A C programozási nyelv

- *Egy egyszerű C program*
- *A C program összetevőinek felépítése*
  - ***Deklarációk***
  - ***Blokk***
  - ***Függvénydefiníció***
- *A C program felépítése*
- *Be- és kiviteli függvények egyszerű alakja*
  - ***Adatbeolvasás formátumellenőrzéssel***
  - ***Egyetlen karakter beolvasása megjelenítés nélkül***
- *Cím, érték, mutató fogalma*

Egy egyszerű C program

```
#include <conio.h>        /* clrscr miatt */
#include <stdio.h>        /* printf, scanf miatt */
#define  PI  3.1415926
main()
{
  float r, kerulet, terulet;
  clrscr();
  printf( "Körjellemzök számítása\n\n" );
  printf( "A kör sugara (méter) = " );
  scanf( "%f" , &r );
  kerulet = 2*r*PI;
  terulet = r*r*PI;
  printf( "\nA kerület= %6.3f méter\n", kerulet );
  printf( "A terület= %6.3f négyzetméter\n", terulet );
}
```

A C program összetevőinek felépítése

- *Deklarációk* *(bármely sor elmaradhat, vagy többször ismétlődhet)*

```
#include <headerfájl-név.h>
#define <konstans, vagy függvénymakró>
typedef <típus> <újtípusnév>;          /* új típus definiálása */
<típus> <változóazonosító>;            /* változódefiniálás */
<típus> <függvénynév(paraméterek)>; /* fv deklaráció */
```

- *Blokk*

```
{
  <deklarációk>    /* lokálisak, blokkon belül érvényesek */
  <utasítások>     /* bármely utasítás lehet blokk */
}
```

A C program összetevőinek felépítése . .

- *Függvénydefiníció*

```
<típus> <függvénynév(paraméterek)>
<blokk>
```

- *A C program felépítése* *Megjegyzés: más felépítés is lehet*

```
<deklarációk>          /*globálisak, programon belül érvényesek */
main()                 /* a fő függvény  */
<blokk>                /* a fő függvény blokkja */
<függvénydefiníciók>   /* a deklarált fv-ek teljes megadása*/
```

Be- és kiviteli függvények egyszerű alakja

Adatbeolvasás formátumellenőrzéssel

scanf(<formátumsztring>, &vált1, &vált2,...);

A függvény a beolvasott adatot a formátumsztringben megadott típusú adatként értelmezi és konvertálja a változó típusára, majd elhelyezi a változó címére. A függvény visszatérési értéke a sikeresen beolvasott értékek száma (int), amelyet nem kötelező felhasználni. Az értékek elhelyezéséhez a változók címét kell megadni a & *operátorral.*

Pl.: **scanf**("%d" , &*egeszvaltozo*);
scanf("%f" , &*valosvaltozo*);
scanf("%u" , &*elojelnelkuli_egesz*);

Be- és kiviteli függvények egyszerű alakja . .

- *Adatkiírás formázott módon*

printf(<formátumsztring>, kifejezés1, kifejezés2,...);

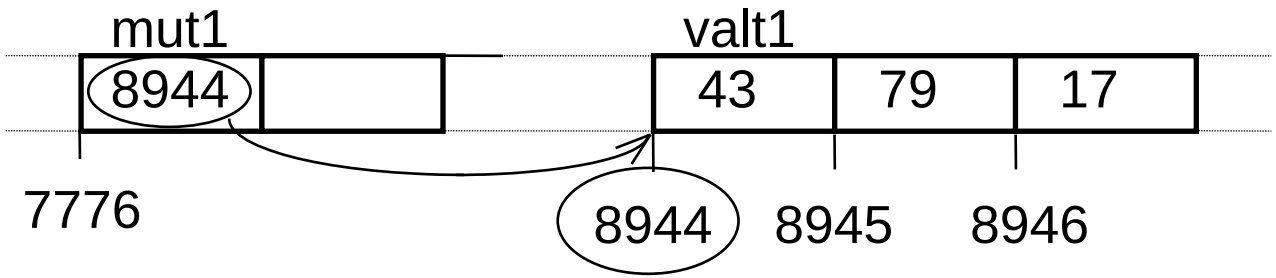
A kiíratás formátumsztringje tájékoztató szövegből és % jellel kezdődő konverziós előírásból állhat.

Cím, érték, mutató fogalma

*A memória egymást követő tárolóhelyek sorozata. Egy tárolóhelynek van sorszáma, amit **cím**nek nevezünk és van tartalma, amit az adott címen tárolt **érték**nek nevezünk. Mivel a címek számértékek és nehezen megjegyezhetők, ezért helyettesítjük azokat nevekkel, ezek a változók. Egy változó címe alatt az általa helyettesített címértéket értjük, értéke alatt pedig az adott címen tárolt értéket.*

*Amennyiben a cím tárolására akarunk változót, azaz egy másik címen található tárolóhelyet létrehozni, ezzel tulajdonképpen egy **mutató**t (pointert) hozunk létre, melynek számunkra az értéke fontos. A pointerváltozóban tárolt érték az a cím, amelyen az eredeti változónk található a memóriában.*

Cím, érték, mutató fogalma . .



A mut1 *pointerváltozónak értékül adhatjuk a* valt1 *változó címét a* & *operátor segítségével:*

mut1 = &*valt1* ;

ami után vált1 *tartalmát, értékét olvashatjuk, írhatjuk közvetett módon a* * (indirekció) *operátorral, mivel:*

valt1 == **mut1* .

Azaz valt1 *definiálása nélkül is tudjuk* mut1 *segítségével írni, olvasni a 8944-es című tárolóhely tartalmát, sőt egyszerűen* mut1 *tartalmának megváltoztatásával hozzáférhetünk pl. a 8945, vagy a 8946 tárolóhelyek tartalmához is.*

Kifejezések, operandusok, operátorok

- *Kifejezések a C nyelvben*
- *Operátorok típusai, kiértékelési sorrendje*
  - ***Aritmetikai operátorok***
  - ***Értékadó operátorok***
  - ***Léptető operátorok***
  - ***Relációoperátorok***
  - ***Logikai operátorok***
  - ***A feltételes operátor***
  - ***A címe és a mutató operátor***
  - ***A sizeof operátor***
  - ***A vessző operátor***
  - ***Típuskonvertáló operátor***

