INFORMATIKA LABORATÓRIUM I.

Tömbök

Tömbök

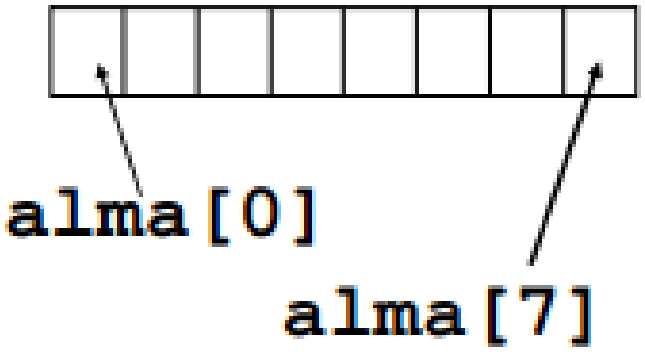
- Mint a legtöbb programozási nyelv, a C is ismeri a tömb fogalmát.
- A tömbök globális kezelését azonban nem biztosítja a C nyelv, azaz nincs olyan utasítás, amely egy tömböt átmásol egy másikba, vagy amely matemetikai műveleteket hajtana végre tömbök között.
- Ezeket a tömb elemenkénti kezelésével tudjuk csak megvalósítani.
- A tömb olyan azonos típusú változók halmaza, amelyekre azonos névvel tudunk hivatkozni.
- A tömb egyes elemeinek elérése indexek segítségével valósítható meg.
- Egy tömb a memóriában folytonosan helyezkedik el.
- A tömb számára lefoglalt memóriaterület legalacsonyabb címén a tömb első eleme helyezkedik el, míg a legmagasabb címen az utolsó eleme.

Tömbök

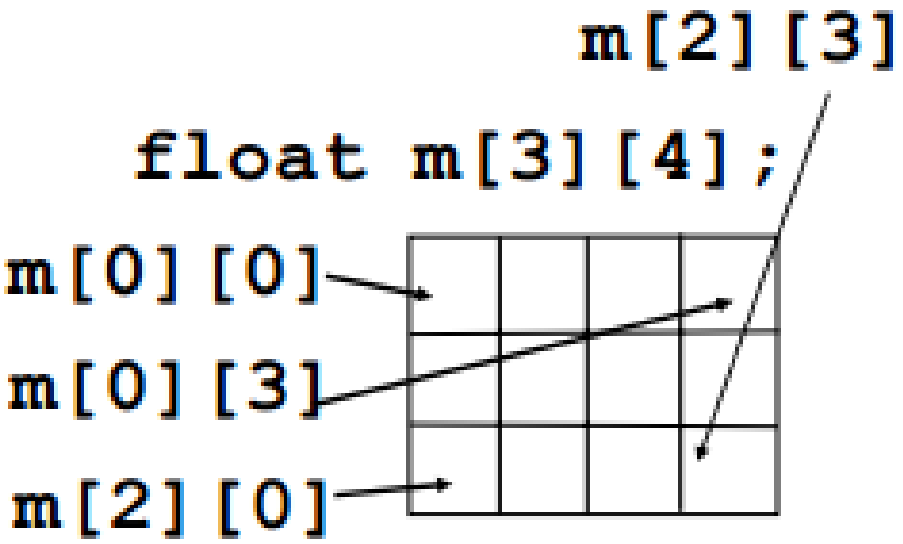
C-ben a tömb indexelése 0-tól kezdődik (figyeljünk a mondatszerű feladatoknál).

Pl:

```
int alma[8];
```




```
float m[3][4];
```



Egydimenziós tömb

A tömb általános definíciója egydimenziós tömb esetén:

típus változó-név[méret];

A tömb elemszámát és a tagok indexét is szögletes zárójelek közé kell zárni. A tömbindexek mindig nullától indulnak és a utolsó elem indexe egy n elemű tömb esetén n-1.

