Programozási nyelvek 1

Szathmáry László
Debreceni Egyetem
Informatikai Kar

10. előadás

- tömbök, realloc
- dinamikus tömb

(utolsó módosítás: 2024. jan. 31.)

ismétlés

```
int main()
{
    int *a;
    int *b;

    a = malloc(sizeof(int));

    *a = 20;
    *b = 13;

    return 0;
}
```

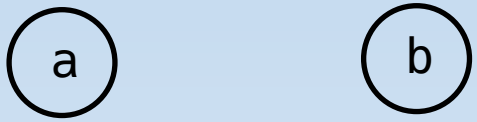
Ebben a kódban van 2 hiba. Melyek ezek?

ismétlés

```
int main()
{
    int *a;
    int *b;

    a = malloc(sizeof(int));

    *a = 20;
    *b = 13;

    return 0;
}
```

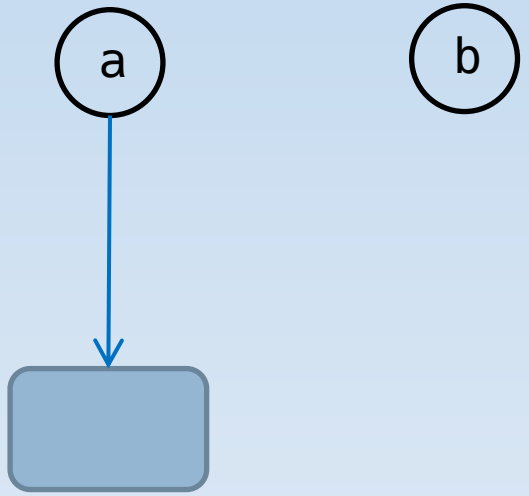
ismétlés

```
int main()
{
    int *a;
    int *b;

    a = malloc(sizeof(int));

    *a = 20;
    *b = 13;

    return 0;
}
```



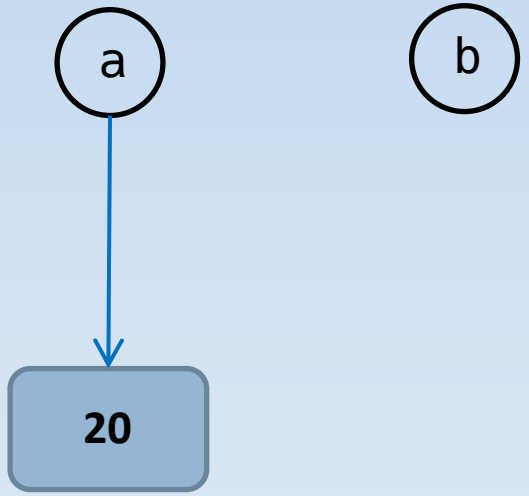
ismétlés

```
int main()
{
    int *a;
    int *b;

    a = malloc(sizeof(int));

    *a = 20;
    *b = 13;

    return 0;
}
```



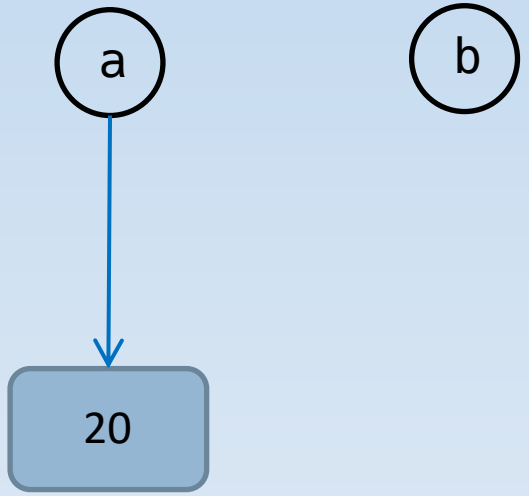
ismétlés

```
int main()
{
    int *a;
    int *b;

    a = malloc(sizeof(int));

    *a = 20;
    *b = 13;

    return 0;
}
```



