Programozási nyelvek 1

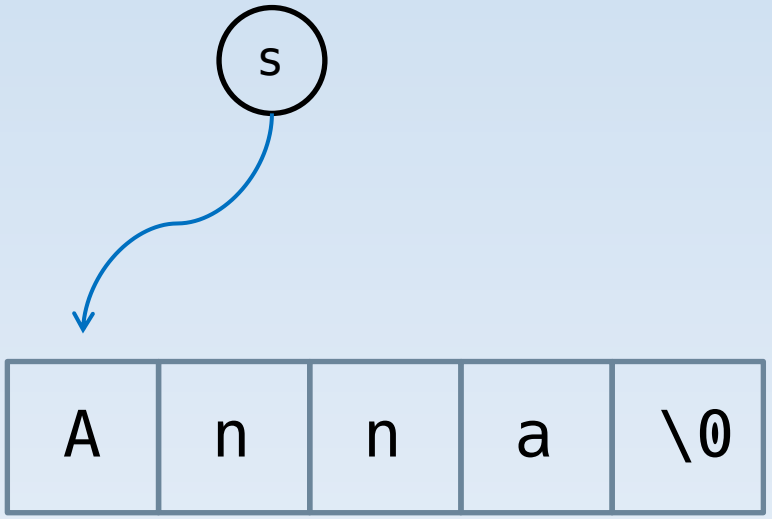
Szathmáry László
Debreceni Egyetem
Informatikai Kar

9. előadás
- mutatók (folyt.)
- memória (stack és heap)

(utolsó módosítás: 2025. febr. 20.)

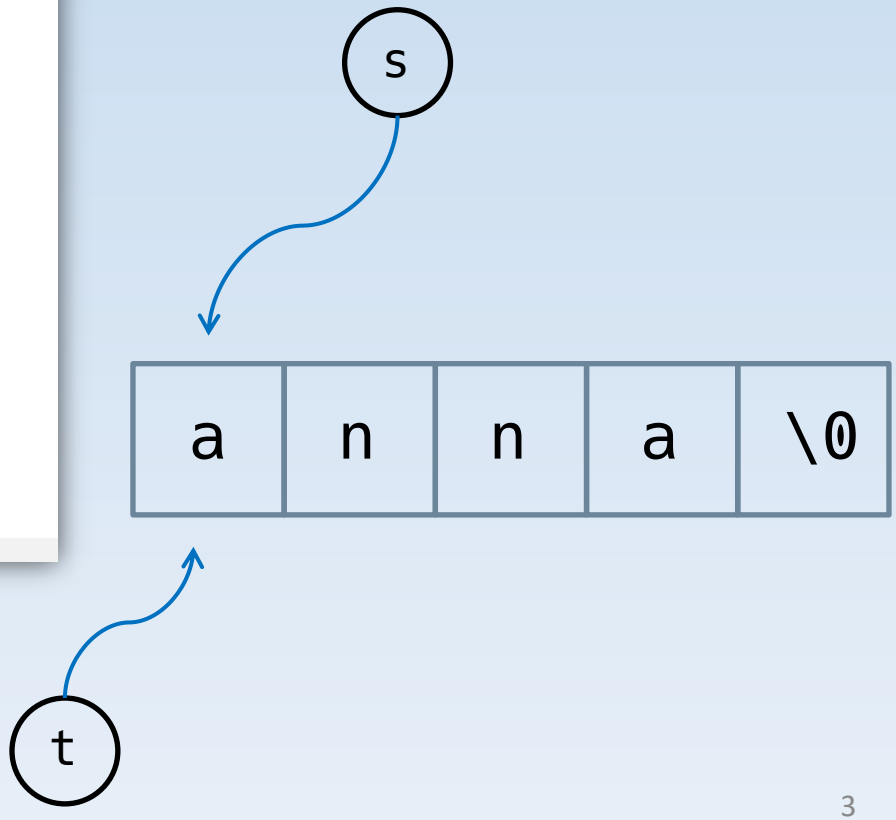
Mutatók és sztringek (folyt.)

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

int main()
{
    char* s = "Anna";

    printf("%s\n", s);

    return 0;
}
```



Mutatók és sztringek

```c
#include "prog1.h"
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main()
{
    char *s = get_string("Név: ");

    char *t = s;

    t[0] = toupper(t[0]);

    printf("%s\n", s);
    printf("%s\n", t);