Példák tömbök deklarálására:

```
int sample [10]; /*10 egész elemet
deklarál: sample[0]...sample[9] */
main()
{
        int x[10];
        int t;
        for (t=0; t<=10; ++t)
                        x[t]=t;
}
```

Ügyeljünk arra, hogy a tömb deklarálásakor megadott méretet a felhasználáskor túl ne lépjük, mert erre vonatkozóan nem történik ellenőrzés.

1 dimenziós tömb

Feladat: írassunk ki egy tömböt fordítva

```c
//Írassunk ki egy tömböt fordítva

#include <stdio.h>
int main()
{
	float tomb[10];
	int i;
	for (i = 0; i < 10; i++)
		scanf(î%fî, &tomb[i]);
	for (i = 9; i >= 0; i--)
		printf(î%f î, tomb[i]);
	return 0;
}
```

1 dimenziós tömb

Feladat: Oldjuk meg az alábbi feladatot C-ben!

TÖLTSÜNK FEL EGY 13 ELEMŰ TÖMBÖT AZ ELSŐ 13 TERMÉSZETES SZÁM NÉGYZETÉVEL!

```
PROGRAM feladat11
        CIKLUS i:=1-TŐL 13-IG  1-VEL
                a[i]:=i*i
        CIKLUS VÉGE
PROGRAM VÉGE
```

1 dimenziós tömb

Feladat: Oldjuk meg az alábbi feladatot C-ben!

Adott egy 25 elemű egész számokkal feltöltött tömb. Határozzuk meg, hogy van-e közöttük egyforma!

```
PROGRAM feladat25
        van:=hamis
        i:=1
        j:=2
        CIKLUS AMÍG (i<=24) ÉS NEM van
                j:=i+1
                CIKLUS AMÍG (i<=25) ÉS NEM van
                        HA a[i]=a[j] AKKOR van:=igaz
                        ELÁGAZÁS VÉGE
                        j:=j+1
                CIKLUS VÉGE
                i:=i+1
        CIKLUS VÉGE
PROGRAM VÉGE
```

1 dimenziós tömb

Feladat: írassunk ki egy tömböt fordítva

```c
//Írassunk ki egy tömböt fordítva

#include <stdio.h>
int main()
{
	float tomb[10];
	int i;
	for (i = 0; i < 10; i++)
		scanf(î%fî, &tomb[i]);
	for (i = 9; i >= 0; i--)
		printf(î%f î, tomb[i]);
	return 0;
}
```

2 dimenziós tömb

Adva van egy 5x5-ös mátrix. Írjunk programot amely a főátlójára tükrözi!

```
#include <stdio.h>
#include <conio.h>
void main(){
int a[5][5] = {         1, 2, 3, 4, 5,
            6, 7, 8, 9,10,
            11,12,13,14,15,
            16,17,18,19,20,
            21,22,23,24,25};
   int i,j,s;

   printf("\nEREDETI : "); printf("\n\n");
   for(i=0; i<5; i++){
            for(j=0; j<5; j++)
               printf(" %2d",a[i][j]);
            printf("\n");
   }
 // Most következik a mátrix tükrözése
   for(i=0; i<5; i++){
                        for(j=i; j<5; j++)
                        {
                           s = a[j][i];
                           a[j][i] = a[i][j];
                           a[i][j] = s;
                        }
   }
   printf("\nTÜKRÖZÖTT : ");printf("\n\n");
   for(i=0; i<5; i++){
            for(j=0; j<5; j++)
                        printf(" %2d",a[i][j]);
            printf("\n");
   }
   getch();
}
```

2 dimenziós tömb

Adva van egy 2x3 - as és egy 3x2 - es mátrix. Írjunk programot, amely össze szorozza őket!

```
#include <stdio.h>
#include <conio.h>
#define N 3
#define M 2

main()
{
    int a[N][M] =   {
                        1, 2,
                        3, 4,
                        5, 6
                    };
    int b[M][N] =   {
                        1, 2, 3,
                        4, 5, 6
                    };
    long c[N][N];
    int i, j, k;

    for(i=0; i<N; i++){
         for(j=0; j<N; j++){
                    c[i][j]=0L;
                    for(k=0; k<M; k++)
                    {
                            c[i][j] += a[i][k]*b[k][j];
                    }
         }
    }
    clrscr();
    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
          printf("C[%d][%d] = %3ld%c",i,j,c[i][j],
             (j == N-1) ? '\n' : ' ');
        }
    }
    getch();
}
```

