Juhász Tibor – Kiss Zsolt:

PROGRAMOZÁSI ISMERETEK

(Műszaki Könyvkiadó, 2011, MK–4462-3)

KIEGÉSZÍTÉSEK A TANKÖNYVHÖZ

Bevezetés

A kiegészítésekben néhány olyan témakört tekintünk át, amely nem fért bele a tankönyvbe, illetve megszakította volna a tankönyv gondolatmenetét. Találunk közöttük elméleti ismereteket csakúgy, mint a Visual Studio, illetve a Visual Basic használatára vonatkozó tudnivalókat.

Az adatok kódolása

A programjainkban felhasznált adatok kódolva kerülnek a memóriába vagy a háttértárra. Az alábbiakban bemutatjuk a leggyakoribb kódolási módokat.

Bináris és kettes komplemens kód

Az egész számokat a leggyakrabban bináris vagy kettes komplemens kódban tároljuk. A bináris kód egyszerűen a szám kettes számrendszerbeli alakját jelenti:

44 → 101100, vagy például 8 bitre kiegészítve: 00101100.

Könnyű belátni, hogy n biten 0-tól 2^n–1-ig ábrázolhatunk egész számokat.

Kettes komplemens kódban pozitív és negatív értékeket is tárolhatunk. A pozitív számok vagy a 0 kettes komplemens kódja megegyezik a bináris kóddal. A $–x$ negatív egész n bites kettes komplemens kódja: $2^n – x$. Így n biten $–2^{n–1}$-től $+2^{n–1} – 1$-ig ábrázolhatunk egész számokat.

A –44 kettes komplemens kódja 8 biten például: $2^8 – 44 = 212$ → 11010100.

Vegyük észre, hogy egy számnak és kettes komplemensének az összege a túlcsorduló 9. bitet elhagyva 0 lesz:

```
    00101100
   +11010100
  (1)00000000
```

Az átváltást kettes számrendszerben úgy szokták végrehajtani, hogy a szám bináris kódját bitenként negálják, majd hozzáadnak 1-et:

00101100 → 11010011 → 11010011 + 1 = 11010100

Az átalakítást természetesen a választott bithosszúsággal kell elvégezni.

Az átváltás „papíron” egyszerűbb, ha észrevesszük, hogy a bináris kódban jobbról indulva az első 1-esig minden bit változatlan marad (beleértve magát az első 1-est is), majd minden további bitet negálunk:

00101100 → 11010100

A kettes komplemens kód előnyei:

- A kódolást és a dekódolást ugyanaz az algoritmus szolgálja.
- Az összeadást és a kivonást ugyanazzal az algoritmussal (áramkörrel) végezhetjük el akár bináris, akár kettes komplemens kódban vannak megadva az operandusok.
- A kettes komplemens kód legnagyobb helyiértékű bitje – bár részt vesz a számításokban – jelzi a szám előjelét (0: pozitív, 1: negatív). Ebből a szempontból a 0 pozitívak számít.

A valós típusú értékek kódolása

A valós (lebegőpontos) értékek kódolását az IEEE 754-es szabvány alapján mutatjuk be. Az egyszeres pontosságú (*Single*, illetve *binary32*) típust 4 bájton tároljuk, a következő forma alapján:

$$\pm 1{,}m \cdot 2^{\pm k}$$

ahol m a bináris érték törtrésze, a k pedig az előjeles kitevő. Az m-et szokás mantisszának[1], a k-t pedig karakterisztikának vagy exponensnek nevezni.

Egyszeres pontosság esetén a 32 bitből a mantisszához 23 bitet, a karakterisztikához 8 bitet, a szám előjeléhez pedig 1 bitet foglalunk le a következő sorrendben (bal oldalon a legnagyobb helyiértékű bit):

[1] A mantisszába gyakran beleértik az egészrészt is. Újabban a mantisszát szignifikandusnak nevezik.

előjelbit karakterisztika mantissza

Az előjelnél a 0 jelenti a pozitív, az 1 a negatív értéket. A karakterisztikánál az előjeles kitevőhöz hozzáadunk 127-et (eltolt érték), így küszöböljük ki a kitevő előjelének a tárolását. Az eltolt értéket binárisan kódoljuk.

A 32 bit négy bájtot foglal el, amelyből az Intel-mikroprocesszorok esetén az alacsonyabb helyiértékű bájt kerül a kisebb memóriacímre („little endian” sorrend).

A szabvány további részletei és speciális esetei megtalálhatók az Interneten (lásd például: Wikipedia.).

Példaként határozzuk meg a –37,25 egyszeres pontosságú, lebegőpontos kódját! Először célszerű átváltani a számot kettes számrendszerbe:

–37,25 → –100101,01

A szám normálalakja „vegyes” formában: $-1{,}0010101 \cdot 2^5$

Így a mantissza (kiegészítve 23 bitre): 00101010000000000000000

az eltolt karakterisztika pedig: 5 + 127 = 132 → 10000100

A lebegőpontos kód tehát:

1 10000100 00101010000000000000000

vagy bájtonként tagolva:

11000010 00010101 00000000 00000000

A lebegőpontos formát szokás hexadecimálisan is felírni:

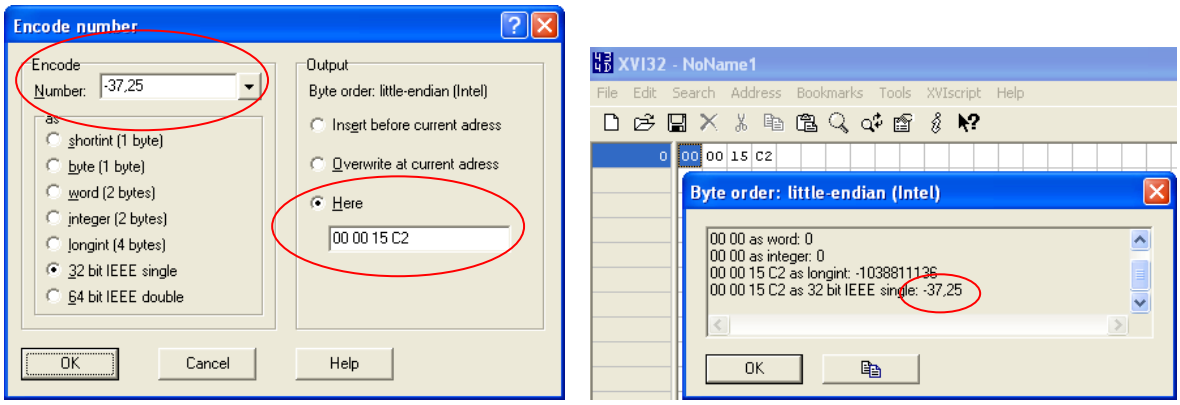
C2 15 00 00

Mivel az Intel-mikroporcesszoroknál az alacsonyabb helyiértékű bájt kerül előre, a memóriában (vagy egy adatfájlban) a bájtok sorrendje:

00 00 15 C2

A kódolást sokféle program segíti. Megemlítjük a telepítés nélkül futtatható, ingyenes XVI32-t, amely a szövegfájlok kódjába is enged betekinteni (lásd a következő szakaszt): http://www.chmaas.handshake.de.

Az XVI32-vel végzett konvertáláshoz először hozzunk létre egy új dokumentumot (*File*/*New*), majd válasszuk a *Tools* menü *Encode number* parancsát. A dekódolást az első bájtra állva a *Decode number* parancscsal hajtjuk végre. Új bájtok beírásához az *Insert* billentyűvel váltsunk beszúró üzemmódba.



Lebegőpontos értékek kódolása és dekódolása az XVI32 programmal

Az Interneten kódolást-dekódolást végző weblapokat is találunk, például: http://www.binaryconvert.com.

Szövegek kódolása

A legtöbb mai karakterkód-rendszer az ASCII (ejtsd: eszki) kódból fejlődött ki. Az ASCII-kód egy bájtot használt fel egy karakter bináris kódolására. Az első 32 kódot 0-tól 31-ig úgynevezett kontroll-karakterekhez rendelték. Ilyen volt például a tabulátor, a soremelés, a csengő (eredetileg a távírón) stb. 32-től 127-ig következtek az írásjelek, a számjegyek, illetve az angol ábécé nagy- és kisbetűi. A 128-tól kezdődő kódokat az angol ábécében nem szereplő betűkre, például az ékezetes magánhangzókra, illetve egyéb karakterek tárolására kezdték felhasználni. Több, egymástól eltérő definíció létezett, ezeket kódlapoknak nevezték. A magyar ábécének például a 852-es kódlap (Latin-1) felelt meg a leginkább, bár eredetileg csak „kalapos” ő-t és ű-t tartalmazott (ô, û).

A Windows operációs rendszernél vezették be a karakterek szintén egy bájtos ANSI-kódolását[2]. Ezzel igyekeztek a legtöbbféle nemzeti karakterkészletet megvalósítani. Az ANSI is kódlapokból állt. Az első 128 kód lényegében megfelelt az ASCII-kódnak, de a többi karakter jelentősen eltért az ASCII-kódlapoktól. Az ANSI-kódlapot az operációs rendszer területi beállításainál választhatjuk ki (a Unicode-ot nem használó

[2] Ne tévesszük össze az amerikai szabványokat felügyelő American National Standards Institute rövidítésével.

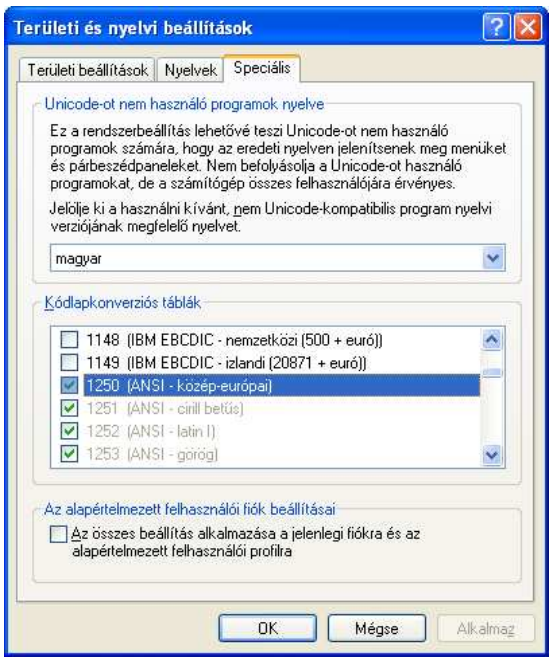
programok számára). A szabványos magyar ékezetes karaktereket a 1250-es (közép-európai) kódlap tartalmazza. Más kódtábla beállításánál vagy alkalmazásánál ezek a karakterek hibásan jelennek meg.[3]