    return 0;
}
```



Dinamikus memóriafoglalás

```
13  int main()
14  {
15      char *s = "anna";
16
17      char *t = malloc(strlen(s) + 1);
18
19      strcpy(t, s);
20
21      t[0] = toupper(t[0]);
22
23      printf("%s\n", s);
24      printf("%s\n", t);
25
26      free(t);
27
28      return 0;
29  }
```

`malloc()`:

Dinamikus memóriafoglalás. A `malloc()` a lefoglalt tárterület címét adja vissza, vagy NULL értéket, ha a foglalás sikertelen volt.

A dinamikusan lefoglalt memóriaterület **NEM** szabadul fel automatikusan!

Ezen területeket nekünk kell majd felszabadítani a `free()` meghívásával!

Ha nem szabadítjuk fel a dinamikusan lefoglalt tárterülete(ke)t, akkor memóriaszivárgás (*memory leak*) lép fel. A memóriaszivárgás **rossz** dolog!

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *p = malloc(20 * sizeof(char));

    return 0;
}
```

```
[11:46:33] ~/work/programozas_1_eloadas_2020_febr/09/sources $ gcc mem_leak.c
[11:46:45] ~/work/programozas_1_eloadas_2020_febr/09/sources $ valgrind ./a.out
==95562== Memcheck, a memory error detector
==95562== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==95562== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==95562== Command: ./a.out
==95562==
==95562==
==95562== HEAP SUMMARY:
==95562==     in use at exit: 20 bytes in 1 blocks
==95562==   total heap usage: 8 allocs, 7 frees, 4,733 bytes allocated
==95562==
==95562== LEAK SUMMARY:
==95562==    definitely lost: 20 bytes in 1 blocks
==95562==    indirectly lost: 0 bytes in 0 blocks
==95562==      possibly lost: 0 bytes in 0 blocks
==95562==    still reachable: 0 bytes in 0 blocks
==95562==         suppressed: 0 bytes in 0 blocks
==95562== Rerun with --leak-check=full to see details of leaked memory
==95562==
==95562== For lists of detected and suppressed errors, rerun with: -s
==95562== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[11:46:49] ~/work/programozas_1_eloadas_2020_febr/09/sources $
```

Lokális és globális változók

Alprogram: függvény vagy eljárás.

Az alprogramban deklarált változók az alprogram **lokális változó**i. A lokális változók az alprogramon kívülről nem láthatók, azokat az alprogram elrejti a külvilág elől.

Léteznek **globális változó**k is, melyeket nem az adott alprogramban deklaráltunk, hanem valahol rajta kívül, de az alprogram törzsében szabályosan hivatkozhatunk rájuk.

Egy alprogram **környezet**e alatt a globális változók együttesét értjük.

Azt a szituációt, amikor egy függvény megváltoztatja a paramétereit vagy a környezetét, a függvény **mellékhatás**ának nevezzük. A mellékhatást általában károsnak tartják.

```c
#include <stdio.h>

#define MAX 10

const double PI = 3.14159;

int counter = 0;

void f2()
{
    counter = 42;
}

void f1()
{
    int counter = 5;
    printf("f1: %d\n", counter);
}

void f0()
{
    printf("f0: %d\n", counter);
}

int main()
{
    printf("MAX értéke: %d\n", MAX);
    printf("PI értéke: %lf\n", PI);
    f0();
    f1();
    f2();
    printf("main: %d\n", counter);

    return 0;
}
```

Kérdés:

Mi a program kimenete?

```c
#include <stdio.h>

#define MAX 10

const double PI = 3.14159;

int counter = 0;

void f2()
{
    counter = 42;
}

void f1()
{
    int counter = 5;
    printf("f1: %d\n", counter);
}

void f0()
{
    printf("f0: %d\n", counter);
}

int main()
{
    printf("MAX értéke: %d\n", MAX);
    printf("PI értéke: %lf\n", PI);
    f0();
    f1();
    f2();
    printf("main: %d\n", counter);

    return 0;
}
```

nevesített konstans

nevesített konstansként használt globális változó

„sima” globális változó, bárki módosíthatja az értékét (nincs előtte a const módosító)