Sztringek

- A tömbök között kitüntetett szerepet játszanak a sztringek. A sztring nem más, mint egy sajátos módon kezelt karakteres tömb.
- Az különbözteti meg a numerikus tömböktől, hogy a karakterkódok között van egy kitüntetett érték, a nulla, amely a sztring végét jelezheti.
- Így egy karakteres tömb végét nemcsak hosszának ismeretében határozhatjuk meg. 10 karakteres sztring részére 11 elemű tömböt kell deklarálnunk:

Példa:

char str[11]

A

| h | e | l | l | o | '\0' |

szöveg részére 6 elemű tömb szükséges.

Ügyeljünk arra, hogy a tömb deklarálásakor megadott méretet a felhasználáskor túl ne lépjük, mert erre vonatkozóan nem történik ellenőrzés.

Sztringek: karaktertömbök

```
#include <string.h>
#include <stdlib.h>     //malloc-hoz
...
      char str1[128] = "asdf";      //létrehozunk fix méretű karaktertömböt
      char str2[] = "Sample string";      //ha így hozunk létre stringet,
      létrejön egy karaktertömb, melynek akkora a mérete, hogy ez a
      megadott string elfér benne
      char* str3 = (char*) malloc(sizeof(char) * 10);      //10 elemű
      karaktertömb dinamikus létrehozása (C++-ban new utasítással is
      lehetne)
...
```

Fontosabb *string.h* függvények

„n-es függvények” : több string.h függvény rendelkezik egy „n”-es változattal is. Az „n”-esek ugyanazt csinálják, mint a „simák”, csak legfeljebb „n” karakterig… Pl. a 2. (strncpy) függvényben.

1. strlen : string hossza

```
size_t strlen ( const char * str );
```

Visszaadja a paraméterül kapott string méretét. Nem a karaktertömb teljes méretét, csupán azt, hogy a benne lévő string hány karakter hosszú (hány karakter van a stringet lezáró \0 karakter előtt).

Példa:

```
char str[100] = "Ez egy string";
int length = strlen(str);
printf("A string hossza: %d\n", length);
```

Output:

A string hossza: 13

Fontosabb *string.h* függvények

- `char * strcpy ( char * destination, const char * source );`

Egy karaktertömb tartalmát egy másikba másolja. A másolandó karaktertömb összes karakterét átmásolja egészen a \0 karakterig (természetesen a cél tömbben tárolt string végére is \0 karakter kerül).

Első paramétere a cél karaktertömb.

Második paramétere a forrás karaktertömb.

Fontos! A cél karaktertömbnek akkorának kell lennie, hogy abban elférjen a forrás string. Ha nem így van, túlindexelődik…

Példa:

```
char str1[] = "Sample string";
char str2[40];
strcpy (str2,str1);
printf ("str1: %s\nstr2: %s\n",str1,str2);
```

Output:

str1: Sample string

str2: Sample string

Fontosabb *string.h* függvények

- `char * strncpy ( char * destination, const char * source, size_t num );`

Hasonló az előzőhöz, de legfeljebb „n” db karaktert fog átmásolni.

Az előzővel ellentétben a cél tömbbe került string végére NEM kerül automatikusan \0 karakter!

Első két paraméter ugyanaz, mint az előzőnél.

Harmadik paraméterben megadhatjuk, hány db karaktert akarunk legfeljebb átmásolni.

