál?"
    CASE 2 TO 4
        PRINT "Matekból bármilyen jegy jó..."
    CASE 5
        PRINT "Te egy kis zseni vagy!"
    CASE ELSE
        PRINT "Érdekesen osztályoz a tanárod..."
END SELECT
```

A két megoldás összehasonlításával meg lehet érteni a SELECT CASE *szerkezet működését...*

Feladatok: (használja a SELECT CASE szerkezetet!)

64. Kérjen be egy számot, és írassa ki, hogy a szám pozitív, negatív vagy nulla!
65. Kérjen be egy számot, és döntse el, hogy lehet-e életkor! Ha igen, akkor adja meg a korcsoportot, amibe tartozik (gyerek–fiatal–felnőtt–idős), egyébként adjon hibaüzenetet!

2.7. Feltételes ugrások

Térjünk vissza az osztásos feladatunkhoz! Emlékeztetőül (bár egy kicsit másképpen felírva):

```
CLS
INPUT "Kérem az első számot: ", a
INPUT "Kérem a második számot: ", b
IF b=0 THEN
    PRINT "Nullával való osztás értelmetlen!!!"
ELSE
    PRINT "A két szám hányadosa: "; a/b
END IF
```

A megoldás egy kicsit suta, hiszen amennyiben a második szám nulla, akkor csak egy hibaüzenetet adunk, hogy az osztás nem elvégezhető. Jó lenne újra megkérdezni a második számot. Csakhogy mi van akkor, ha ez is nulla lesz? Nem tudjuk, hogy hányszor kell újra és újra megkérdezni, mikor kegyeskedik végre a felhasználó valamilyen nullától különböző értéket gépelni. Azt kellene tehát elérni, hogy a vezérlés visszakerüljön arra a sorra, melyiktől kezdve ismételni akarunk! Erre szolgál a GOTO (ugorj) utasítás. Használata:

GOTO cimke

(ahol a cimke azonosítja azt a sort, amelyikre ugrani akarunk.)

48. példa Módosítsuk a fenti feladatot a leírtak szerint!

Megoldás:

```
CLS
INPUT "Kérem az első számot: ", a
cimke1:
INPUT "Kérem a második számot: ", b
IF b=0 THEN
    PRINT "Nullával való osztás értelmetlen!!!"
    GOTO cimke1
END IF
PRINT "A két szám hányadosa: "; a/b
```

Mit is csinál most ez a programocska? Bekéri az első számot, majd bekéri a másodikat is. Ez utóbbit megvizsgálja, hogy nem nulla-e, és ha igen akkor hibaüzenetet ad, és visszaugrik a második szám bekéréséhez (azt a sor azonosítottuk a ***cimke1*** *nevű címkével). Ha a második szám nem nulla, akkor a program végre túljut az elágazáson, és elvégezheti valódi feladatát, az osztást. Fontos, hogy a cimke után mindig kettőspontot (:) kell tenni, a Quick Basic innen ismer rá. (Régebbi Basic változatoknál – főleg régi kis számítógépeken (Spectrum HT-1080Z , Commodore, stb.) – a sorokat számozni kellett. A GOTO utasítással a sor sorszámára kellett ugrani. Erre a Quick Basic is lehetőséget ad, de a sorszámozás alkalmazása eléggé macerás, nem javaslom a használatát.)*

A cimke egyébként bármilyen azonosítót kaphat (eleje, vege, pityuka, stb.), de mindazok a szabályok vonatkoznak rá, amelyek a változók azonosítóira is vonatkoznak. Figyeljük meg, hogy az a szerkezet, amelyet a GOTO segítségével valósítottunk meg, az valójában egy ciklus! A ciklusmagban a második szám bekérése van, a ciklus pedig addig hajtódik végre, amíg csak egy feltétel (b=0) nem válik hamissá. Ezek azonban nem számlálós ciklusok – nem tudom előre, hogy a ciklusmagot hányszor kell végrehajtani –, hanem ún. tesztelős ciklusok. Ezek szervezésére is lehetőséget ad a Quick Basic programozási nyelv; a szerkezeteket részletesen tárgyalja a következő fejezet. A feltételes ugrás azonban egy kicsit egyszerűbb, természetesebb, kezdőknek mindenképpen ajánlatos a megértése, mielőtt belemerülnénk a tesztelős ciklusokkal való programozásba.

Fontos megjegyzés: **a GOTO utasítás alkalmazása csak feltételhez kötötten – tehát egy elágazás THEN vagy ELSE ágán – engedhető meg.** Bár a Basic a használatát nem korlátozza, de a feltétel nélküli ugrások a programot áttekinthetetlenné teszik.

A fenti program alkalmas lehet bármilyen adatbekérés tesztelésére (értékhatárok ellenőrzésére).

A programjaink eddig úgy működtek, hogy a futás végeztével szépen maguktól leálltak. De mi van akkor, ha szeretnénk esetleg többször egymás után használni? Le kell futtatni többször is, ez azonban nem elég kényelmes és elegáns.

49. példa Kérjünk be két számot, írassuk ki a különbségüket, majd kérdezzük meg a felhasználót, hogy akar-e még számolni – ha igen, akkor kezdjük a programot elölről.

Megoldás: Most a program végén lesz szükség egy feltételes ugrásra...

```
eleje:
CLS
INPUT "Kérem az első számot: ", a
INPUT "Kérem a második számot: ", b
PRINT "A két szám különbsége:"; a-b
PRINT "Akarsz még számolni? (i/n) ";
c$ = INPUT$(1)
IF c$="i" OR c$="I" THEN GOTO eleje
```

A programban szerepel egy eleddig még ismeretlen Basic függvény: az `INPUT$(n)`. *Működése hasonlít az* `INPUT` *utasításhoz, ugyanúgy billentyűzetről való adatbekérésre alkalmas. Azonban amíg az* `INPUT` *kérdésére begépelt választ ENTER-rel le kell zárni, addig az* `INPUT$` *függvény annyi billentyű lenyomását várja csak meg, amennyit a zárójelek között megadtunk. A példában szereplő INPUT$(1) tehát egyetlen billentyű lenyomására vár. A lenyomott billentyű a* ***c$*** *változóba kerül (egyetlen karakter is szöveg, ezért kell a $). A következő sorban nézzük meg, hogy a lenyomott billentyű az i vagy I volt-e. Erre azért van szükség – mármint arra, hogy mind a kis i, mind a nagy I betűt teszteljük –, mert a számítógép különbséget tesz a kettő között. Ha csak az egyiket – teszem azt a kis i-t – vizsgálnánk, akkor a másik – példánkban a nagy I – lenyomására a program befejeződne. Fontos, hogy a kérdést úgy tegyük fel, hogy a rá adandó válasz egyértelmű legyen a felhasználó számára. Példánkban a kiíratásban szerepeltettük, hogy az i vagy n billentyűre várunk. Bármilyen egyéb billentyű hatására a program befejeződik. Természetesen módosíthatnánk a működését úgy, hogy csak az i vagy n karaktereket fogadja el a program – ezt egy további feltételes ugrás valósítaná meg. Ekkor a program a következőképpen nézne ki:*

```
eleje:
CLS
INPUT "Kérem az első számot: ", a
INPUT "Kérem a második számot: ", b
PRINT "A két szám különbsége:"; a-b
PRINT "Akarsz még számolni? (i/n) ";
ujrabill:
c$ = INPUT$(1)
IF c$<>"i" AND c$<>"I" AND c$<>"n" AND c$<>"N"
THEN GOTO ujrabill
IF c$="i" OR c$="I" THEN GOTO eleje
```

Látható, hogy a programba még egy cimke bekerült, ennek azonban más a neve, mint az előzőnek. Általános szabály – ugyanúgy, ahogy a változóknál is –, hogy két azonos nevű cimke nem szerepelhet a programban. Ez a programocska már csak az i, I, n, N karaktereket fogadja el a billentyűzetről, és addig várja újra meg újra a leütést, amíg csak meg nem kapja valamelyiket. Tovább finomíthatjuk ezt a megoldást úgy, hogy rossz billentyű leütésekor a programocska adjon hangjelzést (ezt a **BEEP** *utasítással tehetjük meg):*

```
eleje:
CLS
INPUT "Kérem az első számot: ", a
INPUT "Kérem a második számot: ", b
PRINT "A két szám különbsége:"; a-b
PRINT "Akarsz még számolni? (i/n) ";
ujrabill:
c$ = INPUT$(1)
IF c$<>"i"  c$<>"I" AND c$<>"n" AND c$<>"N" THEN
   BEEP
   GOTO ujrabill
END IF
IF c$=„i" OR c$=„I" THEN GOTO eleje
```

Feladatok:

66. Kérjen be egy 1 és 10 közötti számot (tesztelje!), és írassa ki a szám négyzetét (önmagával való szorzatát)!
67. Módosítsa az előző feladatot úgy, hogy a végén kérdezze meg, hogy vége legyen-e már!
68. Kérjen be egy negatív számot (tesztelje!), majd írassa ki a szám (-5)-szörösét!
69. Módosítsa az előző feladatot úgy, hogy a végén kérdezze meg, hogy vége legyen-e már!

70. Kérjen be egy szöveget és egy számot, ami egy szín kódja, és a bekért számot jelenítse meg a bekért színnel (természetesen tesztelje, hogy létező színkódot kap-e)!
71. Módosítsa az előző feladatot úgy, hogy a végén kérdezze meg, hogy vége legyen-e már!
72. Kérjen be egy számot 20 és 30 között, majd egy másikat 100 és 200 között (mindkettőt tesztelje!)! A nagyobbikból vonja ki a kisebbet, és közölje az eredményt!
73. Módosítsa az előző feladatot úgy, hogy a végén kérdezze meg, hogy vége legyen-e már!

2.8. Tesztelős ciklusok

Az előző fejezetben ismertetett feltételes ugrások valójában ciklusokat valósítottak meg. Volt szó arról is, hogy ezek a ciklusok nem lehettek számlálós ciklusok, hiszen nem tudtuk megmondani, hogy hányszor futtassák le a ciklusmagban szereplő utasításokat. A ciklusok minden esetben addig futottak, amíg csak valamilyen feltétel igaz (vagy hamis) nem lett. A feltételek teljesülését természetesen minden egyes végrehajtás előtt vagy után vizsgálni (tesztelni) kell. Ha a ciklusmag előtt tesztelünk, akkor elöltesztelős, ha a ciklusmag után, akkor hátultesztelős ciklusokról beszélünk.

Akár elöl, akár hátul tesztelünk, a ciklus kétféleképpen érhet véget: ha a vizsgált feltétel igaz, illetve ha a vizsgált feltétel nem igaz. Mindkét megoldás jó lehet, mindig azt célszerű alkalmazni, amelyiknél egyszerűbb a feltételt megfogalmazni, bár sokszor a megszokás dönt valamelyik javára.

Ezzel tehát öt különféle ciklust különböztethetünk meg. A számlálós ciklust (FOR–NEXT) akkor alkalmazzuk, ha előre tudjuk, hogy a ciklusmagot hányszor kell végrehajtani. Minden más esetben valamelyik tesztelős ciklust kell alkalmazni. A kettő között az az alapvető különbség, hogy az elöltesztelős ciklus magja lehet hogy egyszer sem fog lefutni (ha a vizsgált feltétel alapján már eleve tudjuk, hogy szükségtelen a ciklusmagot futtatni), míg a hátultesztelős ciklus magja egyszer biztosan lefut. Ez fogja eldönteni, hogy az adott feladathoz melyiket alkalmazzuk.

Tesztelős ciklusokat a DO–LOOP kulcsszavakkal szervezhetünk. A DO szócska jelzi a ciklus kezdetét, a LOOP zárja le (hasonlóan a FOR–NEXT pároshoz). A teszteléshez az UNTIL illetve a WHILE kulcsszavak használatosak: az UNTIL után írt feltétel hatására a ciklus addig

fut, amíg a feltétel igazzá nem válik, a WHILE esetében pedig addig, amíg a feltétel hamissá nem válik (tehát amíg igaz). Mind az UNTIL-t, mind a WHILE-t úgy kell alkalmazni, hogy a DO vagy LOOP után írjuk őket – attól függően, hogy elöl vagy hátultesztelős ciklusokat szeretnéke definiálni. Amennyiben a feltétel olyanná vált, hogy az a ciklus végét jelenti, akkor a vezérlés a ciklus utáni első programsoron fog folytatódni.

A tesztelős ciklusok is rendelkezhetnek ciklusváltozóval. Ebben az esetben a ciklusváltózó kezdőértékéről, illetve a ciklusban való megváltoztatásáról nekünk kell gondoskodnunk.

Még egy megjegyzés: a tesztelős ciklusok közül bármelyik bármelyikre átírható a feltétel megváltoztatásával. A számlálós ciklus is megadható tesztelős ciklusok segítségével.

2.8.1. Elöltesztelős ciklusok

Elöltesztelős ciklusokat a bevezetőben leírtak alapján a következőképpen szervezhetünk:

1.

Ciklus amíg feltétel HAMIS
    ciklusmag
Ciklus vége

DO UNTIL feltétel
    utasítás(ok) (ciklusmag)
LOOP

A ciklusmagban szereplő utasítások addig hajtódnak végre, amíg a feltétel IGAZZÁ nem válik.

2.

Ciklus amíg feltétel IGAZ
    ciklusmag
Ciklus vége

DO WHILE feltétel
    utasítás(ok) (ciklusmag)
LOOP

A ciklusmagban szereplő utasítások addig hajtódnak végre, amíg a feltétel HAMISSÁ nem válik.

50. példa Írassuk ki a számokat 20-tól 40-ig a képernyőre egymás alá!

Megoldás: A programot már megírtuk számlálós ciklus segítségével. Most tesztelős ciklussal valósítjuk meg, mégpedig mindkét (UNTIL ill. WHILE) módszerrel.

UNTIL alkalmazásával:

```
CLS
i = 20
DO UNTIL i > 40
    PRINT i
    i = i+1
LOOP
```

WHILE alkalmazásával:

