Budapesti Műszaki és Gazdaságtudományi Egyetem
Közlekedés- és Járműirányítási Tanszék

Programozás C- és Matlab nyelven
C programozás kurzus
BMEKOKAM603
Mutatók

Dr. Bécsi Tamás

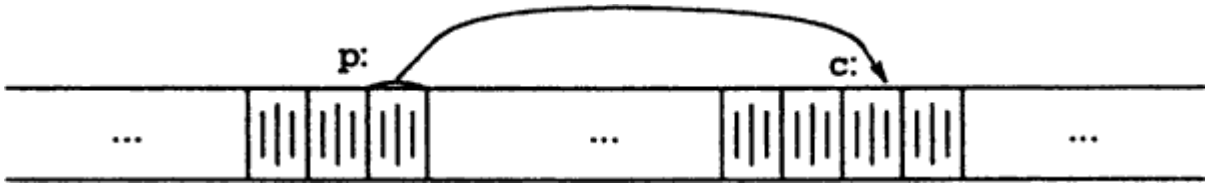
7. Előadás

5.3. Mutatók,tömbök

- A mutató vagy pointer olyan változó, amely egy másik változó címét tartalmazza. A C nyelvű programokban gyakran használják a mutatókat, egyrészt mert bizonyos feladatokat csak velük lehet megoldani, másrészt mert alkalmazásukkal sokkal tömörebb és hatékonyabb program hozható létre.

5.1. Mutatók és címek

- Az **&** unáris (egyoperandusú) operátor megadja egy operandus címét, ezért a `p = &c;` utasítás c címét hozzárendeli a p változóhoz és ilyenkor azt mondjuk, hogy p c-re „mutat”.



- A * unáris operátor neve *indirekció*, és ha egy mutatóra alkalmazzuk, akkor a mutató által megcímzett változóhoz férhetünk hozzá.

Mutatók
Példa


```
int i=10;
int *ip;
ip=&i;
printf("%d\n",*ip);
i++;
printf("%d\n",*ip);
```

Console:

```
10
11
```

Memória cím (bájt sorszám)	Változónév	Érték (int)
1000		?
1004	i	11
1008	ip	1004
1012		?
----	*ip	11
	&i	1004

5.1. Mutatók és címek operátorok példa


```
int x = 1, y = 2, z[10];
int *ip;/* ip int tipushoz tartozó mutató */

ip = &x;/* ip x-re mutat */
y = *ip;/* y most 1 lesz */
*ip = 0;/* most x nulla lesz */
ip = &z[0];/* ip most z[0]-ra mutat */
```

5.1. Mutatók és címek

- Az indirekció alapján látható, hogy **egy mutató mindig meghatározott objektumra mutat**, azaz minden mutató meghatározott adattípust jelöl ki. (Ez alól csak egy **kivétel** van, a **void** típushoz tartozó mutató, ami egy olyan adat, amely bármilyen mutatót tartalmazhat. Erre az a megszorítás érvényes, hogy önmagára nem alkalmazhatja az indirekciót.

5.1. Mutatók és címek precedencia példa


```
int *ip
  *ip = *ip + 10; //*ip-et tízzel növeli
```

Az & és * unáris operátorok szorosabban kötnek, mint az aritmetikai operátorok, ezért az

```
y = *ip + 1
```

kifejezés kiértékelésekor a gép először veszi azt az adatot, amire ip mutat, hozzáad egyet, majd az eredményt hozzárendeli y-hoz.

5.1. Mutatók és címek precedencia példa

Az

```
*ip += 1
```

inkrementálja azt a változót, amire ip mutat, csakúgy, mint a

```
++*ip
```

vagy az

```
(*ip)++
```

5.2. Mutatók és függvényargumentumok

Mivel a C nyelv a függvényeknek érték szerint adja át az argumentumokat, így a hívott függvény nem tudja megváltoztatni a hívó függvény változóit.

```
swap(a, b);

void swap(int x, int y)      /* Hibás!!! */
{ int temp= x;
  x = y;
   y = temp;
}
```

5.2. Mutatók és függvényargumentumok

Mivel a függvényt érték szerint hívjuk, a swap nem képes a hívásában szereplő a és b argumentumokat befolyásolni (azoknak csak egy helyi másolatával dolgozik, ezek cseréje pedig nem befolyásolná az eredeti argumentumok sorrendjét).

