

14.3.1. Jöjjenek újból a LED-ek...

Nézzünk egy mintapéldát a tömbökkel kapcsolatban. A példához használjuk újból az **Explorer 16** fejlesztőpanelen található LED-sort. Az új programunk a LED-eket úgy fogja be- és kikapcsolni, mintha egy zenedobozban lévő henger mintázatát követné. A „henger domborzatának mintáját” egy tömbben fogjuk eltárolni, amit a főprogramunk körbe-körbe fog majd „lejátszani”.

14.5. mintaprogram

```
#include <p24fj128ga010.h>
_CONFIG1( JTAGEN_OFF & FWDTEN_OFF )
_CONFIG2( POSCMOD_HS & FNOSC_PRI )

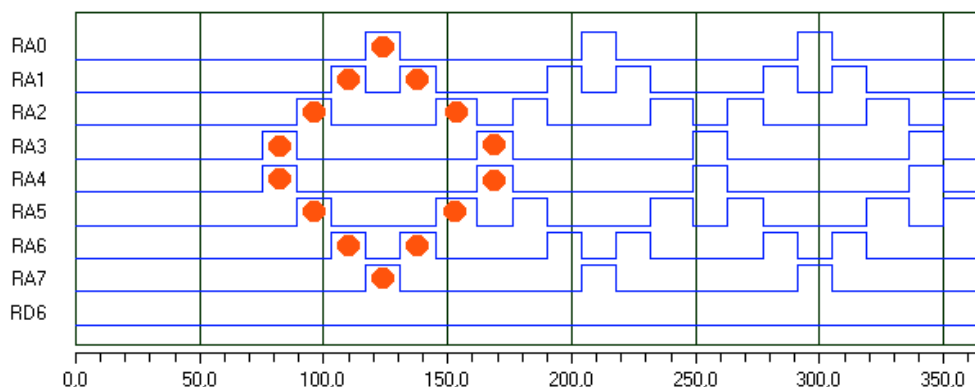
main ( )
{
    unsigned char chLedKep[] =      // A ledmintát tartalmazó táblázat
    {
        0b00011000,
        0b00100100,
        0b01000010,
        0b10000001,
        0b01000010,
        0b00100100
    };
    unsigned int wIndex=0;          // Táblázat elemkijelölő indexe
    unsigned int wTombElemszam = 6; // Táblázat elemszama

    TRISA = 0xFF00;                 // PORTA alsó nyolc lába kimenet lesz.
    LATA = 0x0000;                 // PORTA törlése

    while(1)                       // Végtelen ciklus
    {
        LATA = chLedKep[wIndex++]; // ledkép kihelyezése A ledsorra
        if (wIndex >= wTombElemszam)
        {
            // A tömb indexmutatójának törlése
            wIndex = 0;             // túlcsoordulás esetén
        }
    }
}
```

Szimulátor segítségével futtassuk le a programot. Nyissuk ki a logikai analízátort, és adjuk hozzá a megjelenítendő portlábakhoz az A port alsó nyolc bitjét. Megjelent a sarkára állított négyzet?

14. fejezet: Vissza a változókhoz



14.5. ábra
Tárolt minta megjelenése a LED-soron

Érdeemes a **watch** ablakban a tömbünket is hozzáadni a listához, vagy kinyitni a **View** → **Locals** menüpont alatt található **locals** ablakot (az ablakban az összes lokális változó megjelenik). Mind a két ablakban kinyitható a tömbváltozó, és az egyes elemek értéke egyenként megtekinthető.