```
CLS
i = 20
DO WHILE i <= 40
    PRINT i
    i = i+1
LOOP
```

A fenti két program tökéletesen ugyanazt hajtja végre, csak az alkalmazott megoldás eltérő. Általában az elöltesztelős ciklusoknál a `DO WHILE-LOOP` *szerkezetet szokták csak alkalmazni, de ki-ki kiválaszthatja, hogy melyik a szimpatikusabb. Mindenképpen érdemes az egyiket kiválasztani a kettő közül, és következetesen azt használni. Megfigyelhető, hogy a két megoldás csupán a feltételben tér el egymástól, valamint hogy a második feltétel az első tagadásával egyenlő és viszont. Ez általánosan igaz a kétféle elöltesztelős ciklusra.*

2.8.2. Hátultesztelős ciklusok

Hátultesztelős ciklusokat a bevezetőben leírtak alapján a következőképpen szervezhetünk:

1.

Ciklus
    ciklusmag
amíg feltétel HAMIS
Ciklus vége

DO
    utasítás(ok) (ciklusmag)
LOOP UNTIL feltétel

A ciklusmagban szereplő utasítások addig hajtódnak végre, amíg a feltétel IGAZZÁ nem válik.

2.

Ciklus
    ciklusmag
amíg feltétel IGAZ
Ciklus vége

DO
    utasítás(ok) (ciklusmag)
LOOP WHILE feltétel

A ciklusmagban szereplő utasítások addig hajtódnak végre, amíg a feltétel HAMISSÁ nem válik.

51. példa készítsük el az elöltesztelős ciklusoknál elkészített feladatot: írassuk ki a számokat 20-tól 40-ig a képernyőre egymás alá!

Megoldás:

UNTIL alkalmazásával:

```
CLS
i = 20
DO
    PRINT i
    i = i+1
LOOP UNTIL i > 40
```

WHILE alkalmazásával:

```
CLS
i = 20
DO
    PRINT i
    i = i+1
LOOP WHILE i <= 40
```

A fenti két program tökéletesen ugyanazt hajtja végre, csak az alkalmazott megoldás eltérő. Általában az hátultesztelős ciklusoknál a `DO-LOOP UNTIL` *szerkezetet szokták csak alkalmazni, de ki-ki kiválaszthatja, hogy melyik a szimpatikusabb. Mindenképpen érdemes az egyiket kiválasztani a kettő közül, és következetesen azt használni. Megfigyelhető, hogy a két megoldás csupán a feltételben tér el egymástól, valamint hogy a második feltétel az első tagadásával egyenlő és viszont. Ez általánosan igaz a kétféle hátultesztelős ciklusra.*

2.8.3. feladatok tesztelős ciklusokra

Tesztelős ciklusoknak gyakori alkalmazási területe a bekért adatok tesztelése (értékhatár-helyesség ellenőrzése).

52. példa Kérjünk be egy számot 1 és 10 között, és írassuk ki a képernyő közepére!

Megoldás: A tesztelős ciklust az adatbekérésre alkalmazzuk, hiszen addig kell újra meg újra bekérni az adatot, amíg csak helyes értéket nem kapunk. Elöl- vagy hátultesztelős ciklust alkalmazzunk? A kérdést az dönti el, hogy a ciklusmagot végre kell-e hajtani legalább egyszer. A válasz IGEN, tehát célszerűbb hátultesztelős ciklus alkalmazása. (Természetesen elöltesztelős ciklussal is megoldható a feladat.)

```
CLS
DO
    INPUT "Mondj egy számot 1 és 10 között!", a
    IF a<1 THEN PRINT "Ez túl kicsi!!"
    IF a>10 THEN PRINT "Ez túl nagy!!"
LOOP UNTIL a>=1 and a<=10
CLS
LOCATE 12, 35
PRINT "A számod:"; a
```

A bekérést megvalósító DO–LOOP UNTIL ciklus akkor ér véget, amikor a feltétel (a>=1 AND a<=10) igazzá nem válik, vagyis amikor 1 és 10 közti számot kapunk. A cikluson belül oldjuk meg az esetleges hibaüzenetek kiíratását is (bár ezzel felesleges vizsgálatokat végzünk, hiszen a bekért számot két helyen is teszteljük közel ugyanazokkal a feltételekkel). A feladatot át tudjuk fogalmazni elöltesztelős ciklusra, ez már nem tartalmaz felesleges teszteléseket, de a bekérést kétszer kell szerepeltetni:

```
CLS
INPUT "Mondj egy számot 1 és 10 között!", a
DO WHILE a<1 OR a>10
    PRINT "1 és 10 közötti számot kérek! Próbáld
    újra!"
    INPUT "Mondj egy számot 1 és 10 között!", a
LOOP
CLS
LOCATE 12, 35
PRINT "A számod:"; a
```

Tehát bekérjük a számot, megvizsgáljuk, hogy jó-e, és ha nem, akkor kiíratunk egy hibaüzenetet és újból bekérjük a számot, s tesszük ezt mindaddig, amíg csak jó értéket nem kapunk.

53. példa Kérjünk be két tetszőleges számot, adjuk össze őket, majd a végén kérdezzük meg, hogy legyen-e vége!

Megoldás: A feladatot már megoldottuk a feltételes ugrások segítségével (IF–THEN–GOTO), de most inkább tesztelős ciklust kellene alkalmazni.

```
DO
    CLS
    INPUT "Kérem az első számot: ", a
    INPUT "Kérem a második számot: ", b
    PRINT "A két szám összege:"; a+b
    LOCATE 23
    PRINT "Akarsz még számolni? (i/n) ";
    c$ = INPUT$(1)
LOOP UNTIL c$="i" OR c$="I"
```

Vagyis a programból akkor lépünk ki, ha a leütött billentyű nem i vagy I. Persze ezt a programot is módosíthatjuk úgy, hogy csak az i, I, n, N billentyűt fogadjuk el:

```
DO
    CLS
    INPUT "Kérem az első számot: ", a
    INPUT "Kérem a második számot: ", b
    PRINT "A két szám összege:"; a+b
    PRINT "Akarsz még számolni? (i/n) ";
    DO
        c$ = INPUT$(1)
        IF c$<>"i" AND c$<>"I" AND c$<>"n" AND
c$<>="N" THEN BEEP
LOOP UNTIL c$="i" OR c$="I" OR c$="n" OR c$="N"
LOOP UNTIL c$="i" OR c$="I"
```

54. példa Adjuk össze a számokat 1-100-ig tesztelős ciklusok segítségével!

A régi megoldás: A feladatot más számokkal már megoldottuk, csak ott számlálós ciklust (FOR–NEXT) alkalmaztunk.

```
CLS
osszeg = 0
FOR i = 1 TO 100
    osszeg = osszeg + i
NEXT i
PRINT „1-100-ig a számok összege:"; osszeg
```

Megoldás: Most írjuk át ezt hátultesztelős ciklusra (azért hátultesztelősre, mert a ciklusmagot legalább egyszer biztosan végre kell hajtani).

```
CLS
osszeg = 0
i = 1
DO
    osszeg = osszeg + i
    i = i + 1
LOOP UNTIL i>100
PRINT "1-100-ig a számok összege:"; osszeg
```

Látható, hogy ebben az esetben két dologról nekünk kell gondoskodni: a ciklusváltozó kezdőértékéről (`i=1`*), valamint a ciklusváltozó értékének megváltoztatásáról a ciklusmagban (*`i=i+1`*).*

55. példa Írjunk programot, amely egy tanuló jegyeinek átlagát számítja ki! Az érdemjegyeket egyesével kérjük be, a bekérést a 0 (nulla) zárja le!

Megoldás: Nem tudjuk, hány jegyet kérünk be, de azt tudjuk, hogy egészen addig, amíg nullát nem kapunk. A feladatot tesztelős ciklussal kell tehát megoldani. Kell két segédváltozó is, amelyek az addig bekért jegyek darabszámát (**db**), valamint összegét (**s**) tartalmazzák.

```
CLS
PRINT "Kérem a jegyeidet egyesével, ha vége, ak-
kor a nullát üsd le!"
s = 0
db = 0
DO
    INPUT "jegy: ", jegy
    db = db + 1
    s = s + jegy
LOOP UNTIL jegy = 0
db = db - 1     (Erre azért van szükség, mert az utolsó jegynél (0) is
                 növeltük a db értékét)
PRINT "Az átlagod: "; s / db
```

Feladatok:

74. Módosítsa a fenti példát úgy, hogy a programocska tesztelje a bekért érdemjegyet, hogy az valóban lehet-e érdemjegy (azaz 1 és 5 közötti szám-e)!
75. A ***Feltételes ugrások*** fejezet feladatait írja meg tesztelős ciklusok alkalmazásával!
76. Szorozza össze az egész számokat 1-től 5-ig tesztelős ciklusok segítségével!

2.9. Egy kis matematika...

... de persze nem olyan sok, hogy meg kellene ijedni tőle. A számok kezeléséről lesz szó ebben a fejezetben.

A számításokhoz műveleteket és függvényeket alkalmazhatunk. A fontosabb műveletek: alapműveletek (+, -, *, /), hatványozás: ^ (kalap); egészosztás (olyan osztás, amelynek az eredménye egész típusú): \ (backslash), illetve a MOD (modulus v. maradék). Logikai műveletek: NOT, AND, OR, XOR. Relációk: =, >, >=, <, <=, <> (nem egyenlő).

Egy kifejezésen belül a műveletek sorrendje pontosan meghatározott. Ez annyit jelent, hogy két művelet közül azt fogja a gép előbb elvégezni, amelyik előbb van a prioritási (elsőbbségi) sorban. A logikai műveletek és relációk sorrendje megfelel a fenti sorrendnek (azaz a legnagyobb prioritással a NOT és az = rendelkeznek); az aritmetikai műveletek sorrendje pedig a következőképpen alakul:

- (előjel) → *, / → \ (egészosztás) → MOD → +, -

Azonos prioritású műveleteket balról jobbra haladva végez el a gép. A műveleti sorrendet zárójelek alkalmazásával megváltoztathatjuk.

Egyéb számításokat függvények segítségével végezhetünk el. A legfontosabb matematikai függvények ABC rendben a következők (természetesen ennél jóval több van):

ABS(x)	Abszolútérték
CINT(x)	Egészre kerekítés
COS(x)	Koszinusz
EXP(x)	$=e^x$ ahol e=2,71... (konstans)
FIX(x)	Egész számmá alakítás csonkítással
INT(x)	Egész számmá alakítás – egészrész
LOG(x)	Természetes alapú logaritmus

RND	0 és 1 közötti véletlen szám előállítása
SGN(x)	Előjel (szignum) függvény
SIN(x)	Szinusz
SQR(x)	Négyzetgyök
TAN(x)	Tangens

Az eddigi példák és feladatok csak egész számokkal foglalkoztak, azonban sokszor lehet szükségünk egyéb, ún. valós típusú számokra is. Ezeknél a megjelenítés okozhat időnként problémát...

56. példa Írassuk ki a 180 fok szinuszát!

Megoldás:

```
CLS
PRINT SIN(3.14)
```

A trigonometriai függvények esetén (SIN, COS, TAN) a szöget nem fokokban, hanem radiánban kell megadnunk. 180 fok radiánja π, ennek körülbelüli értéke 3,14). A Quick Basic nyelvben a tizedes-elválasztó nem a vessző, hanem a pont! Az eredmény a következő lett: `1.592448E-03`. *Ez a következőt jelenti: 1,592448*10^{-3}, vagy másképpen felírva 0,00152448.*

Ez az eredmény – bár matematikailag kifogástalan – de mégsem látható jól. Jobb lenne, ha a képernyőre a 0,00152448 alakban (vagy ennek valamennyire kerekített alakjában) kerülne. A kijelzés formátumának megadására szolgál a USING utasítás, amely mindig csak a PRINT-tel együtt használható. Alakja:

PRINT USING "forma"; kiírandó_kifejezés(ek)

ahol a forma egy szöveges kifejezés, amely a kiírandó kifejezés formáját adja az alábbiak szerint:

- # egy helyi értéket jelöl
- . tizedespont
- , ezreselválasztó
- + előjel megjelenítése

A USING használatával a szám értéke nem változik, csak a kijelzését módosítjuk!

57. példa Írassuk ki a 0,0000456, a 12345,567 és -2,345678 számokat formázva a képernyőre!

Megoldás:

```
CLS
PRINT USING "#.######"; 0.0000456
PRINT USING "+##,###.#"; 12345.567
PRINT USING "#######.###"; -2.345678
PRINT USING "##.###"; -3.1
```

Az első számot 6 tizedes jegyre kerekítettük, a másodikat elláttuk előjellel, ezreselválasztóval és 1 tizedes jegyre kerekítettük, illetve a harmadikat 3 tizedes jegy pontossággal a felette levő számhoz igazítottuk azzal, hogy a tizedespont elé 7 db # jelet tettünk. A negyedik számban nem volt csak egyetlen tizedes jegy, de mi kértük három megjelenítését.

58. példa Írjunk programot, amely a 12,67 és -3,51 számokat egészre változtatja mindhárom (CINT, INT és FIX) függvény használatával!

Megoldás:

```
CLS
a = 12.67
b = -3.51
PRINT "Eredeti:", a, b
PRINT "    INT:", INT(a), INT(b)
PRINT "    FIX:", FIX(a), FIX(b)
PRINT "   CINT:", CINT(a), CINT(b)
```

Az **INT** *függvény a szám egészrészét veszi, vagyis a hozzá legközelebbi, de nála kisebb számot (12 és -4). A* **FIX** *függvény a tizedesjegyek egyszerű elhagyásával képez egész számot (12 és -3). Az utolsó, a* **CINT** *függvény szabályosan kerekít, tehát a számhoz legközelebbi egész számot adja eredményül (13 és -4).*

59. példa Kérjünk be egy számot, és jelenítsük meg a négyzetgyökét két tizedesjegy pontossággal!

Megoldás: Mivel csak nemnegatív számnak van négyzetgyöke, ezért a bekért értéket átalakítjuk az ő abszolút értékére, majd a PRINT USING segítségével tudjuk formázottan kiíratni az eredményt.