Kód	*Jelölés*	*Elnevezés*	*Kód*	*Karakter*	*Kód*	*Karakter*	*Kód*	*Karakter*
0	NUL	Null (üres)	32	Szóköz	64	@	96	`
1	SOH	Fejléc kezdete	33	!	65	A	97	a
2	STX	Szöveg kezdet	34	"	66	B	98	b
3	ETX	Szöveg vége	35	#	67	C	99	c
4	EOT	Adatátvitel vége	36	$	68	D	100	d
5	ENQ	Vizsgálat	37	%	69	E	101	e
6	ACK	Visszaigazolás	38	&	70	F	102	f
7	BEL	Csengetés	39	´	71	G	103	g
8	BS	Törlés balra (Backspace)	40	(	72	H	104	h
9	HT	Vízszintes tabulátor	41	)	73	I	105	i
10	LF	Soremelés	42	*	74	J	106	j
11	VT	Függőleges tabulátor	43	+	75	K	107	k
12	FF	Lapdobás (új oldal)	44	,	76	L	108	l
13	CR	Kurzor a sor elejére	45	-	77	M	109	m
14	SO	Nagybetűzár ki	46	.	78	N	110	n
15	SI	Nagybetűzár be	47	/	79	O	111	o
16	DLE	Data Link Escape	48	0	80	P	112	p
17	DC1	Eszközkontroll 1	49	1	81	Q	113	q
18	DC2	Eszközkontroll 2	50	2	82	R	114	r
19	DC3	Eszközkontroll 3	51	3	83	S	115	s
20	DC4	Eszközkontroll 4	52	4	84	T	116	t
21	NAK	Negatív visszaigazolás	53	5	85	U	117	u
22	SYN	Szinkron üresjárat	54	6	86	V	118	v
23	ETB	Adatátviteli blokk vége	55	7	87	W	119	w
24	CAN	Mégsem (Cancel)	56	8	88	X	120	x
25	EM	Adathordozó vége	57	9	89	Y	121	y
26	SUB	Substitute	58	:	90	Z	122	z
27	ESC	Escape	59	;	91	[	123	{
28	FS	Állományelválasztó	60	<	92	\	124	|
29	GS	Csoportelválasztó	61	=	93	]	125	}
30	RS	Rekordelválasztó	62	>	94	^	126	~
31	US	Unit-elválasztó	63	?	95	_		
127	DEL	Delete						

Az eredeti ASCII-kód 0–127 karakterei

Az 1980-as évek végére szükségessé vált egy univerzális kódrendszer bevezetése, amely lehetőség szerint minden nemzeti karakterkészletet magában foglal. A Unicode (ejtsd: junikód) eredetileg két bájttal kódolt egy karaktert, így 65536 jelet tartalmazhatott. 4 bájtos bővítése jelenleg 107 ezer karakter, illetve egyéb jel definícióját öleli fel.[4]

[3] A kódtáblák áttekintését lásd például: http://msdn.microsoft.com/hu-hu/goglobal/bb964653(en-us).aspx
[4] A teljes Unicode-karakterkészletet lásd például: http://www.unicode.org/charts/



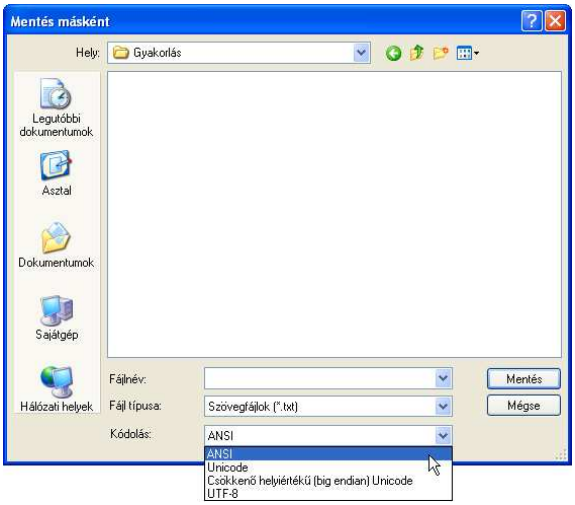
A Windows-1250-es kódlap kiválasztása a területi beállításoknál

Az állandó hosszúságú, négybájtos kód helyett a legtöbb karakter kódolásához elegendő lenne egy, illetve két bájt. A Unicode UTF-8 változata[5] változó hosszon, 1–4 bájttal kódolja a karaktereket. Ez illeszkedik a legjobban az eredeti ASCII-kódhoz. Szükség esetén a kezdőbitekkel jelzi, hogy egy bájtnál hosszabb karakterkód következik. Ékezetes magánhangzóink 2 bájtot igényelnek. Az UTF-16 szintén változó hosszúságú kódot tartalmaz, 2–4 bájttal.

Az UTF kódolású fájlok rendelkezhetnek egy úgynevezett BOM-mal[6], amely az első bájtokon jelzi az Intel-mikroprocesszorokra jellemző sorrendet a több bájtos értékeknél[7]. A Jegyzettömb UTF-8 esetén hexadecimálisan EFBBBF-et ír az első három bájtba. UTF-16 esetén pedig az első két bájt FFFE értéke mutatja a „little endian” bájt-sorrendet.

Az „aáb” karaktereket tartalmazó szövegfájlok mérete például különböző kódolással tárolva:

ANSI:	3 bájt (1-1 bájt karakterenként)
UTF-8:	4 bájt (1 bájtos „a”, 2 bájtos „á”, 1 bájtos „b”)
UTF-8 aláírással:	7 bájt (3 bájtos BOM, 1 bájtos „a”, 2 bájtos „á”, 1 bájtos „b”)
Unicode (UTF-16):	8 bájt (2 bájtos BOM, 2-2 bájtos karakterek)



A kódolás beállítása a Jegyzettömbben

[5] UTF: Unicode Transformation Format

[6] Byte Order Mark: bájtsorrend jelölő. Egyes szövegszerkesztők „aláírásnak” nevezik a BOM-ot (például „UTF-8 aláírással”).