Address	Symbol Name	Value
080A	chLedKep	
080A	[0]	00011000
080B	[1]	00100100
080C	[2]	01000010
080D	[3]	10000001
080E	[4]	01000010
080F	[5]	00100100
0808	wIndex	0x0003
0806	wTombElemszam	0x0006

14.6. ábra
A chLedKep tömb elemei a Locals ablakban

Most helyezzük vissza a már megszokott késleltető rutinunkat a programba, de most az időkorlátot 2000-re csökkentjük. Töltsük le a programunkat a mikrokontrollerbe és indítsuk el. Programunk tesztelését az eddigiektől eltérő módon érdemes elvégezni. A teszteléshez fogjuk a kezünkbe a fejlesztőpanelt, és kezdjük a szemünk előtt jobbra-balra mozgatni. A mozgatást olyan gyorsan végezzük, hogy a szemünk előtt is megjelenjen a szimulátor képernyőjén már látott alakzat. A programot érdemes sötétben kipróbálni, mert a LED-ek fényereje nem olyan erős, hogy nappali fény mellett is megjelenjen a kivetített kép.

14.6. mintaprogram

```
#include <p24fj128ga010.h>
_CONFIG1( JTAGEN_OFF & FWDTEN_OFF )
_CONFIG2( POSCMOD_HS & FNOSC_PRI )

main ( )
{
    unsigned char chLedKep[] = // A ledmintát tartalmazó táblázat
```

```

{
    0b00011000,
    0b00100100,
    0b01000010,
    0b10000001,
    0b01000010,
    0b00100100
};
unsigned int wIndex=0;           // Táblázat elemkijelölő indexe
unsigned int wTombElemszam = 6; // Táblázat elemszama
int iIdo;                       // Késleltetéshez használt változó
TRISA = 0xFF00;                 // PORTA alsó nyolc lába kimenet lesz.
LATA = 0x0000;                 // PORTA törlése

while(1)                        // Végtelen ciklus
{
    LATA = chLedKep[wIndex++]; // ledkép kihelyezése A ledsorra
    if (wIndex >= wTombElemszam)
    {
        // A tömb indexmutatójának törlése
        wIndex = 0; // túlsordulás esetén
    }
    for(iIdo=0; iIdo<2000; iIdo++)
    {
        Nop(); // Késleltetés
    }
}
}

```

Mielőtt a program tesztelése során az ICD2 programozónk a földön végezné, érdemes az ICD2-t debug üzemmódból programozó üzemmódba átváltani. Válasszuk ki a **Programmer** → **Select Programmer** → **MPLAB ICD2** menüpont segítségével az ICD2-t programozóként. Ha így töltjük le a programunkat, akkor végleges változatként tudjuk futtatni, azaz ICD2 nélkül is működni fog a programunk, így már ki lehet húzni a fejlesztőpanelből az ICD2 kábelét. Most már csak arra kell vigyáznunk, hogy a tápkábelt ne rántsuk ki a falból!

14.3.2. Const előtag

A `chLedKep` változónk jelenleg két helyen foglalja a mikrokontroller memóriáját. A tömb kezdőértékei a programmemóriába (flashbe) kerülnek, hogy a tömb inicializálásakor az általunk megadott kezdőértékeket a program be tudja másolni az adatmemóriába (RAM-ba). A C nyelvben a változók alapesetben az adatmemóriában jönnek létre. Mivel a tömbök is változók, ezért a RAM-területen jönnek létre, hogy a program futása közben a tömb elemeinek értékei módosíthatók legyenek. Sok esetben, mint az utolsó mintapéldában is, nincs szükségünk arra, hogy a tömbünk módosítható legyen, ilyenkor érdemes kihasználni a konstans változók létrehozásának lehetőségét.

A Microchip C30 fordítójában, a `const` előtag használatával, a változó a programmemóriában jön létre, így nem foglalja az amúgy is szűkös adatmemóriánkat. Az eredeti C nyelv nem különbözteti meg az adat- és programmemória fogalmát, így általános esetben a `const` előtag csak azt jelenti, hogy a fordító módosítási kísérletek esetén fordítási hibát ad, nem engedi a programunkat lefordítani. A `const` előtaggal rendelkező változóknak szigorúan kezdőértéket kell adni a deklarálásukkor, mert utána nincs lehetőségünk a konstans változók értékeinek módosítására.