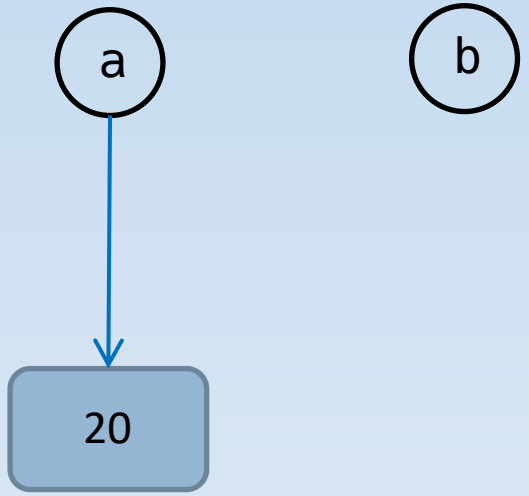
ismétlés

```
int main()
{
    int *a;
    int *b;

    a = malloc(sizeof(int));

    *a = 20;
    *b = 13;

    return 0;
}
```



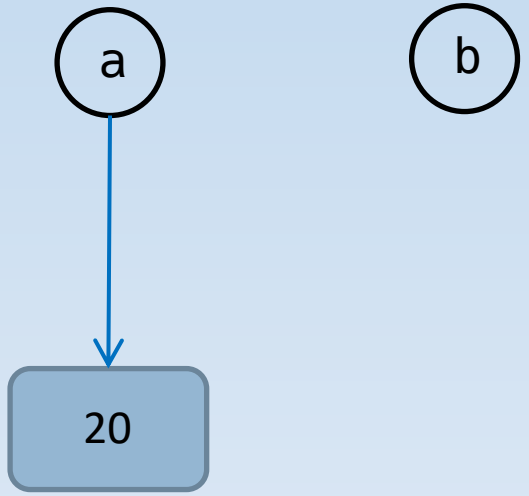
ismétlés

```
int main()
{
    int *a;
    int *b;

    a = malloc(sizeof(int));

    *a = 20;


    return 0;
}
```



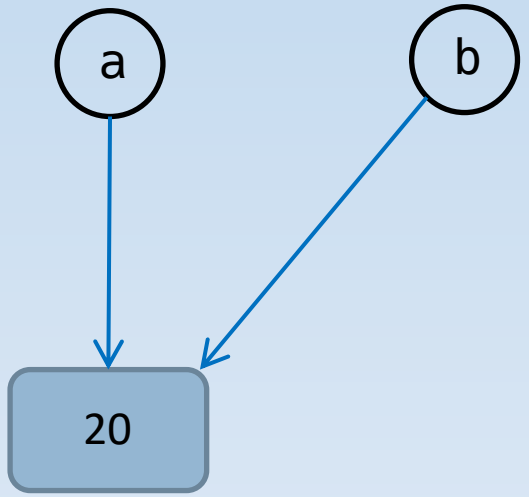
ismétlés

```
int main()
{
    int *a;
    int *b;

    a = malloc(sizeof(int));

    *a = 20;

    b = a;

    *b = 13;

    return 0;
}
```



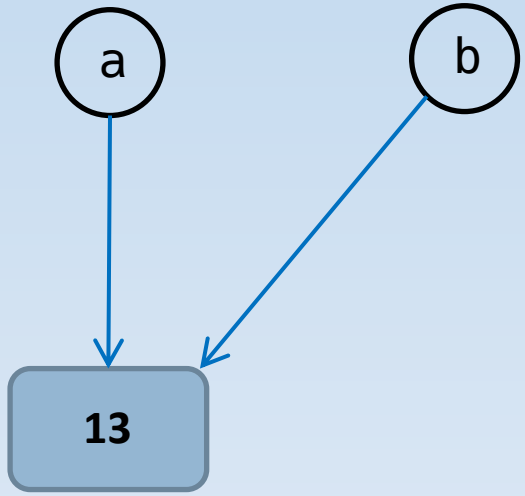
ismétlés

```
int main()
{
    int *a;
    int *b;

    a = malloc(sizeof(int));

    *a = 20;

    b = a;

    *b = 13;

    return 0;
}
```



ismétlés

```
int main()
{
    int *a;
    int *b;

    a = malloc(sizeof(int));

    *a = 20;

    b = a;

    *b = 13;

    return 0;
}
```



tömbök és realloc

1	2	3

Egy tömb mérete fix. A létrehozásakor meg kell adni a méretét, s ezen később sem lehet változtatni. Egy 3 elemű tömbbe nem lehet 4 elemet betenni.