Operátorok típusai, kiértékelési sorrendje

(□) [□] □.□ □ ->□

!□ -□ ++□ □++ --□ □-- &□ *□ (típus)□ sizeof □

□*□ □/□ □%□

□+□ □-□

□<□ □<=□ □>□ □>=□

□==□ □!=□

□&&□

□||□

□?□:□

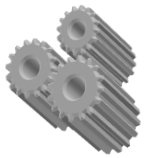
□=□ □+=□ □-=□ □*=□ □/=□ □%=□

□,□

Kiértékelés: jobbról balra ill. balról jobbra haladva. □ = operandus

- **Utasítások, elágazás- és ciklusszervezés**
  - ***C nyelvi utasítások***
    - *Kifejezés utasítás*
    - *Összetett, vagy blokk-utasítás*
    - *Elágazásszervező utasítások*
      - ***Az* if *utasítás***
      - ***Az* if else *szerkezet***
      - ***A* switch *többirányú elágaztató utasítás***
    - *Ciklusszervező utasítások*
      - ***A* while *ciklusszervező utasítás***
      - ***A* for *ciklusutasítás***
      - ***A* do while *ciklus***

C nyelvi utasítások



A C nyelvben is a

program = adatszerkezetek + algoritmusok

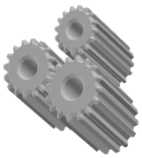
képlet igaz, ahol az algoritmusokat utasítások sorozatával adjuk meg.

Kifejezés utasítás

Bármelyik kifejezésből utasítás lesz, ha pontosvesszőt teszünk utána.

Pl.: *valt1 = a* +25 ; /* *értékadó utasítás* */

C nyelvi utasítások . .



Összetett, vagy blokk-utasítás

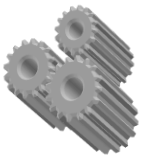
```
{
    <deklarációk és definíciók>
    <utasítások>
}
```

Mind a deklarációk, definíciók, mind az utasítások elmaradhatnak.

Összetett utasítás alkalmazandó olyan helyen, ahol formailag csak egy utasítás állhat, de több utasítást akarunk elvégeztetni.

Láthatóság: *Az összetett utasításban deklarált ill. definiált objektumok csak a blokkon belül láthatók, hivatkozhatók.*

C nyelvi utasítások . .



Elágazásszervező utasítások:

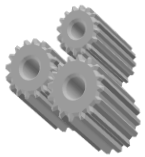
if, if else, switch, goto

Az if utasítás *alkalmas egy feltételtől függően egy programrész végrehajtására, vagy átugrására. Alakja:*

if (<kifejezés> **)**
<utasítás>

Ha a <kifejezés> *igaz (értéke nem nulla), akkor végrehajtódik az* <utasítás>*, egyébként az* <utasítás> *utáni programrészen folytatódik a program futása.*

Elágazásszervező utasítások

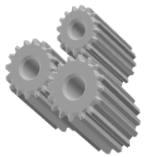


- **Az if else szerkezet** *kétirányú elágazást tesz lehetővé. Alakja:*

```
if ( <kifejezés> )
   <utasítás1>
else
   <utasítás2>
```

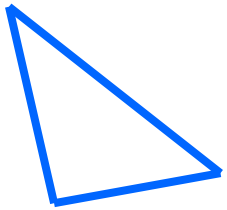
Ha a <kifejezés> *igaz (nem nulla) akkor az* <utasítás1>*, egyébként az* <utasítás2> *hajtódik végre, majd a programfutás az* <utasítás2> *utáni programrészen folytatódik.*

Elágazásszervező utasítások . .

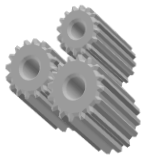


Példa az if else szerkezet *alkalmazására*

```
    /* Háromszög */
#include <stdio.h>
#include <conio.h>
main()
{
 unsigned int a,b,c;
 clrscr();
 printf("Háromszögtípus meghatározása\n\n");
 printf("Adja meg az oldalakat nem növekvö sorrendben ! \n");
 printf("A= "); scanf("%u",&a);
 printf("B= "); scanf("%u",&b);
 printf("C= "); scanf("%u",&c);
```



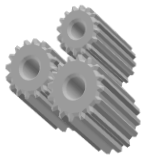
Elágazásszervező utasítások . .



Példa az if else szerkezet *alkalmazására ..*

```
if (a < b+c)
  if (b*b + c*c == a*a)
    if (b==c) printf("egyenlőszárú derékszögű ");
    else printf("derékszögű ");
  else if (a==c && c==b) printf("egyenlő oldalú (szabályos) ");
       else if (a==b || b==c) printf("egyenlőszárú ");
         else printf("általános ");
else printf("nem alkotnak háromszöget ");
printf("\n");
getch();
} /* Megj: egyenlő szárú derékszögű
     csak valós oldalhosszal teljesül.*/
```

Elágazásszervező utasítások . .

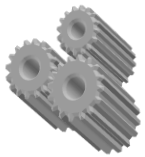


A switch többirányú elágaztató utasítás

Többirányú elágaztatás egymásbatokozott if else *szerkezetekkel is szervezhető, amint az előző példa mutatta. Azonban olyan esetekben, amikor egy egész jellegű kifejezés értékétől függően kell más-más programrészt végrehajtani, a* switch *utasítás áttekinthetőbb szerkezetet eredményez. Az utasítás alakja:*

```
 switch (  <egész jellegű kifejezés> )
{
  case <konstans_kifejezés1> : <utasítások>
  case <konstans_kifejezés2> : <utasítások>
   ...
  default : <utasítások>
}
```

Ciklusszervező utasítások



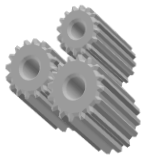
- *A ciklusszervező utasítások szolgálnak ugyanazon utasítások általában eltérő adatokkal történő többszöri végrehajtására. A C nyelvben két elöltesztelős (**while** és **for**) és egy hátultesztelős (**do while**) ciklusszervező utasítás használható, melyek futása még további feltétel nélküli vezérlésátadó utasításokkal (**goto**, **break**, **continue**) módosítható. Ezen utóbbi lehetőségek használata nem eredményez struktúrált programot.*

- **A while ciklusszervező utasítás**:

```
while ( <kifejezés> )
    <utasítás>
```

- *Minden egyes ciklusban előbb kiértékelésre kerül a <kifejezés> és csak akkor hajtódik végre a ciklus magját jelentő <utasítás>, ha a <kifejezés> igaz (nem nulla értékű). Ha a <kifejezés> hamis (értéke nulla), akkor a program futása a **while** utasítás utáni utasítással folytatódik.*

Ciklusszervező utasítások . .