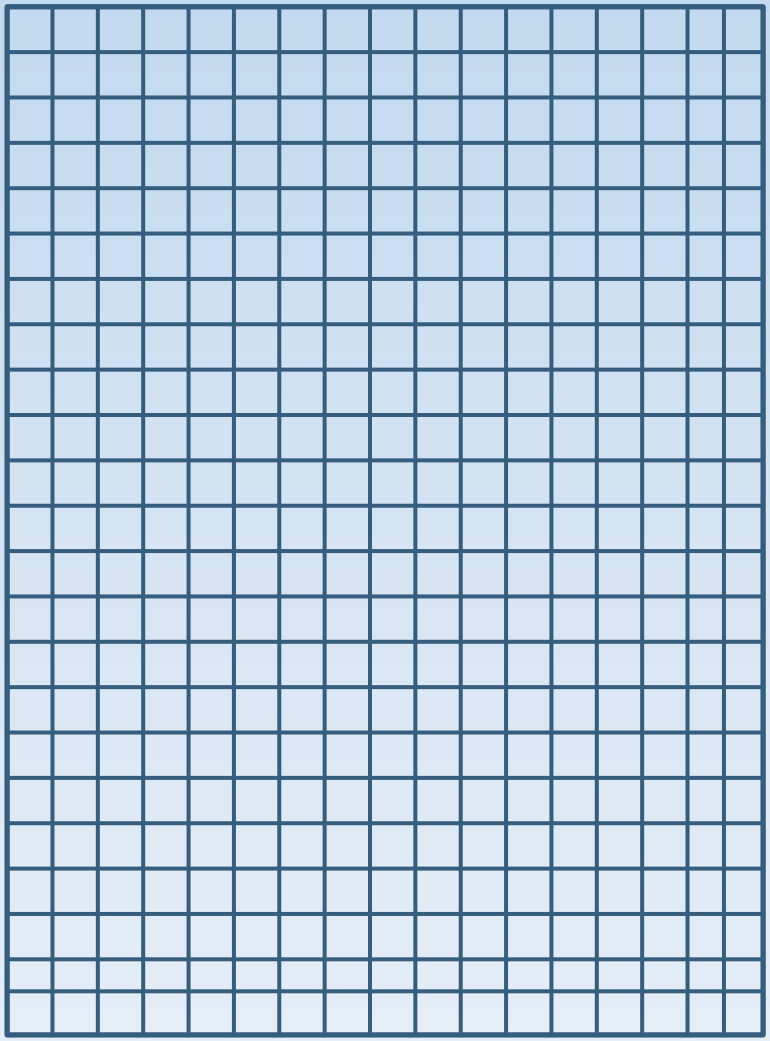
a globális counter értékét módosítjuk

counter itt egy lokális változó (deklaráltuk); a globális counter változót elrejti

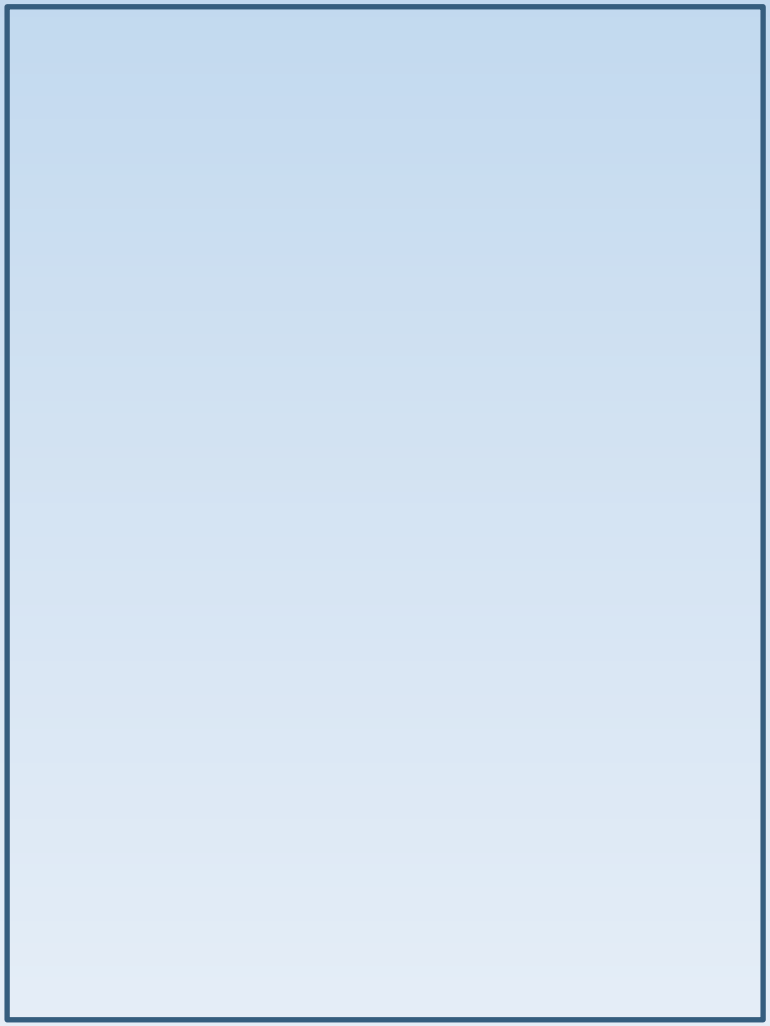
a globális változók minden függvényből látszanak

```
MAX értéke: 10
PI értéke: 3.141590
f0: 0
f1: 5
main: 42
```