[7] Elöl áll az alacsonyabb helyiértékű bájt, szemben például a Motorola-mikroprocesszorokkal.

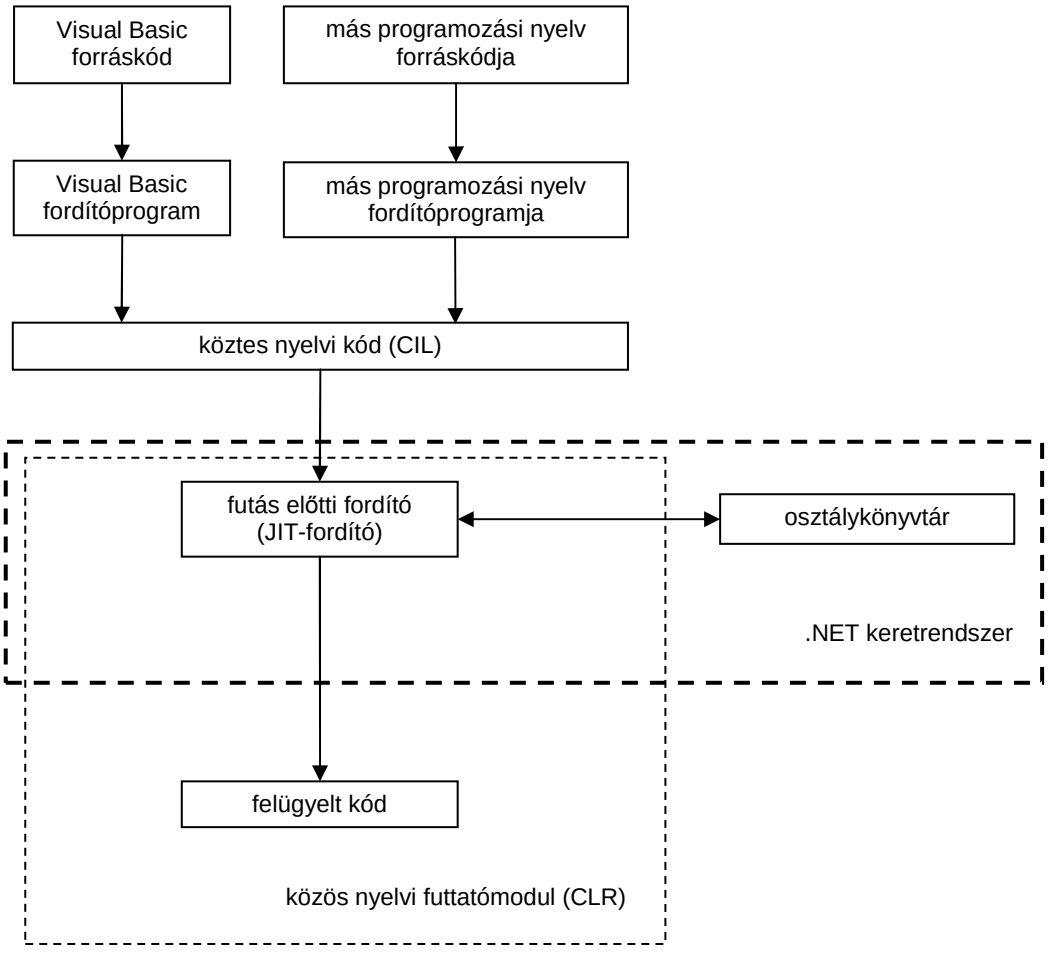
Leginkább a szövegfájlok kezelésénél ütközünk a karakterek kódolásával kapcsolatos problémákba. A Windows Jegyzettömbje például az ANSI-kódot ajánlja fel a mentésnél. A Visual Basic viszont alapértelmezés szerint UTF-8-at alkalmaz. Így a Jegyzettömbbel ANSI-kódban mentett ékezetes karaktereket hibásan értelmezheti a beolvasásnál. A hibák elkerüléséhez mentsük a szövegfájlt Unicode-ban, vagy adjuk meg a fájl megnyitásánál a kódolás módját (részletesebben lásd a Programozási útmutatóban):

```
Fájl = New IO.StreamReader(Út, System.Text.Encoding.Default)
Fájl = New IO.StreamWriter(Út, Hozzáfűz, System.Text.Encoding.Default)
```

A .NET

Köztes nyelv és felügyelt kód

A programozási munka hatékonyságát növeli, ha a forráskód nem kötődik egy meghatározott operációs rendszerhez, illetve egy összetett feladat egyes részeit egymástól eltérő programozási nyelveken készíthetik el a programozók. Ennek a célnak az érdekében a Visual Basicben megírt forráskódot a fordítóprogram a gépi kód helyett egy közbenső nyelvre, az úgynevezett CIL-re fordítja le[8] (Common Intermediate Language: közös közbenső nyelv). A közbenső vagy **köztes nyelv** már független a forráskódban használt programozási nyelvtől, és többé-kevésbé független az operációs rendszertől, illetve a hardverkörnyezettől is.



A .NET keretrendszer szerkezete és a fordítás folyamata

A CIL-változatból a program futtatásakor a **futás előtti fordító** hozza létre a mikroprocesszor számára is érhető gépi kódot (JIT: Just In Time, futás előtt). Bár a JIT-fordító a futtatás elején lép működésbe, de a köztes nyelvről történő fordítás sokkal gyorsabb, mint az eredeti forráskód fordítása gépi kódra.