Az eredeti ANSI C nyelv nem definiálja a programmemória fogalmát, mert a fordító neumanni architektúrára készült, ezért nem különbözteti meg a program- és adat-

14. fejezet: Vissza a változókhoz

memória fogalmát. Ebből következik, hogy a harvardi architektúrára készült C fordítóknak a nyelvet ki kellett bővíteniük ahhoz, hogy különböző adatterületre tudjanak változót elhelyezni. Annak a megadása, hogy egy változó ne az adat-, hanem a programmemóriába kerüljön, implementációfüggő, különböző C fordítóknál más a szintaktikája.

A `const` előtag segítségével deklarált változókat a C30 fordító, a PIC 16 bites kontrollereiben megtalálható **PSV** (*Program Space Visibility*) területre helyezi. A következő sor egy konstans változót deklarál:

```
const float pi = 3.141593;
```

Az előző mintapéldában használt tömbünket a következő deklaráció segítségével tudjuk konstans változóként a programmemóriába elhelyezni:

```
const unsigned char chLedKep[] = // A ledmintát tartalmazó táblázat
{
    0b00011000,
    0b00100100,
    0b01000010,
    0b10000001,
    0b01000010,
    0b00100100
};
```

14.3.3. Többdimenziós tömbök

A C nyelvben lehetőségünk van többdimenziós tömböket is létrehozni. A deklarálás általános alakja a következő:

```
típus név[elemszám_N.Dimenzió]...[elemszám_2.Dimenzió][elemszám_1.Dimenzió];
```

Például a következőképpen lehet létrehozni egy négyszer hármastömböt:

```
int tomb[4][3];
```

A többdimenziós tömb egyes elemeit hasonlóképpen érhetjük el, mint az egydimenziós társaik esetében tettük, csak most az összes dimenziót meg kell adnunk. A tömb elemeinek indexelése többdimenziós tömbök esetén is 0-tól kezdődik, és a (tömb adott dimenziójának elemszáma-1)-ig tart.

```
tomb[2][1] = 3; // A tömb (2;1) indexű elemének értéke
                legyen egyenlő hárommal.
a = tomb[3][2]; // Az a változó értéke legyen egyenlő
                a tömb (3;2) indexű elemének értékével.
```

Többdimenziós tömbök deklarálásakor is lehetőségünk van kezdőérték adására. Az elemeket egymás után is meg lehet adni, nem muszáj belső zárójeleket használni, de ha kizárójelezzük a dimenziókat, akkor sokkal áttekinthetőbb lesz a programkódunk.

```
int tomb[4][3] = { { 1 , 2 , 3 },
                  { 4 , 5 , 6 },
                  { 7 , 8 , 9 },
                  {10 , 11 , 12 } };
```

PIC programozás C nyelven

Az előbb deklarált tömb elemei úgy helyezkednek el egymás után a memóriában, ahogy azt a 14.7. ábra mutatja.

tomb[0][0] = 1	tomb[0][1] = 2	tomb[0][2] = 3
tomb[1][0] = 4	tomb[1][1] = 5	tomb[1][2] = 6
tomb[2][0] = 7	tomb[2][1] = 8	tomb[2][2] = 9
tomb[2][0] = 10	tomb[3][1] = 11	tomb[3][2] = 12

14.7. ábra
Tömb elemeinek elhelyezkedése a memóriában

Kezdőérték adásával összekötött többdimenziós tömbdeklaráció esetén is megtehetjük, hogy nem adjuk meg az összes dimenzió elemszámát. Ilyen esetben elhagyhatjuk a legmagasabb dimenzió elemszámát, mint ahogy azt a következő példa is mutatja.

```
int tomb[][3] = { { 1, 2, 3 },
                  { 4, 5, 6 },
                  { 7, 8, 9 },
                  { 10, 11, 12 } };
```

14.3.4. Karakterláncok

A C nyelvben, ellentétben sok más nyelvvel, nincs külön string típusú változó. A C nyelv a karakterláncokat karaktertömbök formájában tárolja. A karaktertömbök megegyeznek a hagyományos egydimenziós tömbökkel (matematikusok kedvéért: vektorokkal), de egy plusz kikötéssel rendelkeznek: A tömb utolsó elemének **EOS (End of String)** ASCII karakternek kell lennie (**NULL** karakter), amelynek az értéke bináris **nulla**.

