

Programozás C nyelven

(9. ELŐADÁS)

Sapientia EMTE

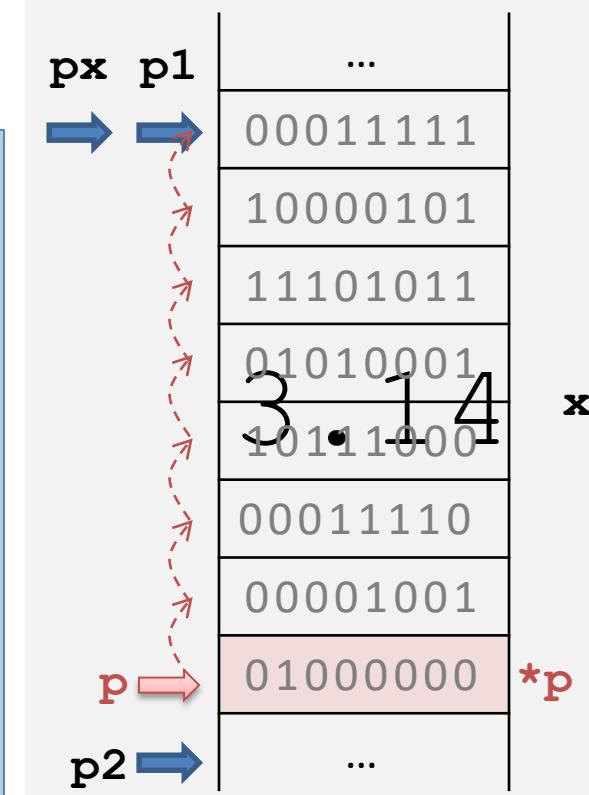
2020-21





POINTEREK – ismétlés

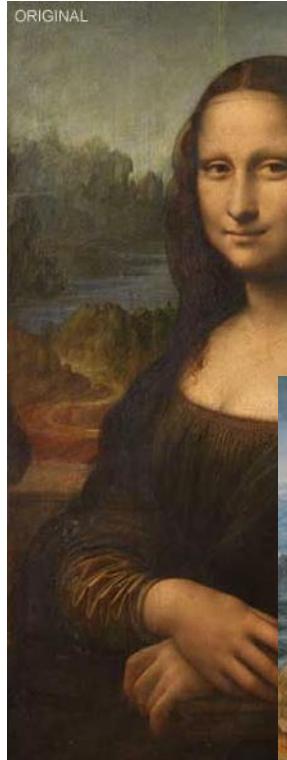
```
int main() {
    double x = 3.14, *px = &x;
    unsigned char *p, *p1, *p2;
    p1 = (unsigned char*)px;
    p2 = p1 + sizeof(double);
    for ( p = p2 - 1 ; p >= p1 ; --p ) {
        binarisan(*p); printf(" ");
    }
    return 0;
}
```



```
void binarisan (unsigned char c){
    int n = sizeof(unsigned char) * 8;
    int i, b[n];
    for ( i = 0 ; i < n ; ++i ) { b[i] = c%2; c /= 2; }
    for ( i = n-1 ; i >= 0 ; --i ) { printf("%i", b[i]); }
}
```

01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111

POINTER-ek alkalmazásai (címszerinti paraméterátadás)



75001 Paris, France

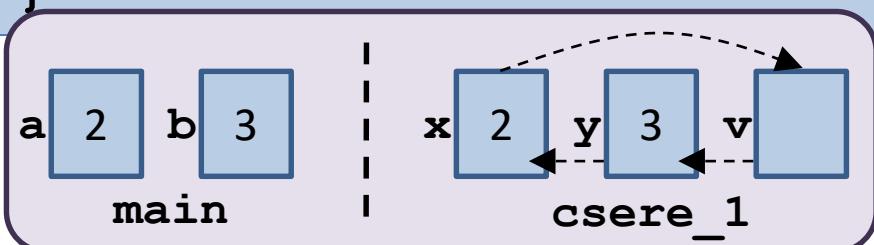


POINTER-ek alkalmazásai (címszerinti paraméterátadás)

Írj függvényt két változó tartalmának kicserélésére!