B	e	a	\0	B	e
a	\0	1	2	3	B
e	a	\0	B	e	a
\0					

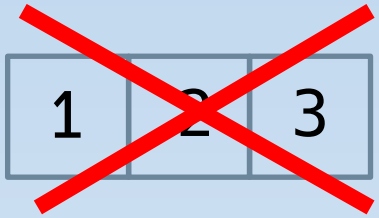
tömbök és realloc

tömbök és realloc

tömbök és realloc

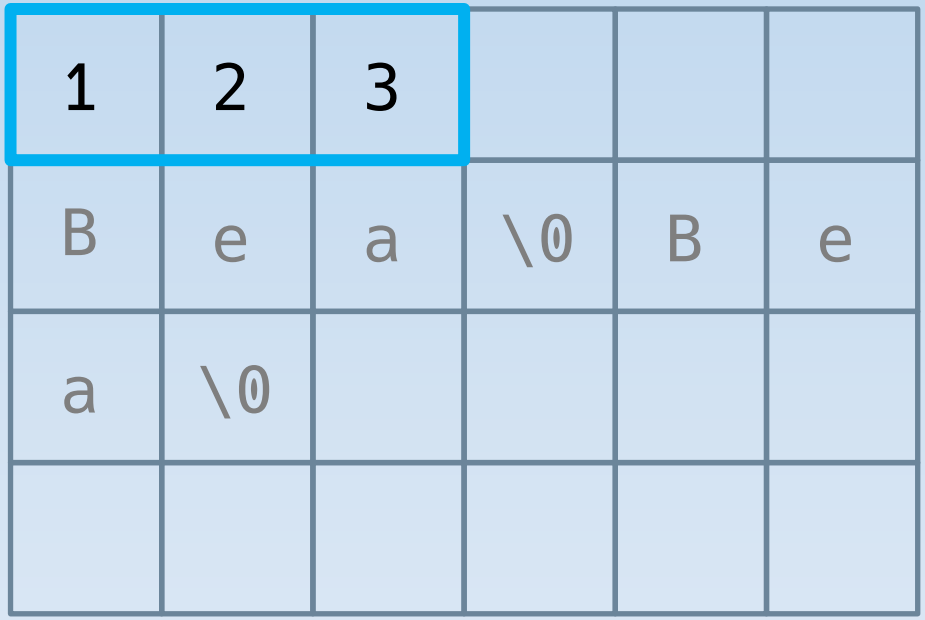
tömbök és realloc

tömbök és realloc



tömbök és realloc

tömbök és realloc




```
int *list = malloc(3 * sizeof(int));

list[0] = 1;
list[1] = 2;
list[2] = 3;
```

