

## 8. gyakorlat

A gyakorlat témája a stringek, stringkezelő függvények.

### A string fogalma

A szövegeket karakterláncban, stringben tárolhatjuk. A C nyelvben a string egy egyszerű tömböt jelent, amiben karakterek vannak, és a végén egy záró karakter (a 0 numerikus értékű \0) áll.

Szöveg létrehozása szöveggént és tömbként:

```
char szoveg1[] = "Hello!";  
char szoveg2[] = {'H', 'e', 'l', 'l', 'o', '!', 0}
```

Az első féle értékadásban szöveggént töltjük fel a tömböt. Egyszerűbb beírni, és a záró 0 karaktert automatikusan kirakja. A második féle értékadásban egyszerű tömbként határozzuk meg az értéket.

### Módosítható és nem módosítható stringek

```
char* literal = "Hello!";  
char chararray[] = "Hello!";
```

A `char* nev = "..."` utasítással létrehozott szövegek nem módosítható memóriaterületre kerülnek, ezeknek elemeit nem tudjuk átírni. Ha mégis megpróbáljuk, `segmentation fault` futásidejű hibával találkozhatunk.

A `char nev[] = "..."` utasítással, tömbként létrehozott stringek betűit szabadon módosíthatjuk.

### string.h

A szövegek kezelését végző függvények a `string.h` könyvtárban laknak.

#### Szöveg hossza - `strlen()`

A szöveg hosszát kiszámoló függvény. A záró karakterig megszámolja az összes karaktert. Eredménye `size_t`, vagyis egy egész szám.

```
size_t strlen(const char* string)
```

#### Keresés stringben

##### Karakter keresése a stringben - `strchr()`

```
char* strchr(const char* string, char c)
```

Megkeresi `c` karakter első előfordulását. Eredménye a megtalált karakterre mutató pointer.

```
char* strchr(const char* string, char c)
```

Ugyanez, csak visszafelé keres.

##### String keresése stringben - `strstr()`

```
char* strstr(const char* miben, const char* mit)
```

Megkeresi a miben stringben a mit rész-string első előfordulását. Eredménye a megtalált rész-string első karakterére mutató pointer.

### Bizonyos betűk bármelyikének megkeresése - strchr()

```
size_t strchr(const char* miben, const char* miket)
```

A miben stringben megkeresi a miket string karaktereinek bármelyikét. Amikor talált egyet, visszaadja a karakter indexét (*nem pointert*)

### Stringek összehasonlítása - strcmp()

Stringek összehasonlítása.

```
int strcmp(const char* string1, const char* string2)
```

Az eredmény

- < 0, ha string1 < string2
- = 0, ha string1 = string2
- > 0, ha string1 > string2

Az összehasonlítás karakterérték szerint történik, kisbetűk-nagybetűk meg vannak különböztetve. Az "A" nagyobb mint a "z". A "9" nagyobb mint a "10".

Stringek első n karakterének összehasonlítása

```
int strncmp(const char* string1, const char* string2, size_t n)
```

### Stringek másolása - strcpy()

```
char* strcpy(char* cel, const char* forras)
```

A cel stringbe beírja forras tartalmát. A célnak elég nagynak kell lennie! A záró karakternek is bele kell férnie. A visszatérési érték a cel végére mutató pointer, általában nem kell használni. Ha a forrás hosszabb a célnál, a programunk összeomlik.

```
char* strncpy(char* cel, const char* forras, size_t n)
```

A forras első n karakterét másolja át. Ha a forrás hosszabb n-nél, zárókaraktert nem tesz ki. Ha forrás rövidebb n-nél, a maradék helyeket 0-val tölti fel.

Biztonságos másolás készíthető, ha  $n = \text{strlen}(\text{cel}) - 1$ . Így a forrás végét levághatja, de nem fut túl a célon kívülre.

