QBASIC

Írta: Csík Tibor

Tartalom Jegyzék

Előszó

Jelen írásomat főként azoknak a figyelmébe ajánlom akik még csak most ismerkednek meg a programozás rejtelmeivel. Sokan indítják úgy a könyvüket, hogy egy példa program után egyből bele a mély vízbe vagyis tömény elkezdik leírni a parancsokat, típusokat és függvényeket. Ennek azonban az a hátránya hogy akik nem megszállottak de azért szeretnék megismerni a programozást egyszerűen abba hadják és ezzel elesnek a lehetőségtől hogy valóban megismerjék és megtanulják a programozást.

Néhány dolog melyet érdemes tudni!

A Basic az első magasabb szintű programozási nyelvek közé tartozik, Sikerét az 1970-es évek végén szerezte amikor az IBM PC-ket a Microsoft Basic értelmezővel látták el. Utólag ezen értelmezők szerkezete vált standardá vagyis Szabványá. A Basic 2. sikerét az MS-DOS-ba való integrálása jelentette. Sikeréhez hozzá tartozott az akkoriban még elterjedt Assambly és fortan nyelvekhez képest egyszerű felépítése. Nem típusos nyelv mely azt jelenti hogy a különböző adat formákat a fordító konvertálta ezzel is csökkentve a programozó terhét. A 1980-as évek vége felé vége szakadt a sikerének. Ekkoriban a C nyelv kezdet elterjedni melynek utódai uralják napjainkban a piacot. Szintén Ismert a Pascal programozási nyelv, az ő utódai közé tartok az Object Pascal és a Delphi nyelv is. Napjainkban azonban kezd egyre erőteljesebben betörni a piacra a Sun Microsystem saját fejlesztésű, inkább C-re hasonlító fejlesztése, a JAVA.

Első programunk!

Az első programunk a klasszikus „Hello World” mely lényege az hogy a "Hello World!" szöveget írja ki a képernyőre.

KÓD:

```
PRINT „Hello World!”
```

Ehhez elegendő mindössze egyetlen parancsot kiadnunk a PRINT-et. A PRINT után Idézőjelek közé kell zárni a szöveget melyet ki akarunk íratni a képernyőre.

Szöveg színezése

A kirandó szövegeket a COLOR utasítással színezhetjük át. Két paramétere van melyeket szóközzel kell elválasztani. Az első a betűszín míg a második a szöveg hátterét határozza meg. Most a szövegünk Piros színű lesz míg a háttér sötét kék. A pirus kódja:12, a kéké:1.

KÓD:

```
COLOR 12,1
PRINT ''Hello World!''
```

Szöveg pozicionálása

Egy adott szöveget a LOCATE paranccsal lehet pozicionálni, Első paramétere a sorok- míg második paramétere az oszlopok számát határozza meg. De mielőtt ezt használnánk meg kell ismerkednünk a karakteres képernyővel. Alapvetően a képernyő az un. 0-dik módban van mely a karakteres felület. Ez 25 sorból és 80 Oszlopból áll. A sorok és az oszlopok számozása a bal felső sarokban kezdődik. A 25 sort nem szokás használni mert ezt a fordító tartja fent saját maga számára. A mégis használjuk akkor az ide írt szöveg eggyel fentebb ugrik és így az összes többi sor és a legelső nem lesz látható. A következő forrás azt eredményezi hogy a "hello word" a képernyő közepén jelenik meg.

KÓD:

```
COLOR 12,0
LOCATE 12,35
PRINT „HELLO WORLD!”
```

A Képernyő törlése

Sajnos a képernyőnk a kicsit zsúfolttá vált így hát valamilyen módon meg kell tisztítani. Ezt a Dosból is ismert CLS (Clear Screen) parancsal tehetjük. Ennek a használata érdekesebb hisz a COLOR beállításának megfelelően törli a képernyőt. Ha a háttér pirosnak van állítva akkor az egész képernyő piros lesz.

KÓD:

```
COLOR 15,0
CLS
```

Adatok olvasása a billentyűzetről

A Billentyűzetről az INPUT parancsal lehet adatokat beolvasni. Az olvasás akkor fejeződik be amikor a felhasználó ENTER-t üt. Az utasítás két paraméteres. Az első egy szöveg melyet kiír a program a második egy változó melybe beolvassuk az adatokat.

Változók:A változók a memóriában lefoglalt egy egy területet jelképeznek, melyen megadott mennyiségű adatott tudunk tárolni. Szöveges adatok tárolásához ún. String típust alkalmazunk mely 255 karakter hosszúságú szöveget jelent. Ezen változók jelölésére a $ karatert használjuk melyet a változónév után kell írni. A változónév lehet egy betű betűvel kezdődő számsorozat, vagy betűkből és számokból álló karaktersorozat. Erre a Célra csak az angol abc kis és nagy betűit lehet használni és számokat ill. a _ karkatert.