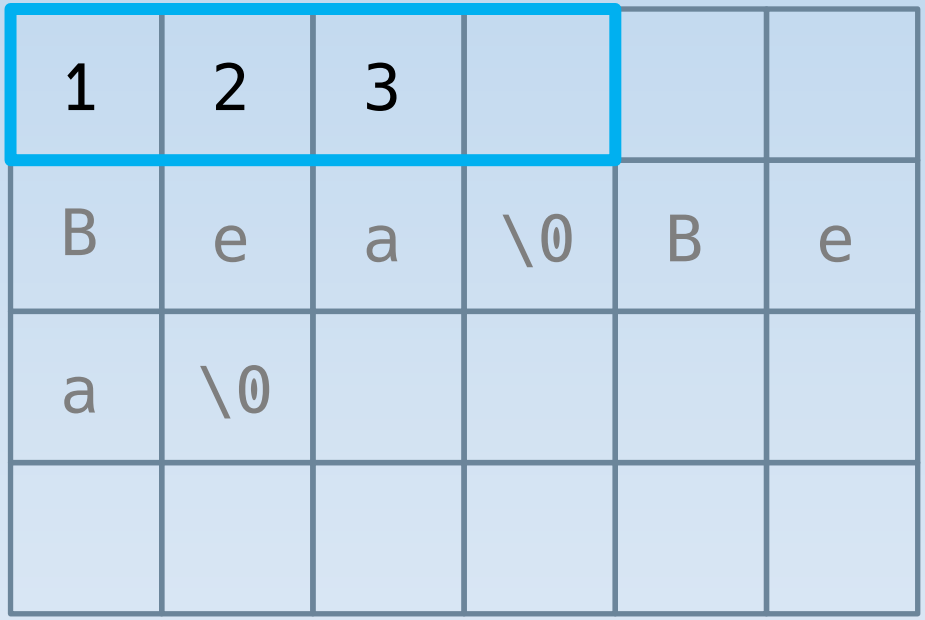
`realloc()`

Ha a lefoglalt terület után van elég szabad terület, akkor a lefoglalt területet kiterjeszti, az elemeket nem mozgatja át.

Ha a helyben való kiterjesztéshez nincs elég szabad hely, akkor keres egy megfelelő méretű szabad területet, lefoglalja, az elemeket átmásolja, s a régi területet felszabadítja.

Minden esetben a lefoglalt terület címével tér vissza.

tömbök és realloc




```
list = realloc(list, 4 * sizeof(int));
```

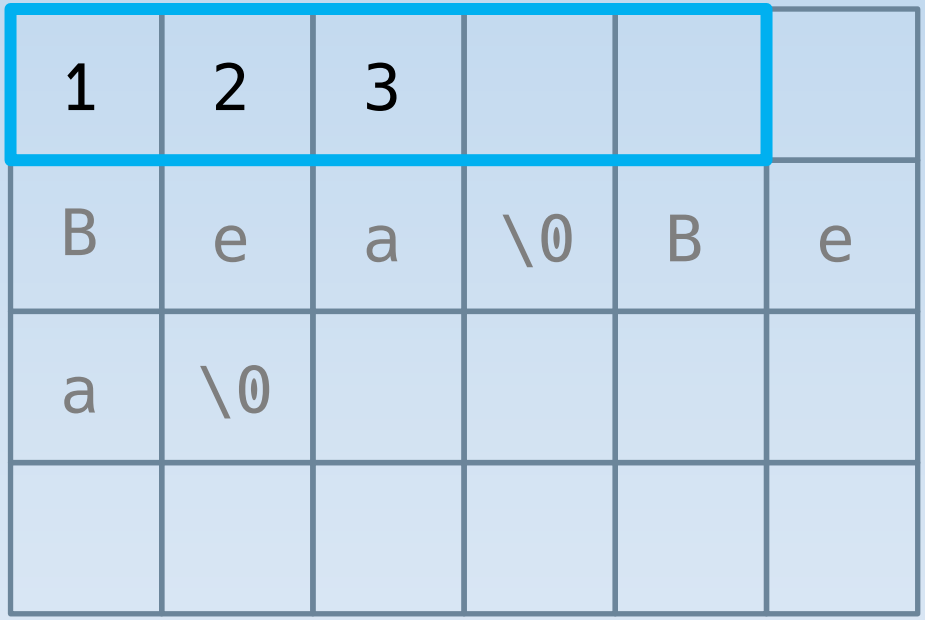
`realloc()`

Ha a lefoglalt terület után van elég szabad terület, akkor a lefoglalt területet kiterjeszti, az elemeket nem mozgatja át.

Ha a helyben való kiterjesztéshez nincs elég szabad hely, akkor keres egy megfelelő méretű szabad területet, lefoglalja, az elemeket átmásolja, s a régi területet felszabadítja.

Minden esetben a lefoglalt terület címével tér vissza.

tömbök és realloc




```
list = realloc(list, 5 * sizeof(int));
```

`realloc()`

Ha a lefoglalt terület után van elég szabad terület, akkor a lefoglalt területet kiterjeszti, az elemeket nem mozgatja át.

Ha a helyben való kiterjesztéshez nincs elég szabad hely, akkor keres egy megfelelő méretű szabad területet, lefoglalja, az elemeket átmásolja, s a régi területet felszabadítja.