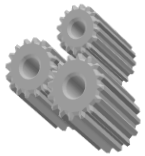


- *A **while** használata akkor célszerű, ha a ciklusmag előre ismeretlen számban - lehet, hogy egyszer sem - hajtódhat végre.*

Pl.:

```
#include <conio.h>        /* getch miatt */
#include <stdio.h>        /* printf, scanf miatt */
main()
{
  float adat, szumma = 0;
  while ( printf("\nVan adat? (I/N):") , getch() == 'I' )
  {
     printf("\nAdat=");
     scanf("%f", &adat );
     szumma += adat ;
  }
  printf("\nAz összeg= %f", szumma);
}
```

Ciklusszervező utasítások . .



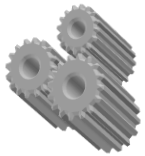
- **A for ciklusutasítás** *legelőnyösebben akkor használható, ha a végrehajtandó ciklusok száma előre ismert. A C nyelv* **for** *ciklusa sokféle alakot ölthet. Ez szerkezetéből is látható:*

```
for ( <kezdőért_kif>;<feltétel_kif>;<léptető_kif>)
      <utasítás>
```

Bármelyik kifejezés elmaradhat. Ha a <feltétel_ kif> *hiányzik, igaz értéket helyettesít a program.*

Működése: *legelőször egyszer kiértékelődik a* <kezdőért_kif>*, majd ciklusban kiértékelődik a* <feltétel_kif>*, végrehajtódik az* <utasítás> *és kiértékelődik a* <léptető_kif>*, mindaddig, amig a* <feltétel_kif> *igaz (értéke nem nulla). Ha a* <feltétel_kif> *nem igaz, a program futása a* **for** *ciklus utasítását követő program-utasításon folytatódik.*

Ciklusszervező utasítások . .

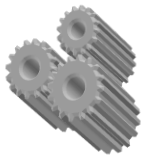


- Példa **for** ciklusutasításra

```
#include <stdio.h>
main()
{
   unsigned int  szumma, k ;
   for ( szumma = 0, k = 1 ; k <= 100 ; k++)
         szumma += k;
   printf("Számok összege százig= %u", szumma);
 }
```

1+2+3+4+5+6+7+8+9+10+11+12+ .. +92+93+94+95+96+97+98+99+100

Ciklusszervező utasítások . .

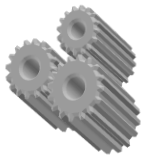


- **A do while ciklus** *előnyös alkalmazási területe az előre nem ismert számban, de legalább egyszer végrehajtódó ciklustörzsű ciklusok. Alakja:*

```
do
 <utasítás>
while ( <kifejezés> ) ;
```

A ciklusban először végrehajtódik az <utasítás>*, majd kiértékelődik a* <kifejezés>*. Ha a* <kifejezés> *igaz (nem nulla), akkor az előbbiek ismétlődnek. Ha a* <kifejezés> *nem igaz (értéke nulla), a programvégrehajtás a* **do while** *utasítás utáni utasításon folytatódik.*

Ciklusszervező utasítások . .

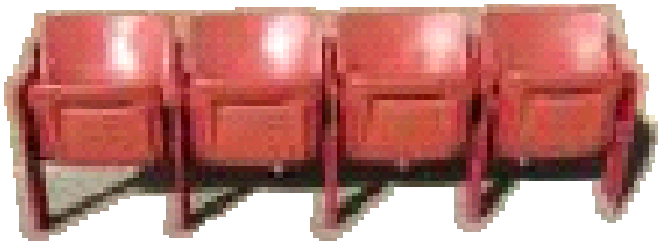


- Példa **do while** ciklusra

```
#include <stdio.h>
#include <stdlib.h>      /* randomize, random miatt */
main()
{
  int szam, tipp, n = 0;
  randomize(); szam = random(10); szam++;
  printf("\nGondoltam egy számot 1-10 között \n");
  do
   {
     printf("\nA tippje= ");
     scanf("%d",&tipp);
     n++;
   }
  while ( tipp != szam );
  printf("\n%d lépésben kitalálta!", n);
}
```

Egyméretű tömbök, vektorok

- *Vektorok definiálása*
- *Vektorok definiálása kezdőértékekkel*
- *Egyméretű tömbök és a mutatók*
- *Pointer-aritmetikai műveletek*
- *Példaprogram vektorhasználatra*
- *Szövegkezelés karaktervektorokkal*
- *Példaprogram karaktervektorokkal*



Vektorok definiálása

Az egydimenziós tömbök, vagy másnéven vektorok a memóriában egymást követően elhelyezkedő azonos típusú elemekből állnak.

A definiálás formája:

<típus> <azonosító> **[** <elemek_száma> **]** ;

Az <elemek_száma> *fordításkor kiértékelt egész konstans kifejezés.*

Egy adott tömbelemre a sorszámával (indexével) lehet hivatkozni, amely a 0 .. (<elemek_száma> -1) *tartományból választható.*

A tartományon kívüli indexmegadások nem okoznak hibajelzést és nehezen kideríthető hibákat eredményeznek. Emiatt célszerű az elemszámot (a vektor méretét) előre definiált konstanssal megadni.

Vektorok definiálása . .

Pl.:

```
...
#define ESZ 7
float vektor[ ESZ ] ;
int i ;
for ( i = 0 ; i < ESZ ; i++)  vektor[ i ] = 2*i ;
...
```

Nagyobb tömböket célszerű a programra nézve globális változókként megadni, a stack memória túlcsordulásának megelőzésére.