Egy karaktertömbben az egyes karakterek ASCII kódja kerül eltárolásra. Ha egy karakter ASCII kódját szeretnénk megadni, akkor azt két egyes aposztróf (') között, az adott karakter beírásával tudjuk megadni. Például, ha a z karakter ASCII kódjára van szükségünk, akkor azt a 'z' kifejezés segítségével tudjuk megadni. Az alap ASCII kódtábla csak hétbites, 128 karaktert tartalmaz. A felső 128 karaktert is használó ASCII táblákat kiterjesztett ASCII tábláknak hívjuk. A kiterjesztett ASCII táblákat már ritkán használjuk, mert a helyébe lépett a UNICODE karakterábrázolás.

A 14.8. ábra az alap ASCII táblázat egyes karaktereinek hexadecimális értékeit mutatja. Az oszlopok tetején lévő számok a nagyobb, a sorok elején lévő számok a kisebb helyi értékű számjegyek. A táblázatból kiolvasható a 'z' karakter értéke 0x7A.

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(8	H	X	h	x
9	HT	EM)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

14.8. ábra
ASCII karaktertáblázat

14. fejezet: Vissza a változókhoz

Üres karaktertömböt a már megismert tömbdeklarációval lehet létrehozni:

```
char szSzoveg[8]; // 7 karakter tárolására, az utolsó karakternek
// bináris nullának (NULL) kell lennie!
```

A karaktertömb deklarálásakor lehetőségünk van kezdőérték adására is.

```
char szSzoveg[] = {'C', ' ', 'n', 'y', 'e', 'l', 'v', 0};
```

A nyelv egy egyszerűbb formát is biztosít a karakterláncok megadására. A nullával záródó karaktertömböket idézőjelek (") között is megadhatjuk.

```
char szSzoveg[] = "C nyelv"; // Az így létrehozott tömb is
// nyolc bájt hosszúságú
```

A karakterláncok megadásakor bizonyos vezérlő karaktereket adhatunk meg, fordított törvonal után. A 14.9. ábra által felsorolt karakterek főleg a terminálprogramokkal való kommunikációkor kapnak szerepet.

Jelölés	Beillesztett vezérlőkarakter
\a	csengő karakter
\b	visszaléptetés karakter
\f	lapdobás
\n	új sor (soremelés)
\r	kocsivissza
\t	tabulátor karakter
\v	függőleges tabulátor

Jelölés	Beillesztett vezérlőkarakter
\e	escape karakter (csak gcc)
\\	backslash karakter
\'	aposztróf
\"	idézőjel
\?	kérdőjel
\ooo	ooo oktális kódú karakter
\xhhh	hhh hexadecimális kódú karakter

14.9. ábra
Vezérlőkarakterek kódjai

Hivatalosan nem jelenik meg a C30 fordító dokumentációjában, de a fordító képes kezelni a UNICODE karaktereket is. A UNICODE karakterek 16 bit hosszúságúak, ezért ezeket unsigned short típusként kell definiálni. Ha az stdlib.h fejlécfájlományt is betöltjük, akkor pedig a szokásos wchar_t származtatott típust is használhatjuk. (A származtatott változótípusról a 15.1. fejezetben lesz szó.) A UNICODE karaktereket és karakterláncokat „L” előtaggal kell ellátni. Sajnos konstansként csak ékezet nélküli karaktereket lehet megadni, mert az ékezetes karaktereket a fordító nem tudja lefordítani.

```
unsigned short wCh = L'z';
unsigned short wszSzoveg[] = L"C nyelv";
vagy az #include<stdlib.h> után használva:
wchar_t wCh = L'z';
wchar_t wszSzoveg[] = L"C nyelv";
```

14.3.5. Készítsünk fényűságot!

A következő mintapéldában az utoljára megismert többdimenziós tömböket és a karakterláncokat is ki fogjuk használni. Az új program az előző programunk folytatása, habár a teljes belső magot lecseréljük, csak az alapötletet fogjuk megtartani. Kirajzoltuk a LED-

PIC programozás C nyelven

sorra az előző programunk élére állított négyzeteket. A fejlesztőpanel jobbra-balra mozgásával a négyzetek meg is jelentek a szemünk előtt.