Kód:

```
CLS
INPUT „Add meg a neved:” , a$
PRINT „A te neved:”, a$
```

Műveletek változókkal.

A változókkal való müveletek elvégzése előtt meg kell ismerkedni az alapvető típusokkal.

STRING	$
Egész szám, INTEGER	%
Hosszú szám, LONG-INTEGER	&
Lebegő pontos szám, Single precision	!
Double Precision	#

Az integer típusú számok egész számok. Míg a lebegőpontos számok tizedes számok is lehetnek. Más nyelvekben REAL-nak vagy EXTENDED-nek ill DOUBLE-nek is hívják. A kettő közt még van egy fontos különbség. A lebegőpontos zámok elvileg nem érhetik el a 0-át. Csak megkölíthetik.

Műveletek:

Stringek:
+ : két szöveg összefűzése
= : két szöveg összehasonlítása. 0 ha nem egyezik és -1 ha igen
< : Két szöveg közül melyik a nagyobb. Az a nagyobb ahol eltérés van és a krakter hátrébb van az abc-ben
> : Az előző fordítva.
<> : Két szöveg nem egyezik. Ha legalább egy karater eltérés van köztük.

Számok:
= :a két szám egyezik-e
+ : két szám összege
- : két szám különbsége

> : Két szám közül a kisebb
< : két szám közül a nagyobb
<>: A két szám nem egyezik

Gyakorlat

Most néhány példa a váltózókkal való műveletekre:

KÓD:

```
CLS
PRINT „MÜVELETEK VÁLTOZÓKKAL”
a$=”abc”
b$=”def”
c$=a$+b$
d!=1
e!=2
PRINT a$
PRINT b$
PRINT c$
PRINT a$<b$
PRINT (a$+b$)=c$
PRINT d!
PRINT e!
PRINT d!+e!
```

Az eredmény:

```
MŰVELETEK VÁLTOZÓKKAL
abc
def
abcdef
0
-1
1
2
3
```

Elágazások a programban

A számítástechnikában szinte nincs is olyan program amelyben nem található elágazás. Az elágazás valójában egy olyan megoldás, melyben egy nem a programozó által beállított értéktől függ a program további kimenetele. A legegyszerűbb példa ha elakarunk menni valahova. Ez teljesen egyszerű akkor elindulunk, ezt nem változtatjuk mert ez biztos esemény. De mi van ha esik az eső, akkor esőkabátot veszünk és úgy indulunk el. Ez nekünk természetes de a számítógépnek nem. Neki elkel magyarázni hogy mit tegyen bizonyos helyzetekben.

Ha esik az eső akkor esőkabát kell, ellenkező esetben nem. Ezt a pc-nek az **IF-THEN-ELSE** szerkezettel magyarázhatjuk el. Felépítése:

IF (*feltétel*) **THEN**
(*utasítás*)
ELSE
(*utasítás ha feltétel nem igaz*)
END IF

A legjobban egy példán keresztül tudjuk megérteni. A következő kis kód bekér egy számot. Ha a szám kisebb 10-nél akkor kiírja a számot ellenkező esetben egy hiba üzenetett. Ám mielőtt neki fognánk meg kell ismerkednünk egy új függvénnyel amely a Szöveges adatokat számmá alakítja. Ez a **VAL(**„szöveg”**)**.

KÓD:

```
CLS
INPUT „Kérek egy számot, max:10 ”, c$
IF VAL(c$)<=10 THEN
  PRINT „A beírt szám:”;c$
ELSE
  PRINT „A beírt szám túl nagy!:”;c$
END IF
```

Ciklusok

sokszor találkozhatunk a mindennapi életben olyan dolgokkal melyet újból és újból meg kell csinálni. Jó példa erre amikor papírból ugyanazt az alakzatot kell kivágnunk. Ilyenkor nem rajzoljuk meg újra és újra. Elég egyszer és a többit már csak az elsőről másoljuk, mert azt mintának használjuk. A programozásban is valahogy így van.

Tehát a ciklus nem más mint egy adott folyamat megismétlése. A kérdés már csak az hogy hányszor kell ismételni. E szerint három ciklust különböztetünk meg.

A léptető ciklus (FOR ciklus)

Ezt a ciklust akkor használjuk ha tudjuk hogy hányszor kell megismételni az adott dolgot. Felépítése egy kicsit bonyolult de nem nagyon.