Definiálhatunk saját tömbtípust is:

Pl.:

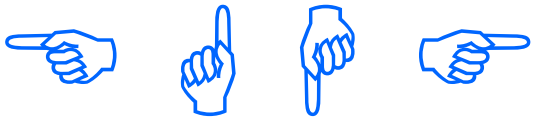
```
typedef float vekttip [ 7 ] ;
vekttip torpek_kora ;      /*  vekttip típusú vektor definiálása */
```

Egyméretű tömbök és a mutatók

*A mutatók a C nyelvben az általuk mutatott adat típusának megfelelő típusúak. Ha típus nélküli mutatót akarunk definiálni, a **void** típust kell megadni.*

Pl.:

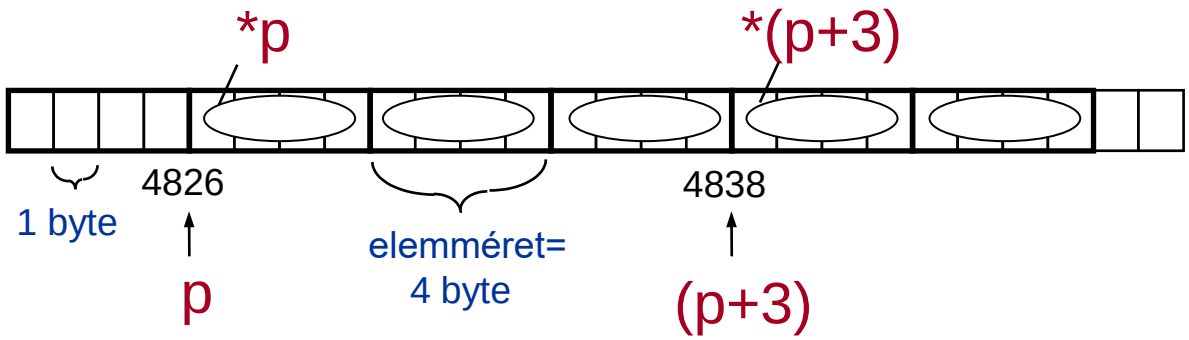
```
int * egeszremutato;
float * lebegopontosra_mutato;
double * mutdbl;
void * mutato;
```



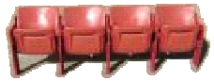
Pointer-aritmetikai műveletek

A (nem void) pointerekre a következő aritmetikai műveletek vannak értelmezve, melyek kifejezetten tömbelem-mutatóknál alkalmazhatóak előnyösen:

p++	*inkrementálás*
p - -	*dekrementálás*
p + k	*egész kifejezés értékének hozzáadása*
p - k	*egész kifejezés értékének levonása*
p1 - p2	*két mutató kivonása.*



Pointer-aritmetikai műveletek . .



A tömbök és a mutatók rokonságának bemutatására tekintsük a következő példát:

```
int vekt[ 6 ] , * p ;        /* p definiálatlan helyre mutat */
int elso, masodik, harmadik;
p = &vekt[ 0 ];              /*   vagy   p = vekt    */
elso = vekt[ 1 ];            /*   vagy   p[ 1 ]      */
masodik = vekt[ 2 ];         /*   vagy   *(vekt+2)   */
harmadik = vekt[ 3 ];        /*   vagy   *(p + 3)    */
```

*A C nyelvben a vekt[k] és a *(vekt + k) kifejezések egyenértékűek.*

A vekt tömbnév és a p mutató közti eltérés, hogy amíg p egy változó, a tömbnév egy konstans mutató, igy nem is adható neki érték.

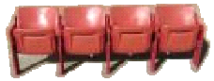
Példaprogram vektorhasználatra

1. alapalgoritmus: összegzés

```
#include <stdlib.h>      /* randomize(); random();  */
#include <stdio.h>
int vekt [ 20 ];
main()
{
    int i, szum;
    randomize();
    for ( i = 0; i < 20; i++)
        vekt [ i ] = random(31);
    for ( szum = 0, i = 0; i < 20; i++)
        szum += vekt [ i ];
    printf("Átlag= %f",szum / 20.);
}
```

5+7+11+23+30+5+9+13+22+13+21+17+4+2+0+9+1+11+7+18 = 228; 228/20.=11.4

Szövegkezelés karaktervektorokkal



A C nyelv nem ismer önálló szöveges típust, a szövegek tárolását karaktervektorokkal oldja meg.

A karaktervektorok között nincs közvetlen értékátadás, a sztringek átadását karaktermásoló ciklussal kell megoldani.

~~szoveg1 = szoveg2;~~

Egy karakterfüzér tárolására kijelölt memóriaterület hossza eltérhet az aktuálisan benne tárolt szöveg hosszától. A szöveg hosszának tárolása helyett a C nyelv a szöveg végét jelöli a \0 karakterrel (0 ASCII kód). A karaktervektor-változó definiálásakor ezt a karaktert is számba kell venni a méret megadásánál.

Szövegkezelés karaktervektorokkal . .

Példák:

```
char str20[21];
char s_teli[ 8]      = {'M', 'o', 'n', 'i', 't', 'o', 'r', '\0' };
char s_min[ ]        = {'E', 'g', 'é', 'r', '\0'};
char s_nagy[ 22 ] = "Joystick";
char s_smart[ ]      = "Modem";
```

A két utolsó esetben a \0 karaktert a fordító helyezi el a szövegek végén.

Amennyiben mutatóval definiáljuk a karakter vektort, ügyelni kell a mutatónak történő értékadással, mert felülírhatjuk az eredeti stringre mutató értéket.

Megszámlálás, kiválasztás alapalgoritmusok

- *Vektoron értelmezett 2. alapalgoritmus: a megszámlálás algoritmusa*
- *Vektoron értelmezett 3. alapalgoritmus: a kiválasztás algoritmusa*
  - *Példa egyszerű logikai feltételnek eleget tevő elem kiválasztására*
  - *Elemkiválasztás összetett logikai feltétel alapján*

A megszámlálás algoritmusa . .

```
#include <conio.h>
#include <stdio.h>
unsigned int vektor[100];
void main(void)
{
    unsigned int i, n, ketjegyu, haromjegyu ;
    clrscr(); puts("  Megszámlálás\n");
    printf("A számok összes darabszáma= ");
    scanf("%u", &n);
    /* Beolvasás:  */
    for (i = 0; i < n ; i++)
   {
      printf("A %u. szám = ", i +1);
      scanf("%u", &vektor[ i ]);        /* vagy scanf("%u",vektor+i); */
   }
```

A megszámlálás algoritmusa . .

```
/* Megszámlálás: */
for (ketjegyu = 0, haromjegyu = 0, i = 0; i < n; i++)
if (10<=vektor[ i ] && vektor[ i ]<=99)  ketjegyu++;
else if (100<=vektor[ i ] && vektor[ i ]<=999)
        haromjegyu++;
/* Eredmény kiíratása: */
printf("\n%u darab kétjegyű és %u darab háromjegyű \
        szám van.\n", ketjegyu, haromjegyu);
getch();
}
```