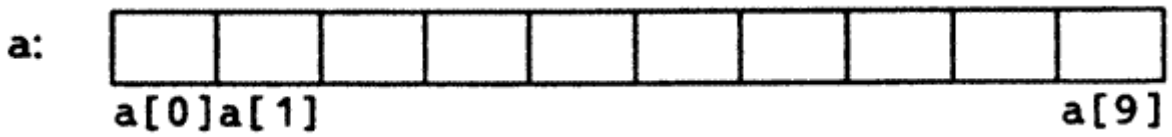
```
swap(&a, &b);

void swap(int *px, int *py) /* Helyes   */
{ int temp=*px;
  *px = *py;
  *py = temp;
 }
```

5.3. Mutatók és tömbök

A C nyelvben a mutatók és a tömbök között szoros kapcsolat van. Az

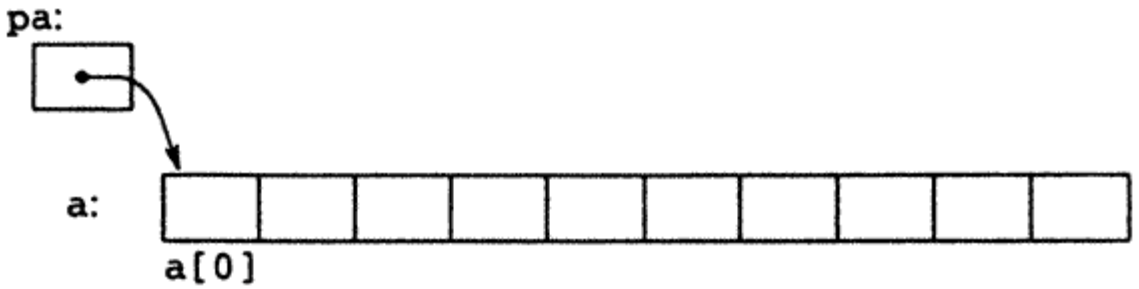
```
int a[10];
```

deklaráció egy tízelemű tömböt jelöl ki, azaz tíz egymást követő, a[0]...a[9] névvel ellátott objektumot.



5.3. Mutatók és tömbök

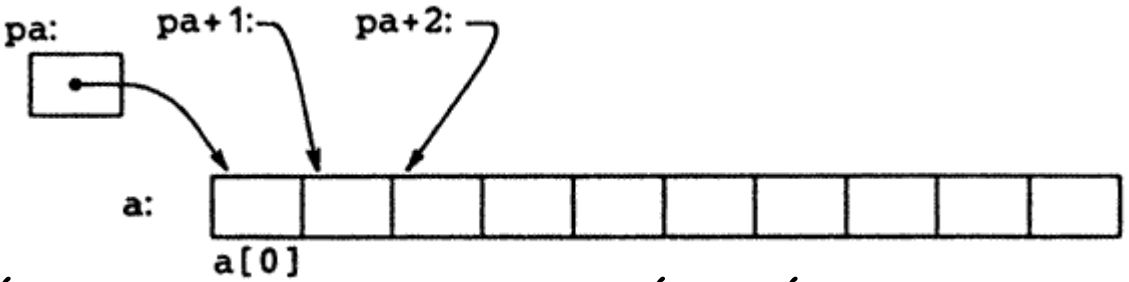
- Az `a[i]` jelölés a tömb i-edik elemére hivatkozik. Ha pa egy egész típushoz tartozó mutató, amit `int *pa;` módon deklaráltunk, akkor a `pa = &a[0];` értékadás hatására pa az a tömb nulladik elemére fog mutatni, vagyis pa az a[0] címét fogja tartalmazni.



5.3. Mutatók és tömbök

Ennek megfelelően az `x=*pa;` értékeadás a[0] értékét másolja x-be.

A `*(pa+1)` `a[i]`-ik elemére mutat



`a[0]` és `a` ugyanaz a mutató, ezért:

`pa = &a[0];` ekvivalens a `pa=a;` kifejezéssel.

`a[i]` ekvivalens `*(a+i)` hivatkozással.