A mostani program nemcsak négyzeteket fog majd kirajzolni a mozgó LED-sorra, hanem egy általunk megadott szöveget is. Nézzük, mire van szükségünk ahhoz, hogy elkészítsük a fényújságprogramunkat.

- Először szükségünk van egy karaktermintákat tartalmazó táblázatra. A `chLedKep` változó egy kétdimenziós tömb, amely minden egyes dimenzióban egy betű bitmintáját tartalmazza. Egy karakterminta öt bájtól áll. Az első négy bájt az adott karakter bitmaszkját tartalmazza, az ötödik bájt nulla, ami az egyes betűk közötti elválasztáshelyet biztosítja. Az egyszerűség kedvéért most csak az ábécé első három nagybetűjének bitképét készítjük el. A táblázatot mindenki a saját kreativitása szerint folytassa.
- A `wKarakterHossz` változó egy konstans ötös értéket tartalmaz, egy betű karaktermintájának hosszát adja meg.
- A `szSzoveg` változó egy nullával záródó karakterláncot tartalmaz. Ez a szöveg fog a programunk által a LED-sorra kirajzolódni.
- A `wStrIndex` változó a `szSzoveg` tömb aktuális karakterére mutat.
- A `wIndex` változó az aktuális karakter aktuális fénymintájára mutat.
- Az `iIdo` változót a késleltető rutin használja.

A program a LED-sor inicializálásával kezdődik. A programunk magját képező ciklus a `szSzoveg` karakterlánc egyes karakterein megy végig. A belső ciklus az előző ciklus által kiválasztott betű képét teszi ki a kimenetre.

A belső ciklus magját egy utasítás képezi (a késleltetési rutinon kívül). Érdekes ezt az utasítást egy kicsit tüzetesebben átnézni.

```
LATA = chLedKep[szSzoveg[wStrIndex]-'A'][wIndex];
```

A `chLedKep` tömb alsó dimenziójának indexét a `wIndex` változó határozza meg. A felső dimenziót a `szSzoveg[wStrIndex]-'A'` kifejezés határozza meg. A `szSzoveg[wStrIndex]` az éppen kiírás alatt álló karakter ASCII kódját adja vissza. Ebből az értékből azért kell az 'A' karakter értékét kivonni, mert a `chLedKep` tömbben az ábécé első három betűje van csak megvalósítva.

14.7. mintaprogram

```
#include <p24fj128ga010.h>
_CONFIG1( JTAGEN_OFF & FWDTEN_OFF )
_CONFIG2( POSCMOD_HS & FNOSC_PRI )

main ( )
{
    const unsigned char chLedKep[3][5] = // Karaktermintákat
    {
        {
            // tartalmazó táblázat
            // A betű
            0b11111100,
            0b00010010,
            0b00010010,
            0b11111100,
            0b00000000
        },
        // B betű
        0b11111110,
        0b10010010,
        0b10010010,
        0b01101100,
        0b00000000
    }
}
```