5, 7, 11 , 23 , 30 , 5 , 9 , 133 , 22 , 13 , 21 , 127 , 4 , 2 , 0 , 91 , 1 , 11 , 7 , 168

ketjegyu == 8; haromjegyu == 3

A minimális elem kiválasztás programja

```
#include <conio.h>
#include <stdio.h>
float vektor[100];
void main(void)
{
    unsigned int n, i, index ;
    float minertek ;
    clrscr(); puts("  Kiválasztás\n");
    printf("Elemek száma= "); scanf("%u", &n);
    /* Beolvasás a vektorba:  */
    for (i = 0; i < n; i++)
    {
        printf("Vektor[%u]= ", i +1);
        scanf("%f ", &vektor[ i ]);          /* vagy scanf(%f",vektor+ i ); */
    }
```

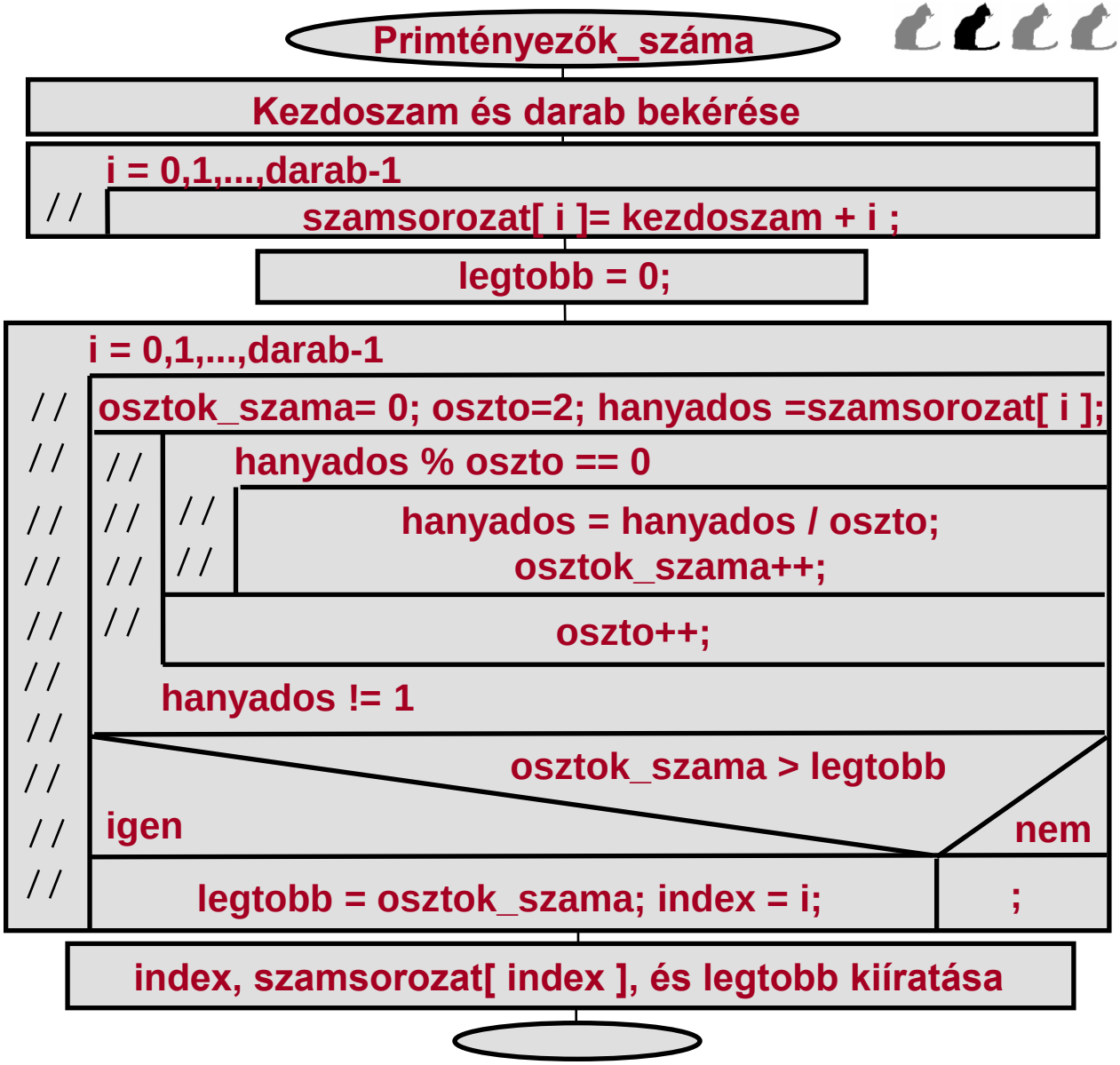
A minimális elem kiválasztás programja . .

```
/*  A minimális elem meghatározása:  */
for (minertek =  vektor [0], index =0, i =1; i < n; i++)
if (vektor[ i ]<minertek)
{
      minertek = vektor [ i ]; index= i ;
}
/* Az eredmény kiíratása:  */
printf("\nIndex= %u, minimális érték =%10.3f", index, minertek);
getch();
}
```

- ***Elemkiválasztás összetett logikai feltétel alapján***

Példa:

Adott egy egyesével növekvő számsorozat a kezdőszámmal és az elemek számával. Meghatározandó annak a számnak a sorszáma, amely a legtöbb prímtényezőre bontható fel.



Elemkiválasztás összetett logikai feltétel alapján . .

A primtényezők számát megadó program:

```
#include <conio.h>
#include <stdio.h>
unsigned int szamsorozat [100];
void main(void)
{
    unsigned int kezdoszam, darab, oszto, i, index,
                 osztok_szama, legtobb, hanyados;
    clrscr(); puts("Legtöbb prímtényező \n");
    printf("Kezdő szám="); scanf("%u",&kezdoszam);
    printf("\nSzámok darabszáma="); scanf("%u", &darab);
    for (i = 0; i < darab; i++)  szamsorozat [ i ]= kezdoszam+i ;
    legtobb= 0;
```

kezdoszam= 11; *darab*= 2; *szamsorozat* ☞ { 11, 12 }

Elemkiválasztás összetett logikai feltétel alapján . .

```
for (i = 0; i < darab; i++)
{     osztok_szama = 0; oszto = 2;  hanyados = szamsorozat [ i ];
      do
      {     while (hanyados % oszto == 0)
            {     hanyados |= oszto;                /* egészosztás! */
                  osztok_szama++;
            }
            oszto++;
      } while (hanyados != 1);
      if (osztok_szama> legtobb)
      {legtobb= osztok_szama; index= i ;
      }
}
printf("\nIndex= %u, a szám= %u, tényezők száma= %u\n",
            index, szamsorozat[index], legtobb);
getch();
}
```

Rendezés minimális elem kiválasztással

- *Az algoritmus működése*
- *A rendezés programja*

Az algoritmus működése

Rendezés: *célunk egy adott számsorozat növekvő sorrendbe rendezése.*

Példa:

Adott a rendezendő sorozat:
[7,8,4,9,1]

Rendezés után ezt várjuk:
[1,4,7,8,9].