### Stringek összefűzése - strcat()

A concatenate (összefűzés) szó rövidítése.

```
char* strcat(char* mihez, const char* mit)
```

A mihez végére beírja mit tartalmát. A mihez zárókarakterét törli, beírja mit karaktereit a zárókarakterével együtt. A mihez stringnek elég nagynak kell lennie! Eredménye a mihez-re mutató string (de nem kell használni).

```
char* strncat(char* mihez, const char* mit, size_t n)
```

A mihez végére  $n$  darab karaktert másol a `mit`-ből. Az eredmény mindenképpen tartalmazza a záró karaktert, ezért a miheznek legalább  $\text{strlen}(\text{mihez}) + n + 1$  méretűnek kell lennie.

## String darabolása

A string darabolása alatt azt értjük hogy bizonyos karakterei mentén elválasztjuk, és a részeiből új stringeket gyártunk.

A `"a,bb,ccc"` string feldarabolható a vesszők mentén erre a három stringre: `"a"`, `"bb"`, `"ccc"`.

Az `strtok` függvény kicsit furcsán működik. A bemeneti stringjét módosítja, az elválasztó karaktereket kicseréli záró karakterekre, majd a string egyes részeire mutató pointereket ad vissza. Ezeket a részeket különálló stringekként tudjuk olvasni.

```
char* strtok(char* string, const char* elvalasztok)
```

A string a bemenet, aminek módosíthatónak kell lennie. Az elválasztók 1 vagy több karaktert tartalmazó string.

A függvényt először a stringre kell meghívni. Ekkor visszaadja az első darabot, és a maradékot eltárolja egy belső változóba. A többi darab megtalálásához a string helyére `NULL`-t kell írni. Amikor már nincs több darab, a visszatérési érték `NULL` lesz.

## Példa

```
char sor[] = "a,bb,ccc"; // NEM char*
char* darab1 = strtok(sor, ",")
char* darab2 = strtok(NULL, ",")
char* darab3 = strtok(NULL, ",")
```

A sor ezek után így néz ki:

```
"a\0bb\0ccc"
  ^   ^   ^
  |   |   | darab3
  |   |   | darab2
  |   |   | darab1
```

Több elválasztó karaktert is használhatunk: `",";`

Az `strtok` üres darabot nem talál. Ha két vagy több elválasztó karakter van egymás után, akkor azokat átugorja. CSV olvasáshoz ezért csak akkor lehet használni, ha garantáltan ki van töltve minden cella.

## stdio.h

### Sor olvasása - `gets_s()` és `fgets()`

*Létezik `gets()` függvény is, de azt tilos használni.*

### Windowson: `gets_s()`

```
char* gets_s(char* sor, size_t n)
```

Legfeljebb n-1 karaktert olvas be. Enternél vagy fájl végénél fejezi be az olvasást. Az entert is beolvassa.

### Nem-Windows: fgets()

```
char* fgets(char* sor, size_t n, FILE* stream)
```

Billentyűzetről olvasáskor a stream = stdin (*standard input*). A string méretét is meg kell adni. Legfeljebb n-1 karaktert olvas be. Enter nyomásánál vagy a fájl végénél fejezi be az olvasást, az entert is beolvassa a sor végére. Visszatérési értéke siker esetén a buffer címe, és NULL ha már nem tud mit beolvasni.

Az entert gyakran ki kell törölni, pl így:

```
sor[strcspn(str, "\n")] = 0;
```

### Értékek írása stringbe - sprintf()

```
int sprintf(char* str, const char *format, ...)
```

Olyan mint a printf, de egy stringbe ír bele. A stringnek elég nagynak kell lennie!

Létezik egy snprintf() párja is, ami legfeljebb n karaktert ír.

### Értékek olvasása stringből - sscanf()

```
int sscanf(const char* str, const char* format, ...)
```

Olyan mint a scanf, de stringből olvas be.

### stdlib.h

Konvertálások. Ezek helyett lehet sscanf-et is használni. Bonyolultabb dolgokra sscanf.

### Stringből int - atoi()

```
int atoi(const char* str)
```

### Stringből double - atof()

```
double atof(const char* str)
```