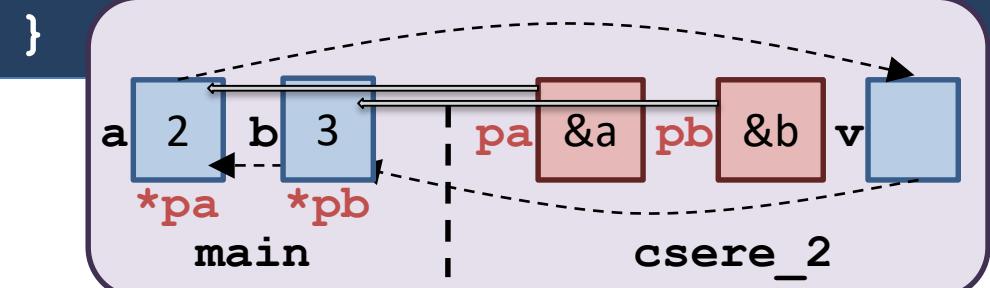
```
void csere_1(int,int);
int main(){
    int a = 2, b = 3;
    csere_1(a,b);
    printf("%i %i", a, b);
    return 0;      23_
}
```

```
void csere_1(int x, int y){
    int v;
    v = x; x = y; y = v;
}
```



```
void csere_2(int*,int*);
int main(){
    int a = 2, b = 3;
    csere_2(&a,&b);
    printf("%i %i", a, b); 32_
    return 0;
}
```

```
void csere_2(int *pa, int *pb){
    int v;
    v = *pa; *pa = *pb; *pb = v;
}
```

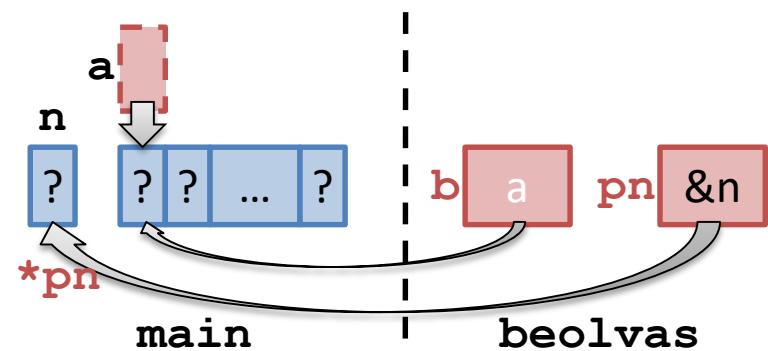


POINTER-ek alkalmazásai

(1D-tömbök átadása függvénynek)

Írj függvényeket számSORozat beOLVASÁSáRA/kiíRÁSáRA!

```
void beolvas(int*,int*);  
void kiir(int*,int);  
  
int main(){  
    int n, a[100];  
    beolvas(a,&n);  
    kiir(a,n);  
    return 0;  
}
```



```
void beolvas(int *b, int *pn){  
    freopen("be.txt","r", stdin);  
  
    int i;  
    scanf("%i", pn);  
    for( i = 0 ; i < *pn ; ++i ){  
        scanf("%i", &b[i]); //b+i  
    }  
}  
  
freopen("CON","r", stdin);
```

A tömbök implicity cím-szerint adódnak át!

be.txt
5
2 3 51 0 -6

POINTER-ek alkalmazásai (1D-tömbök átadása függvénynek)

Írj függvényeket számSORozat beOLVASÁSáRA/kiíRÁSáRA!

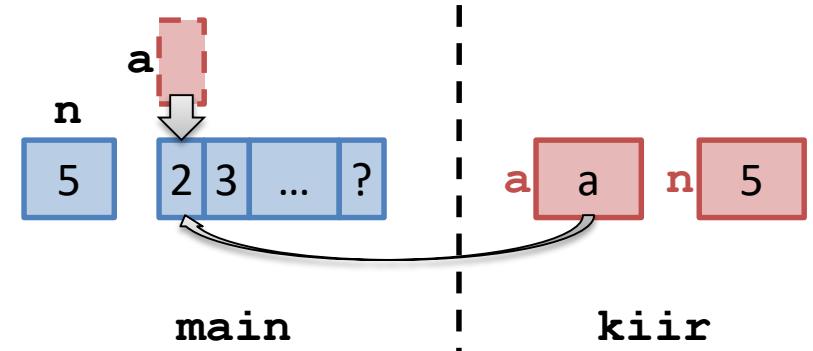
```
void beolvas(int*,int*);  
void kiir(int*,int);  
  
int main(){  
    int n, a[100];  
    beolvas(a,&n);  
    kiir(a,n);  
    return 0;  
}
```

```
void kiir(int *a, int n){  
    int i;  
    for( i = 0 ; i < n ; ++i ){  
        printf("%i ", a[i]);  
    }  
    printf("\n");
```

2 3 51 0 -6
-

szamsor.txt

```
5  
2 3 51 0 -6
```



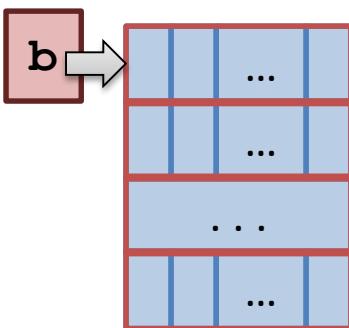
POINTER-ek alkalmazásai (2D-tömbök átadása függvénynek)