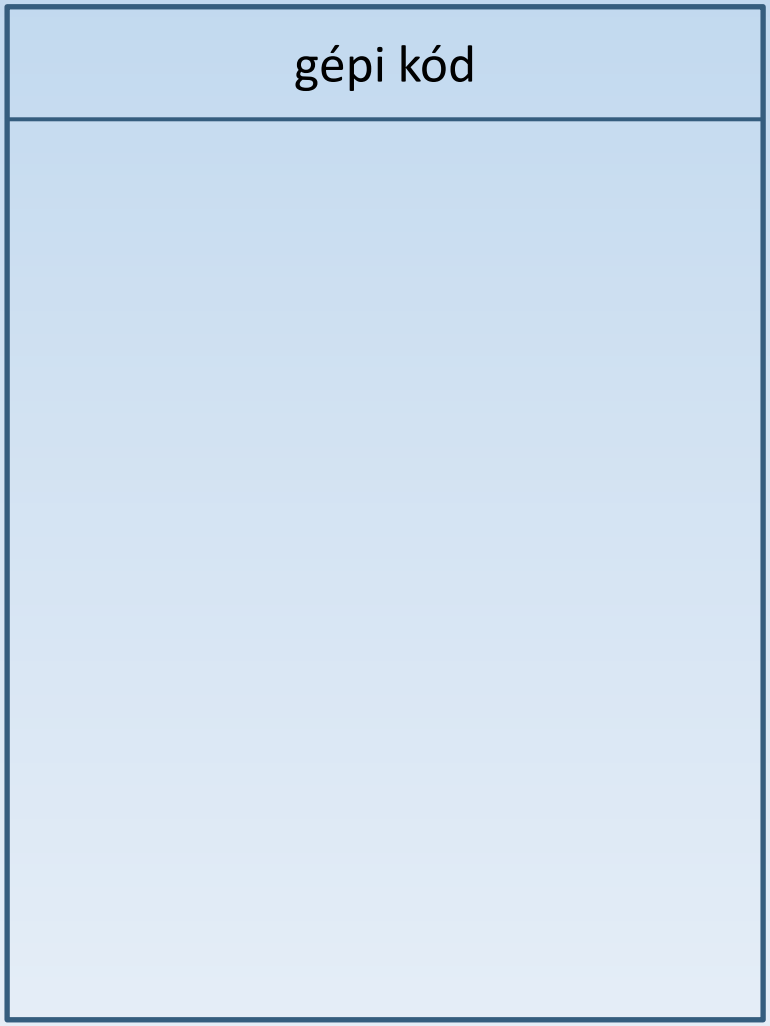
A memória (RAM) kiosztása



A memória (RAM) kiosztása



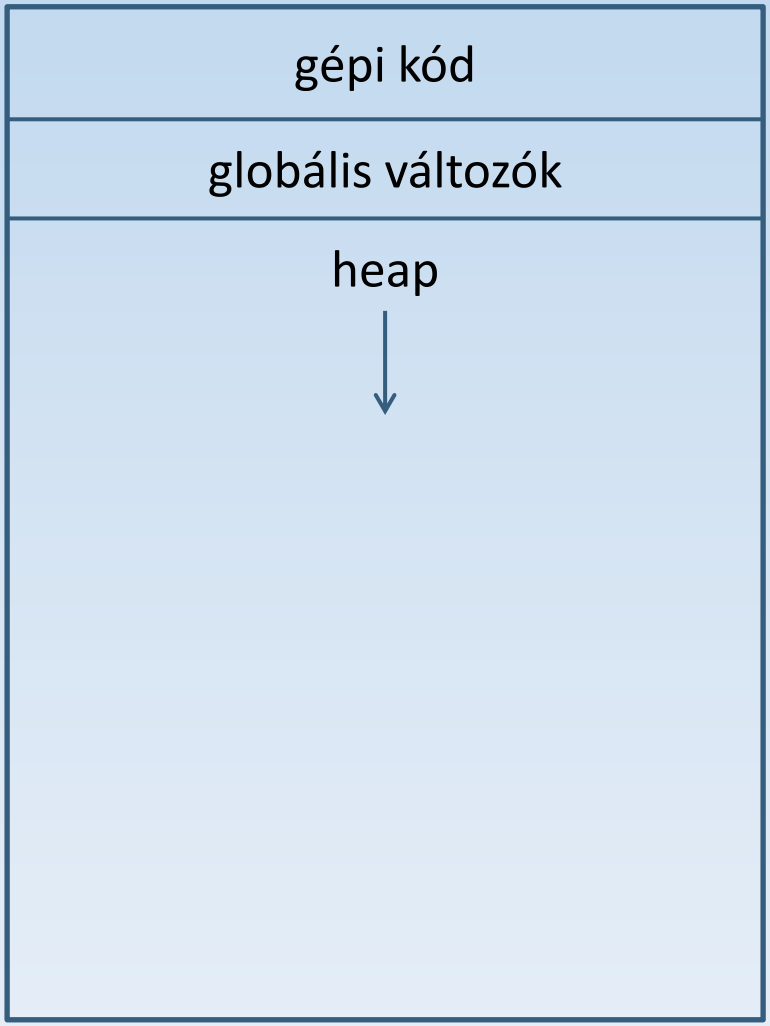
A memória (RAM) kiosztása



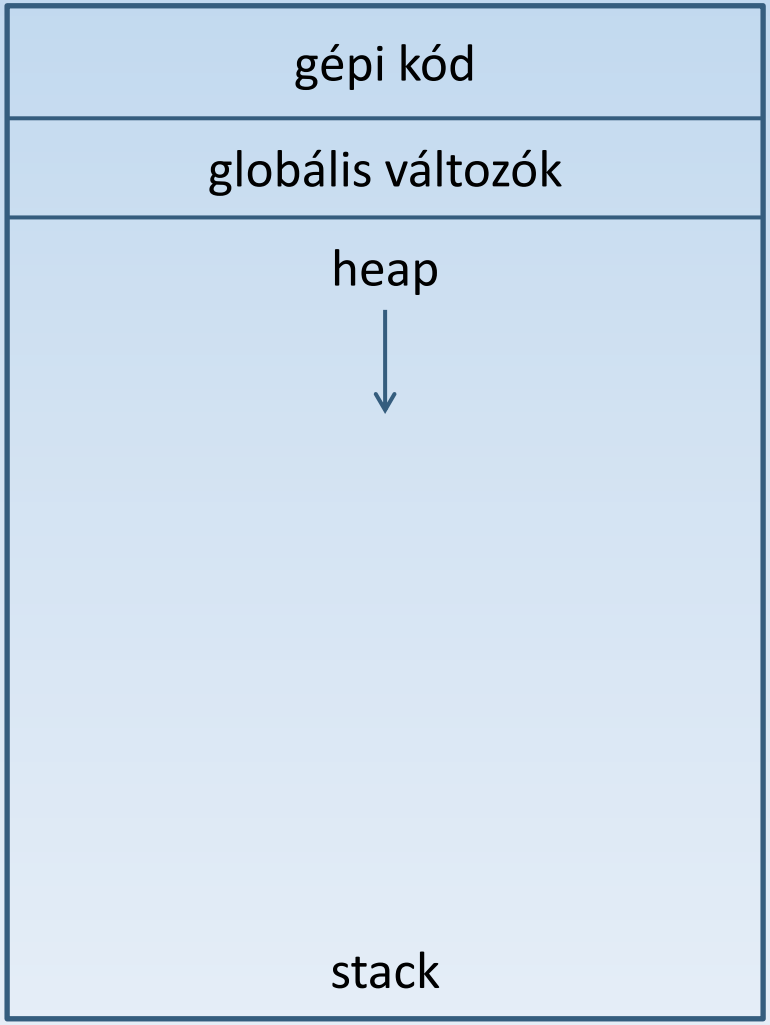