```
CLS
INPUT "Mondj egy számot, megmondom a gyökét!", a
a = ABS(a)
gyok = SQR(a)
PRINT a;"gyöke:"; USING "######.##"; gyok
```

60. példa Számítsuk ki egy bekért szög szögfüggvényeit!

Megoldás: A szög értékét először át kell alakítani radiánra.

```
CLS
INPUT "Mondj egy szögértéket!"; szog
pi = 3.1415
rad = szog*pi/180
PRINT "A szög szinusza: "; USING "+##.###";
SIN(rad)
PRINT "A szög koszinusza: "; USING "+##.###";
COS(rad)
PRINT "A szög tangense: "; USING "+##.###";
TAN(rad)
```

Feladatok:

77. Számítsa ki egy négyzet kerületét és területét, ha bekéri az oldalát (Tesztelje, hogy pozitív számot kapott-e!)!
78. Módosítsa az előbbi feladatot úgy, hogy nem teszteli a bekért értéket, hanem automatikusan az abszolút értékével számol!
79. Számítsa ki egy kör kerületét és területét, ha bekéri a sugarát (tesztelje!)!
80. Kérjen be egy számot! Ha a szám negatív, vegye az abszulút értékét! ossza el a számot 13-mal, és két tizedesjegy pontossággal írja ki az eredményt a képernyőre!
81. Kérjen be egy tetszőleges számot, és kerekítse két tizedesjegyesre!
82. Kérjen be három számot, és írja ki az átlagukat!
83. Írja ki a képernyőre egymás alá a szögeket 5°-onként 0-90-ig! A szögek mellé írja ki a színuszukat, koszinuszukat és tangensüket! Ne feledkezzen meg a radiánra való átváltásról!

2.10. Szövegek kezelése

A számok után most nézzük meg, hogy hogyan kezelhetjük a szövegeinket. A szöveg mindig karakterek sorozata (maximum 32767 karakter lehet egy szöveges változó hossza).

A legfontosabb szövegkezelő függvények ABC rendben a következők:

ASC	Karakter ASCII kódjának megadása
CHR$	ASCII kódból karakter
INSTR	Karaktersorozat keresése szövegben
LCASE$	Szöveg átalakítása kisbetűssé
LEFT$	A szöveg bal oldala
LEN	A szöveg hossza
MID$	A szöveg egy része
RIGHT$	A szöveg jobb oldala
SPACE$	Szóközökből álló karaktersorozat
SPC	Szóközök kiíratása
STR$	Átalakítás számból szöveggé
UCASE$	Szöveg átalakítása nagybetűssé
VAL	Átalakítás szövegből számmá

61. példa Kérjünk be egy szöveget és azt pontosan a képernyő közepére írassuk ki!

Megoldás: Ahhoz, hogy a képernyő közepére írassunk, ismernünk kell a szöveg hosszát, ezt adja nekünk a LEN függvény. A kiíratást pontosan annyival kell a képernyő közepétől balra elkezdeni, amennyi a szóban forgó szöveg hosszának a fele.

```
CLS
INPUT "írj be egy szöveget! ", sz$
hossz = LEN(sz$)
LOCATE 12, 40 - (hossz \ 2)
PRINT sz$
```

Emlékeztetőül: a szöveges típusú változók azonosítója végén szerepeltetni kell a $ jelet! A program 3. sorában a ***hossz*** *nevű változóba tesszük el a szöveg hosszát, amelyet a* LEN *függvény ad meg a számunkra. A* LOCATE *sorban a képernyő középső oszlopától (40) a hossz felével kell visszafelé elkezdeni a kiírást. Az osztást az ún. egészosztással végeztük el (\), amelynek az eredménye a helyes eredmény egészrésze. Erre azért volt szükség, mert ha a szöveg páratlan számú karaktert tartalmaz, akkor a fele nem lenne egész szám* .*

* Meg kell jegyezni, hogy a ha LOCATE utasításban nem egész számot adunk meg, akkor a LOCATE automatikusan elvégzi az egészre konvertálást.

62. példa Kérjünk be egy szöveget, és írassuk ki, hogy található-e benne szóköz!

Megoldás: A szövegben **keresni** kell a szóközt, ezért az INSTR függvény fogjuk alkalmazni. A függvény alakja:

INSTR (kezdőpoz, szöveg, keresett_szöveg)

Eredményül azt a számot kapjuk vissza, ahányadik pozíción kezdődik a **szöveg**ben a **keresett_szöveg**. Amennyiben a **keresett_szöveg** nem található, akkor az INSTR függvény a nulla értéket adja vissza. A **kezdőpoz** paraméterrel (megadása nem kötelező) a keresés kezdő pozícióját adhatjuk meg.

```
CLS
INPUT "Kérek egy szöveget: ", sz$
hol = INSTR(sz$, " ")
IF hol>0 THEN
    PRINT "A szövegben van szóköz, mégpedig a ";
hol; ". helyen"
ELSE
    PRINT "A szövegben nincs szóköz!!"
END IF
```

63. példa Írassunk ki egy bekért nevet karakterenként, minden karakter után egy kis szünetet tartva!

Megoldás: a karakterenkénti kiíratás nagyon hatásos eszköz. A megoldásához egy ciklus kell, amely a szövegen végigmegy. A szöveg egy karakterét (vagy egy részét) a MID$ függvény adja. Alakja:

MID$ (szöveg, kezdőpoz, hossz)

A függvény eredményül a **szöveg** szövegben a **kezdőpoz**tól kezdődő **hossz** számú karakterből álló szöveget ad vissza. Ez utóbbi elhagyható, ekkor a szöveg végéig tart az eredményül adott karaktersorozat. Példánkban a **hossz** 1 lesz...

```
CLS
INPUT "Kérek egy szöveget: ", sz$
CLS
LOCATE 5
hossz = LEN(sz$)
FOR i = 1 TO hossz
    kar$ = MID$(sz$, i, 1)
    PRINT kar$;
    FOR j = 1 TO 1500 : NEXT j
NEXT i
```

*A ciklus belsejében a **kar$** változó tartalmazza a bekért szöveg (**sz$**) **i**-edik karakterét. Ezt íratjuk ki a következő sorban, majd a **j**-ciklus segítségével várakoztatjuk a programot (itt a ciklusváltozó végértékét, amit én 1500-ra definiáltam, a kedves olvasó a saját ízlésének megfelelően állítsa be, hogy a szöveg kiíratása a lehető leglátványosabb legyen).*

64. példa Kérjünk be egy szöveget, és számoljuk meg, hogy hány „s" betűt tartalmaz!

Megoldás: A szövegen karakterenként végig kell menni, közben egy **db** nevű változóban számlálunk.

```
CLS
INPUT "Kérek egy szöveget: ", sz$
hossz = LEN(sz$)
db  = 0
FOR i = 1 TO hossz
   IF MID$(sz$, i, 1) = "s" THEN db=db+1
NEXT i
PRINT "A szövegben"; db; "db s betű van."
```

Feladatok:

84. Kérjen be egy mondatot, és adja meg, hogy hány szóból áll! (Ötlet: mit érdemes számolni?)
85. Kérjen be egy szöveget, és számolja meg, hogy hány számjegyet tartalmaz! (Ötlet: szövegekre is lehet alkalmazni a <, > jeleket.)
86. Kérjen be egy szöveget, majd titkosítsa a következő módon: minden karakter helyett vegye az 1-gyel utána következőt! Írja ki az eredeti szöveget, majd alá a titkosítottat is!

65. példa Kérjük be a felhasználó teljes nevét, és írassuk ki külön a vezeték-, külön a keresztnevét (pillanatnyilag nem foglalkozunk azokkal, akiknek három vagy több részből áll a neve)!

Megoldás: A bekért szöveg bal oldala a vezetéknév, a jobb oldala a keresztnév, az alkalmazott függvények tehát a LEFT$ és RIGHT$ lesznek. A LEFT$ függvény alakja a következő:

LEFT$ (szöveg, hossz)

Eredményül a függvény a **szöveg** első **hossz** számú karakteréből álló szöveget adja vissza. A **RIGHT$** függvény ugyanígy működik, csak a **szöveg** végéről ad vissza **hossz** számú karaktert.

A megoldás első lépése az lesz, hogy megállapítsuk, hol végződik a vezetéknév, és hol kezdődik a keresztnév, vagyis keressük a szóközt.

```
CLS
INPUT "Mi a teljes neved"; nev$
hossz = LEN(nev$)
szpoz = INSTR(nev$, " ")
vnev$ = LEFT$(nev$, szpoz-1)
knev$ = RIGHT$(nev$, hossz-szpoz)
LOCATE 5
PRINT "A vezetékneved: "; vnev$
PRINT "A keresztneved: "; knev$
```

*Az **szpoz** változó tartalmazza a névben a szóköz pozícióját, ebből adjuk meg a keresztnevet és a vezetéknevet.*

Feladatok:

87a. Módosítsa a fenti programot úgy, hogy tesztelje, tényleg teljes-e a név!

88b. Módosítsa a fenti példaprogramot úgy, hogy a háromnevűeket is figyelembe vegye (vezetéknév, középső név, keresztnév)! Ha valaki pedig azt állítja magáról, hogy háromnál több részből áll a neve, ne higgye el!

66. példa Készítsünk egy kis programot, amelyben jelszót alkalmazunk! A jelszó legyen a „BASIC" szó. A program első lépéseként bekérjük a jelszót, és kiíratjuk, hogy egyezik-e az eltárolt eredetivel.

Megoldás: A megoldás során egy problémával kell szembenéznünk, mégpedig azzal, hogy a begépelt szót el kellene fogadni „Basic", „BASIC", „basic", stb. alakokban egyaránt. Erre a megoldást az jelentheti, ha a szöveget átalakítjuk nagybetűssé az UCASE$ függvénnyel. Használata:

UCASE$ (szöveg)

Eredménye a **szöveg**ből nagybetűssé alakított karaktersorozat lesz.

```
jelszo$ = "BASIC"
CLS
INPUT "Kérem a jelszót! ", pw$
pw$ = UCASE$(pw$)
IF pw$<>jelszo$ THEN
```

```
    PRINT "Nem helyes jelszó!!"
ELSE
    PRINT "Helyes jelszó!"
END IF
```

*A **pw** változó tartalmazza a begépelt jelszót (**pass**word=jelszó), ezt a bekérés után rögtön nagybetűssé alakítjuk, és így hasonlítjuk össze az eredetivel.*
Az `UCASE$` *függvényhez hasonlóan működik az* `LCASE$` *függvény is, ez azonban a szöveget kisbetűssé alakítja. Figyelem: az* `UCASE$` *és a* `LCASE$` *függvények csak az angol ABC kis és nagybetűit kezelik helyesen!*

67. példa Kérjünk be egy karaktert, és jelenítsük meg az ASCII kódját!

Megoldás: Az ASC függvényt kell alkalmazni.
Megadása: **ASC** (szöveg)
Eredményül a **szöveg** első karakterének ASCII kódját kapjuk vissza.

```
CLS
PRINT "Kérek egy karaktert! "
c$ = INPUT$(1)
kod = ASC(c$)
PRINT "A leütött billentyű ASCII kódja:"; kod
```

68. példa Sok olyan karakter van, amely nem található meg a billentyűzeten, és csak az ASCII kód segítségével (ALT+kód) lehet begépelni. Az alábbi példaprogram a 32-128 kódú karaktereket jeleníti meg.

Megoldás: A CHR$ függvényt kell használni. Megadása:
CHR$ (kód)
Eredménye a **kód** ASCII kódú karakter.

```
CLS
SOR = 4 : OSZL = 10
FOR I = 32 TO 127
    LOCATE SOR, OSZL
    PRINT i; ": "; CHR$(i)
    SOR = SOR+1
    IF SOR = 20 THEN
        SOR = 4 : OSZL = OSZL+10
    ENDIF
NEXT i
```

Feladatok:

89. Kérjen be egy szöveget, és titkosítsa a következőképpen: minden karakter helyett a karakter ASCII kódját írja ki!

69. példa Az iskolások a vakáció előtti napokon a táblára felírják a **vakáció** szót úgy, hogy minden nap egy betűvel többet írnak hozzá. Írjuk meg ezt a programot úgy, hogy a képernyőre egymás alá, jobbra igazítva írassuk ki az **ó–ió–ció–...–akáció–vakáció** sorozatot!

Megoldás: Összesen annyi sort kell kiíratni, amilyen hosszú a vakáció szó. Ez tehát egy ciklus lesz, mégpedig számlálós ciklus. Minden egyes lépésben az utolsó **i** db. karaktert kell kiíratni. A probléma már csak az igazítással van. Ezt megtehetnénk úgy, hogy minden lépésnél a LOCATE utasításban kiszámítanánk a kiíratandó szöveg helyét (ekkor a `LOCATE ,45-i` formulát kellene használni, ahol a 45 az az oszlop, amelyhez a szövegeket igazítjuk), de a megoldásban inkább más módszert alkalmazunk...

```
CLS
sz$ = "vakáció"
hossz = LEN(sz$)
LOCATE 10
FOR i = 1 TO hossz
    ksz$ = SPACE$(hossz-i)+ RIGHT$(sz$,i)
    LOCATE ,37
    PRINT ksz$
    SLEEP(1)
NEXT i
```

*A ciklusban a **ksz$** változó tartalmazza az egyes lépésekben kiíratandó szöveget. Ezt úgy kapjuk meg, hogy az eredeti szöveg (**sz$**) utolsó **i** db karaktere (`RIGHT$(sz$,i)`) elé annyi szóközt teszünk, amennyivel minden lépésben azonos hosszúságot kaphatunk (`SPACE$(hossz-i)`). Ezt a kettőt összefűzzük – ezt a műveletet nevezzük konkatenációnak –, mégpedig a + jel segítségével. Az összefűzés a két (vagy több) szöveg egyszerű egymás után írását jelenti. Ha ez megvan, akkor már csak az a dolgunk, hogy az egyes karaktersorozatokat ugyanarról a kezdő pozícióról (37. oszlop) kezdve (hiszen egyenlő hosszúak) kiírassuk. A `SLEEP(1)` – amelyről már tudjuk, hogy 1 mp-es várakoztatást jelent – csak azért kell, hogy a kis program látványosabb legyen.*

(A látványosságot szolgálhatja még, ha minden lépésben más-más színt alkalmazunk, ehhez a COLOR utasításban a színeket szintén a ciklusváltozó segítségével adhatjuk meg.)

70. példa Kérjünk be egy pozitív számot, majd alakítsuk szöveggé és fűzzük hozzá a "Ft" mértékegységet!

Megoldás: Szöveggé alakításhoz a STR$ függvényt alkalmazhatjuk. Használata:

STR$ (szám)

Eredménye a **szám** numerikus kifejezés karakteres alakja (pl. a 324 számból a „324" karaktersorozatot állítja elő).