FOR *változó=kezdőérték* **TO** *végérték* **STEP** *lépések mértéke*
[*utasítás blokk*]
NEXT *változó*

A STEP azt határozza meg hogy hányasával hajtódjon végre a ciklus, pl ha a step után 2 van akkor minden 2 lépésben fut le az utasítás blokk. Ezt nem kötelező használni. A következő példa 1-50-ig kiírja számokat a képernyőre:

KÓD:

```
CLS
FOR i=1 TO 50
  PRINT i
NEXT i
```

Elől tesztelős ciklus

Az elől tesztelős ciklust akkor használjuk ha csak adott körülmények között szabad végrehajtódnia az utasításoknak. A következő példa csak akkor fut le ha a=2, de ha lefut akkor 'a' értéke 1lessz különben végtelen ciklus alakul ki.

KÓD:

```
CLS
a=2
DO WHILE a=2
  PRINT „a=2”
  a=1
LOOP
```

Szintén elől tesztelős ciklus a WHILE-WEND szerkezet. Használata nagyon hasonlít az első változatra.

KÓD:

```
CLS
a=2
WHILE a=2
  PRINT „a=2”
  a=1
WEND
```

Hátul tesztelős ciklus

A hátul tesztelős ciklust a benne lévő utasítás blokk eredménye vezérli. Ez addig tart míg a feltétel hamis.

KÓD:

```
CLS
DO
  INPUT „A kilépéshez írd be EXIT :”, c$
LOOP UNTIL c$=”exit”
```

Bonyolultabb példa program

Az eddigiek folyamán a legáltalánosabb dolgokat megtanultuk így már magunk is írhatunk kisebb programokat. Ez a fejezet egy egyszerű, könnyen használható, a négy alapművelet elvégzésére alkalmas számológép elkészítését mutatja be részletesen.

Az első lépésben egy kis egészítő eddigi tanulmányainkhoz. A basic rendelkezik egy **END** utasítással ami a program futásának befejezését jelenti. Ezt a program legvégére szokás tenni, de máshova is lehet de ezt majd később. Ha ezt kitesszük akkor az end után úgy **SUB** programokat vagyis alprogramokat is írhatunk. Ezt a ***név:*** módon kezdjük és a **RETURN** zárja le. Ezeket arra lehet használni hogy egy adott programrészt többször felhasználhassuk anélkül hogy újra írnánk.

Ezt a programban a **GOSUB** *név*: formában tudjuk elérni. Ez csak azért került ide mert így a számológép forrását több mint felére tudjuk csökkenteni, ez egy nagyobb program esetén több ezer sorral rövidítheti forrásunkat, és még átláthatóbbá is válhat.

A számológépet úgy alakítjuk ki hogy a lehető legkevesebb adatott keljen megadni. A műveleteket egy-egy szimbólum jelképezi majd. **Összeadás(+), Kivonás(-), Szorzás(*), Osztás(/)**.

A programot egyetlen hátul tesztelős ciklus vezérli mely akkor fejezi be futását ha a **c$** változónk értéke”**exit**”.

```
DO
  [parancsok]
LOOP UNTIL UCASE$(c$)=”EXIT”
```

Az **UCASE$(„szöveg”)** a megadott szöveget átalakítja nagybetűssé így mindegy hogy a felhasználó mily módon írja azt be.

Ahhoz a hogy egy idegen is tudja használni a programot szükségünk lesz egy jelmagyarázatra. Ezt a legfelső sorokba írjuk egy, a program forrásának végén meghatározott alprogram segítségével.

```
...
fejlec:
  CLS
  LOCATE 1
  PRINT "+--------------+-----------+-----------+----------+--------------+"
  PRINT "| osszeadas: + | Kivonas: - | Szorzas: * | Osztas: / | Kilepes: exit |"
  PRINT "+--------------+-----------+-----------+----------+--------------+"
RETURN
...
```

Ezt majd a **GOSUB** fejlec utasítással futtatjuk le. A **RETURN** pedig jelzi a programnak hogy az alprogram befejezte futását így visszatér ahhoz a ponthoz ahonnan hívtuk.
A fejléc kiírása után már csak az adatokat kell bekérnünk. Ezt az INPUT-tal végezzük.

```
CLS
DO
  GOSUB fejlec
  LOCATE 10
  INPUT "Add meg a muvelet tipusat:", c$
…
```

Most már szinte mindent tudunk, csak a számokat nem amelyekkel a műveleteket akarjuk végezni. Ezt megint egy alprogramban végezzük el, melynek neve: SZAMBEKERES.

```
szambekeres:
LOCATE 12, 25
INPUT "Add meg az elso szamot:", szam1$
LOCATE 13, 25
INPUT "Add meg a masodik szamot:", szam2$
RETURN
```

Most már tudjuk a számokat is. A feladatunk csak annyi hogy ezeket megjelenítjük, ill. kiszámítjuk a helyes megoldást.