A memória (RAM) kiosztása

A memória (RAM) kiosztása

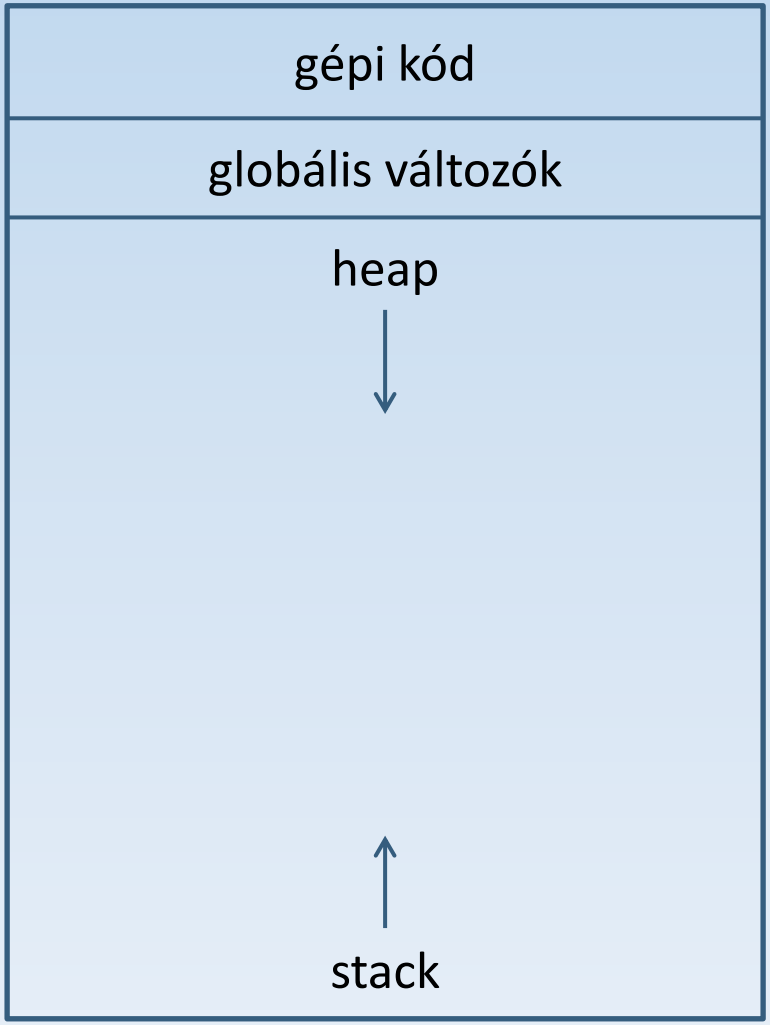
A memória (RAM) kiosztása



A memória (RAM) kiosztása

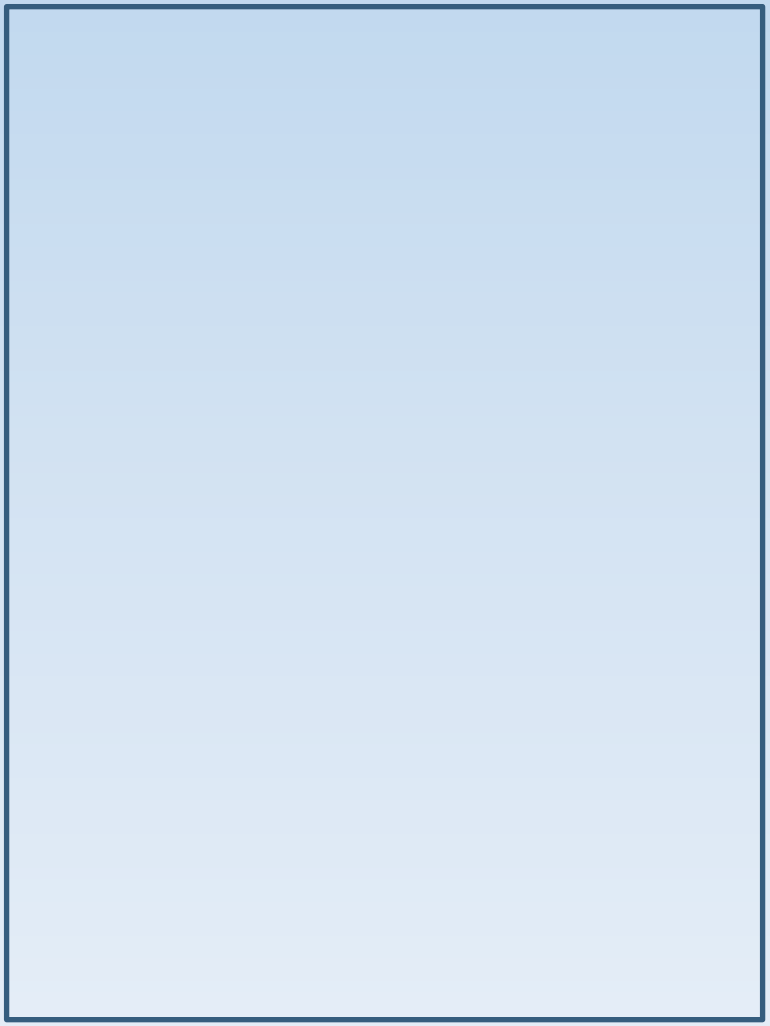