```
CLS
INPUT "Kérek egy pozitív számot: ", a
szam$ = STR$(a)+" Ft"
PRINT szam$
```

Az adatbekéréskor, amikor numerikus adatra várakozunk az INPUT utasítás segítségével, gondot okozhat, ha nem számot kapunk. Ilyenkor a „Redo from start" figyelmeztetéssel jelentkezik a Quick Basic, és újra meg lehet kísérelni a szám beírását. Ez nem elegáns, és a gondosan megtervezett kiíratásainkat is elronthatja. A probléma lekezelhető azzal az egyszerű módszerrel, hogy nem numerikus, hanem minden esetben szöveges adatot kérünk be – hiszen az bármi lehet –, és utólag alakítjuk számmá. Erre szolgál a VAL függvény. Használata:

VAL (szöveg)

Eredménye a **szöveg** számmá alakított változata (pl. a "1234Ft" szöveget 1234-re alakítja, ez már szám lesz). A VAL függvény a szöveget addig alakítja, amíg meg nem akad valamilyen nem szám jellegű karakternél (az előző "1234Ft" példánál ezért lett ebből az 1234 szám).

71. példa Kérjünk be egy számot, de kezeljük szövegként! Teszteljük, hogy kapott szöveg átalakítható-e valóban számmá!

Megoldás: A bekért szöveg akkor szám, ha az első karaktere számjegy vagy negatív előjel. Ezt kell tesztelni. Az **ek** nevű változóban a bekért szöveg első karakterét fogjuk tárolni.

```
CLS
DO
    INPUT "Kérek egy számot!", sz$
    ek$ = LEFT$(sz$,1)
LOOP UNTIL ek$="-" OR (ek$<="9" AND ek$>="0")
szam=VAL(sz$)
PRINT SZAM
```

A példában megfigyelhetjük azt is, hogy a szövegekre is értelmezhetjük a relációs jeleket. Az összehasonlítás alapja a karakterek ASCII kódja, így lesz értelme annak, hogy az egyik szöveg „kisebb" mint a másik.

Feladatok:

90. Kérjen be egy szöveget, és írassa ki a képernyőre karakterenként függőlegesen!
91. Kérjen be egy szöveget, és írassa ki visszafelé!
92. Kérjen be egy szöveget, és írassa ki a középső karakterét! Ha nincs ilyen, akkor a két középsőt írassa ki!
93. Kérjen be egy teljes nevet, és írassa ki a monogramját!
94. Kérjen be egy szöveget, és minden harmadik karakterét cserélje ki a "@" karakterre!
95. Kérjen be egy mondatot, és írassa ki szavanként egymás alá!

2.11. Több adat együttes kezelése

2.11.1. Konstans adatok

Több adat kezelése eléggé nehézkes lehet változókkal, hiszen sok változót kell alkalmazni, és egy idő után az ember elfelejti, melyik mit is jelent. Amennyiben előre tudjuk, hogy a programunk milyen adatokkal fog dolgozni, akkor a **DATA** kulcsszó után elsorolhatóak ezek az adatok, és ahol szükséges, ott a **READ** utasítással kiolvashatók.

72. példa Készítsünk névjegyet a következő adatok felhasználásával: Zseni Alfonz programozó; cím: 9999. Sajóhajós Éteri Eduárd u. 3.; mh: TutiProgram Kft.; Tel: (06-1)123-4567!

Megoldás: Az egyes adatokat a DATA kulcsszó után soroljuk fel, a READ-del olvassuk ki, így csupán két változót kell felhasználni: egy numerikus és egy szöveges változót.

```
CLS
READ sz$
LOCATE 8, 40-LEN(sz$)\2
PRINT sz$
READ sz$
LOCATE 9, 40-LEN(sz$)\2
PRINT sz$
READ sz$
LOCATE 14, 3
PRINT „Lakáscím: „; sz$
READ sz$
LOCATE 15, 3
PRINT „Munkahely: „; sz$
READ sz$
LOCATE 16, 3
PRINT „Telefon: „; sz$

DATA "Zseni Alfonz", "programozó"
DATA "9999. Sajóhajós Éteri Eduárd u. 3"
DATA "TutiProgram Kft.", "(06-1)123-4567"
```

A következőkre kell odafigyelni: a READ sorban olvassa ki a DATA sorokban leírt adatokat. Egy DATA után több adatot is szerepeltethetünk, vesszővel elválasztva.

Amennyiben újra az adatsor elejéről szeretnénk adatokat olvasni, a **RESTORE** utasítást kell alkalmazni; ennek hatására a mutató visszaáll az adatsor elejére (az első DATA utasításra). Lehetőség van bármely DATA sorra ráállni, ekkor a GOTO-hoz hasonlóan az illető sort meg kell jelölni egy címkével, és a RESTORE utasításban a címkét is meg kell adni.

A megoldást ritkán alkalmazzuk, de pl. egy kis zene tárolására kiválóan alkalmas. Használható még programok tesztelésére is, ha nem akarunk minden egyes futás alkalmával egy jó csomó adatot bekérni.

Feladatok:

96. Kérje be egy hónap sorszámát, és írja ki szövegesen a nevét! Az egyes hónapok neveit tárolja DATA sor(ok)ban! Tesztelje, hogy tényleg hónap-e!

2.11.2. Tömbök

Sok, azonos típusú – numerikus vagy szöveges – adat tárolására tömböket alkalmazunk. A tömb lehet egy adatsor (ekkor vektornak nevezzük), és lehet egy táblázat is.

Egy 8 elemű vektort mutat a következő ábra, amelynek az elemei 2,3,6,2,5,6,9 és 2.

2	3	6	2	5	6	9	2

Egy 3 soros és 8 oszlopos tömböt mutat a következő ábra:

2	3	6	2	5	6	9	2
8	5	6	9	7	1	5	3
4	5	8	9	2	2	5	6

A tömböket is változókkal azonosítjuk, azonban itt először a tömböt létre kell hozni, deklarálni kell. A tömbök deklarálására a DIM utasítást kell alkalmazni.

DIM változóazonosító (kezdőérték **TO** végérték, kezdőérték **TO** végérték...)
ahol a kezdőérték és a végérték az oszlop- és sorazonosítók kezdő- és végértékét jelenti, nem a beírt adatokét!

DIM változóazonosító (végérték)
itt egy olyan tömböt hozunk létre, amelynél a kezdőérték 0 (nulla). Amennyiben teljesen elhagyunk mindent a zárójeleken belül, akkor egy 0-10-ig dimenzionált tömb jön létre.

Amennyiben a tömb szöveges adatokat fog tartalmazni (pl. nevek listája) akkor az azonosítójához ugyanúgy hozzá kell fűzni a $ jelet, mint bármely más változó nevéhez!

A fenti két tömb deklarációja a következő lesz:

DIM v (1 TO 8) illetve

DIM t (1 TO 3, 1 TO 8)

Ezzel a **v** változó fogja azonosítani a fenti vektort, a **t** változó pedig a fenti tömböt.

Amennyiben egy tömböt újra akarunk dimenzionálni a program egy későbbi szakaszában, akkor a **REDIM** utasítást kell alkalmazni. Működése megegyezik a DIM működésével, de csak már létező tömbváltozóra adható meg. A feleslegessé vált tömböt pedig törölhetjük a memóriából az **ERASE** utasítással.

A tömböket értékekkel vagy DATA sorokból, vagy adatbekéréssel lehet feltölteni. Nézzünk erre egy példát!

73. példa Kérjünk be 5 számot egy vektorba, és számoljuk ki az összegüket és átlagukat!

Megoldás:

```
CLS
DIM a(1 TO 5)
REM adatbekérés:
FOR i=1 TO 5
PRINT "Kérem a(z)"; i; ". adatot: "; INPUT "", a(i)
NEXT i
REM Az összeg számítása:
osszeg=0
FOR i=1 TO 5
      osszeg=osszeg+a(i)
NEXT i
atlag=osszeg/5
REM eredmények kiíratása:
PRINT "Összeg:"; osszeg
PRINT "Átlag:"; USING "###.##" atlag
```

A programban szereplő **REM** *utasítások megjegyzések elhelyezését teszik lehetővé a programban, ugyanis a Quick Basic a REM-mel kezdődő sorokat a végrehajtás során nem veszi figyelembe (alkalmazható még hasonló célra a sor előtt az aposztróf (') karakter is). A képernyőtörlés után létrehoztunk egy ötelemű tömböt (DIM), majd az első FOR–NEXT ciklus az adatbekérést, a második az összeg számítását végzi el. A tömb elemeire a*

változónév(elemszám)

formulával hivatkozhatunk (pl. **a***(3) az* **a** *tömb 3. elemét jelenti).*

74. példa Definiáljunk egy 10 elemű tömböt, töltsük fel számokkal DATA sorokból, majd állapítsuk meg, hogy a tömb hányadik eleme a legnagyobb!

Megoldás: A legnagyobb elem kereséséhez végig kell menni a tömb minden elemén, tárolni az addigi legnagyobb elemet, és ezt összehasonlítani az aktuális tömbelemmel. Ha ez nagyobb az eddigi maximumnál, akkor cseréljük az értéket...

```
CLS
REM Tömb létrehozása és feltöltése adatokkal:
DIM a(1 TO 10)
FOR i=1 TO 10
    READ a(i)
NEXT i
REM Maximumkeresés:
max = a(1)
maxindex = 1
```

```
FOR i = 2 TO 10
    IF a(i)>max THEN
        max = a(i)
        maxindex = i
    END IF
NEXT i
REM Eredmények kiíratása:
PRINT "Az adatsor legnagyobb eleme: ";max
PRINT "Ez az elem a tömb "; maxindex; ". helyén
van"
REM Adatok:
DATA 5, 3, 9, 12, 23, 100, 23, 1, 0, 62
```

75. példa Gyakori feladat az adatok sorba rendezése is. Rendezzünk egy 10 elemű tömböt a következő módszerrel: haladjunk végig a tömb elemein először 10-től 2-ig, majd 10-től 3-ig, stb., és ha egy elem kisebb az előtte lévőnél, akkor cseréljük meg őket! Az első menet végére az első elem a helyére kerül, a második végére a másodikelem kerül a helyére, stb. (A módszert buborék-rendezésnek hívják.)

Megoldás:

```
CLS
PRINT "Az adatsor rendezés előtt: "
REM Tömb létrehozása és feltöltése adatokkal:
DIM a(1 TO 10)
FOR i=1 TO 10
    READ a(i)
    PRINT a(i);
NEXT i
REM Rendezés:
FOR i=2 TO 10
    FOR j=10 TO i STEP -1
        IF a(j)<a(j-1)THEN SWAP a(j), a(j-1)
    NEXT j
NEXT i
REM Az eredmény kiíratása:
LOCATE 4,1
PRINT "Rendezés után:"
FOR i=1 TO 10
    PRINT a(i);
NEXT i
REM Adatok:
DATA 5, 3, 9, 12, 23, 100, 23, 1, 0, 62
```

A programban két változó cseréjét a **SWAP** *utasítás segítségével oldottuk meg. Használata:* **SWAP v1, v2**. *Eredményeként a* ***v1*** *változó értéke* ***v2****-be kerül, és viszont.*

Feladatok:

97. Készítsen programot az előző példa alapján, amely nem maximumot, hanem minimumot keres!
98. Kérjen be egy tömbbe 10 számot, és adja meg, van-e a számok között 0 (nulla)!
99. Módosítsuk az előző feladatot úgy, hogy amennyiben talált nullát, akkor azt is meg tudja mondani, hogy hány darab van belőle!
100. Kérjen be egy tömbbe 10 számot, és az összes negatív számot változtassa pozitívvá!
101. Kérjen be egy tömbbe 10 számot, majd egy másik tömbbe tegye a számok 2-szeresét!
102. Kérjen be egy tömbbe 10 számot, majd egy másik tömbbe tegye át ezeket a számokat, de nagyság szerint rendezve! Alkalmazza a következő eljárást: első lépesben keresse meg a legkisebb elemet és tegye át a másik tömbbe, majd az eredeti helyén cserélje ki valami speciális – célszerűen valami jó nagy – számra! Ezt ismételje összesen 10-szer!
103. Módosítsa az előbbi feladatot úgy, hogy a rendezés helyben menjen végre (a legkisebb elemet minden lépésben cserélje ki először az első, majd a második, stb. elemmel)!

2.12. Írjunk játékprogramot!

A játékprogramok írása a programozás legizgalmasabb feladatai közé tartozik. Kezdetben persze nem írunk csillogó-villogó kalandjátékokat, de egyszerűbb játékok – főleg szerencsejátékok – megírása már nem okozhat gondot. Az alábbiakban néhány egyszerű és jópofa játékocskát fogunk elkészíteni. A feladatok megoldásai pedig remek programozási gyakorlatokat jelentenek – főleg azért, mert érdekesek.

Az első lépcsőfok mindenképpen az, hogy a véletlent megtanuljuk programozni. A számítógép elő tud állítani ún. véletlen számot, erre szolgál az RND függvény. Használata:

RND

Az RND függvény egy olyan véletlen számot állít elő, amely nulla és egy közé esik, de míg a nulla értéket felveheti, addig az egyet nem. Természetesen nem egész, hanem valós típusú számról van szó. Matematikailag leírva:

$$0 \leq \text{RND} < 1$$

Általában nekünk egy és valami között kell előállítani véletlen számot (pl. LOTTO-számoknál 1 és 90 között, kockadobásnál 1 és 6 között, stb.). Ekkor az RND eredményét egy kis számítással módosítani kell. Állítsunk elő például LOTTO-számot. A LOTTO-számok 1 és 90 közötti egész számok. Első lépésben az RND értékét szorozzuk 90-nel:

$$0 \leq \text{RND} \cdot 90 < 90$$

Ezzel egy nulla és 90 közötti számot kaptunk, ami nulla lehet, de 90 nem. Második lépésben csináljunk ebből egész számot; ezt a már ismert INT függvénnyel elintézhetjük:

$$0 \leq \text{INT}(\text{RND} \cdot 90) \leq 89$$

Most már egy nulla és 89 közötti egész számunk van, amely mindkét határértéket felveheti. Egyet kell már csak hozzáadni, és pontosan a kívánt számokhoz jutottunk.

$$1 \leq \text{INT}(\text{RND} \cdot 90) + 1 \leq 90$$

Tehát az INT(RND*90)+1 formulával egy 1 és 90 közötti egész típusú véletlen számot állítunk elő. Általában is igaz, hogy ha 1 és „valami" között kell egy véletlen számot előállítani, akkor az

INT(RND*"valami")+1

formulával tehetjük meg.

Két tetszőleges érték közti véletlen szám előállítása (legyenek ezek **k** és **v**) a következő képlettel állítható elő (levezetését az olvasóra bízom):

INT(RND*(v-k+1))+k (ebben mindkét határ benne lesz)

Például 50 és 100 közötti véletlen szám esetén: INT(RND*51)+50. Ez a fajta megadás azonban meglehetősen ritka.

Ez a szám, amit az RND előállít, csak ún. álvéletlen, vagy pszeudovéletlen szám. Ez annyit jelent, hogy a véletlen szám egy számítás eredményeképpen áll elő, vagyis nem valódi „véletlen" alapján készül. Ez természetes is, hiszen a gép nem tud „találomra" mondani egy számot, vagy „kihúzni egyet a kalapból". Egy véletlen szám mindig az előző alapján kerül kiszámításra, ezért az első számításakor létezik egy kezdőérték. Ez a kezdőérték állandó, és ez azt jelenti, hogy a ki-

számított véletlen számok is mindig ugyanazok lesznek. Ez viszont nem felel meg nekünk, mert ez a „véletlen" túl kiszámítható lenne... A megoldás az, ha be tudjuk állítani a véletlenszám-generátor kezdőértékét. De ha egy másik számot adunk kezdőértéknek, akkor ugyanott vagyunk, ahol voltunk: megint mindig ugyanazt a számsorozatot fogjuk megkapni, csak most egy másikat, mint az előbb. Tehát a generátor kezdőértékét is véletlenszerűen kellene beállítani...

A véletlen szám kezdőértékét a **RANDOMIZE** eljárással lehet beállítani. Ahhoz, hogy ez minden esetben más legyen – ha nem is véletlenszerű –, segítségül hívunk egy olyan függvényt, amely az éjfél óta eltelt másodpercek számát adja meg. Ez a függvény a **TIMER**.

Tehát minden olyan programban, amelyben véletlen számo(ka)t állítunk elő, a program elején – de mindenképpen az első RND előtt – a RANDOMIZE TIMER sort meg kell adni, amely a véletlenszám-generátor kezdőértékét beállítja „véletlenszerűen".

76. példa Rendezzünk LOTTO-sorsolást! Írassunk ki a képernyőre egymás mellé öt véletlenszerű 1 és 90 közötti számot!

Megoldás:

```
CLS
RANDOMIZE TIMER
FOR i = 1 TO 5
    lsz = INT(RND*90)+1
    PRINT lsz,
NEXT i
```

A program elején megadtuk a RANDOMIZE TIMER sort, amely a véletlenszerűségről gondoskodik. A ciklusban az ***lsz*** *nevű változóba generálunk egy véletlen számot (ötször), majd kiíratjuk.*

Feladat:

Egy öt elemű tömbbe generáljon öt lottó számot és mindegyik lépésben ellenőrizze, hogy volt-e már ilyen! Az eredmény öt különböző szám legyen!

Most már ennyi előzetes után igazán itt az ideje a játéknak! Csak még egy megjegyzés: innentől kezdve már minden programhoz készítünk tervet. Erre eddig nem volt szükség, hiszen minden programunk nyúlfarknyi volt, nem volt értelme a terv készítésének. Egy „komoly" programot azonban nem lehet elkezdeni terv készítése nélkül! A továbbiakban mindig elkészítjük a program tervét is, majd kódoljuk Basic nyelvre. A feladatok elkészítésénél se felejtsen el tervet készíteni!

77. példa Készítsünk programot, amelyben a gép „gondol" egy számot 1 és 3 között, mi tippelünk egy számot, és ha eltaláltuk, akkor nyertünk!

Megoldás: Készítsünk először programtervet!

A gép "gondol" egy számot (szam)
Tippelünk (tipp)
Összehasonlítjuk a kettőt (Ha szam=tipp, akkor...)

részletesebben:

szam=véletlen(1-3)
Be: tipp
Ha tipp=szam akkor Ki: "Nyertél"
különben Ki: "Vesztettél"

és most a program:

```
CLS
RANDOMIZE TIMER
szam = INT(RND*3)+1
PRINT "Gondoltam egy számra 1 és 3 között. Találd
ki!"
INPUT "Kérem a tippedet! ", tipp
IF szam=tipp THEN
    PRINT "Nyertél!!! Valóban erre gondoltam!"
ELSE
    PRINT "Nem talált! A gondolt szám a"; szam;
"volt..."
END IF
```

A programtervet úgy célszerű elkészíteni, hogy először nagy vonalakban megfogalmazunk egy vázlatot, majd ezt finomítjuk, vagyis kisebb lépésekre bontjuk. A terv nem hosszadalmas, egy kiíratásnál nem tartalmazza a helyet és a színt, ezt mind a program kódolása során kell megadni. A vezérlési szerkezetek leírásánál (elágazások és ciklusok) bemutattam, hogy az illető szerkezetet hogyan írhatjuk le, de a fenti példából is jól látszik. Egyéb leíró eszközök a következők: Ki (kiíratás, vagyis PRINT), Be (bekérés, azaz INPUT).

Feladatok:

104. Módosítsa a fenti programot úgy, hogy a végén megkérdezze, hogy „Akarsz még játszani?", és ha igen akkor kezdjük elölről. Ha lehet, alkalmazzon hozzá tesztelős ciklust!
105. Módosítsa a fenti programot úgy, hogy a kiíratások szebbek legyenek (hely, szín, stb.)!

78. példa Módosítsuk az előző kis játékot úgy, hogy ne egy játszmát játsszunk, hanem mondjuk ötöt, és számoljuk az összeredményt!

Megoldás: Az előző programot egy ciklusba kell ágyazni, és az összeredmény számolására szükségünk van egy új változóra. Megint készítsünk tervet!

```
pont=0 (kezdőérték)
Ciklus j=1-től 5-ig
      szam=véletlen(1-3)
      Be: tipp
      Ha tipp=szam akkor Ki: "Eltaláltad!"
                                  pont=pont+1
            különben Ki: "Nem találtad el"
      Várakozás egy billentyű nyomására
Ciklus vége
Ki: pont
Ha pont >=3 akkor Ki: "Nyertél"
      különben Ki: "vesztettél"
```

és most a program:

```
CLS
RANDOMIZE TIMER
pont = 0
FOR i = 1 TO 5
    szam = INT(RND*3)+1
    PRINT "Gondoltam egy számra 1 és 3 között.
    Találd ki!"
    INPUT "Kérem a tippedet! ", tipp
    IF szam=tipp THEN
        PRINT "Eltaláltad!"
        pont = pont+1
    ELSE
        PRINT "Nem talált! A gondolt szám a";
        szam; "volt..."
    END IF
    PRINT "Nyomj egy billentyűt a folytatáshoz"
    c$=INPUT$(1)
    PRINT
NEXT i
CLS
PRINT pont; "találatod volt az ötből."
IF pont>=3 THEN
    PRINT "Nyertél!!!"
ELSE
    PRINT "Vesztettél!"
END IF
```

A program a terv alapján egy kis tanulmányozás után érthető, egyetlen megjegyzés csak: a `c$=INPUT$(1)` *sor valósítja meg a*

várakozást. A ***c$*** *változóba bekerül a lenyomott billentyű, de nincs rá szükségünk, nem is használjuk fel a későbbiekben. Várakoztatásra alkalmas lehet még az* **INKEY$** *függvény is, amely csak abban különbözik az INPUT$(1)-től, hogy nem várakozik billentyű nyomására, hanem amennyiben történt ilyen, akkor visszaadja a lenyomott billentyű kódját. Miután nem várakozik, ezért egy ciklust kell köré szervezni:*

```
DO
LOOP UNTIL INKEY$<>""
```

Vagyis ciklus addig, amíg az INKEY$ már tartalmaz valamit, és nem üres szöveg ("") az értéke. Vigyázat, nem " " (szóköz karakter), hanem "" (üres szöveg vagy üres sztring).

Feladatok:

106. Szebbítse meg ezt a kis játékot is, valamint itt is tegye lehetővé az újra játszást!

A következő kis játék az előző továbbfejlesztése, de itt a nyeréshez már nem csupán a vakszerencsére támaszkodhatunk...

79. példa Készítsünk játékprogramot, amelyben a gép által „gondolt" számot kell kitalálnunk. Többet tippelhetünk (egészen addig, amíg ki nem találjuk a számot), a gép pedig folyamatosan tájékoztat bennünket, hogy „alá, vagy fölé lőttünk".

Megoldás: Lássuk a tervet:

```
szam=véletlen(1-100)
Ciklus
      Be: tipp
      Ha szam>tipp akkor Ki: "Nagyobbra gondoltam!"
      Ha szam<tipp akkor Ki: "Kisebbre gondoltam!"
amíg szam<>tipp
Ciklus vége
Ki: "Eltaláltad!"
```

és most a program:

```
CLS
RANDOMIZE TIMER
szam = INT(RND*100)+1
PRINT "Gondoltam egy számra 1 és 100 között. Ta-
láld ki!"
DO
    INPUT "Kérem a tippedet! ", tipp
    IF szam>tipp THEN PRINT "Nagyobbra gondoltam!"
    IF szam<tipp THEN PRINT "Kisebbre gondoltam!"
```

```
LOOP UNTIL szam=tipp
PRINT "Nyertél, erre gondoltam!"
```

Feladatok:

107. Szebbítse meg ezt a kis játékot is, valamint itt is tegye lehetővé az újra játszást!
108. Ha úgy érzi, hogy elsajátította a nyerő stratégiát – minél kevesebb lépésből kitalálni a gondolt számot –, próbálja meg megírni a játék fordított változatát, vagyis most a gépnek kell kitalálnia a számot! (A feladat nehezebb a példánál!)

80. példa Megint egy kis szerencsejáték – kockázzunk! Készítsünk programot, amelyben a gép is „dob" egyet, mi is „dobunk" egyet, és az nyer, aki nagyobbat dob! (Döntetlen esetén a gép nyer.)

Megoldás: Az előzőek megértése után csupán csak az a gond, hogyan oldjuk meg azt, hogy mi tudjunk dobni, és ne a gép dobjon helyettünk. Erre azt a nagyon egyszerű, de nagyon látványos módszert alkalmazzuk, hogy a gép addig készít véletlen számot és jeleníti meg a képernyőn (a szám „pörög"), amíg egy billentyű lenyomásával meg nem állítjuk. Első ugyebár a terv...

```
gepdob=véletlen(1-6)
Ciklus
    endob=véletlen(1-6)
    Ki: endob
amíg nincs lenyomott billentyű
Ciklus vége
Ha endob>gepdob akkor "Te nyertél" különben Ki: "Én nyertem""
```

és most a program:

```
CLS
RANDOMIZE TIMER
gepdob = INT(RND*6)+1
PRINT "Az én dobásom:"; gepdob
DO
    endob = INT(RND*6)+1
    LOCATE 2,1
    PRINT "A te dobásod:"; endob
LOOP UNTIL INKEY$<>""
IF endob>gepdob THEN PRINT "Nyertél!" ELSE PRINT
"Vesztettél!"
```

Feladatok:

109. Módosítsa a programot úgy, hogy lehetőséget adjon az újrajátszásra!
110. Módosítsa az előbbi feladatot úgy, hogy a döntetlent is kiírja!
111. Módosítsa az előbbi feladatot úgy, hogy döntetlen esetén újradobás legyen, egészen addig, amíg el nem dől a játszma!
112. Módosítsa az eredeti programot úgy, hogy most két kockával kelljen dobni, és a két kocka összeredménye döntsön!
113. Készítse el ennek a programocskának is a döntetlent lekezelő és újrajátszást megengedő változatait!
114. Mind az egy, mind a két kockás változatnál írja át a programot úgy, hogy egy ötjátszmás „bulit" játsszunk, és értékelje az összeredményt!

Az eddigi példák alapján ki-ki megpróbálkozhat egyéb – akár bonyolultabb – játékocska megírásával. A fenti példákat kedvcsinálónak szántam; látható, hogy viszonylag egyszerű eszközökkel szórakoztató játékokat tudtunk készíteni.

2.13. Menü készítése

Menüről beszélünk akkor, ha programunk több lehetőséget kínál fel, amelyből – legtöbbször egy billentyű lenyomásával – választhatunk, majd a választás alapján más és más programrészeket hajt végre. Ebben a fejezetben elkészítünk egy menüt, amelyet bármely programban lehet majd alkalmazni (persze egy kis „fazonigazítás" után).

A menük általános sémája a következő:

1. Menüpontok kiíratása a képernyőre
2. Várakozás egy billentyű lenyomására, majd a lenyomott billentyű vizsgálata: csak a felkínált lehetőségeket fogadjuk el!
3. A lenyomott billentyű alapján (ez leggyakrabban egy sorszám vagy egy kezdőbetű) elágazás szervezése, ahol az elágazás egy-egy ágán készítjük el a végrehajtandó programrészeket

81. példa Kérjünk be két számot, majd kérdezzük meg, hogy melyik alapműveletet végezzük el a két számmal! A kiválasztott műveletet végezzük el!

Megoldás: A bekérés után egy menüt készítünk. A program terve a következő:

```
Be: a,b
Menüpontok kiíratása (választási lehetőségek: 1-4 sorszámok)
Ciklus
      Be: c$ (lenyomott billentyű)
amíg c$<"1" vagy c$>"4"
Ciklus vége
Elágazás
      Ha c$="1" akkor Ki: a+b
      Ha c$="2" akkor Ki: a-b
      Ha c$="3" akkor Ki: a*b
      Ha c$="4" akkor
            Ha b=0 akkor Ki: "Nullával nem lehet osztani!"
                  különben Ki: a/b
Elágazás vége
```

A programlista:

```
REM Bekérés:
CLS
INPUT "Kérem az első számot: ", a
INPUT "Kérem a második számot: ", b
REM Menüpontok kiíratása a képernyőre:
CLS
PRINT "1. Összeadás"
PRINT "2. Kivonás"
PRINT "3. Szorzás"
PRINT "4. Osztás"
PRINT
PRINT "Melyiket választod? "
REM Várakozás egy billentyű lenyomására:
DO
    c$ = INPUT$(1)
LOOP UNTIL c$>="1" AND c$<="4"
REM Elágazás:
CLS
SELECT CASE c$
    CASE "1"
        PRINT "A két szám összege: "; a+b
    CASE "2"
        PRINT "A két szám különbsége: "; a-b
    CASE "3"
        PRINT "A két szám szorzata: "; a*b
    CASE "4"
        IF b=0 THEN
            PRINT "Nullával nem lehet osztani!!"
        ELSE
            PRINT "A két szám hányadosa: "; a/b
        END IF
END SELECT
```

Ez a legegyszerűbb menüséma, a lényegét kell megtanulni ez alapján. A lenyomott billentyűt egyszerűbb bekérni, mint egy számot, ezért használunk szöveges változót (c$), azonfelül ez a változat mentes a típushibáktól (számot várunk, de egy betűt kapunk), illetve általánosabb – hiszen lehet, hogy nem egy sorszám alapján szeretnénk elágaztatni.

A fenti példa nem volt eléggé elegáns, hiszen a választást egyszer kínálja fel a program, majd a kiválasztott tevékenység végrehajtása után leáll. A valódi, menüt tartalmazó programokban mindig el kell helyeznünk egy „Kilépés" vagy hasonló értelmű menüpontot, és csak akkor szabad kilépni, ha ezt választjuk ki. Ellenkező esetben a menüt újra meg kell jeleníteni mindaddig, amíg csak a felhasználó nem választja a kilépést.

A megoldás az lesz, hogy a fenti menüsémát úgy ahogy van, egy hátultesztelős ciklus belsejébe helyezzük.

82. példa Módosítsuk az előző példaprogramot úgy, hogy a menüpontok közé a „Kilépés"-t is felvesszük!

Megoldás: Lássuk a tervet:

```
Be: a,b
Ciklus
        Menüpontok kiíratása (választási lehetőségek: 0-4 sorszámok; 0:kilépés)
        Ciklus
                Be: c$ (lenyomott billentyű)
        amíg c$<"0" vagy c$>"4"
        Ciklus vége
        Elágazás
                Ha c$="1" akkor Ki: a+b
                Ha c$="2" akkor Ki: a-b
                Ha c$="3" akkor Ki: a*b
                Ha c$="4" akkor
                        Ha b=0 akkor Ki: "Nullával nem lehet osztani!"
                                különben Ki: a/b
        Elágazás vége
        Várakozás egy billentyű lenyomására
amíg c$<>"0"
Ciklus vége
```

A programlista:

```
REM Bekérés:
CLS
INPUT "Kérem az első számot: ", a
INPUT "Kérem a második számot: ", b
DO
    REM Menüpontok kiíratása a képernyőre:
    CLS
    PRINT "1. Összeadás""
```

```
        PRINT "2. Kivonás"
        PRINT "3. Szorzás"
        PRINT "4. Osztás"
        PRINT "0. Kilépés"
        PRINT
        PRINT "Melyiket választod? "
        REM Várakozás egy billentyű lenyomására:
        DO
            c$ = INPUT$(1)
        LOOP UNTIL c$>="0" AND c$<="4"
        REM Elágazás:
        CLS
        SELECT CASE c$
            CASE "1"
                PRINT "A két szám összege: "; a+b
            CASE "2"
                PRINT "A két szám különbsége: "; a-b
            CASE "3"
                PRINT "A két szám szorzata: "; a*b
            CASE "4"
                IF b=0 THEN
                    PRINT "Nullával nem lehet oszta-
                          ni!!"
                ELSE
                    PRINT "A két szám hányadosa: "; a/b
                END IF
        END SELECT
        IF C$<>"0" THEN
        LOCATE 23, 1
        PRINT "Nyomj egy billentyűt a folytatáshoz"
        a$=INPUT$(1)
        ENDIF
    LOOP UNTIL c$="0"
```

Ez a programocska már egy komplett menüt készít, bármilyen programban alkalmazható.
A programot megírás után feltétlenül mentsük el, még szükségünk lesz rá!

Feladatok:

115. Készítse el a fenti programok szebbített változatait! A menü kerüljön nagyjából a képernyő közepére, alkalmazzon színeket, bővítse a kiíratásokat, stb.
116. Készítsen el egy valutaváltó programot! A menüben lehessen kiválasztani, hogy Ft-ról $-ra, vagy visszafelé szeretnénk-e átváltani! Az elágazás egy-egy ágán szerepeljen a két kis programocska, bekéréssel együtt (ezeket már előbb megírtuk)! Ne felejtse el lekezelni a kilépést is!

Menü jellegű programról beszélünk akkor is, ha nem íratunk ki a szó szoros értelmében vett menüpontokat, de más billentyű lenyomására más tevékenység történik. Persze a képernyő alján azért nem árt a lehetőségeket felsorolni...

83. példa Készítsünk programot, amely egy * karaktert mozgat a képernyőn! A mozgatást a J(obbra), B(alra), F(el) és L(e) billentyűkkel végezzük!

Megoldás: A megoldás során egy elágazással ugyanúgy le tudjuk kezelni az egyes eseteket, ahogyan azt a menünél láttuk. Szükségünk van két változóra, amely a * aktuális helyét jelzi (**sor**, **oszl**). Vigyáznunk kell arra, hogy a * előző helyéről eltűnjön, és egy másik helyre kerül. Mozgatni pedig csak akkor tudunk, ha a képernyőről nem visszük ki a *-ot. A programból a K(ilépés) billentyűvel lehessen kilépni. Ennél a programnál már csak a tervet készítjük el közösen, a program megírását az olvasóra bízom (persze a lemezmellékleten ez a program is megtalálható).

```
Ki: képernyő aljára a „menüpontok"
sor = 12: oszl = 40 (alapbeállítások a képernyő közepére)
Ki (sor, oszl): „*"
Ciklus
        Be: c$
        Ki (sor, oszl):" „ (szóköz kiíratása a csillag előző helyére)
        Elágazás
        Ha c$="b" vagy c$="B" akkor
                Ha oszl>1 akkor oszl=oszl-1   (ha nem az első oszlopban vagyunk,
                                               akkor az oszl változó értékét
                                               1-gyel csökkentjük)
        Ha c$="j" vagy c$="J" akkor
                Ha oszl<80 akkor oszl=oszl+1  (ha nem az utolsó oszlopban vagyunk,
                                               akkor az oszl változó értékét 1-gyel
                                               növeljük)
        Ha c$="f" vagy c$="F" akkor
                Ha sor>1 akkor sor=sor-1      (ha nem az első sorban vagyunk, akkor az
                                               sor változó értékét 1-gyel csökkentjük)
        Ha c$="l" vagy c$="L" akkor
                Ha sor<22 akkor sor=sor+1     (ha nem az utolsó sorban vagyunk, akkor
                                               az sor változó értékét 1-gyel növeljük)
        Elágazás vége
        Ki (sor, oszl): "*"
amíg c$<>"K"
Ciklus vége
Program vége
```

*Tehát először kiíratjuk a képernyő közepére (12,40) a *-ot. Majd nyitunk egy ciklust, amely egészen addig fut, amíg csak a „K" betűt nem nyomjuk le. Ebben a ciklusban először várakozunk egy*

*lenyomott billentyűre. Ha megkaptuk, akkor 1. a * helyére kiíratunk egy szóközt, hogy eltűnjön, hiszen ez a „régi" helye; 2. a lenyomott billentyű alapján megváltoztatjuk a* ***sor*** *és* ***oszl*** *változók értékeit (persze figyelemmel a képernyő széleire); 3. az új helyre (ezt mutatja a két változó) kiíratjuk a *-ot. Az egyes esetek szétválasztásakor mind a nagy-, mind a kisbetűkre figyelemmel kell lenni! Ha bármely más billentyűt nyomunk le, akkor csak az történik, hogy 1. a * eltűnik; 2. a két változó értéke nem változik; 3. a *-ot újra kiírjuk változatlan helyre. Mivel ez eléggé gyors már egy régebbi gépen is, ezért felesleges lenne külön lekezelni, de mindig meg kell gondolni, hogy szükséges-e valamilyen egyéb tevékenység erre az esetre. Próbálja ki, hogy mi történik, ha a szóköz kiíratását a programból kivesszük!*

Feladatok:

117. Próbálja meg átírni ezt a kis programot úgy, hogy a képernyő széléhez érve a *-ot a másik oldalon belépteti (pl. ha már elért a * a jobb szélre, akkor a J gomb lenyomására a bal szélen jelenjen meg)!
118. Próbálja meg bővíteni a programot úgy, hogy a S(zín) billentyű lenyomására a csillag színe megváltozzon! (Olyan változatok is elképzelhetők, amelyekben egy billentyű lenyomására megváltozik a kirajzolt karakter (pl. #, %, +, stb.), illetve egy adott billentyűre megváltozik a lépésköz (nem egyesével, hanem kettesével vagy hármasával mozog a *). Ha van kedve hozzá, érdemes kísérletezni velük!)
119. Egészítse ki az alapműveleteket tartalmazó példát egy Adatbekérés nevű menüponttal! Itt lehessen az adatokat megadni. Ügyeljen arra, hogy a többi menüpont csak akkor legyen választható, ha már megtörtént az adatbekérés!

2.14. Alprogramok

Az alprogramok a programtól függetlenül megírt, de ott felhasznált, önálló programrészek. Akkor alkalmazhatók jól, ha

- a főprogram egy részét, amely önállóan is megállja a helyét, az áttekinthetőség kedvéért alprogramban írjuk meg, vagy

- olyan programrészeknél, amik sokszor ismétlődnek, és a többszöri megírás helyett az adott részt csak egyszer készítjük el, és többször meghívjuk.

Az alprogramok kétfélék lehetnek: szubrutinok és függvények.

2.14.1. Szubrutinok

A szubrutinok a Basic nyelvre jellemző alprogramok. Egy-egy alprogram elkészítése 3 lépésből áll:

1. Az alprogram megírása
2. Az alprogram ún. deklarálása, ami a program elején egy megjegyzés: jelzés a Basic rendszernek, hogy van alprogram ilyen és ilyen néven, mert a végrehajtás során csak így fogja megtalálni
3. Az alprogram megfelelő helyről való meghívása, vagyis lefuttatása

Egy szubrutin általános szerkezete a következő:

```
SUB alprogram_azonosító
    az alprogramot alkotó utasítás(ok)
END SUB
```

Az alprogramo(ka)t a program végén kell elhelyezni.

A kész alprogramot deklarálni kell, ennek érdekében a következő sort kell a program elején (vagy nem az elején, de mindenképpen az adott alprogram felhasználása előtt) elhelyezni:

```
DECLARE SUB alprogram_azonosító ()
```

Az alprogramokat a nevükkel tudjuk lefuttatni, vagyis a programban ott, ahol az adott szubrutint szeretnénk lefuttatni, az alprogram nevét kell szerepeltetni.

A Quick Basic-ben a szubrutinok készítése és deklarációja automatizált, lépéseit a következő példából ismerhetjük meg.

84. példa Az alapműveleteket végző menüs programunkat átalakítjuk alprogramosra, amelyben egy alprogramba tesszük a menüpontok kiíratását.

Megoldás: Nyissuk meg az elmentett menüt tartalmazó programunkat! Első lépésben elkészítjük az új alprogramot. Ehhez válasszuk az **Edit** menü **New SUB...** menüpontját! A megjelenő ablakba írjuk be az alprogram

nevét, legyen ez MenuKiir. (Az alprogramok azonosítójára ugyanazon szabályok érvényesek, mint a változók neveire.) Erre megjelent egy ablak, amelynek a fejlécében a láthatjuk a program nevét, majd egy kettősponttal elválasztva a szubrutin nevét. Az ablakban a szubrutin fejét, és lezárását már ott találjuk, a kettő közé kell beírnunk azokat az utasításokat, amelyek a szubrutint felépítik. Ezt már megírtuk egyszer, tehát csak át kell tenni ide a főprogramból a vágólap segítségével: A **View** menü **SUBs...** menüpontját kiválasztva vissza tudunk térni a főprogramba. Jelöljük ki a `REM Menüpontok kiíratása:` sortól a `PRINT "Melyiket választod?"` sorig terjedő programrészt, ezt fogjuk áttenni az alprogramba. Az **Edit** menü **Cut** menüpontját választva a programrészt kitesszük a vágólapra, majd a **View/SUBs...** menüpont segítségével átmegyünk a szubrutinba. A két már ott lévő sor közé állva az **Edit** menü **Paste** menüpontjával a vágólapra tett szöveget beillesztjük az alprogramba. Ezzel az alprogram létrejött. (Természetesen egy újonnan létrehozott alprogramba általában be kell gépelni a sorokat, de a már megírt programrészt kár lenne újra gépelni.) Most térjünk vissza a főprogramba! Ahhoz, hogy a szubrutint le is tudjuk futtatni, ahhoz a programnak arra a részére, ahonnak kivettük a szöveget – a DO utáni sorba – beírjuk az alprogram nevét: `MenuKiir`. Ezzel készen vagyunk, futtassuk le a programot! Ha mindent jól csináltunk, akkor a program ugyanúgy fut le, ahogyan eddig. A futtatás után mentsük le a programot, és a mentés után látható a program első sorában a deklaráció: `DECLARE SUB MenuKiir()`. Ezt tehát a Quick Basic automatikusan elvégzi helyettünk. És most lássuk a teljes programlistát (a változások félkövérrel szedetten láthatók):