Minden esetben a lefoglalt terület címével tér vissza.

tömbök és realloc

1	2	3			
B	e	a	\0	B	e
a	\0				

```
list = realloc(list, 6 * sizeof(int));
```

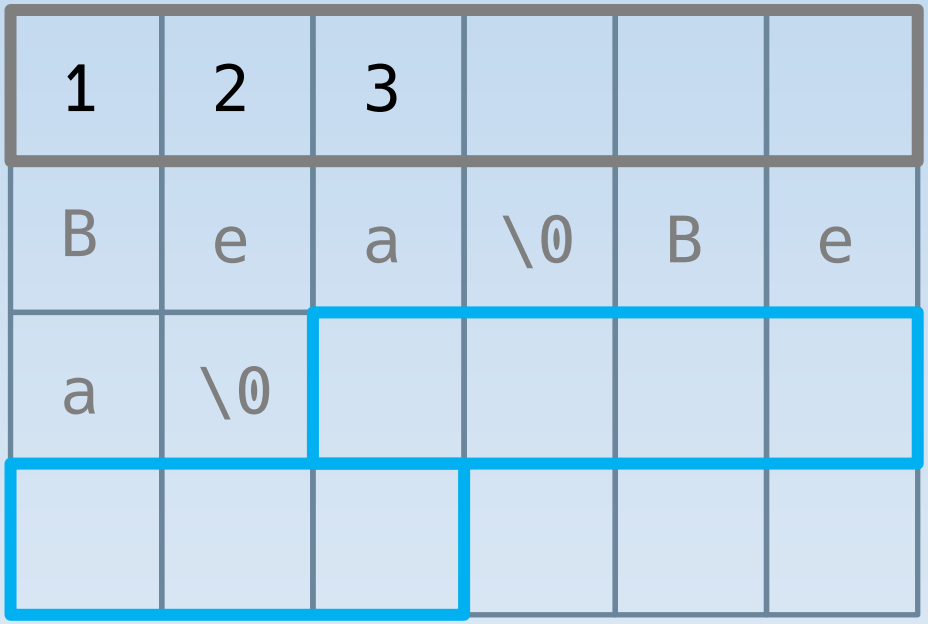
`realloc()`

Ha a lefoglalt terület után van elég szabad terület, akkor a lefoglalt területet kiterjeszti, az elemeket nem mozgatja át.

Ha a helyben való kiterjesztéshez nincs elég szabad hely, akkor keres egy megfelelő méretű szabad területet, lefoglalja, az elemeket átmásolja, s a régi területet felszabadítja.

Minden esetben a lefoglalt terület címével tér vissza.

tömbök és realloc




```
list = realloc(list, 7 * sizeof(int));
```

`realloc()`

Ha a lefoglalt terület után van elég szabad terület, akkor a lefoglalt területet kiterjeszti, az elemeket nem mozgatja át.

Ha a helyben való kiterjesztéshez nincs elég szabad hely, akkor keres egy megfelelő méretű szabad területet, lefoglalja, az elemeket átmásolja, s a régi területet felszabadítja.

Minden esetben a lefoglalt terület címével tér vissza.