A memória (RAM) kiosztása



Stack

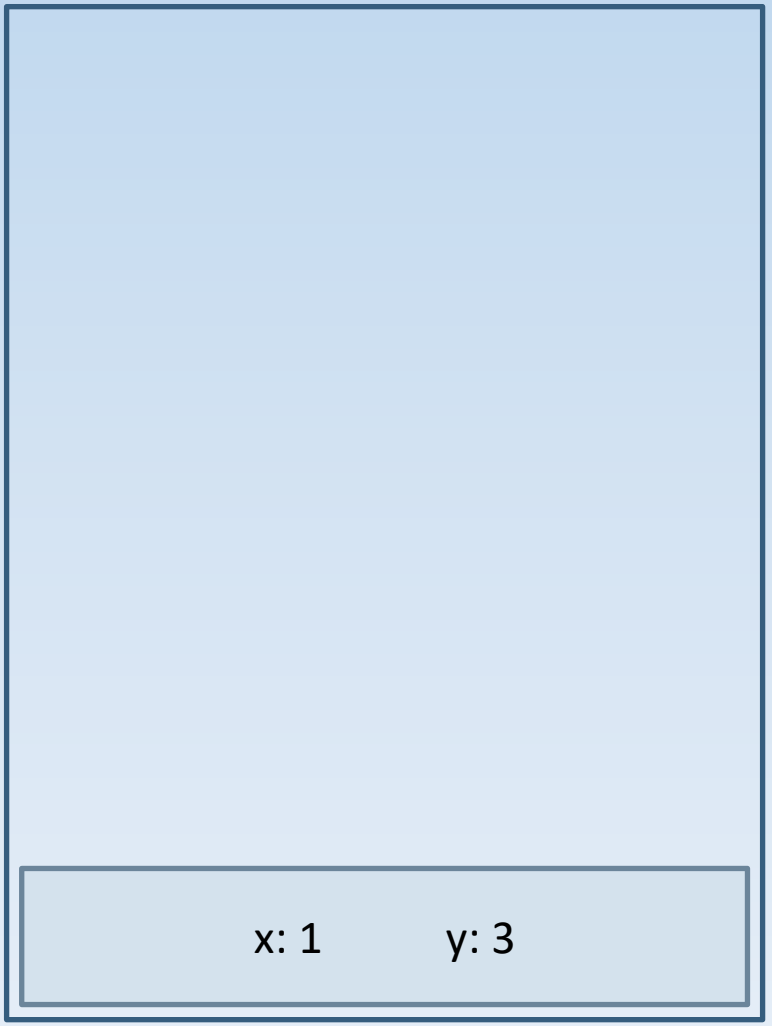
```c
#include <stdio.h>

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(x, y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