```
DECLARE SUB MenuKiir ()
REM Bekérés:
CLS
INPUT "Kérem az első számot: ", a
INPUT "Kérem a második számot: ", b
```

```
DO
    MenuKiir
    REM Várakozás egy billentyű lenyomására:
    DO
        c$ = INPUT$(1)
    LOOP UNTIL c$>="0" AND c$<="4"
    REM Elágazás:
    CLS
    SELECT CASE c$
        CASE "1"
            PRINT "A két szám összege: "; a+b
        CASE "2"
            PRINT "A két szám különbsége: "; a-b
        CASE "3"
            PRINT "A két szám szorzata: "; a*b
        CASE "4"
            IF b=0 THEN
                PRINT "Nullával nem lehet oszta-
                      ni!!"
            ELSE
                PRINT "A két szám hányadosa: "; a/b
            END IF
    END SELECT
    IF c$<>"0" THEN
    LOCATE 23, 1
    PRINT "Nyomj egy billentyűt a folytatáshoz"
    a$=INPUT$(1)
    ENDIF
LOOP UNTIL c$="0"

SUB MenuKiir
    REM Menüpontok kiíratása a képernyőre:
    CLS
    PRINT "1. Összeadás"
    PRINT "2. Kivonás"
    PRINT "3. Szorzás"
    PRINT "4. Osztás"
    PRINT "0. Kilépés"
    PRINT
    PRINT "Melyiket választod? "
END SUB
```

A főprogram ezzel áttekinthetőbbé vált, hiszen rövidebb. (Hosszabb és bonyolultabb programok főprogramjai sokszor semmi mást nem tartalmaznak, csak alprogramok hívását.)

A szubrutinok nagyon gyakran a főprogram bizonyos adataival dolgoznak. Ehhez a szubrutin hívásakor az alprogram részére át kell adni

azokat a paramétereket, amelyekkel dolgoznia kell. A szubrutin fejében zárójelek között fel kell sorolni azokat a változókat, amelyek értékeivel a szubrutin dolgozni fog – ezeket hívjuk formális paramétereknek, a szubrutin hívásakor pedig át kell adni ezt a paramétert.

85. példa Írjuk tele a képernyő első 20 sorát # karakterekkel úgy, hogy egy sor kiírását szubrutin végezze!

Megoldás: Figyeljük meg a megoldást:

```
DECLARE SUB SorKiir (sor)
CLS
FOR i = 1 TO 20
    SorKiir(i)
NEXT i

SUB SorKiir (sor)
FOR oszl = 1 TO 80
    LOCATE sor, oszl
    PRINT "@"
NEXT OSZL
END SUB
```

A szubrutin egy sor kiíratását végzi el, formális paramétere a ***sor*** *változó. A főprogramban semmi más nincs, mint hogy 20-szor ki kell írni egy teljes sort, mégpedig mindig az* ***i****-ediket. Ez az* ***i*** *változó tehát az, ami paraméterként a SorKiir nevű szubrutinba átkerül. Konkrétan a következő történik a szubrutin hívásakor: a főprogram* ***i*** *változójának értékét átveszi a szubrutin* ***sor*** *nevű változója, majd ezzel az értékkel számol tovább.*

Egy alprogramnak egyszerre több paramétere is lehet, ekkor ezeket egymástól vesszővel elválasztva fel kel sorolni. A szubrutin hívásakor viszont ügyelni kell arra, hogy egyrészt a paraméterátadásban ugyanannyi paramétert kell átadni, mint az alprogram formális paramétereinek a száma, másrészt ezt ugyanolyan sorrendben is kel megtenni.

Az alprogramban alkalmazott változók és formális paraméterek lokálisak, ami azt jelenti, hogy csak abban az alprogramban van értékük. Ugyanaz a változó előfordulhat a főprogramban is és az alprogramban is, mindkét helyen megtartva az ő eredeti értékét (például a fenti példaprogram szubrutinjában szereplő **oszl** ciklusváltozó helyett alkalmazhattuk volna újra az **i** változót is, ez a program működése szempontjából nem jelentene változást, de furcsa és zavaró lenne). A paraméterek eme furcsa sajátosságát a következő programban tanulmá-

nyozhatja (a programocska semmi érdemlegeset nem csinál, csak ezt hivatott bemutatni).

86. példa Gépelje be a következő programot (vagy nyissa meg a lemezről), és futtassa le. Figyelje meg az **a** és **b** változók értékeit, kövesse végig figyelmesen a programot

```
DECLARE SUB Alprogram1 ()
DECLARE SUB Alprogram2 ()
CLS
a = 5
b = 9
PRINT "                a   b"
PRINT "  Főprogram: "; a; b
Alprogram1
PRINT "  Főprogram: "; a; b
a = a + b
PRINT "  Főprogram: "; a; b
Alprogram2
PRINT "  Főprogram: "; a; b

SUB Alprogram1
a = 12
b = -450
PRINT "Alprogram 1: "; a; b
END SUB

SUB Alprogram2
a = b
PRINT "Alprogram2: "; a; b
END SUB
```

Feladatok:

120. Az első alprogramos példát módosítsa úgy, hogy az adatbekérés is alprogramban legyen! Majd gyakorlásképpen az elágazás egyes ágain szereplő műveleteket is tegye egy-egy alprogramba!

2.14.2. Függvények

Az alprogramok másik típusa a függvény. A függvények abban különböznek a szubrutinoktól, hogy mindig van egy visszaadott érték; az, amit a függvény számol. A függvények szervezése pontosan ugyanúgy megy végbe, mint a szubrutinoké, csak a **SUB** kulcsszó helyett a **FUNCTION** kulcsszót kell alkalmazni mind a függvénydefinícióban,

mind a deklaráció során. A másik – de annál lényegesebb – különbség pedig az, hogy a függvény definíciójában szerepelnie kell egy **Függvénynév=visszaadott érték** sornak.

87. példa Kérjünk be két tetszőleges számot, és írassuk ki a nagyobbat! A nagyobbik szám megadásához használjunk függvényt!

Megoldás: Készítünk egy Max nevű függvényt két bemenő paraméterrel. A kimenő érték a nagyobb szám lesz...

```
DECLARE FUNCTION Max (x,y)

CLS
INPUT "Mondj egy számot! ", a
INPUT "Mondj egy másikat is! ", b
nagyobb = Max(a,b)
PRINT "A nagyobbik szám: "; nagyobb

FUNCTION Max (x,y)
    IF x>=y THEN Max=x ELSE Max=y
END FUNCTION
```

Az általunk elkészített függvények – ugyanúgy, ahogyan a Quick Basic saját függvényei – nem alkothatnak önálló programsort, hanem csak egy értékadás jobb oldalán, vagy egy kiíratásban szerepelhetnek!

Egyes egyszerűbb függvényeket elkészíthetünk a DEF FN kulcsszó segítségével is. Alakja a következő:

DEF FNfüggvénynév (paraméterlista) = kimenő érték

vagy

DEF FNFüggvénynév (paraméterlista)
    utasítás(ok)
    Függvénynév=kimenő érték
END DEF

A DEF FN -nel szervezett függvényt ugyanúgy kell használni, mint a FUNCTION kulcsszóval szervezettet, de itt nem kell a függvényt deklarálni, a függvény megadása egyben deklarációt is jelent. Egyetlen fontos szabály: a függvény megadásának meg kell előznie a függvény hívását (a gyakorlatban az összes függvénydefiníció a program legelején szokott szerepelni). Fontos, hogy a függvénynév közvetlenül az FN után íródik (szóköz nélkül), és így az FN a függvény nevének része lesz!

88. példa Készítsük el a fenti példának a DEF FN -nel szervezett változatát!

Megoldás:

```
DEF FNMax (x,y)
    IF x>=y THEN FNMax=x ELSE FNMax=y
END DEF

CLS
INPUT "Mondj egy számot!", a
INPUT "Mondj egy másikat is!", b
nagyobb = FNMax(a,b)
PRINT "A nagyobbik szám: "; nagyobb
```

Feladatok:

121. Készítsen programot, amelyben függvények segítségével számolja ki egy téglalap kerületét és területét! Próbálja ki mindkét (FUNCTION, DEF FN) módszert!
122. Készítsen függvényt, amely átvált mondjuk dollárt forintra! A függvény bemenő paraméterei legyenek a dollárösszeg, illetve az árfolyam, kimenő értéke a forintösszeg!
123. Készítsen programot, amely egy függvény segítségével kiszámítja egy bekért szám négyzetét!

2.15. Grafika a Quick BASIC-BEN

Az operációs rendszer kétféle képernyőt képes kezelni: a karakteres v. szöveges, illetve a grafikus képernyőt. Az előbbivel az eddigi fejezetek során már megismerkedtünk; jellemzője, hogy – alaphelyzetben – 25 sorban és 80 oszlopban tudunk karaktereket elhelyezni rajta. A képernyőt tehát csak kis téglalaponként (karakterhelyenként) tudjuk kezelni, képpontonként nem. Ezt a grafikus képernyőn tehetjük meg. A grafikus képernyőn tehát már tetszőleges képpontot (pixelt) is programozhatunk.

A képpontok száma a képernyőn, illetve a használható színek száma a grafikus kártya lehetőségeitől függ. Néhány fontosabb típust az alábbiakban bemutatok:

CGA	320*200 képpont 4 szín
EGA	640*350 képpont 16 szín
VGA	640*480 képpont 16 szín
Hercules	720*348 képpont monochrom

A ma elterjedt videokártyák már legnagyobbrészt VGA vagy SVGA kártyák (ez utóbbi lehetővé tesz 800*600, 1024*768, 1280*1024, stb. felbontásokat, illetve 256, 64 millió, stb. szín alkalmazását), ezért a továbbiakban a grafikát a VGA szabvány alkalmazásával fogom bemutatni. (A Quick Basic – és általában a magas szintű programnyelvek – természetesen lehetővé teszik egyéb grafikus képernyők kezelését is. Bizonyos utasítások eltérően működnek a más-más grafikus módok használata mellett.)

A VGA képernyő tehát 480 pixelsort és 640 oszlopot tartalmaz. Az origó a képernyő bal felső sarka, ez a (0;0) pozíció. A sorok és oszlopok számozása nullától kezdődik, tehát a sorok a 0-479, míg az oszlopok a 0-639 tartományba eshetnek. Az oszlopok koordinátáját az x, a sorokét az y jelöli általában. Egy karakter egy 8*16 képpontból álló téglalapba kerülhet, ezekből 30 sor és 80 oszlop áll össze.

Amennyiben grafikával szeretnénk dolgozni, az első teendőnk az, hogy áttérjünk a grafikus képernyőre. Erre szolgál a SCREEN utasítás.

SCREEN képernyőmód

a lehetséges képernyőmódokat az alábbi táblázat mutatja:

Képernyőmód	Grafikus kártya	Felbontás
0	minden	Szöveges mód
1	CGA, MCGA, EGA, VGA	320 x 200
2	CGA, MCGA, EGA, VGA	640 x 200
3	Hercules (monochrom)	720 x 348
4	Olivetti, AT&T	640 x 400
7	EGA, VGA	320 x 200
8	EGA, VGA	640 x 200
9	EGA, VGA	640 x 350
10	EGA, VGA (monochrom)	640 x 350
11	MCGA, VGA (monochrom)	640 x 480
12	VGA	640 x 480
13	MCGA, VGA	320 x 200

A VGA monitorokhoz a 12 kód tartozik, tehát a `SCREEN 12` utasítást kell kiadni minden olyan program elején, amelyben a grafikus lehetőségeket szeretnénk kihasználni. Visszatérni a szöveges módhoz a SCREEN 0 utasítással lehet. A SCREEN utasítás mindig törli is a képernyőt!

Az alábbi táblázat a legfontosabb grafikus módban használható utasításokat tartalmazza:

Leírás	Utasítás / Függvény
Képernyőtörlés	**CLS**
Írás a képernyőre	**PRINT** szöveg
A kiíratás helyének meghatározása	**LOCATE** sor, oszlop
Szín megadása	**COLOR** szín
Egy képpont kigyújtása	**PSET** (x,y),szín
Egy képpont eloltása	**PRESET** (x,y),szín
Egy képpont színének lekérdezése	**POINT** (x,y) (Függvény)
Vonal rajzolása	**LINE** (x1,y1)-(x2,y2),szín
Téglalap rajzolása	**LINE** (x1,y1)-(x2,y2),szín,B
Kitöltött téglalap rajzolása	**LINE** (x1,y1)-(x2,y2),szín,BF
Kör rajzolása	**CIRCLE** (x,y),sugár,szín
Körcikk rajzolása	**CIRCLE** (x,y),sugár,szín,kezdőszög,végszög
Ellipszis rajzolása	**CIRCLE** (x,y),nagytengely,szín,,,tengelyek aránya
Képernyőterület feltöltése színnel	**PAINT** (x,y),szín,keret

89. példa Írassuk ki a képernyő közepére a nevünket zölddel, majd keretezzük be pirossal!

Megoldás: A programot a SCREEN utasítással kell kezdenünk. A kiíratás a hagyományos módon megy. A keretezés egy téglalap rajzolása lesz.