Írj függvényt 2D-tömbben tárolt mátrix kiírására!

```
void kiir ( int(*)[100], int, int );
int main() {
    int n, m, b[50][100];
    ...
    kiir(b,n,m);
    return 0;
}
```

int *x[100];// int-pointer TÖMB
int (*x)[100];// POINTER int-tömbre

```
void kiir ( int(*b)[100], int n, int m ){
    int i,j;
    for( i = 0 ; i < n ; ++i ){
        for( j = 0 ; j < m ; ++j ){
            printf("%i ", b[i][j]);
        }
        printf("\n");
    }
}
```



Generálj n hosszú véletlen-számSORozatot (n<10000) ,
melynek elemei a [0,10) intervallumba esenek. Készíts
szám-előfordulási statisztikát!

```
#include <stdlib.h>
#include <time.h>
int main(){
    int n,i,a[10000],st[10]={0};
    srand(time(NULL));
    scanf("%i", &n);
    for( i = 0 ; i < n ; ++i ){
        a[i] = rand() % 10;
        ++st[a[i]];
    }
    for( i=0 ; i<10 ; ++i ){ printf("%i:%i\n",i,st[i]); }
    return 0;
}
```

The diagram illustrates the execution of the C program through four states of memory:

- Initial State:** An array of 10 slots labeled 0 to 9. Each slot contains a question mark '?'.
- After srand and first loop iteration:** The array shows the first few elements being populated. Slots 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 contain 0, 0, 0, 0, 0, 0, 0, 0, 0, and 0 respectively. A red box highlights the first ten slots. A speech bubble labeled "pazarlás" points to the slot at index 5.
- After the first loop iteration:** The array shows the state after one iteration. Slots 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 contain 2, 3, 5, 3, 3, ?, ?, ?, ?, and ? respectively. A red box highlights the first ten slots.
- Final State:** The array shows the final state after all iterations. Slots 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 contain 0, 0, 1, 3, 0, 1, 0, 0, 0, and 0 respectively. A red box highlights the first ten slots.

A vertical column of numbers on the right lists the counts for each digit from 0 to 9, with a '-' entry at the bottom:

5
0:0
1:0
2:1
3:3
4:0
5:1
6:0
7:0
8:0
9:0
-

Dinamikus helyfoglalás/változók (malloc, calloc, realloc, free)

```
#include <stdlib.h> // tartalmazza malloc,... deklarációit
#include <time.h>
int main(){
    int n, i, *a, *st;
    srand(time(NULL));
    scanf("%i", &n);
    a = (int*)malloc(n*sizeof(int)); if(a==NULL) {... return 0;}
    st = (int*)calloc(10,sizeof(int)); if(st==NULL) {...}
    for( i = 0 ; i < n ; ++i ){
        a[i] = rand() % 10;
        ++st[a[i]];
    }
    for( i=0 ; i<10 ; ++i ) { printf("%i:%i\n", i, st[i]); }
    free(a); free(st);
    return 0;
}
```

The diagram illustrates the state of variables and memory after each iteration of the loop. It shows three pointers: `a` (blue box), `st` (red box), and `n` (not shown). The variable `a` points to a memory block of size `n` containing integers from 0 to 4. The variable `st` points to a memory block of size 10 containing zeros. The variable `n` contains the value 5. The code then enters a loop where it fills the array `a` with random values (0-9) and increments the corresponding index in the `st` array. After the first iteration, the `a` array contains [2, 3, 5, 3, 3] and the `st` array contains [0, 0, 1, 3, 0, 1, 0, 0, 0, 0]. This pattern repeats until the `st` array is filled with values from 0 to 9.

5
0:0
1:0
2:1
3:3
4:0
5:1
6:0
7:0
8:0
9:0
-

HEAP

Dinamikus helyfoglalás/változók (**malloc**, **calloc**, **realloc**, **free**)

- **void* malloc (size_t size);**
 - Size of the memory block, in bytes
 - `size_t` is an unsigned integral type
 - If the function failed to allocate the requested block of memory, a *null pointer* is returned
- **void* calloc (size_t num, size_t size);**
 - ... and initializes all its bits to zero
- **void* realloc (void* ptr, size_t size);**
 - Changes the size of the memory block pointed to by `ptr`
 - The function may move the memory block to a new location (whose address is returned by the function)
- **void free (void* ptr); // `ptr = NULL;`**