A program kezdete:

```
CLS
DO
  GOSUB fejlec
  LOCATE 10
  INPUT "Add meg a muvelet tipusat:", c$
  IF c$ = "+" THEN
    GOSUB fejlec
    LOCATE 10, 30
    PRINT "osszeadas!"
    GOSUB szambekeres
    LOCATE 15, 25
    PRINT "Eredmeny:" + szam1$ + "+" + szam2$ + "="; VAL(szam1$)
+ VAL(szam2$)
    LOCATE 24
    INPUT "A folytatashoz nyomj ENTER-t", z$
  END IF
…
```

Most már kész is vagyunk. Lehet hogy kicsit zavaros így elsőre de ha jól áttanulmányozzuk akkor minden világossá válik. Egyébként ez meg nem egy teljesen tömör megoldás mert még mindig a felére lehetne csökkenteni a méretét. Ugyan letisztultabb lenne, de ez a kezdőket csak jobban összezavarná:

Most pedig a <u>teljes</u> forrás:

```
CLS
DO
  GOSUB fejlec
  LOCATE 10
  INPUT "Add meg a muvelet tipusat:", c$
  IF c$ = "+" THEN
    GOSUB fejlec
    LOCATE 10, 30
    PRINT "osszeadas!"
    GOSUB szambekeres
    LOCATE 15, 25
    PRINT "Eredmeny:" + szam1$ + "+" + szam2$ + "="; VAL(szam1$) + VAL(szam2$)
    LOCATE 24
    INPUT "A folytatashoz nyomj ENTER-t", z$
  END IF
  IF c$ = "-" THEN
    GOSUB fejlec
    LOCATE 10, 30
    PRINT "Kivonas!"
    GOSUB szambekeres
    LOCATE 15, 25
    PRINT "Eredmeny:" + szam1$ + "-" + szam2$ + "="; VAL(szam1$) - VAL(szam2$)
    LOCATE 24
    INPUT "A folytatashoz nyomj ENTER-t", z$
  END IF
  IF c$ = "*" THEN
    GOSUB fejlec
    LOCATE 10, 30
    PRINT "Szorzas!"
    GOSUB szambekeres
    LOCATE 15, 25
    PRINT "Eredmeny:" + szam1$ + "*" + szam2$ + "="; VAL(szam1$) * VAL(szam2$)
    LOCATE 24
    INPUT "A folytatashoz nyomj ENTER-t", z$
  END IF
  IF c$ = "/" THEN
    GOSUB fejlec
    LOCATE 10, 30
    PRINT "Osztas!"
    GOSUB szambekeres
    LOCATE 15, 25
    PRINT "Eredmeny:" + szam1$ + "/" + szam2$ + "="; VAL(szam1$) / VAL(szam2$)
    LOCATE 24
    INPUT "A folytatashoz nyomj ENTER-t", z$
  END IF
LOOP WHILE UCASE$(c$) <> "EXIT"
END

fejlec:
CLS
LOCATE 1
PRINT "+--------------+-----------+-----------+----------+---------------+"
PRINT "| ”sszead s: + | Kivon s: - | Szorz s: * | Oszt s: / | Kil,p,s: exit |"
PRINT "+--------------+-----------+-----------+----------+---------------+"
RETURN
```

```
szambekeres:
LOCATE 12, 25
INPUT "Add meg az els‹ sz mot:", szam1$
LOCATE 13, 25
INPUT "Add meg a m sodik sz mot:", szam2$
RETURN
```