tömbök és realloc


```
list = realloc(list, 7 * sizeof(int));
```

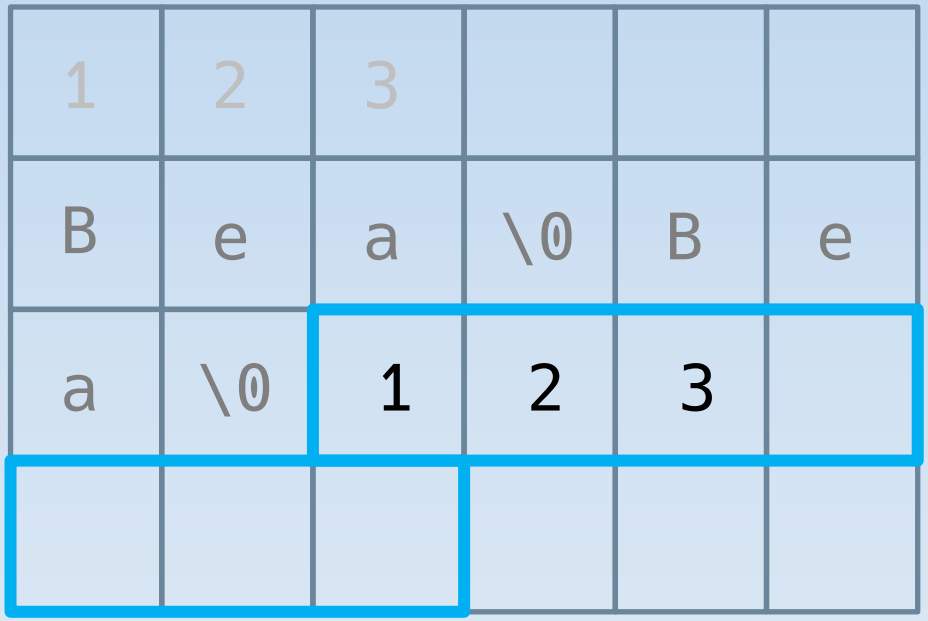
`realloc()`

Ha a lefoglalt terület után van elég szabad terület, akkor a lefoglalt területet kiterjeszti, az elemeket nem mozgatja át.

Ha a helyben való kiterjesztéshez nincs elég szabad hely, akkor keres egy megfelelő méretű szabad területet, lefoglalja, az elemeket átmásolja, s a régi területet felszabadítja.

Minden esetben a lefoglalt terület címével tér vissza.

tömbök és realloc




```
list = realloc(list, 7 * sizeof(int));
```

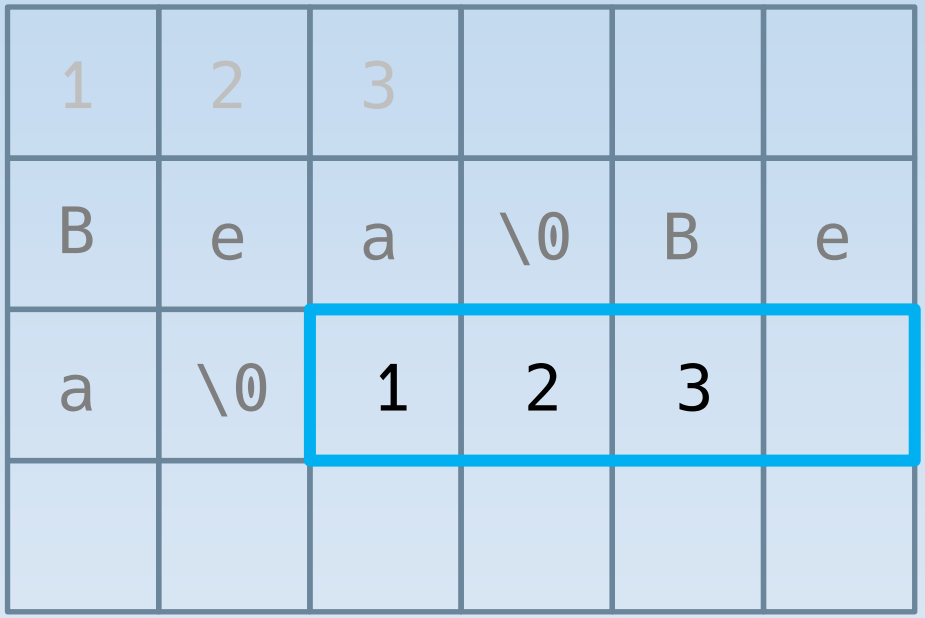
`realloc()`

Ha a lefoglalt terület után van elég szabad terület, akkor a lefoglalt területet kiterjeszti, az elemeket nem mozgatja át.

Ha a helyben való kiterjesztéshez nincs elég szabad hely, akkor keres egy megfelelő méretű szabad területet, lefoglalja, az elemeket átmásolja, s a régi területet felszabadítja.

Minden esetben a lefoglalt terület címével tér vissza.

tömbök és realloc




```
list = realloc(list, 4 * sizeof(int));
```

A `realloc()` -kal akár csökkenteni is lehet a lefoglalt terület méretét (*shrink*).