Dinamikus helyfoglalás/változók **(malloc, calloc, realloc, free)**

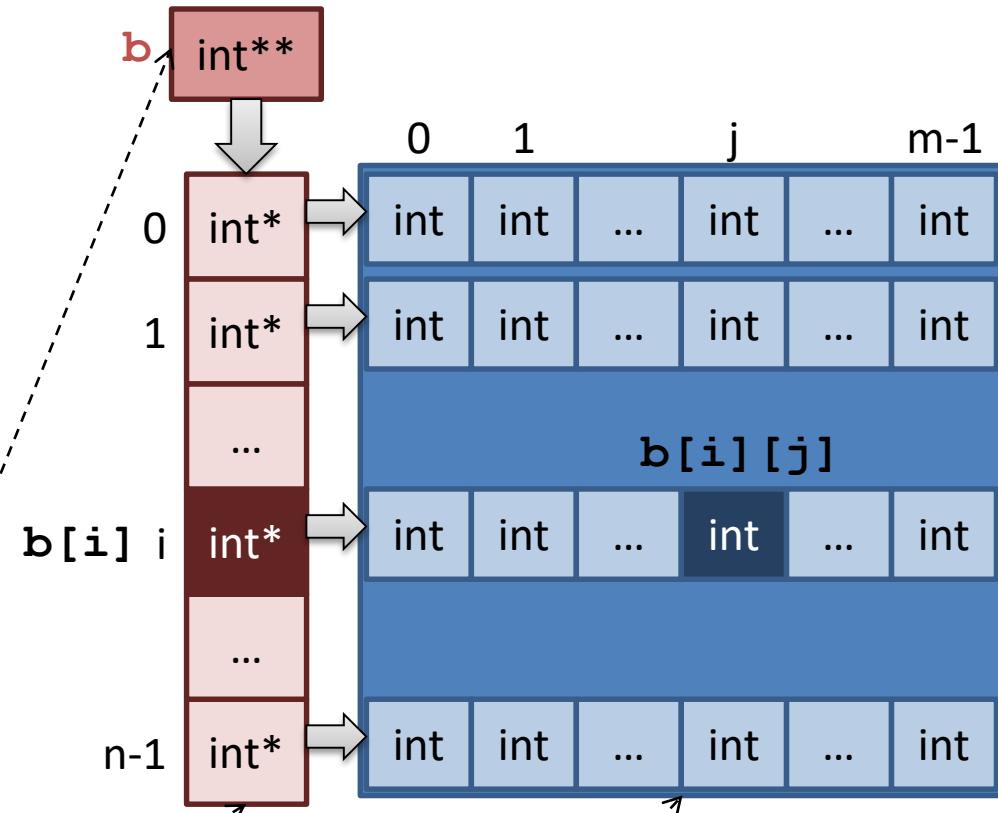
- minden malloc/calloc/realloc-ot kövessen NULL-tesztelő if.
- minden malloc/calloc/realloc-nak legyen meg a free párja.
- tanácsos a free utasítást követően a pointer nullázni.
 - megtévesztő, ha egy pointer nem 0, mégse mutat lefoglalt területre.
- biztonságosabb, ha a NULL makró helyet 0-t használunk.
 - C++-ban a null kulcsszót.

Biztonságos állomány megnyitás

```
. . .
if( !freopen ("bemenet.txt", "r", stdin) ) {
    printf("Sikertelen allomany megnyitas");
    return 0;
}
. . .
```



2D dinamikus tömbök



```
int n, m, i, **b;  
scanf("%i%i", &n, &m);  
b = (int**)malloc(n*sizeof(int*));  
for( i = 0 ; i < n ; ++i ) {  
    b[i] = (int*)malloc(m*sizeof(int));  
}
```

```
for( i=0 ; i<n ; ++i ) {  
    free(b[i]);  
}  
free(b);
```

Függvény- pointerek (qsort)

```
int double_cmp ( const void *, const void * );
int main(){
    . . .
    int n, i; double *a;
    scanf("%i", &n);
    a = (double*)malloc(n*sizeof(double)); ...
    for ( i = 0 ; i < n ; ++i ){
        scanf("%i", &a[i]);
    }
    qsort ( a, n, sizeof(double), double_cmp );
    for ( i = 0 ; i < n ; ++i ){
        printf("%i ", a[i]);
    }
    free(a);
    return 0;
}
```

```
int double_cmp ( const void *p1, const void *p2 ){
    double *q1 = (double *)p1;
    double *q2 = (double *)p2;
    if ( *q1 < *q2 ) { return -1; }
    else if ( *q1 > *q2 ) { return 1; }
    else { return 0; }
}
```



Összefoglalás

Mit tesznek lehetővé a POINTERek?



- Címszerinti-paraméterátadás
 - A tömbök impliciten címszerint adódnak át
 - A 2D-tömbök nevei, mint pointerek, a 0. sor címét tárolják
- Dinamikus helyfoglalás:
 - **malloc**, **calloc**, **realloc**, **free**
 - dinamikus 1D, 2D-tömbök
- **void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))**
 - compar: címszerint átadott függvény