Példa:

```
char str1[] = "Sample string";
char str2[40];
strncpy (str2, str1, 5);
str2[5] = '\0';   //le kell zárnunk '\0' karakterrel, ugyanis
                  strncpy-nél nem kerül be automatikusan a '\0' a
                  string végére
printf ("str1: %s\nstr2: %s\n",str1,str2);
```

Output:

str1: Sample string

str2: Sampl

Fontosabb *string.h* függvények

- `char * strcat ( char * destination, const char * source );`

Összefűz két stringet. Első paraméterben a cél karaktertömböt, másodikban a hozzáfűzendő karaktertömböt kell megadnunk.

Az első paraméterben lévő string végéről törli a \0 karaktert, hozzáfűzi a 2. paraméterben lévő karaktersorozatot, majd a legvégére tesz egy \0 karaktert.

Ahogy az strcpy-nél, itt is figyelni kell arra, hogy a cél karaktertömbben elférjen a két string! Ha nem fér el, túlindexelés lesz…

Példa:

```
char str1[] = "Sample string";
char str2[40] = "Destination ";
strcat(str2, str1);
printf ("str1: %s\nstr2: %s\n",str1,str2);
```

Output:

str1: Sample string

str2: Destination Sample string

Fontosabb *string.h* függvények

- `char * strncat ( char * destination, const char * source, size_t num );`

Ugyanaz, mint az előző, de csak „n” db. karakterig füzi hozzá a stringet a másikhoz.

(Az előző „n-es” strcncpy függvénnyel ellentétben itt nem kell külön kiraknunk a ’\0’ karaktert)

Példa:

```
char str1[] = "Sample string";
char str2[40] = "Destination ";
strncat(str2, str1, 5);
printf ("str1: %s\nstr2: %s\n",str1,str2);
```

Output:

str1: Sample string

str2: Destination Sampl

Fontosabb *string.h* függvények

- `int strcmp ( const char * str1, const char * str2 );`

Összehasonlítja két string tartalmát, karakterenként.

Ha a két string tartalma azonos, 0-val tér vissza.

0-nál nagyobb értékkel tér vissza, ha a két string első eltérő karaktere közül az első stringben nagyobb a karakter (magyarul az első string eltérő karaktere hátrébb van az ascii táblában, mint a második stringé).

0-nál kisebb az érték különben.

Példa:

```
char str1[] = "Sample string";
char str2[40] = "Samplf string";
int cmp = strcmp(str1, str2);
if (cmp == 0) {
      printf("A ket string egyforma\n");
} else if (cmp > 0) {
      printf("str1 a 'nagyobb'\n");
} else {
      printf("str2 a 'nagyobb'\n");
}
```

Output:

```
str2 a 'nagyobb'
```

Fontosabb *string.h* függvények

- `int strncmp ( const char * str1, const char * str2, size_t num );`

Ugyanaz, mint az előző, de legfeljebb „n” karakterig vizsgálja a két stringet.

Példa:

```
char str1[] = "Sample string";
char str2[40] = "Samplf string";
int cmp = strncmp(str1, str2, 5);
if (cmp == 0) {
      printf("A ket string egyforma\n");
} else if (cmp > 0) {
      printf("str1 a 'nagyobb'\n");
} else {
      printf("str2 a 'nagyobb'\n");
}
```

Output:

```
A ket string egyforma
```

Fontosabb *string.h* függvények

- `const char * strchr ( const char * str, int character );`

Megkeresi egy karakter első előfordulását a stringben.

Első paraméter a string.

Második paraméter a keresett karakter.

Karakter pointer-t fog visszaadni, mely a megtalált karakterre mutat. Ha nem talált ilyen karaktert a stringben, null pointert fog visszaadni.

Példa:

```
char str1[] ="sample string";
char* c = strchr(str1, 's');
if (c != NULL) {
      int index = c - str1;   //a két címet kivonva egymásból
                              megkapjuk, hanyadik indexen volt
                              eredetileg a karakter
      printf("A megtalalt karakter a %d. indexen van\n", index);
} else {
      printf("Nem talalhato a karakter");
}
```

Output:

```
A megtalalt karakter a 0. indexen van
```

Fontosabb *string.h* függvények

- `const char * strrchr ( const char * str, int character );`

Hasonló az előzőhöz, de nem az első, hanem az utolsó előfordulását keresi meg a karakternek!