A következő forrás a lerövidített. Ehhez még nem tanultunk mindent de nem s nehéz megérteni. Aki komolyabban szeretne foglalkozni a programozással annak jól jön majd a későbbiekben mert az OOP-ben csak ilyennel lehet találkozni.

```
DECLARE SUB muvelet (message$, muvelet$, sz1$, sz2$, eredmeny$)

CLS
DO
  GOSUB fejlec
  LOCATE 10
  INPUT "Add meg a muvelet tipusat:", c$
  IF c$ = "+" THEN
    GOSUB szambekeres
    muvelet "Osszeadas", "+", szam1$, szam2$, STR$(VAL(szam1$) + VAL(szam2$))
  END IF
  IF c$ = "-" THEN
    GOSUB szambekeres
    muvelet "Kivonas", "-", szam1$, szam2$, STR$(VAL(szam1$) - VAL(szam2$))
  END IF
  IF c$ = "*" THEN
    GOSUB szambekeres
    muvelet "Szorzas", "*", szam1$, szam2$, STR$(VAL(szam1$) * VAL(szam2$))
  END IF
  IF c$ = "/" THEN
    GOSUB szambekeres
    muvelet "Osztas", "/", szam1$, szam2$, STR$(VAL(szam1$) / VAL(szam2$))
  END IF

LOOP WHILE UCASE$(c$) <> "EXIT"

END

fejlec:
CLS
LOCATE 1
PRINT "+--------------+------------+------------+-----------+---------------+"
PRINT "| ”sszead s: + | Kivon s: - | Szorz s: * | Oszt s: / | Kil,p,s: exit |"
PRINT "+--------------+------------+------------+-----------+---------------+"
RETURN

szambekeres:
GOSUB fejlec
LOCATE 12, 25
INPUT "Add meg az els‹ sz mot:", szam1$
LOCATE 13, 25
INPUT "Add meg a m sodik sz mot:", szam2$
RETURN

SUB muvelet (message$, tipus$, sz1$, sz2$, eredmeny$)
    LOCATE 10, 30
    PRINT message$
    LOCATE 15, 30
    PRINT "Eredmeny:" + sz1$ + tipus$ + sz2$ + "=" + eredmeny$
    LOCATE 24
    INPUT "A folytat shoz nyomj ENTER-t", z$
END SUB
```

Fájl kezelés Basic-ben

A fájlkezlés a legtöbb programozási nyelvben hasonlóan történik. Először meg kell nyitni a fájlt. Ehez különböző módokat lehet használni attól függően hogy mire akarjuk használni. Ezt követően már műveletett tudunk vele végezni, Olvasás, írás. És végül le kell zárni a fájlt. A basic-ben azeket a következő utasítások végzik el:

- **OPEN**: beolvassa a fájlt egy fájlváltozóba a megfelelő móddal.
- **INPUT, LINE INPUT**: Beolvassa a fájl tartalmát. Egy szöveges változóba.
- **PRINT, WRITE**: Írás a fájlba.
- **SEEK**:Fájlba pozíció változtatása.
- **CLOSE**: A fájl lezárása, ezzel megszüntetjük a program és a fájl közötti kapcsolatott.

Fájl nyitási módok:

- **APPEND**:A fájl megnyitása írásra. A kurzor a fájl végére kerül.
- **BINARY**: Megnyitja a fájlt bináris módban vagyis bitenkénti műveletvégzés lehetséges. Ilyenkor az írás és olvasás a **GET** és **PUT** parancsokkal tötténik.
- **INPUT**:Megnyitja a fájlt olvasásra.
- **OUTPUT**:Fájl nyitása írásra, a fájl teljes tartalma felül íródik.
- **RANDOM**:Ezt nem tudtam értelmezni :)

A fájlváltozót a **#** jel vezeti be. A módot a **FOR** mód **AS** szerkezetben kell megadni.
Most egy-egy példa olvasásra és írásra.

KÓD:

```
CLS
OPEN „test.dat” FOR OUTPUT AS #1
PRINT #1, „Ez az első szöveges fájlom”
CLOSE #1
OPEN „test.dat” FOR INPUT AS #1
INPUT #1, a$
PRINT a$
CLOSE #1
```

Szöveges fájl nézegető program

Ahogy látjátok most hogy már valamennyire ismerjük az alapokat így konkrét programokkal mutatom be azok használatát, hogy lássátok mire is használhatóak az adott parancsok.

A következő program egy a felhasználó által meghatározott fájlt (nem feltétlen szövegest) olvas be és jelenít meg a képernyőn oldalanként.

A program működéséhez újabb utasításokat és funkciókat kell megismernünk.

Az első az **INKEY$** mely a billentyűzetről olvas be egy karaktert. Így önmagában nem szoktuk alkalmazni. A legtöbb esetben elől vagy hátul tesztelős ciklus feltételeként ill. a cikluson belül alkalmazzuk. Ennek segítségével tudjuk elérni hogy egyes billentyűk lenyomásakor történjen valami és nem kell az **ENTER**-re várni. A következő példa folyamatosan kiírja képernyőre a „nyomj ESC-et” szöveget amíg nem nyomjuk meg az **ESC** billentyűt. Ehhez a **CHR$()** függvényt használjuk mely bemenete az **ASCII** kód egy eleme, a kimenete pedig egy karater. Az **ESC** a **27**-es ebben a táblában. Vagyis **CHR$(27)**. És most következzen a program.