A minimális elem kiválasztásával működő algoritmus jobbra követhető:

i =	*0*	*1*	*2*	*3*	*4*
j = 0	7	8	4	9	1
	4	8	7	9	1
	1	8	7	9	4
j = 1	1	8	7	9	4
	1	7	8	9	4
	1	4	8	9	7
j = 2	1	4	8	9	7
	1	4	7	9	8
j = 3	1	4	7	9	8
	1	4	7	8	9
Eredmény:	1	4	7	8	9

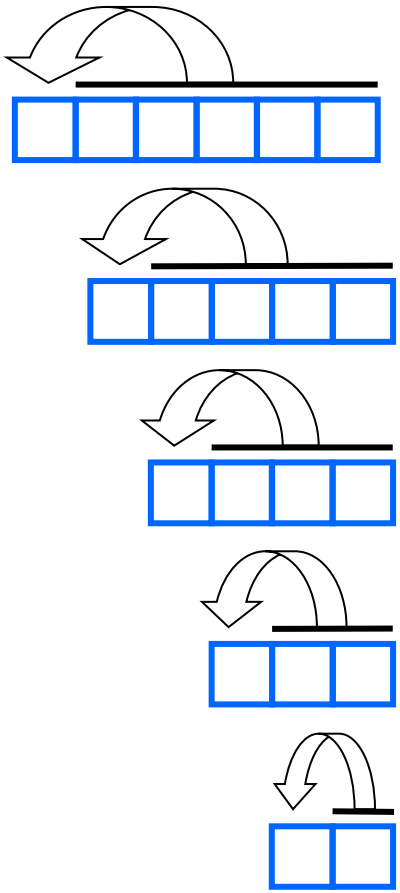
?
V[j] > V[i]

j. és i. elemek összehasonlítása

: már rendezett elemek

A rendezés egymásba ágyazott ciklusa:

Mivel egyre rövidülő részsorozatok minimális elemének az aktuális részsorozat első pozíciójába vitele a célunk, az egyetlen sorozaton működő előző algoritmust ciklusba kell ágyazni.



Figyeljük majd meg, hogy a belső for ciklus kezdőértékét a külső ciklusváltozó aktuális értéke határozza meg.

A rendezés programja:

```
#include <stdio.h>
#include <conio.h>
int V [100];
void main()
{int i, j, n, Segedvaltozo;
 clrscr();
 puts("Rendezés minimális elem kiválasztással\n");
 printf("Elemek száma="); scanf("%d",&n);
 for (i =0; i < n; i++)
  {printf("V [ %d ]=", i+1); scanf("%d",&V [ i ]);
  }
```

A rendezés programja ..

```
for (j =0; j <n -1; j++)
 for (i =j +1; i <n; i++)
 if (V [ i ]<V [ j ])            /*  csere  */
  { Segedvaltozo= V [ j ];
    V [ j ]= V [ i ];
    V [ i ]= Segedvaltozo;
  }
puts("\nA rendezett vektor elemei:");
for (i=0; i < n; i++)
    printf("  V [ %d ]= %d", i+1, V [ i ]);
getch();
}
```

Függvények

- *Függvények deklarációja és prototípusa*
- *A függvénydefiníció*
- *Hivatkozás függvényre*
- *Mintaprogram függvénydefiniálással*
- *Címszerinti argumentumátadás*

Függvények deklarációja és prototípusa

A C nyelvben a függvény központi fogalom. Egy függvénynek a main()*-nek minden programban egyszer és csak egyszer szerepelnie kell. Egy függvény egy általában többször kiszámítandó értéket határoz meg, vagy logikailag egységet képező feladatot végez el. A fejlesztőrendszer részét képező könyvtári függvényeket, vagy az általunk készítetteket egyaránt használhatjuk. A függvényeket felhasználás (hivatkozás) előtt deklarálni kell, és saját függvényeinket egy helyen a programban teljesen meg kell adni, azaz definiálni szükséges. A korszerűbb prototípus a deklarációtól abban több, hogy megadja a függvény paramétereit is:*

<típus> függvénynév (<paraméterek>) ;

Pl.: **int** *minimum*(**int** *a*, **int** *b*) ;

A <típus> *határozza meg a függvény által visszaadott érték típusát. Ez* egyszerű, **struct** *és* **union** *típus lehet, valamint* **void***, amely értéket vissza nem adó függvényt (eljárást) eredményez. A függvény visszatérési értékét a függvény definíciós blokkjában elhelyezett* **return** *utasítás után álló kifejezés értéke adja. A* **return void** *függvényeknél elmaradhat.*

A <paraméterek> *megadásánál ügyelni kell arra, hogy minden paraméter előtt meg kell adni a típusát, amely az eddig tanultak bármelyike lehet. A prototípust a végén álló* ***;*** *különbözteti meg a függvénydefiníció fejlécétől. A prototípusok a program, vagy a blokkok deklarációs részében helyezhetők el.*

- **A függvénydefiníció** *a prototípusra emlékeztető fejlécből és egy blokkból áll:*

```
<típus> függvénynév (<paraméterek>)
{
   <deklarációk>     // lokálisak, blokkon belül érvényesek
   <utasítások>      // köztük lehet a return utasítás
}
```

A paraméterek csak érték szerinti átadásúak.
A cím szerinti átadást mutatóval kell megvalósítani.
A függvénydefiníciók általában a program végén helyezkednek el, a prototípusok és a hivatkozások után. Amennyiben a definíció megelőzi a hivatkozásokat, akkor a prototípus elmarad. Függvénydefiníció nem adható meg más függvény blokkjában, csak deklaráció. A függvénydefiníciók egymás mellé rendeltek, a main() függvénnyel azonos szinten, moduláris módon.

Példa egy függvénydefiniálásra:

```
int minimum( int a, int b)
{
    return (a < b) ? a : b ;
}
```

Hivatkozás függvényre

A C nyelv nem void függvényei pontosvessző nélkül függvénykifejezésként kifejezésben alkalmazhatók, de sok esetben pontosvesszővel lezárva utasításként kerülnek felhasználásra.