A JIT-et a Windows bővítése, az úgynevezett **.NET** (ejtsd: dotnet) tartalmazza (.NET Framework: .NET keretrendszer). A .NET része továbbá a **közös nyelvi futtatómodu**l, amely a futás előtti fordításon kívül a programok futtatásáért, az erőforrások elosztásáért felelős (CLR: Common Language Runtime, közös nyelvi futtató). A CLR alkalmazkodik a számítógép hardver- és szoftverelemeihez, ezekkel nem kell a programozó-

[8] Eredetileg: MIL (Microsoft Intermediate Language, Microsoft köztes nyelv).

nak törődnie. Mivel a CLR felügyeli a program végrehajtását és az erőforrások biztonságos felhasználását, a CLR által futtatott kódot **felügyelt kódnak** nevezik.

A .NET-ben találunk még egy több ezer osztály definícióját tartalmazó gyűjteményt[9]. Ezeket az osztályokat felhasználhatjuk programjainkban. Segítségükkel rengeteg részfeladat programozásától kímélhetjük meg magunkat.

A .NET Framework a Vistától kezdve már beépült a Windows operációs rendszerbe. Windows XP használata esetén a .NET keretrendszer letölthető a Microsoft webhelyéről. Telepítését elvégzik azok a programok is, melyeknek szüksége van rá. A Visual Studio (Visual Basic) Express Edition például szükség esetén telepíti a .NET Framework megfelelő változatát.

Megjegyezzük, hogy az első, platformfüggetlenségre törekvő nyelv, a Java az 1990-es évek elején jött létre. A Java fordítóprogramja által előállított köztes kódot bájtkódnak nevezik. Ezt a Java futtatómodulja hajtja végre (Java virtuális gép). Újabban a Microsoftnál is kezdik bájtkódnak nevezni a CIL-t.

Táblázatok megjelenítése

Nagyobb méretű táblázatokat nem célszerű címkeobjektumon megjeleníteni. A címkéhez nem rendelhető gördítősáv, automatikus méretezésének engedélyezésével fedésbe kerülhet más elemekkel, a méretezés tiltásával pedig lemaradhatnak a táblázat részei.

A táblázatok kiírásához használjunk inkább többsoros szövegdobozt vagy *DataGridView* vezérlőelemet.

Többsoros szövegdoboz

Többsoros szövegdoboz létrehozásához helyezzünk egy *TextBox* vezérlőt az űrlapra, majd *Multiline* tulajdonságát állítsuk *True*-ra. Táblázatok megjelenítéséhez általában fix szélességű karaktereket, például *Courier* betűtípust használunk. A szövegdoboz *ScrollBars* tulajdonságával célszerű engedélyezni a gördítősávokat, a sortörést viszont tiltsuk le (*WordWrap* = *False*). A *ReadOnly* = *True* beállításával megakadályozzuk, hogy a felhasználó módosíthassa a szövegdoboz tartalmát. A kijelölést a következő metódushívással törölhetjük:

```
TextBox1.Select(0, 0)
```

Egyszerűbb esetben tabulátorokkal tagolhatjuk az egyes sorokat (*vbTab*). A következő utasítások viszont a *String.Format* metódust használják a *Diákok.vb* kódfájlban található adatok kiírásához:

```
For I = 0 To Név.GetUpperBound(0)
  Temp = String.Format("{0, -20} {1, 5:F0} cm {2, 6:F1} kg", _
                       Név(I) & ":", Magasság(I), Tömeg(I))
  TextBox1.Text &= Temp & vbNewLine
Next
TextBox1.Select(0, 0)
```

A diákok adatainak megjelenítése többsoros szövegdobozban.
A vízszintes gördítősáv további oszlopokra utal.

A többsoros szövegdoboz *Text* tulajdonsága helyett mind értékadásnál, mind kiolvasásnál használhatjuk a *Lines* tulajdonságot, amely egy sztringtömbben tárolja a sorokat. Így egy-egy sort utólag is módosíthatunk!

A *Lines* tulajdonság valójában egy *csak olvasható* másolata a szövegdobozba írt soroknak. A sorok futásidejű módosításához egy megfelelően feltöltött sztringtömböt rendeljük hozzá a tulajdonsághoz, például:

```
TextBox1.Lines = New String() {"első", "második", "harmadik"}
```

[9] osztálykönyvtárat

A tömböt természetesen más módon is inicializálhatjuk:

```
Dim Doboz(4) As String
For I = 0 To 4
    Doboz(I) = I
Next
TextBox1.Lines = Doboz
```

A hozzárendelést a sztringtömb minden módosítása után el kell végezni!

A diákok táblázatának megjelenítése a *Lines* tulajdonság alkalmazásával:

```
Dim Sor(Név.GetUpperBound(0)) As String
For I = 0 To Név.GetUpperBound(0)
  Sor(I) = String.Format("{0,-20} {1,5:F0} cm {2,6:F1} kg", _
                         Név(I) & ":", Magasság(I), Tömeg(I))
Next
TextBox1.Lines = Sor
TextBox1.Select(0, 0)
```

A *Lines* tulajdonság segítségével hozzáférhetünk a *TextBox* soraihoz. Ehhez rendeljük hozzá egy tömbhöz a tulajdonságot:

```
Dim Sor() as String
Sor = TextBox1.Lines
```

A *Sor* tömb módosítása után ismét vissza kell adni az elemeket a szövegdoboznak:

```
TextBox1.Lines = Sor
```