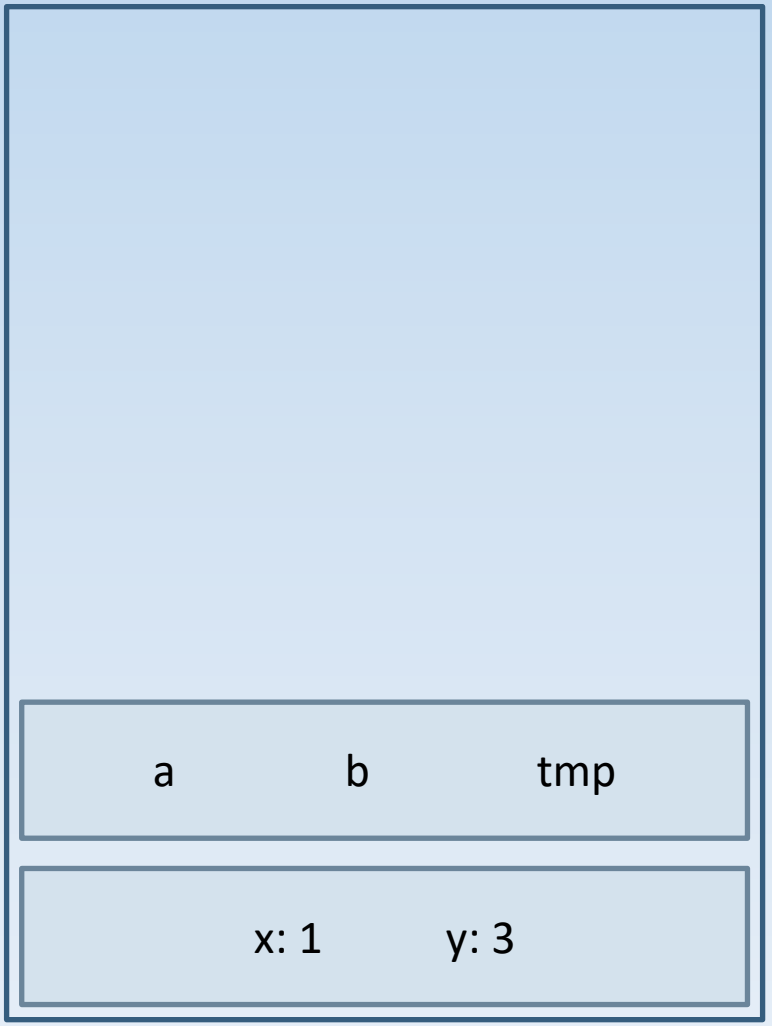
```c
#include <stdio.h>

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(x, y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