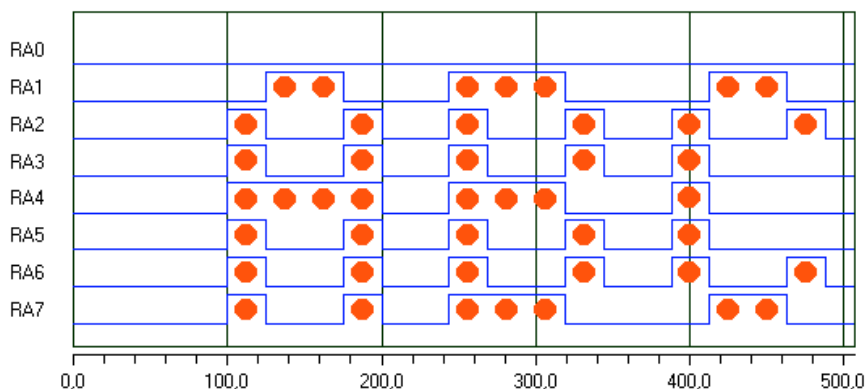
14. fejezet: Vissza a változókhoz

```
    },
    {
        // C betű
        0b01111100,
        0b10000010,
        0b10000010,
        0b01000100,
        0b00000000
    }
};
const unsigned int wKarakterHossz=5; // Egy karakter hosszúsága
const char szSzoveg[]="ABC"; // Kiírásra kerülő szöveg
unsigned int wStrIndex=0; // Karakterkijelölő index
unsigned int wIndex=0; // Fénymintát kijelölő index
int iIdo; // Késleltetéshez használt változó

TRISA = 0xFF00; // PORTA alsó nyolc lába kimenet.
LATA = 0x0000; // PORTA törlése

while(1) // Végtelen ciklus
{
    // Addig lépkedjünk karakterenként végig a karakterláncon,
    // amíg véget nem ér, azaz nullás karakter nem jön.
    for( wStrIndex=0; szSzoveg[wStrIndex] > 0; wStrIndex++ )
    {
        // Jelezzük ki a ledsoron a kiválasztott karaktert
        for( wIndex=0; wIndex < wKarakterHossz; wIndex++ )
        {
            // Fényminta kihelyezése a ledsorra
            LATA = chLedKep[szSzoveg[wStrIndex]-'A'][wIndex];
            for(iIdo=0; iIdo<2000; iIdo++)
            {
                Nop(); // Késleltetés
            }
        }
    }
}
}
```

A késleltető rutin nélkül program szimulációjának eredményét a 14.10. ábra mutatja.



14.10. ábra
Fényújság szimulációjának eredménye

14.4. MUTATÓARITMETIKA

A mutatókkal foglalkozó fejezet részben csak értékeket adtunk mutatóknak vagy hivatkoztunk mutatókkal. Ezeket a műveleten kívül lehetőségünk van a mutatókkal bizonyos matematikai műveleteket is elvégezni. A mutatóaritmetikai műveletek három csoportra oszthatók:

- Mutatóhoz egész szám hozzáadása vagy kivonása:
 - pointer + int
 - pointer - int
- Mutató inkrementálása vagy dekrementálása:
 - pointer++, pointer--
 - ++pointer, --pointer
- Két mutató kivonása egymásból:
 - pointer - pointer

A mutatók alapegysége a mutató típusának bájtban értelmezett nagysága. Ha egy mutató értékét növeljük vagy csökkentjük, akkor a mutató alapegységével nő vagy csökken. Például egy int típusú mutatóhoz hozzáadunk egyet, akkor az értéke kettővel nő, mert az int típus nagysága (memóriaigénye) kettő bájt.

```
int *ptr;           // int típusú változóra mutató
                  // Az int típus memóriaigénye 2 bájt
ptr++;            // a ptr értéke 1×2=2 bájttal nő
ptr = ptr + 3;    // a ptr értéke 3×2=6 bájttal nő
```

Az általános típusú mutató (void*) alapegysége 1 bájt. A void* mutató segítségével bájtonként elérhetjük a teljes memóriaterületet.

Két mutató különbsége csak ugyanolyan típusú mutatók között van értelmezve, eredménye a két mutató címértékének különbsége lesz, de nem bájtban, hanem az adott változó bájtban mért alapegységében értelmezve. Máshogy megfogalmazva: a két mutató különbsége azt mondja meg, hogy hány változónyira van egymástól a két mutató által mutatott két változó.