A *DataGridView* vezérlőelem

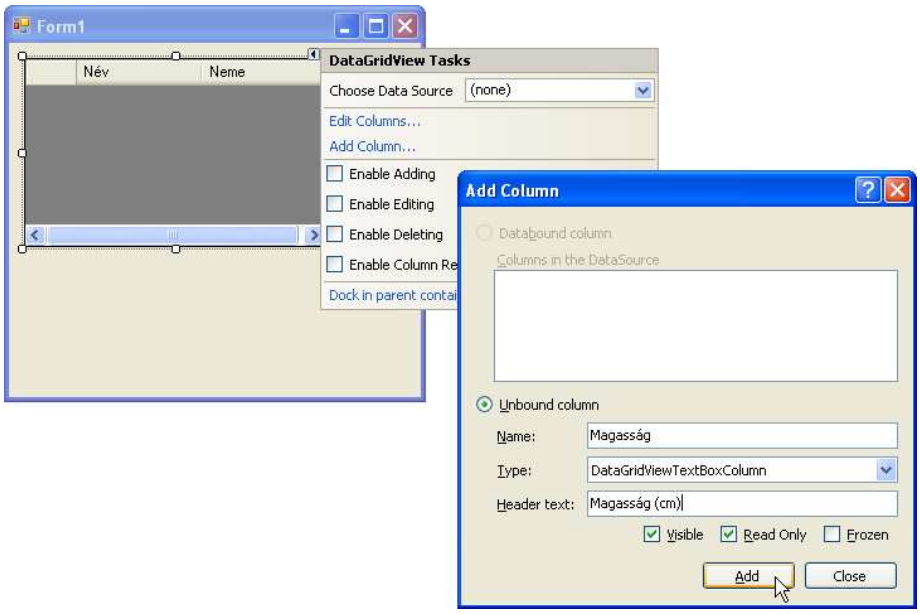
Táblázatok áttekinthető kiírásához használjuk a *DataGridView* vezérlőelemet, melyet az eszközkészlet *Data* csoportjában találunk! Az összetett tulajdonságokkal és metódusokkal rendelkező vezérlővel akár adatbázisok lekérdezéseinek eredménytábláit is megjeleníthetjük. Az alábbiakban egy egyszerű példát mutatunk a használatára.

Hozzunk létre egy új Windows-alkalmazást, és helyezzünk el az űrlapon egy *DataGridView* vezérlőelemet. Adjuk hozzá a projekthez a forrásfájlok között található *Diákok.vb* kódfájlt.

A vezérlőelem menüjében tiltsuk le, hogy a felhasználó felvehessen, szerkeszthessen vagy törölhessen cellákat a táblázatban (*Enable Adding*/*Editing*/*Deleting* jelölőnégyzetek).

A vezérlőelem menüjének *Add column* parancsával adjuk hozzá a táblázathoz a *Név*, *Neme* és *Magasság* oszlopokat. Mindegyik oszlopot tegyük láthatóvá (*Visible*), és tiltsuk le a módosítást (*Read Only*).

Mindezen tulajdonságokat utólag a tulajdonságablakban, illetve futás közben is beállíthatjuk.



Oszlopok hozzáadása a táblázathoz

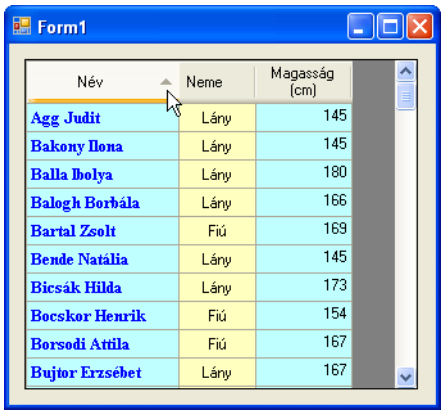
Ha engedélyezzük a táblázat adatainak a módosítását, ez természetesen nem lesz automatikusan hatással az eredeti adatokra.

A táblázat kitöltéséhez definiáljunk egy sztringtömböt, melynek annyi eleme van, ahány oszlopot szeretnénk megjeleníteni a táblázatban. Töltsük fel a sztringtömb elemeit a táblázat egy sorának adataival, majd a *Rows.Add* metódussal adjuk hozzá a táblázathoz.

A megjelenítést végző kódrészlet:

```
Dim Sor(2) As String
For I = 0 To Diákok.Név.GetUpperBound(0)
  Sor(0) = Diákok.Név(I)
  If Diákok.Fiú(I) Then
    Sor(1) = "Fiú"
  Else
    Sor(1) = "Lány"
  End If
  Sor(2) = Diákok.Magasság(I)
  DataGridView1.Rows.Add(Sor)
Next
```

Szükség esetén méretezzük át a vezérlőelemet, majd futtassuk a programot. Figyeljük meg, hogy a felhasználó is módosíthatja az oszlopszélességet. Az oszlopfejlécre kattintva pedig a szokásos módon, növekvő, illetve csökkenő sorrendbe rendezheti az adatokat.



A formázott táblázat névsor szerint rendezve

Az oszlopok programozott rendezéséhez hívjuk meg a táblázat *Sort* metódusát:

`DataGridView1.Sort(`*`oszlopnév`*`, `*`rendezésiránya`*`)`

Például:

```
DataGridView1.Sort(Név, System.ComponentModel.ListSortDirection.Ascending)
```

A táblázat formázása

A tulajdonságablak számos tulajdonságot tartalmaz. Az alábbiakban csak néhány lehetőséget sorolunk fel.