KÓD:

```
WHILE INKEY$<>CHR$(27)
  PRINT „Nyomd meg az ESC-et”
WEND
```

Ennek egy kicsit bonyolultabb változata már csak akkor írja ki újra a szöveget ha leütünk egy billentyűt. Először csinálunk egy ciklust mely csak akkor adja vissza a vezérlést ha leütünk egy billentyűt. Ehhez létrehozunk egy **KEY**$ változót melynek átadjuk **INKEY**$ értékét, Ez azért szükséges hogy a cikluson kívül is elérjük az értéket és ne keljen ismét meg ívni az **INKEY**$-t. A külső ciklus hasonlít a legelsőre csak a feltételt nem az **INKEY**$-re nézzük hanem a **KEY**$-re és a belső ciklus előtt töröljük a **KEY**$-t mert ha ezt nem tennék meg akkor a belső ciklus el sem indulna mert a feltétel hamis lenne mivel **KEY**$ nem üres.

KÓD:

```
CLS
WHILE KEY$<>CHR$(27)
  PRINT „Nyomd meg az ESC-et!”
  KEY$=””
  WHILE KEY$=””
    KEY$=INKEY$
  WEND
WEND
```

Ezzel meg van a programot működtető ciklusunk. Az **INKEY**$-nek azonban van egy másik visszatérési értéke is melyet a funkcionális billentyűknél add vissza.(A számok és a betűk kivételével az összes ilyen). Ezek felépítése annyiban tér el, hogy két karaktert tartalmaz. Az első az **ASCII** 0 és utána a **BASIC SCAN** kódja. Ezeket a **súgó Contents** és „**Keyboard Scan Code”** alatt lehet megnézni. Nekünk csak három kell. Az **F3=61**, a **PageUp=73** és a **PageDown=81**. Használatuk a következő: **Inkey$=CHR$(0)+CHR$(59)** //az F1 esetén

A program funkciói: először kell egy fejléc melyen a billentyűparancsokat jelenítjük meg. Ez lesz a **FEJLEC**

```
fejlec:
CLS
COLOR 0, 10
LOCATE 1
PRINT "<F3=Megnyitas> <Page Down=Kovetkezo oldal> <Page Up=Elozo oldal> <ESC=Kilepes>"
COLOR 7, 0
RETURN
```

Hogy a főprogramban csak a legfontosabbak legyenek ezért a billentyűzet figyelést is alprogramba tesszük.

KÓD:

```
billolvasas:
key$ = ""
WHILE key$ = ""
  key$ = INKEY$
WEND
RETURN
```

Ezt követően be kell olvasnunk a fájlt. Itt azonban meg kell határoznunk hogy hány sorból is áll, hogy könnyen tudjunk benne ugrálni majd. Egyben itt kérjük be a fájl nevet is. És végül meghatározzuk hogy a szövegpozíció mekkora legyen (max. 20 mert ennyi hely van a képernyőn.).

```
openfile:
  szovegcount% = 0
  szovegpos = 20
  LOCATE 15, 30
  INPUT "Kerem a fajl nevet:", filename$
  OPEN filename$ FOR INPUT AS #1
  WHILE NOT EOF(1)
    szovegcount% = szovegcount% + 1
    LINE INPUT #1, szoveg$
  WEND
  IF szovegcount% <= 20 THEN
    szovegpos = szovegcount%
  END IF
  CLOSE #1
RETURN
```

Ezzel meg is volnánk már csak a megjelenítést kell valahogy megoldani. Mivel ezt a program állandóan futtatja ezért érdemes egy vizsgálattal indítani, melyből megállapítjuk, hogy megvan-e nyitva fájl, ill. annak tartalma nem-e üres. Ezután az első sortól a szövegpozícióig beolvassuk és kiírjuk a képernyőre.

KÓD

```
printszoveg:
IF szovegcount% <> 0 THEN
  OPEN filename$ FOR INPUT AS #2
  FOR j = 1 TO szovegpos
    LINE INPUT #2, szoveg$
    PRINT szoveg$
  NEXT j
  CLOSE #2
END IF
RETURN
```

Most hogy már minden lényeges alprogram kész van jöhet a főprogram. A főciklus a fejlév megjelenítésével indul majd pedig egy feltétel hogy az általunk szügséges billentyűk közül valamelyik le lett-e nyomva, ha igen akkor megállapítjuk hogy melyik és az alapján elvégezzük az aktuális műveletet. A fájl nyitás az nem kíván különösebb mondani valót, de az oldal léptetés igen. Itt vizsgálni kell hogy a pozíció nem túl kicsi vagy nem túl nagy mert ez végzetes hibát okozz a programunk számára. A feltételek után már csak meg kell jeleníteni a fájl tartalmát és lefutatni a belső ciklust a billentyűzet figyelés véget.