14.5. KAPCSOLAT A TÖMBÖK ÉS A MUTATÓK KÖZÖTT

A C nyelv a tömböket és a mutatókat hasonlóképpen kezeli. Ha nagyon sarkítva fogalmazzunk, akkor azt is mondhatjuk, hogy igazából nincs különbség a mutatók és a tömbök között. A tömb típusú változók nevét (szögletes zárójelek nélkül) a tömb első elemére mutató pointerként kezeli a fordító. Ahhoz, hogy jobban megértsük az összefüggést, nézzünk egy példát:

14.8. mintaprogram

```
main ( )
{
    int tomb[3]={10, 20, 30}; // 3 elemű, int típusú tömb
    int *ptr;                // int típusú mutató
    int a, b, c, x, y, z;

    a=*tomb; // tomb 0. indexű elemének értékével tér vissza
    b=tomb[1]; // tomb 1. indexű elemének értékével tér vissza
    c=(tomb+2); // tomb+2 által mutatott memóriaterület értékével tér vissza

    ptr = tomb; // ptr mutasson a tomb-re

    x=*ptr; // ptr által mutatott memóriaterület értékével tér vissza
    y=ptr[1]; // ptr (tömb) 1. indexű elemének értékével tér vissza
    z=(ptr+2); // ptr+2 által mutatott memóriaterület értékével tér vissza
}
```

14. fejezet: Vissza a változókhoz

A programot lefuttatva a változók értékei a következőképpen alakulnak:

Address	Symbol Name	Value
0814	tomb	
0814	[0]	10
0816	[1]	20
0818	[2]	30
0812	ptr	0x0814
0810	a	10
080E	b	20
080C	c	30
080A	x	10
0808	y	20
0806	z	30

14.11. ábra

Az a, b, c, x, y, z változók értékei tömbbel és mutatókkal történő értékadások után

Az a, b, c változók értékét a tomb változó, amíg az x, y, z változók értékeit a ptr mutató segítségével adtuk meg. Az a, b, c változók értékei rendre megegyeznek az x, y, z változók értékeivel, holott különböző módon állítottuk be az értékeiket. A ptr = tomb; utasításból jól látható, hogy a tomb típusú változó egyben mutató is. Az utasítás végrehajtása után a tomb és a ptr változó is a 0x0814 memóriacímre mutat. Az a, b, c és az x, y, z változók értékadásánál megfigyelhető, hogy a tomb típusú változók és a mutatók is egyformán kezelhetők. A mutatókat lehet tömbindexelő operátorokkal címezni, míg a tomb típusú változókra is lehet címaritmetikát végezni.

A tömbindexelő operátor és a mutatók címaritmetikája között egy az egyes összefüggés írható fel:

```
pointer[egész] ≡ *(pointer + egész)
```

Karakterláncok estén is megvan a tömbök és a mutatók kettősége. A karakterláncokról szóló 14.3.4. fejezetben kétféleképpen deklaráltunk karakterláncot. A második deklarációról, amikor idézőjelek között adtuk meg a karakterlánc értékét (`char szSzoveg[] = "C nyelv";`), akkor valójában létrejött egy különálló karaktertömb, amelynek címét a szSzoveg (mint mutató) kapta meg. Ebből következik, hogy lehetőségünk van egy harmadik fajta karaktertömb deklarálására is. Ilyen esetben nem tömböt deklarálunk, hanem egy mutatót, ami a fordító által létrehozott karaktertömbre mutat.

```
char *szSzoveg = "C nyelv";
```

Az idézőjelek közé elhelyezett szöveg ideiglenes karaktertömbként is működni. Ezt a tulajdonságot akkor tudjuk kihasználni, ha egy függvény bemeneti paramétere karaktertömböt vár. Ilyen esetekben nem kell egy karaktertömböt külön létrehozni, hanem a függvény hívásakor, a paraméterek között egyenesen csak a karakterláncot kell megadni. (A függvények létrehozásáról és használatáról a 16.1. fejezetben lesz szó.)