`realloc()`

Ha a lefoglalt terület után van elég szabad terület, akkor a lefoglalt területet kiterjeszti, az elemeket nem mozgatja át.

Ha a helyben való kiterjesztéshez nincs elég szabad hely, akkor keres egy megfelelő méretű szabad területet, lefoglalja, az elemeket átmásolja, s a régi területet felszabadítja.

Minden esetben a lefoglalt terület címével tér vissza.

tömbök és realloc

```
 4  int main()
 5  {
 6      int *list = malloc(3 * sizeof(int));
 7      if (list == NULL) {
 8          exit(1);
 9      }
10      list[0] = 1;
11      list[1] = 2;
12      list[2] = 3;
13
14      list = realloc(list, 4 * sizeof(int));
15
16      list[3] = 4;
17
18      for (int i = 0; i < 4; ++i)
19      {
20          printf("%d\n", list[i]);
21      }
22
23      free(list);
24
25      return 0;
26  }
```

dinamikus tömb

A magas szintű programozási nyelvek általában rendelkeznek **dinamikus tömb** adatszerkezettel. Ez általában objektum-orientált nyelvekre jellemző adatszerkezet.
A dinamikus tömb tehát általában egy **objektum**. Az objektumon belül valójában egy statikus tömb található (ami dinamikusan lett lefoglalva). Ennek a tömbnek van egy **elemszám**a (hány elemet tettünk bele), s van egy **kapacitás**a (hány elem befogadására képes fizikailag).
Egy dinamikus tömb a létrehozásakor lefoglal magának egy N méretű statikus tömböt (ez lesz a kapacitás), de az elemszám 0 (még semmit se tettünk bele).
Ahogy tesszük be az elemeket, előbb-utóbb megtelik a tömb. Ha ekkor be akarunk tenni még egy elemet, akkor ez már nem fér be a statikus tömbbe.

dinamikus tömb

Az objektum figyeli, hogy mikor telítődik a statikus tömb, s ha már nem fér be több elem, akkor lefoglal egy nagyobb memóriaterületet (realloc!), átmásolja az eddigi elemeket, felszabadítja a régi (kisebb) tömb területét, majd ezt az újonnan lefoglalt (nagyobb) tömböt használja tovább. Ebbe már befér az új elem.

Vagyis a dinamikus tömb egy olyan tömb, ami igény esetén **automatikusan** megnöveli magát úgy, hogy nekünk, programozóknak ezzel nem kell foglalkoznunk.

Programozóként mit látunk? Annyi elemet pakolhatunk bele, amennyit csak akarunk (persze a RAM mérete miatt lesz egy limit).

Ötvözi a statikus tömb előnyös tulajdonságait (O(1) hozzáférés), ill. a láncolt listákhoz hasonlóan nyugodtan pakolhatunk bele.

dinamikus tömb

```
>>> li = []
>>> len(li)
0
>>> li.append(3)
>>> li.append(5)
>>> len(li)
2
>>> li.append(8)
>>> len(li)
3
>>> li
[3, 5, 8]
>>>
```

Python példa

dinamikus tömb

```
>>> li = []
>>> len(li)
0
>>> li.append(3)
>>> li.append(5)
>>> len(li)
2
>>> li.append(8)
>>> len(li)
3
>>> li
[3, 5, 8]
>>>
```

length: 0
capacity: 2

dinamikus tömb

```
>>> li = []
>>> len(li)
0
>>> li.append(3)
>>> li.append(5)
>>> len(li)
2
>>> li.append(8)
>>> len(li)
3
>>> li
[3, 5, 8]
>>>
```

length: **1**
capacity: 2

dinamikus tömb

```
>>> li = []
>>> len(li)
0
>>> li.append(3)
>>> li.append(5)
>>> len(li)
2
>>> li.append(8)
>>> len(li)
3
>>> li
[3, 5, 8]
>>>
```