5.3. Mutatók és tömbök
Példa


```
int t[3]={2,4,6},i,*pi;
pi=t;
for(i=0;i<3;i++)
printf("%d %d %d\n",t[i],*(t+i),*(pi+i));
do{
printf("%d\n",*(pi));
}while (++pi-t<3);
```

Cím	Cím hivatkozás			Érték hivatkozás			Érték
1000	&t[0]	t	pi	t[0]	*t	*pi	2
1004	&t[1]	t+1	pi+1	t[1]	*(t+1)	*(pi+1)	4
1008	&t[2]	t+2	pi+2	t[2]	*(t+2)	*(pi+2)	6

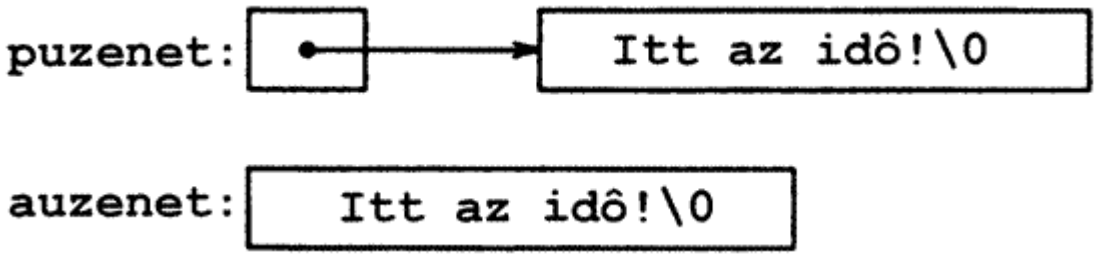
5.4. Címaritmetika

- Ha p egy tömb valamelyik elemének mutatója,
  akkor `p++` inkrementálja a p mutatót, hogy az a tömb következő elemére mutasson és `p+=i` pedig úgy növeli p-t, hogy az az aktuális elem utáni i-edik elemre mutasson.
  Mutatók esetén általában igaz, hogy az int típusra alkalmazható operátorok alkalmazhatóak rá. (bár bizonyos eseteknek nem nagyon van értelmük (pld.: /,%))

5.5. Karaktermutatók és függvények

Az `"Ez egy karaktersorozat"` alakban írt karaktersorozat állandók valójában karakterekből álló tömbök, amelyeket a belső ábrázolásban egy null-karakter (`'\0'`) zár.

```
char auzenet[] = "Itt az ido"; /* ez egy tömb */
char *puzenet = "Itt az ido"; /* ez egy mutató */
```



Ennek megfelelően stringek esetén az `s=t;` utasítás csak a pointert másolja a karaktereket nem helyezi el új tömbben, ez külön ciklussal kell megoldani:

5.5. Karaktermutatók és függvények

- Az strcpy függvény a standard könyvtárban (a <string.h> headerben) található, és visszatérési értéke az átmásolt karaktersorozat.
- A fent ismertetett megoldás nem foglalkozik azonban a különböző méretű tömbök problémakörével.

5.5. Karaktermutatók és függvények


```
void strcpy(char *s, char *t)
  {
     int i = 0;
      while ((s[i] = t[i]) != '\0') i++;
  }
```

vagy egyszerűbben:

```
void strcpy(char *s, char *t)
{
   while(*s++ = *t++) ;
}
```

B3. Karaktersorozat-kezelő függvények: a <string.h> header

- `char *strcpy(s, ct)` Az strcpy függvény a ct karaktersorozatot átmásolja az s karaktersorozatba, beleértve act-t záró '\0' végjelet is. A függvény visszatérési értéke s mutatója.
  `char *strncpy(s, ct, n)` Az strncpy függvény a ct-ből n karaktert átmásol s-be és visszatér s mutatójával. Az s végét '\0' végjelekkel tölti fel, ha ct n karakternél rövidebb volt.
  `char *strcat(s, ct)` Az strcat függvény a ct karaktersorozatot az s karaktersorozat végéhez fűzi (konkatenálja) és visszatér s mutatójával.
  `char *strncat(s, ct, n)` Az strncat függvény a ct karaktersorozatból n karaktert az s karaktersorozat végéhez fűz, s-t lezárja a '\0' végjellel és visszatér s mutatójával.