A függvényhívás formája:

függvénycímkifejezés (<argumentumok>)

A függvénycímkifejezés lehet a konstans mutatóként viselkedő függvénynév, vagy más, a függvény címét szolgáltató kifejezés. Az <argumentumok> vesszővel elválasztott kifejezések, melyek számban és (esetleges automatikus konvertálás után) típusban egyeznek a prototípus, vagy a definíció paramétereivel. Amennyiben a paraméterlista helyén a void (=üres) szó szerepelt, nem adható meg argumentum.
Az érték szerinti átadás miatt a függvény nem tud visszaadni megváltoztatott értéket az argumentumváltozóknak.

Példa függvényhivatkozásra:

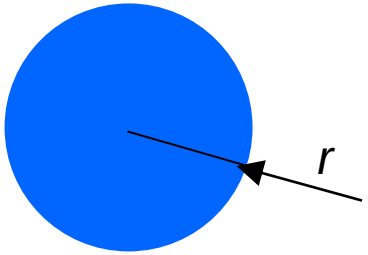
```
printf("k és m közül a kisebb értéke= %d", minimum( k , m ) );
```

- ***Mintaprogram függvénydefiniálással:*** hatványozás

```
#include <stdio.h>
#include <math.h>          // pow( ), exp( ), log( ) miatt
#include <conio.h>
double hatvany(double x, double y);   // prototípus
void main()
{    double alap, kitevo;
     clrscr();
     printf("Hatványalap = ");  scanf( "%lf", &alap );
     printf("Hatványkitevö = ");  scanf( "%lf", &kitevo );
     printf("\nEredmény= %lf", hatvany(alap,kitevo) );
     printf("\nPow-rel= %lf\n", pow(alap,kitevo) );
     getch();
}
double hatvany(double x, double y)      // definíció
{
     return exp(y*log(x));
}
```

Címszerinti argumentumátadás

Írjunk alprogramot kör területének és kerületének számítására a sugár ismeretében.



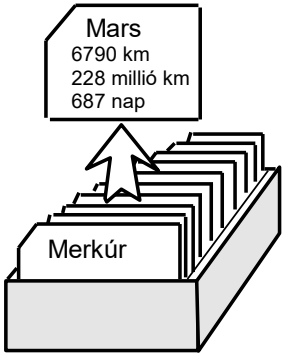
A terület és kerület argumentumváltozók másolata jön létre a függvénybe való belépéskor, melyek megváltozott értéküket a függvényből való kilépéskor elveszítik, változatlanul hagyva eközben a terület és kerület változókat. Megoldás: a változók címét adjuk át a függvénynek, melyet, mint mutatóértéket használva módosítja a mutatott terület és kerület változókat.

Címszerinti argumentumátadás ..

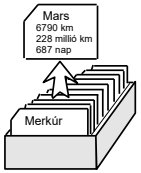
```
#include <stdio.h>
void kor ( float r, float* ter ,float* ker );     // prototípus
main()
{
  float sugar, t, k;
  printf( "Sugar =" );
  scanf( "%f", &sugar );
  kor(sugar, &t, &k);                              // hivatkozás
  printf("Terulet= %10.2f, kerulet= %10.2f ",  t, k);
}
void kor( float r, float* ter ,float* ker )      // definíció
{
  float Pi = 3.1415;
  *ter = r * r * Pi;
  *ker = 2 * r * Pi;
}
```

A struct és az enum típus

- *A struktúra adattípus*
- *Struktúra típus definiálása*
- *Struktúra típusú változó definiálása*
- *Hivatkozás a struktúratagokra*
- *Mintapélda struktúrák alkalmazására*
- *Az enum típus*
- *Mintaprogram enum típus alkalmazására*



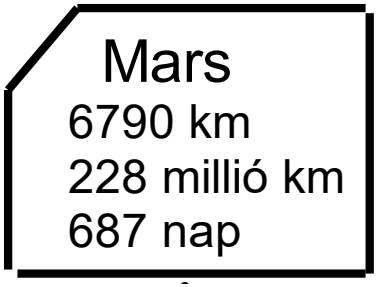
A struktúra adattípus



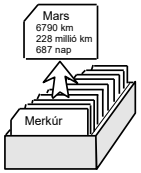
A struktúra adattípus egy egyed logikailag összetartozó, viszonylag kisszámú, különböző adattípusokkal ábrázolható jellemzőinek megadására szolgál.

A jellemzők tárolására önállóan is használható struktúratagok szolgálnak.

A struktúra összes adattagjával egyetlen egységként is kezelhető, pl. értékátadásban, függvények paraméterátadásában vagy visszatérő értékeként.



Példaként a Nap bolygói:

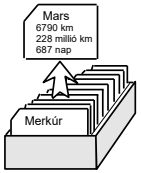


a struktúra adattagjai

A bolygó neve	Átmérője [km]	Távolsága a Naptól [km]	Keringési ideje	év/ nap
Merkúr	4 880	58 millió	88	nap
Vénusz	12 100	108 millió	225	nap
Föld	12 756	150 millió	365	nap
Mars	6 790	228 millió	687	nap
Jupiter	142 800	778 millió	12	év
Szaturnusz	120 860	1427 millió	29,5	év
Uránusz	52 000	2870 millió	84	év
Neptunusz	48 400	4497 millió	165	év
Plútó	3 000	5950 millió	248	év

egy struktúra

Struktúra típus definiálása



Több lehetőség:

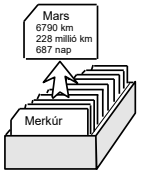
```
struct <struktúra_típusnév>
{  <típus1> struktúratag1 ;
   <típus2> struktúratag2 ;
            . . .
   <típusK> struktúratagK ;
 } ;
```

vagy ***typedef*** *utasítással kétféle alakban:*

```
typedef struct <struktúra_típusnév>
                 {  <típus1> struktúratag1 ;
                    <típus2> struktúratag2 ;
                             . . .
                    <típusK> struktúratagK ;
                 } ;
```

vagy

```
typedef struct{  <típus1> struktúratag1 ;
                 <típus2> struktúratag2 ;
                          . . .
                 <típusK> struktúratagK ;
               } <STRUKTÚRA_TÍPUSNÉV>;
```



Példák struct típus definiálására:

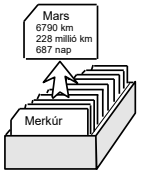
```
struct ponttipus { float x ; float y ; } ;
typedef struct komplextip { double Re ; double Im ; } ;
typedef struct { char nev[ 40 ] ;
                 char tankor [ 6 ] ;
               }  HALLGATOREKTIP ;
```

Struktúra változó definiálása

a típus-definiálási formától függően eltérő, a struct szó után megadott típusnév csak a struct szóval együtt jelenti az adott struktúratípust, míg a harmadik alak típusneve önállóan szerepelhet struktúraváltozók definiálásában:

struct <struktúra_típusnév> <változónév> ;

<STRUKTÚRA_TÍPUSNÉV> <változónév> ;



Példák struktúraváltozók definiálására:

struct ponttipus P1, P2 ;
struct komplextip cmp , * cmpmut ;
HALLGATOREKTIP *hallgatovekt* [120] ;

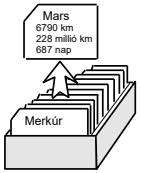
Az utolsó példa egy struktúrákból álló vektort definiál.