3	**5**

length: **2**
capacity: 2

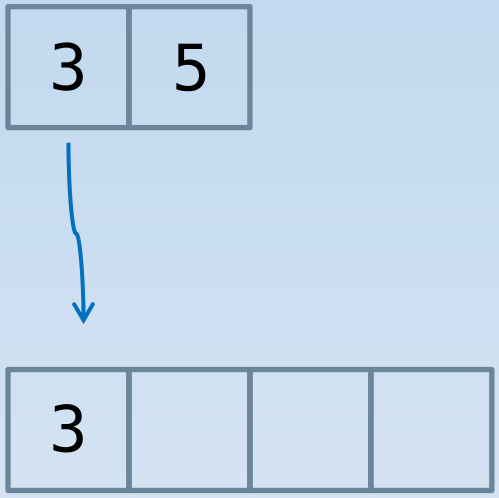
dinamikus tömb

```
>>> li = []
>>> len(li)
0
>>> li.append(3)
>>> li.append(5)
>>> len(li)
2
>>> li.append(8)
>>> len(li)
3
>>> li
[3, 5, 8]
>>>
```

length: 2
capacity: **4**

dinamikus tömb

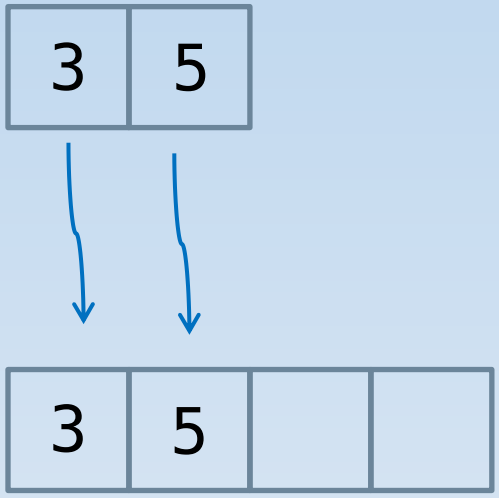
```
>>> li = []
>>> len(li)
0
>>> li.append(3)
>>> li.append(5)
>>> len(li)
2
>>> li.append(8)
>>> len(li)
3
>>> li
[3, 5, 8]
>>>
```



length: 2
capacity: 4

dinamikus tömb

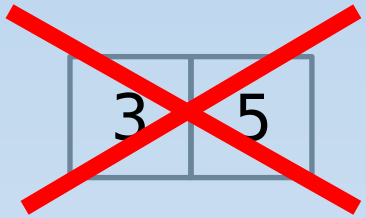
```
>>> li = []
>>> len(li)
0
>>> li.append(3)
>>> li.append(5)
>>> len(li)
2
>>> li.append(8)
>>> len(li)
3
>>> li
[3, 5, 8]
>>>
```



length: 2
capacity: 4

dinamikus tömb

```
>>> li = []
>>> len(li)
0
>>> li.append(3)
>>> li.append(5)
>>> len(li)
2
>>> li.append(8)
>>> len(li)
3
>>> li
[3, 5, 8]
>>>
```



length: 2
capacity: 4

dinamikus tömb

```
>>> li = []
>>> len(li)
0
>>> li.append(3)
>>> li.append(5)
>>> len(li)
2
>>> li.append(8)
>>> len(li)
3
>>> li
[3, 5, 8]
>>>
```

length: 2
capacity: 4

dinamikus tömb

```
>>> li = []
>>> len(li)
0
>>> li.append(3)
>>> li.append(5)
>>> len(li)
2
>>> li.append(8)
>>> len(li)
3
>>> li
[3, 5, 8]
>>>
```

length: **3**
capacity: 4

dinamikus tömb

```
60  int main()
61  {
62      DynArray *li = da_create();
63
64      // li.append(1);
65      da_append(li, 1);
66      for (int i = 2; i <= 20; ++i) {
67          da_append(li, i);
68      }
69
70      for (int i = 0; i < li->length; ++i) {
71          printf("%d ", li->elems[i]);
72      }
73      puts("");
74
75      li = da_destroy(li);
76
77      return 0;
78  }
```

C-ben is tudunk készíteni dinamikus tömb adatszerkezetet.

A teljes forráskód a tárgy GitHub oldalán található meg.

Házi feladat

- A K & R-féle „C Bibliában” nézzék át azokat a részeket, amikről szó volt az előadáson.
- Juhász István jegyzetéből nézzék át azokat a fogalmakat, amikről szó volt az előadáson (link).