KÓD:

```
szovegpos = 0
szovegcount% = 0
WHILE key$ <> CHR$(27)
  GOSUB fejlec
  IF (key$ = CHR$(0) + CHR$(61)) OR (key$ = CHR$(0) + CHR$(73)) OR (key$ =
CHR$(0) + CHR$(81)) THEN
    IF (key$ = CHR$(0) + CHR$(61)) THEN
      CLS
      GOSUB fejlec
      GOSUB openfile
      CLS
      GOSUB fejlec
    END IF
    IF (key$ = CHR$(0) + CHR$(73)) THEN
      szovegpos = szovegpos - 20
      IF szovegpos <= 0 THEN
        szovegpos = 20 AND szovegcount%
      END IF
    END IF
    IF (key$ = CHR$(0) + CHR$(81)) THEN
      szovegpos = szovegpos + 20
      IF szovegpos > szovegcount% THEN
        szovegpos = szovegcount%
      END IF
    END IF
  END IF
  GOSUB printszoveg
  GOSUB billolvasas
WEND
END

fejlec:
CLS
COLOR 0, 10
LOCATE 1
PRINT "<F3=Megnyitas> <Page Down=Kovetkezo oldal> <Page Up=Elozo oldal> <ESC=Kilepes>"
COLOR 7, 0
RETURN

billolvasas:
key$ = ""
WHILE key$ = ""
  key$ = INKEY$
WEND
RETURN
```

```
openfile:
  szovegcount% = 0
  szovegpos = 20
  LOCATE 15, 30
  INPUT "Kerem a fajl nevet:", filename$
  OPEN filename$ FOR INPUT AS #1
  WHILE NOT EOF(1)
    szovegcount% = szovegcount% + 1
    LINE INPUT #1, szoveg$
  WEND
  IF szovegcount% <= 20 THEN
    szovegpos = szovegcount%
  END IF
  CLOSE #1
RETURN

printszoveg:
IF szovegcount% <> 0 THEN
  OPEN filename$ FOR INPUT AS #2
  FOR j = 1 TO szovegpos
    LINE INPUT #2, szoveg$
    PRINT szoveg$
  NEXT j
  CLOSE #2
END IF
RETURN
```

Grafika

Most már megtanultuk az alapvető IO műveleteket, a változókkal való művelet végzéseket és a fájl kezelés alapjait. Ideje végre a legtöbb kezdő programozót érdeklő Grafikai utasítások áttanulmányozása.

Alapvető grafika utasítások

A legfontosabb utasítások a PONT-, VONAL- és KÖR rajzoló parancsok. Ezeknek egyetlen közös paraméterük a szín melyet nem kötelező megadni. A koordináták ugyanazok, de mindnél más szerepe van.

A koordináták

A programozásban a matematikában megismert koordinációs rendszer 4-dik negyedét használjuk egy kis módosítással. A 0,0 a Bal felső sarokban található. Mind az X és Y a képernyőn látható területen pozitív értéket vessz fel.

Képernyők

Igen! Képernyők! A programozási nyelvekben külön külön képernyőnek kell elképzelni az egyes módokat. A Basicben 14 található. Jelölésük 0-13-ig történik. A képernyő módot a SCREEN utasítással lehet megváltoztatni. De ahhoz, hogy módosítsuk meg kell ismernünk őket. A következő táblázat felsorolja az egyes módok tulajdonságait

SCREEN x	Tulajdonságai
0	Alapértelmezett. Karakteres mód ahol 25x80 karakter fér el. Az első a sorok száma
1	Grafikus üzemmód. 320X200 felbontás 16 szín, 25x40 karakter
2	Grafikus üzemmód. 640X200 felbontás 16 szín, 25x80 karakter
3	Grafikus üzemmód. 720X348 felbontás 2 szín, 25x80 karakter
4	Grafikus üzemmód. 640X400 felbontás 16 szín,25x80 karakter
5	Még a basicben sincs leírás
6	Még a basicben sincs leírás
7	Grafikus üzemmód. 320X200 felbontás 16 szín, 25x40 karakter
8	Grafikus üzemmód. 640X200 felbontás 16 szín, 25x80 karakter
9	Grafikus üzemmód. 640X350 felbontás 16 Szín, 25x80 karakter
10	Grafikus üzemmód. 640X350 felbontás 9 szín, 25x80 karakter, Monochrome monitorra
11	Grafikus üzemmód. 640X480 felbontás 16 szín, 30x80 karakter
12	Grafikus üzemmód. 640X480 felbontás 16 szín, 30x80 karakter
13	Grafikus üzemmód. 320X200 felbontás 256 szín, 25x40 karakter

Színek kezelése

A basicben mi a 12-es módot vagyis 640x480-as felbontást alkalmazunk. Itt 16 szint tudunk használni. A színek a 0-val kezdődnek és a 15-el végződnek. A 0 a fekete a 15 a fehér. 0-7 a sötét színek, 8-15-ig ezeknek világosabb megfelelője.