Példa:

```
char str1[] = "sample string";
char* c = strrchr(str1, 's');
if (c != NULL) {
      int index = c - str1;   //a két címet kivonva egymásból
                              megkapjuk, hanyadik indexen volt
                              eredetileg a karakter
      printf("A megtalalt karakter a %d. indexen van\n", index);
} else {
      printf("Nem talalhato a karakter");
}
```

Output:

A megtalalt karakter a 7. indexen van

Fontosabb *string.h* függvények

```
char * strstr (char * str1, const char * str2 );
```

Hasonló az strchr-hez, azonban nem egy karakter előfordulását, hanem egy string előfordulását (substring) keres egy másik stringben.

Az első paraméter a string, amelyben keresünk.

A második paraméter a keresett string.

Karakter pointerrel fog visszatérni, mely NULL, ha nem találta meg a substringet. Különben rámutat a substring kezdő karakterére.

Példa:

```
char str[] ="This is a simple string";
char * c;
c = strstr (str,"simple");   //"simple" stringet keresünk
if (c != NULL) {
      //Ha megtaláltuk a substringet, kicseréljük "sample"-ra!
      //Gyakorlatilag a megtalált pointer helyére bemásoljuk a
      "sample"-t, ezzel felül is írjuk az eredeti "simple"-t
      strncpy (c,"sample",6);
} else {
      printf("Nem talalhato a substring!\n");
}
printf("%s\n", str);
```

Output:

```
This is a sample string
```

Fontosabb *string.h* függvények

`char* strrev(char* s)`

Megfordítja egy string tartalmát.

Paraméterül a megfordítandó stringet kell megadnunk.

Visszatérési értéke a megfordított string kezdetére mutató karakter pointer.

Példa:

```
char str1[] = "Vajon ez egy palindroma?";
char str2[] = "indulagorogaludni";
printf("%s\n%s\n\n", str1, str2);
strrev(str1);
strrev(str2);
printf("%s\n%s\n", str1, str2);
```

Output:

Vajon ez egy palindroma?

indulagorogaludni

?amordnilap yge ze nojaV

indulagorogaludni

Fontosabb *string.h* függvények

```
char *strset( const char *str, char ch );
```

Egy string összes karakterét egy megadott karakterre állítja.

Első paramétere a string.

Második paramétere a karakter, melyre állítani szeretnénk a string összes karakterét.

Visszatérési értéke egy karakter pointer, mely a string kezdetére mutat.

Példa:

```
char str1[] ="sample string";
printf("%s\n\n", str1);
strset(str1, 'x');
printf("%s\n", str1);
```

Output:

sample string

xxxxxxxxxxxxx

```
char *strnset(char *string, int c, size_t n);
```

Ugyanaz, mint az előző, de csak „n” db karaktert fog átállítani.

Példa

```
/*
A forrás stringből a célba csak karaktereket másol át, ha azok nagy betűsek voltak,
kicsivé alakítja Pl. "Indul a gorog aludni" = "indulagorogaludni"
*/
void strCopyWithoutSpaceAndLowerCase(char *dest, char *src) {
        int length = strlen(src);
        int i, j = 0;
        /*A különbség 'a' és 'A' betű között a kódtáblában*/
        int diffBetweenLowerAndUpper = 'a' - 'A';
        for (i = 0; i < length; i++) {
                //ha kisbetűs, azt beletehetjük a célba
                if (src[i] >= 'a' && src[i] <= 'z') {
                        dest[j] = src[i];
                        j++;
                        /*ha nagybetű, azt kicsivé alakítjuk*/
                } else if (src[i] >= 'A' && src[i] <= 'Z') {
                        dest[j] = src[i] + diffBetweenLowerAndUpper;
                        j++;
                }
                /*más esetben nem rak karaktert a cél stringbe*/
        }
        /*string végét lezárjuk*/
        dest[j] = '\0';
}
```