Hivatkozás a struktúratagokra

A hivatkozás operátora a **.** *(pont). Példák:*

*P1.x = 6.234; P1.y = 12*34.5 ;*
cmp.Re = cmp.Re + 4.67 ; cmp.Im = 12.45 ;
strcpy(*hallgatovekt* [*12*] **.** *nev , "Nagy Tamás"*);
strcpy(*hallgatovekt* [*12*] **.** *tankor , "G1BV3"*);

Megj.: *Az* strcpy() *függvénnyel másolhatjuk át a szövegeket a struktúratagokba, mivel karaktervektorok számára nincs értelmezve az egyben történő értékadás.*



Mutatóval elérhető struktúrák tagjainak hivatkozása hagyományos módon, vagy a -> nyíl operátorral történhet:

```
(*cmpmut) . Re = 34.7;
  cmpmut ->Re = 34.7;
```

Azonos típusú struktúrák között értelmezve van az értékadás, pl.:

```
P1 = P2 ;
```

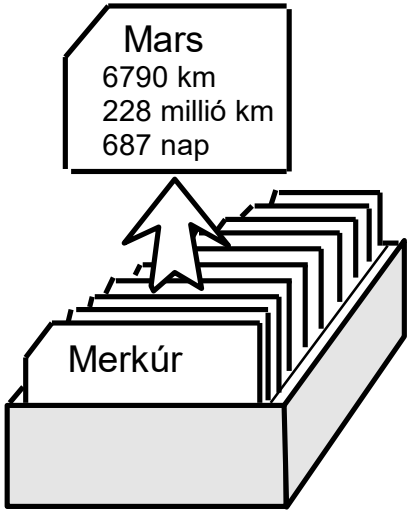
A struktúrák is elláthatók definiáláskor kezdőértékkel, pl.:

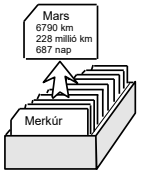
```
HALLGATOREKTIP  diak = {"Gál Éva" ,"G1BG1" };
struct komplextip  cmp = { -123.44 , 254.3 } ;
```

Mintapélda struktúrák alkalmazására:

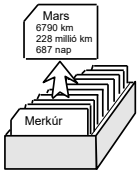
Hozzuk létre a Nap bolygóinak adatait tároló struktúra-vektort, majd irassuk ki a Mars adatait!

```
#include <stdio.h>
#include <conio.h>
#define  bsz  9                    //  bolygók száma
typedef struct {                   char nev [15 ] ;
                        unsigned int atmero ;
                              long int tavolsag ;
                                  float kering_ido ;
                                  char evvagynap[ 4 ] ;
                        } BOLYGOTIP ;
BOLYGOTIP bolygok[ bsz ] ;
```

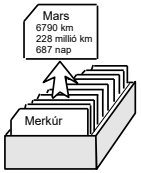





```
main()
{
    int i;
    char nev [15];
    clrscr();
    printf("Bolygók adatbázisa\n\n");
    for (i = 0; i < bsz ; i++)
    {
      printf( " \nA bolygó neve: " );
      scanf( "%s", bolygok[ i ].nev );
      printf( "Átmérője=" );
      scanf( "%u", &bolygok[ i ].atmero );
      printf( "Naptól való távolsága=" );
      scanf( "%ld", &bolygok[ i ].tavolsag );
      printf("Keringési idö=");
      scanf("%f",&bolygok[ i ].kering_ido);
      printf("Években, vagy napokban? (év/nap):");
      scanf("%s",bolygok[ i ].evvagynap);
    }
```




```
printf( "\n\nA keresett bolygó neve:" );
scanf( "%s", nev );
for (i = 0; i < bsz ; i++)
{   int j = 0;
    while (nev[ j ] == bolygok[ i ].nev[ j ] && nev[ j ] && bolygok[ i ].nev[ j ] )
        j++;      /*   a while ciklus magja   */
    if (nev[ j ] == bolygok[ i ].nev[ j ])    /*   a két név azonos     */
    {
        printf( "\nA bolygó átmérője= %u", bolygok[i].atmero );
        printf( "\nNaptól való távolsága= %ld millió km", bolygok[i].tavolsag );
        printf( "\nKeringési ideje= %f  %s",
                        bolygok[ i ].kering_ido, bolygok[ i ].evvagynap);
        break;    /*   kiugrik a for ciklusból    */
    }
}
getch();
}
```



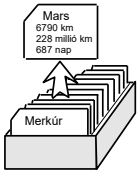
Az enum típus

Az enum típus felsorolt azonosítók automatikus sorszámozására, konstans sorszámérték hozzárendelésére alkalmas.

A 0-val kezdődő automatikus értékhozzárendelést megváltoztathatjuk, ha valamelyik azonosító számára előírjuk az értéket, mely ***int*** *típusú lehet. A nem megadott értékű azonosítók az előttük levőtől eggyel nagyobbak.*

Az ***enum*** *típus konstansai egész értékekként viselkednek. Amennyiben önálló típusnévvel látjuk el, a felsorolt értékek egy ilyen típusú adat értékkészleteként foghatók fel.*

hétfő kedd szerda csütörtök péntek szombat vasárnap



*Példák **enum** típusú definíciókra:*

```
enum { false, true } ;   /*  false = 0, true = 1 értékű    */
enum fibonacci { a, b, c = 1, d = 2, e = 3, f = 5 };
```

Definiálás önálló típusnévvel:

```
typedef enum evszaktip { tavasz, nyar, osz, tel };
enum evszaktip evszak;   /* evszak változó definiálása   */
```