Strcpy példa


```
str1: Sample string
str2: Sample string
str3: copy successful
```

```c
/* strcpy example */
#include <stdio.h>
#include <string.h>
int main ()
{
  char str1[]="Sample string";
  char str2[40]; char str3[40];
  strcpy (str2,str1);
  strcpy (str3,"copy successful");
  printf ("str1: %s\nstr2: %s\nstr3: %s\n"
          ,str1,str2,str3);
return 0;
}
```

Strncpy példa


```
To be or not to be
To be or not to be
To be
```

```c
#include <stdio.h>
#include <string.h>
int main ()

{ char str1[]= "To be or not to be";
  char str2[40]; char str3[40];
  /* túlcsordulás biztos másolás: */
  strncpy ( str2, str1, sizeof(str2) );
  /* részleges másolás (csak 5 karakter): */
  strncpy ( str3, str2, 5 );
  str3[5] = '\0';
  /* null karakter manuálisan beállítva */
  printf("%s \n%s \n%s\n ", str1, str2, str3);
return 0;
}
```

Strcat,strncat példa


```
ezeket a stringeket ossze
```

```
#include <stdio.h>
#include <string.h>
int main ()
{
   char str[80];
   strcpy (str,"ezeket ");
   strcat (str,"a ");
   strcat (str,"stringeket ");
   strncat (str,"osszefuztem. ",5);
   printf("%s",str);
   return 0;
}
```

B3. Karaktersorozat-kezelő függvények: a <string.h> header

- `int strcmp(cs, ct)` Az strcmp függvény összehasonlítja a cs karaktersorozatot a ct karaktersorozattal és visszatér negatív értékkel, ha cs < ct, nulla értékkel, ha cs == ct és pozitív értékkel, ha cs > ct.
  `int strncmp(cs, ct, n)` Az strncmp függvény összehasonlítja a cs karaktersorozat legfeljebb n karakterét a ct karaktersorozattal és visszatér negatív értékkel, ha cs < ct, nulla értékkel, ha cs == ct és pozitív értékkel, ha cs > ct.
  `char *strchr(cs, c)` Az strchr függvény a c karakter cs-beli első előfordulási helyének mutatójával, ill. ha c nem található meg cs-ben, akkor NULL értékű mutatóval tér vissza.
  `char *strrchr(cs, c)` Az strrchr függvény a c karakter cs-beli utolsó előfordulási helyének mutatójával, ill. ha c nem található meg cs-ben, akkor NULL értékű mutatóval tér vissza.

Strcmp példa


```
#include <stdio.h>
#include <string.h>

int main()
{
  char szKey[] = "alma";
  char szInput[80];
  do
  {
    printf ("Mi a kedvenc gyumolcsom? ");
    scanf("%s",&szInput);
    } while (strcmp (szKey,szInput) != 0);
  printf("Helyes!"); return 0;
}
```

```
Mi a kedvenc gyumolcsom? narancs
Mi a kedvenc gyumolcsom? alma
Helyes!
```

Strncmp példa


```
#include <stdio.h>
#include <string.h>
```

R2-es droidokat keresunk...
Egyezes: R2D2
Nem ez a droid, amit keresnek!
Egyezes: R2A6

```
int main () {
  char str[][5] = { "R2D2" , "C3PO" , "R2A6" };
  int n;
  printf("R2-es droidokat keresunk...");
  for (n=0 ; n<3 ; n++)
     if (strncmp (str[n],"R2xx",2) == 0)
        printf ("Egyezes: %s\n",str[n]);
        else
        printf("Nem ez a droid, amit keresnek!\n");
  return 0;
}
```

Strchr példa


```
#include <stdio.h>
#include <string.h>
int main ()
{
  char str[] = "This is a sample string";
  char * pch;
  pch=strchr(str,'s');
  while (pch!=NULL)
  {
     printf ("Talalt: %d. karakter\n",pch-str+1);
     pch=strchr(pch+1,'s');
  }
  return 0;
}
```

```
Talalt: 4. karakter
Talalt: 7. karakter
Talalt: 11. karakter
Talalt: 18. karakter
```

B3. Karaktersorozat-kezelő függvények: a <string.h> header

- `char *strstr(cs, ct)` Az strstr függvény visszatérési értéke a ct karaktersorozat cs-beli első előfordulásának kezdetét címző mutató, vagy NULL, ha a ct nem található meg cs-ben.

  `size_t strlen(cs)` Az strlen függvény visszatérési értéke a cs karaktersorozat hossza.

  `char *strerror(n)` Az strerror függvény az n hibaszámhoz tartozó, a gépi megvalósítástól függő hibaüzenet karaktersorozatának mutatójával tér vissza.

Strstr példa


```
This is a simple string
This is a sample string
```

```
#include <stdio.h>
#include <string.h>
int main ()
{
  char str[] ="This is a simple string";
  char * pch;
  printf("%s\n",str);
  pch = strstr (str,"simple");
  strncpy (pch,"sample",6);
  printf("%s\n",str);
  return 0;
}
```