```
SCREEN 12
COLOR 10
LOCATE 15, 34
PRINT "Zseni Alfonz"
LINE (259,221)-(363,240), 12, B
```

*A kiíratással különösebb probléma nincsen, csak az utolsó sor szorul magyarázatra. A kiírt szöveg bal felső sarka a (33*8+1; 14*16+1) koordinátákon van (a 34. oszlop előtt, illetve a 15. sor előtt). Ahhoz, hogy a keret szép legyen, valamennyit ki kell vonni ezekből az értékekből (259=33*8-5; 221=14*14-3). A kiírt szöveg jobb alsó sarkának x koordinátáját úgy kapjuk meg, hogy a bal felső sarok x koordinátájához (264) hozzáadunk annyiszor 8-*

*at, ahány karakter van a szövegben (363=33*8+12*8+3). Az y koordinátához csak a 15*16 szorzat eredményére (240) van szükségünk (a 15. sor alsó képpontsora). A LINE utasításban a 12 a szín kódja, a B pedig a téglalapot jelenti (Box).*

90. példa Készítsünk a képernyőre sakktáblát fehérrel és kékkel színezve!

Megoldás: A sakktábla 8*8 mezőt tartalmaz. A feladatot két lépésre bontjuk: először megvonalazzuk kékkel (9 vízszintes és 9 függőleges vonalat kell rajzolni egyenlő távolságokra egymástól), majd minden mezőt kitöltünk a megfelelő színűre.

```
SCREEN 12

FOR y = 120 TO 360 STEP 30
    LINE (200, y)-(440, y), 1
NEXT y

FOR x = 200 TO 440 STEP 30
    LINE (x, 120)-(x, 360), 1
NEXT x

FOR y = 1 TO 8
    FOR x = 1 TO 8
        a = (x + y) / 2
        IF a = INT(a) THEN szin = 15 ELSE szin = 3
        PAINT (185 + x * 30, 105 + y * 30), szin, 1
    NEXT x
NEXT y
```

Az első ciklus húzza meg a 9 vízszintes vonalat, a második ciklus pedig a 9 függőlegeset. Minden kocka 30 képpont nagyságú. A harmadik ciklus (amely magába zár egy negyediket) valósítja meg a színezést. Itt a két egymásba ágyazott ciklus végigmegy a sakktábla mind a 64 mezőjén, mégpedig sorfolytonosan. Az első lépésben eldöntjük, hogy milyen színűre kell kitölteni az x. sor y. oszlopának mezőjét, majd a PAINT utasítással kitöltjük. A PAINT paraméterei a következők: egy belső pont két koordinátájával megadva, a kitöltőszín, és végül a kitöltendő terület határoló vonalainak színe. Annak eldöntésére, hogy éppen milyen színnel kell kitölteni, kihasználjuk azt a tényt, hogy a sakktáblánkon fehér minden olyan mező, amelynek mind sor-, mind oszlopszáma azonosan páros vagy páratlan (azonos parosságúak). Az azonos parosságú számokat összeadva biztos, hogy páros számot ka-

*punk, különböző párosságú számok összege pedig mindig páratlan. Ezt alkalmazzuk a következőképpen: az **a** változó tartalmazza a két szám összegének a felét. Ha a két szám összege páros, akkor az **a** értéke egész, különben tört. Ezt vizsgáljuk az IF elágazásban, egész pontosan azt, hogy az **a** értéke megegyezik-e a saját egészrészével (azaz egész szám-e). Ha egész, akkor az **x+y** páros volt, vagyis **x** és **y** azonos párosságúak, tehát fehérrel kell színezni, ellenkező esetben kékkel. (A programnak ezt a részét természetesen más – talán egyszerűbb – módszerekkel is el lehet készíteni, például több ciklus alkalmazásával.)*

91. példa Rajzoljunk a képernyőre egy vigyorgó fejet!

Megoldás: A fej és a szemek körök, az orr ellipszis, a száj egy félkör.

```
SCREEN 12
pi = 3.14159
CIRCLE (320,240), 200
CIRCLE (250,180), 40
CIRCLE (390,180), 40
CIRCLE (320,250), 25, , , , 1.5
CIRCLE (320,250), 150, , pi, 2*pi
```

A kör rajzolásánál a középpontot (annak x és y koordinátáját), és a sugarat kell megadni. Körívnél a kezdő és végszöget radiánban kell megadni (a kezdőirány, tehát a 0° az óra 3-asának irányába esik, a szög forgási iránya az óramutató járásával ellentétes). Ellipszisnél a kis- és nagytengely arányát adhatjuk meg, mégpedig úgy, hogy a megadott sugár a nagytengely lesz, a kistengelyt az arány alapján számítja a gép. Ha az arány 1-nél nagyobb, akkor az ellipszis áll; ha 1-nél kisebb, akkor fekszik. A CIRCLE utasítás negyedik paramétere (a sugár után) a szín lenne, de ez most minden sorból kimaradt.

92. példa A gázmolekulák rendezetlen mozgását Brown-féle mozgásnak nevezik. Készítsünk programot, amely ezt a rendezetlen mozgást mutatja be először egyetlen molekulával (szimuláció)!

Megoldás: A molekulának egy képpont fog megfelelni, a helyét két változóban (mx, my) tároljuk. Első lépésben a molekula helyét beállítjuk a képernyő közepére (320,240). Ezután következhet a mozgatás. Egy lépésben a molekula a körülötte lévő 8 képpont valamelyi-

kére kerülhet, azaz mind az x, mind az y koordinátáját véletlenszerűen növeljük a -1, 0, 1 értékek valamelyikével. Mivel a molekula egyetlen kirajzolt helyzetét sem töröljük, ezért a teljes útvonal látható lesz! A szimuláció mindaddig fut, amíg egy billentyű lenyomásával meg nem állítjuk.

```
RANDOMIZE TIMER
SCREEN 12
mx = 320 : my = 240
COLOR 14
DO
    mx = mx + INT(RND*3)-1
    my = my + INT(RND*3)-1
    PSET (mx, my)
LOOP UNTIL INKEY$<>""
```

Feladatok:

124. Próbálja módosítani a fenti kis szimulációt úgy, hogy a molekula ne mehessen ki a képernyőről!
125. Módosítsa a fenti szimulációs feladatot úgy, hogy 20 db molekulát mozgat egyszrre! (Ötlet: a molekulák adatait egy 20*2-esre d imenzionált tömbben tárolja, és minden mozgatás előtt törölje le az aktuális molekulát a PRESET utasítással.)
126. Rajzoljon a képernyőre minél szebb olimpiai ötkarikát!
127. Rajzoljon a képernyőre egy házikót! Alkalmazzon benne minél több geometriai idomot (vonal, téglalap, kitöltött téglalap, kör, ellipszis), használjon színeket!
128. Készítsen egy kis rajzprogramot a *-ot mozgató programocska alapján! Próbáljon minél több funkciót beépíteni!

3. Mellékletek

3.1. ASCII kódtábla

32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	□ 127
Ç 128	ü 129	é 130	â 131	ä 132	à 133	å 134	ç 135	ê 136	ë 137	è 138	ï 139	î 140	ì 141	Ä 142	Å 143
É 144	æ 145	Æ 146	ô 147	ö 148	ò 149	û 150	ù 151	ÿ 152	Ö 153	Ü 154	¢ 155	£ 156	¥ 157	₧ 158	ƒ 159
160	í 161	ó 162	ú 163	ñ 164	Ñ 165	ª 166	º 167	¿ 168	⌐ 169	¬ 170	½ 171	¼ 172	¡ 173	« 174	» 175
░ 176	▒ 177	▓ 178	│ 179	┤ 180	╡ 181	╢ 182	╖ 183	╕ 184	╣ 185	║ 186	╗ 187	╝ 188	╜ 189	╛ 190	┐ 191
└ 192	┴ 193	┬ 194	├ 195	─ 196	┼ 197	╞ 198	╟ 199	╚ 200	╔ 201	╩ 202	╦ 203	╠ 204	═ 205	╬ 206	╧ 207
╨ 208	╤ 209	╥ 210	╙ 211	╘ 212	╒ 213	╓ 214	╫ 215	╪ 216	┘ 217	┌ 218	▉ 219	▄ 220	▌ 221	▐ 222	▀ 223
α 224	ß 225	Γ 226	π 227	∑ 228	σ 229	µ 230	τ 231	Φ 232	Θ 233	Ω 234	δ 235	∞ 236	ø 237	ε 238	∩ 239
≡ 240	± 241	≥ 242	≤ 243	⌠ 244	⌡ 245	÷ 246	≈ 247	° 248	• 249	· 250	√ 251	ⁿ 252	² 253	■ 254	□ 255

3.2. A könyvben szereplő Quick BASIC utasítások és függvények jegyzéke

Megnevezés	Típus*	Leírás	Oldalszám
ABS()	F	Szám abszolút értéke	83
ASC()	F	karakter ASCII kódját adja	91
BEEP	U	Hangjelzés	75
CASE	U	Többfelé elágazásnál az egyes esetek megadása	70
CHR$()	F	ASCII kód alapján karaktert ad	91
CINT()	F	Egész számmá alakítás kerekítéssel	85
CIRCLE	U	Kör vagy ellipszis rajzolása	122
CLS	U	Képernyőtörlés	40, 120
COLOR	U	Aktuális karakterszín és háttérszín megadása	42, 120
COS()	F	Radiánban adott szög koszinuszának számítása	86
DATA	U	Állandó (konstans) adatok tárolása	94
DECLARE	U	Alprogram deklarálása	112
DEF FN	U	Függvény definiálása	117
DIM	U	Tömb dimenzionálása	96
DO	U	Tesztelős ciklus kezdete	77, 78
ELSE	U	Elágazás HAMIS ágának megadása	67
ELSEIF	U	Elágazás HAMIS ágán újabb elágazás	70
END DEF	U	Függvény vége	117
END FUNCTION	U	Függvény vége	116
END IF	U	Elágazás vége	68
END SUB	U	Szubrutin vége	112
ERASE	U	Tömb törlése a memóriából	96
EXP()	F	Exponenciális függvény	83
END SELECT	U	Többfelé elágazás lezárása	70
FIX()	F	Egész számmá alakítás csonkolással	85
FOR	U	Számlálós ciklus kezdete	53

* F: függvény U: utasítás

Megnevezés	Típus*	Leírás	Oldal-szám
FUNCTION	U	Függvény megadása	116
GOTO	U	Vezérlés átadása egy cimkével megjelölt programsorra	72
IF	U	Elágazás kezdete	66
INKEY$	F	Egy karakter beolvasása a billentyűzetről	104
INPUT	U	Adat beolvasása a billentyűzetről	50
INPUT$()	F	Adott darabszámú karakter beolvasása a billentyűzetről	74
INSTR()	F	Szövegrész keresése szövegben	88
INT()	F	Egész számmá alakítás – egészrész fügvény	85
LEFT$()	F	Szöveg bal oldalát adja	89
LCASE$()	F	Szöveget kisbetűssé alakít	90
LEN()	F	Szöveg hosszát adja	87
LET	U	Értékadás változónak	48
LINE	U	Vonal vagy téglalap rajzolása	120
LOCATE	U	Kurzor pozícionálása a képernyőn	40, 120
LOG()	F	Logaritmus függvény	83
LOOP	U	Tesztelős ciklus lezárása	77, 78
MID$()	F	Szöveg egy részét adja	88
NEXT	U	Számlálós ciklus vége	53
PAINT	U	Képernyőterület feltöltése adoot színnel	120
POINT()	F	Képpont színének lekérdezése	120
PRINT	U	Adatok képernyőre írása	39, 120
PRINT USING	U	Kiíratás beállított formátummal	84
PRESET	U	Képpont kioltása	120
PSET	U	Képpont kigyújtása	123
RAMDOMIZE	U	Véletlenszám-generátor kezdőérték beállítása	101
READ	U	DATA sor(ok)ból adatok kiolvasása	94
REDIM	U	Tömb újradimenzionálása	96
REM	U	Megjegyzések elhelyezése a programban	97
RESTORE	U	DATA sor(ok)ban tárolt adatok ismételt használatát teszi lehetővé	95
RIGHT$()	F	Szöveg jobb oldalát adja	89
RND()	F	Véletlenszám előállítása	99
SCREEN	U	Képernyőmód beállítása	119

Megnevezés	Típus*	Leírás	Oldalszám
SELECT CASE	U	Elágazás többfelé	70
SGN()	F	Előjelfüggvény (signum)	84
SIN()	F	Radiánban adott szög szinuszának számítása	86
SPACE$()	F	Szóközökből álló szöveg	92
SPC()	F	Szóközök kiíratása	87
SQR()	F	Szám négyzetgyöke	86
STEP	U	Számlálós ciklusban a lépésköz megadása	61
STR$()	F	Számot szöveggé alakít	93
SUB	U	Szubrutin megadása	112
SWAP	U	Változók értékének cseréje	99
TAN()	F	Radiánban adott szög tangensének számítása	86
THEN	U	Elágazás IGAZ ágának megadása	66
TIMER	F	Az éjfél óta eltelt másodpercek számát adja	101
UCASE$()	F	Szöveget nagybetűssé alakít	90
UNTIL	U	Tesztelős ciklus feltételének megadása	77, 78
VAL()	F	Szöveget számmá alakít	93
WHILE	U	Tesztelős ciklus feltételének megadása	77, 79
WIDTH	U	Képernyő méretének megváltoztatása	46

Jegyzet

Informatikai füzetek

A sorozat kötetei:

1. Alapismeretek
2. Operációs rendszerek
3. Kiegészítő ismeretek
4. Szövegszerkesztés
5. Táblázatkezelés
6. Adatbázis-kezelés
7. Programozás

Sorozatszerkesztő:
Bártfai Barnabás

ISSN 1418-8791

PROGRAMOZÁS

ISBN 963 03 5286 9

BBS-E Számítástechnikai és Könyvkiadó Betéti Társaság

Könyvsorozatunk segítségével alapszintről elindulva, a számítógép megvásárlásától és első bekapcsolásától kezdve a DOS parancsain és üzenetein át, a különböző felhasználói programok alkalmazásáig mindent könnyedén megtanulhat.

A leírtak tanfolyamok tapasztalataira épülve, gyakorlati példákat bemutatva segítenek elsajátítani a számítógép kezelését, megismerni részegységeinek használatát olymódon, hogy azt Ön a mindennapi munkájában is kamatoztathassa.

Kiadványaink a nagy sikerű Hogyan használjam? Című könyv alapján készültek, teljesen kezdő felhasználókhoz szólnak, s önálló tanulásnál is jól használhatóak. Továbbra is fontosnak tartottuk, hogy ne azt mutassuk meg, hogy egy adott programfunkció mire való, hanem azt, hogy egy adott feladatot milyen módon tudunk megoldani.