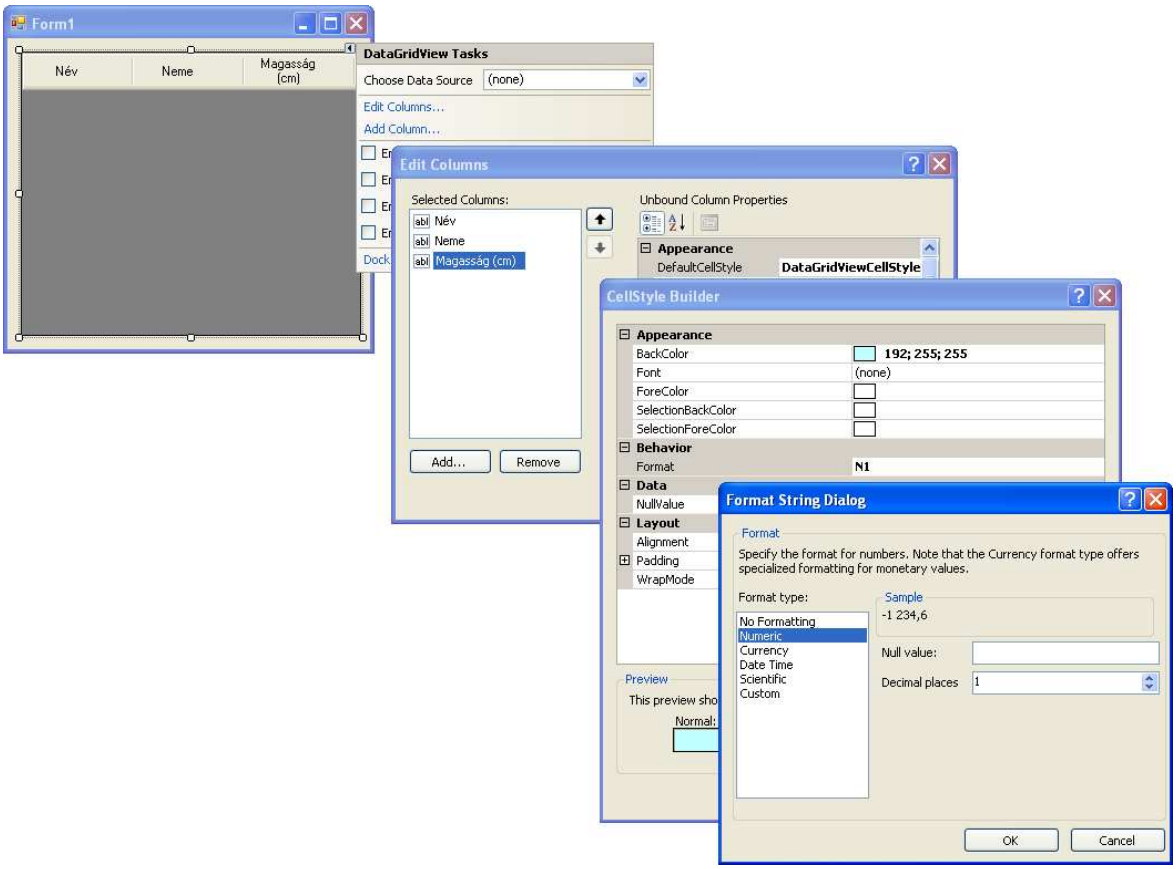
Tulajdonság	Javasolt érték	Magyarázat
AllowUserToAddRows	False	Új sort hozzáadásának engedélyezése
AllowUserToDeleteRows	False	Sor törlésének engedélyezése
AllowUserToOrderColumns	False	Oszlopok átrendezésének engedélyezése
AllowUserToResizeColumns	True	Oszlopok átméretezésének engedélyezése
AllowUserToResizeRows	False	Sorok átméretezésének engedélyezése
AutoSizeColumnsMode	AllCells	Az oszlopszélesség automatikus igazítása a tartalomhoz
BackgroundColor		Háttérszín
BorderStyle		A táblázatszegély típusa
CellBorderStyle		A cellaszegély típusa
ColumnHeadersBorderStyle		A fejlécszegély típusa
ColumnHeadersDefaultCellStyle		A fejlécsor formátuma (oszloponként is állítható)
ColumnHeadersHeight		A fejlécsor magassága
ColumnHeadersHeightSizeMode	Autosize	A fejlécsor automatikus magasságállítása

Tulajdonság	Javasolt érték	Magyarázat
ColumnHeadersVisible	True	A fejlécsor megjelenítése
DefaultCellStyle		A cellák formátuma (oszloponként is állítható)
GridColor		A cellaszegély színe
ReadOnly	True	Az adatok módosításának tiltása (a módosításhoz az egész táblázatra is ki kell adni az engedélyt)
RowHeadersVisible	False	Sorfejlécek engedélyezése
ScrollBars	Both	Gördítősávok engedélyezése

A DataGridView vezérlőelem fontosabb tulajdonságai

Az egyes oszlopok formázásához nyissuk meg a vezérlőelem menüjét, majd válasszuk az *Edit Columns* parancsot. Itt rengeteg tulajdonságot állíthatunk be. Módosíthatjuk az oszlopok sorrendjét, törölhetünk, illetve felvehetünk további oszlopokat. A kiválasztott oszlopra felülírhatjuk a táblázat tulajdonságainál megadott értékeket.