Rajzoljunk!

Kezdjük a legegyszerűbbel a pixel rajzolásával. Ezt a PSET-tel tudjuk kirajzoltatni. Használata: PSET (x,y),szín.

A következő a sokk pontból álló egyenes. Ugyan pontokból is fellehet rajzolna de az jóval lassabb. Vonalat a LINE függvénnyel rajzolunk. Használata: LINE (x1,y1)-(x2,y2), szín, ezt üresen hadjuk, &H és hexadecimális számok. Az utolsó a vonal szaggatotságát határozza meg.

A következő a NÉGYZET. Érdekes mert ezt is a LINE-nal kell kirajzolni, csak most ahol előbb nem írtunk semmit oda a B betűt kell írni, ha a B helyet a BF-et írjuk akkor egy olyan négyzetett rajzol ahol a megadott színnel kitölti a belsejét.

LINE (x1,y1)-(x2,y2),szín,B vagy BF

Most jöhet egy kör. Ehhez a Circle-t használjuk. Az első paraméter a a kör középpontja míg a többi a sugara és a színe. CIRCLE (x,y),sugár, szín
Most pedig a példák:

KÓD:

```
SCREEN 12
CLS
PSET (10,10),12
LINE (10,100)-(10,200),12
LINE (10,300)-(100,400),12,B
LINE (200,10)-(300,100),12,BF
CIRCLE (500,200),50,12
```

Most megismerkedünk egy hasznos függvénnyel a PAINT-tel. Ez a megadott koordinátákat tartalmazó mezőt melynek a kerete a megadott színű az kitölti a kitöltési színnel.
A következő példa kirajzolja a számokhoz tartozó színeket.

KÓD:

```
SCREEN 12
CLS
j = 0
FOR i = 0 TO 3
  FOR k = 0 TO 3
    LINE (100 * k, 100 * i)-(100 * k + 50, 100 * i + 50), j, BF
    j = j + 1
  NEXT k
NEXT i
LOCATE 2, 4
PRINT "0"
LOCATE 2, 16
PRINT "1"
LOCATE 2, 28
PRINT "2"
LOCATE 2, 41
PRINT "3"
LOCATE 8, 4
PRINT "4"
```

```
LOCATE 8, 16
PRINT "5"
LOCATE 8, 28
PRINT "6"
LOCATE 8, 41
PRINT "7"
LOCATE 15, 4
PRINT "8"
LOCATE 15, 16
PRINT "9"
LOCATE 15, 28
PRINT "10"
LOCATE 15, 41
PRINT "11"
LOCATE 21, 4
PRINT "12"
LOCATE 21, 16
PRINT "13"
LOCATE 21, 28
PRINT "14"
LOCATE 21, 41
PRINT "15"
```

És ezzel vége is könyvemnek!

Utószó

Ahogy már leírtam az előszóban ezt a könyvet teljesen kezdők számára írtam. Lehet hogy kicsit nehéznek tűnik a teljes programok leírása de ez azért van mert megpróbáltam olyan megoldásokat alkalmazni mellyel átlátható jól strukturált programokat kapunk. A Basic-ben, sőt a többi nyelvben is lehetőség van feltétel nélküli ugrásokra. Ez megkönnyíti a kezdő programozók munkáját de később egy nagyobb projectnél átláthatatlan lesz a forrás. Én ezektől szeretnélek titeket megóvni. Sok megoldása van egy adott problémának én megpróbáltam az általam legjobbnak vélt legegyszerűbb megoldást bemutatni.

Aki ezek után úgy gondolja, hogy ő komolyan megszeretne ismerkedni a programozással annak azt tanácsolom, hogy sose adja fel akár milyen nehéznek is tűnik az adott feladat. Sokszor találkozom én is olyan feladatokkal és hibákkal melyek fölött napokat ülök és akkor se jövök rá, de végül mindig kiderül hogy egyszerű logikai hibát követtem el. Aki foglalkozni fog vele az sokszor fog hasonló helyzetekbe kerülni.

Ha tényleg komolyak a szándékaid a programozás terén akkor ismerkedj meg a PASCAL/DELPHI, C/C++/Visual Basic és Java nyelvekkel és válaszd ki magadat azt amelyik a legjobban illik hozzád. A nyelvek nagyon hasonlóak csak néhány szabály különbözik.

MEGJEGYZÉS: A Visual Basic csak nevében basic, mert inkább a C++-ra hasonlít.

Utasítások