A *Frozen* tulajdonság *True* értéke azt jelöli, hogy az oszlop a vízszintes gördítősáv használatakor is látható marad a táblázat bal szélén. A *SortMode* tulajdonság *NotSortable* értéke esetén a felhasználó nem tudja rendezni a sorokat. Kísérletezzünk a többi tulajdonság módosításával! Figyeljük meg a hatásukat a táblázat megjelenésére.



Az oszlopok formázását segítő párbeszédablakok

Jelzés a felhasználónak

A hosszan tartó eljárások, ciklusok során célszerű a felhasználónak jelezni, hogy a program rendben működik, nem került végtelen ciklusba. Az ablak frissítésére alapértelmezés szerint ugyanis csak az eseménykezelő eljárások befejezésekor kerül sor.

Hozzunk létre egy Windows-alkalmazást egy címkével (*Label1*) és egy parancsgombbal (*Button1*). A parancsgomb *Click* eseménykezelőjébe írjuk be a következő utasításokat:

```
Label1.Text = ""
For I = 1 To 100
  For J = 1 To 20000000
    ' üres ciklus
  Next
  Label1.Text &= I & vbNewLine
Next
```

A belső, üres ciklus csak lassítja a végrehajtást. Figyeljük meg, hogy egy darabig semmi látható nem történik, majd a számok szinte egyszerre kerülnek ki a címkére.

A vezérlőelemek frissítése

A hosszas műveletsor alatt az eredményeket megjelenítő vezérlőelem frissítésével érhetjük el a részeredmények megjelenítését. A frissítéshez hívjuk meg a vezérlő *Refresh* metódusát:

```
Label1.Text &= I & vbNewLine
Label1.Refresh()
```

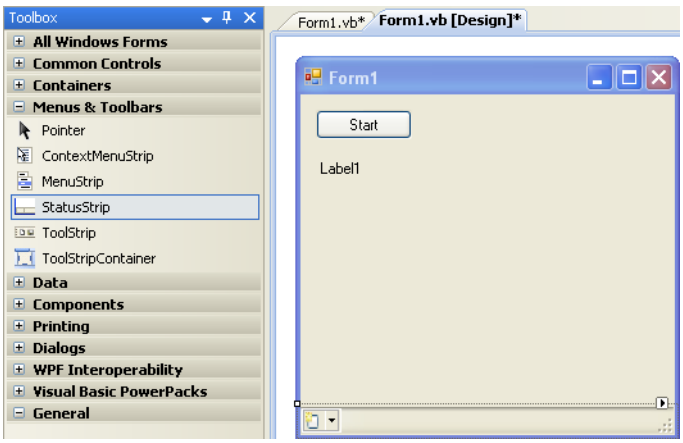
Ha az előző program külső ciklusát kiegészítjük a frissítéssel, akkor a számok – némi késleltetéssel – folyamatosan megjelennek a címkén.

Több vezérlőelem módosítsa esetén a programablakot is frissíthetjük:

```
Me.Refresh()
```

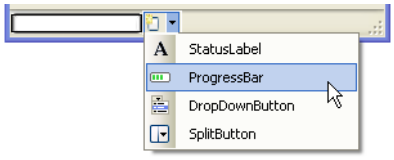
Folyamatjelző az állapotsorban

A vezérlőelemek, illetve az ablak frissítése – főleg sok kiírás esetén – jelentősen lelassítja a program futását. Helyette célszerű az állapotsorban tájékoztatni a felhasználót a végrehajtás folyamatáról. Az állapotsor megjelenítéséhez helyezzünk az űrlapra egy *StatusStrip* vezérlőt, amit az eszköztár *Menus & Toolbars* csoportjában találunk. A vezérlő az űrlap alsó széléhez kerül. Helyzetét a *Dock* tulajdonság módosításával befolyásolhatjuk.



Az állapotsorral kiegészített űrlap

Az állapotsor különböző vezérlőelemeket tartalmazhat. A szokásos folyamatjelző sáv megjelenítéséhez az állapotsor legördülő menüjéből válasszuk a *ProgressBar* elemet. Méretét a *Size* tulajdonsággal módosíthatjuk.



Folyamatjelző kiválasztása az állapotsor vezérlőelem legördülő menüjéből

A tulajdonságok ablakban adjuk meg a *Minimum* és a *Maximum* tulajdonság értékét, illetve a *Step* lépésközt, amellyel a módosítás gyakoriságát szabályozhatjuk. A folyamatjelző helyzetét a programban a *Value* tulajdonság segítségével állíthatjuk be.

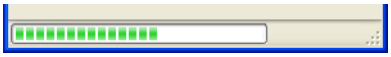
Fenti példaprogramunkban a következő értékeket választottuk:

Size = 200, *Mininum* = 0, *Maximum* = 100, *Step* = 5

A külső ciklusban elhelyeztük a

```
ToolStripProgressBar1.Value = I
```

utasítást. A program futása során megfigyelhetjük az állapotjelző működését. A frissítéshez nincs szükség a *Refresh* metódus meghívására.



Folyamatjelző az állapotsorban

A vezérlőelem eredeti nevét célszerű átírni rövidebb és kifejezőbb azonosítóra.

Ha szöveget akarunk írni az állapotsorba, akkor a *StatusStrip* legördülő menüjéből válasszuk a *StatusLabel* elemet.

Példaprogramunkban a ciklusváltozó kiírásához adjuk meg a címke *Text* tulajdonságát, majd hívjuk meg az állapotsor (!) *Refresh* metódusát:

```
ToolStripStatusLabel1.Text "I = " & I
StatusStrip1.Refresh()
```

Megjegyezzük, hogy az állapotsor egyszerre több vezérlőelemet is tartalmazhat.

Tartalom