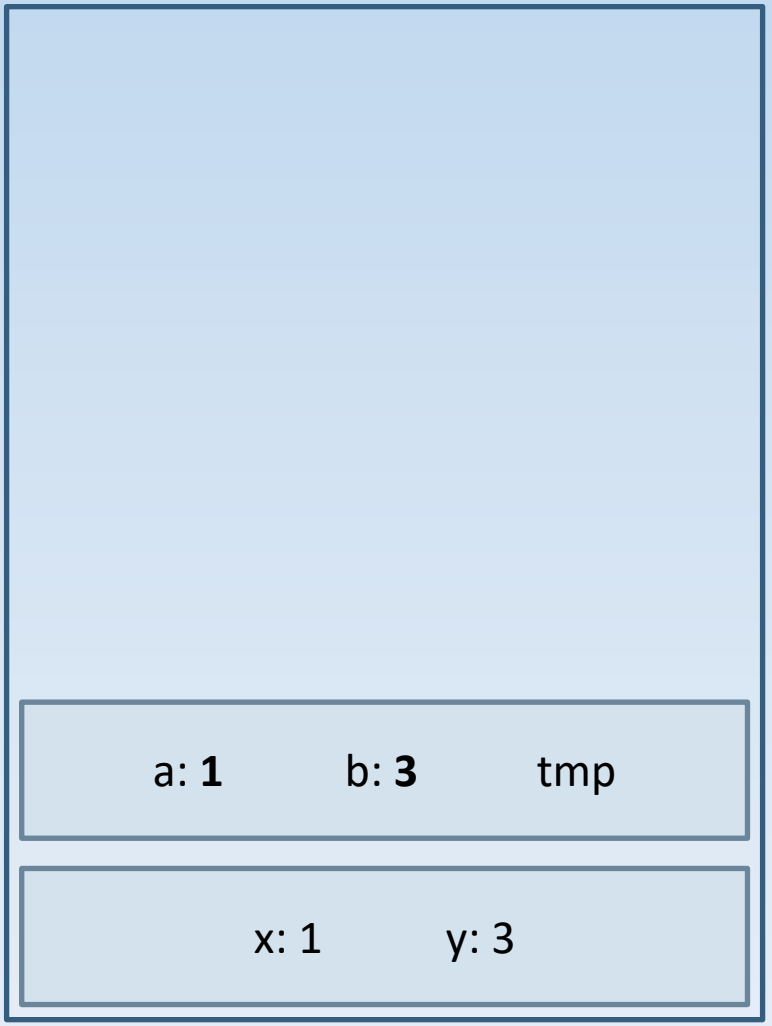
```c
#include <stdio.h>

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(x, y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

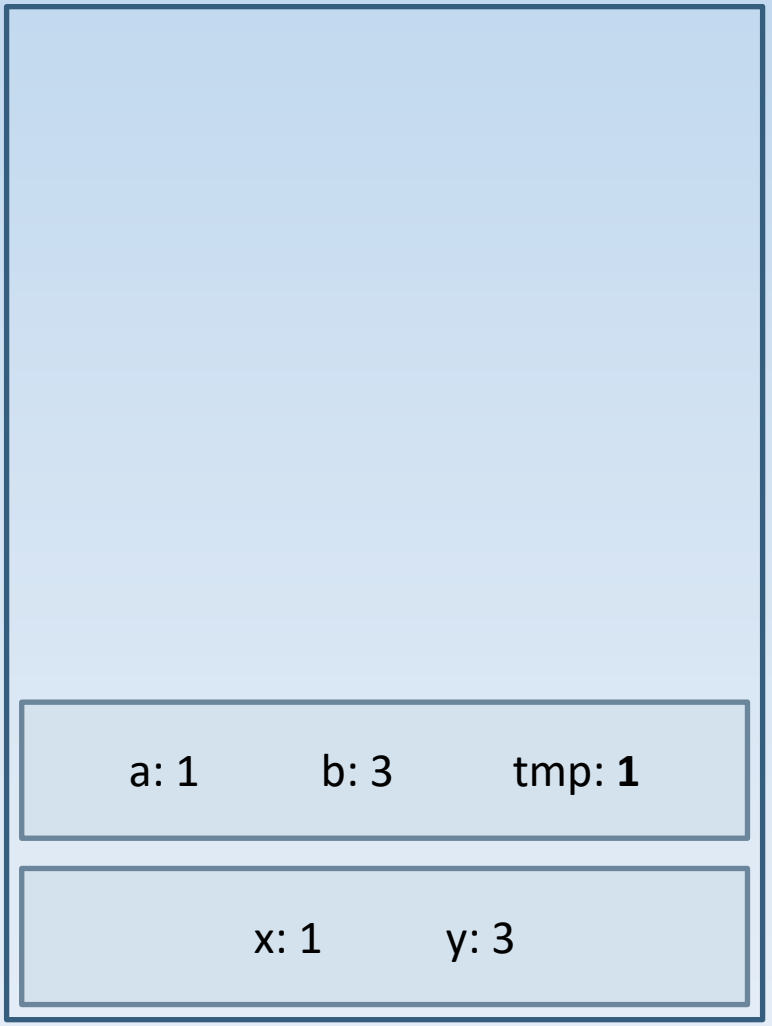
```c
#include <stdio.h>

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(x, y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

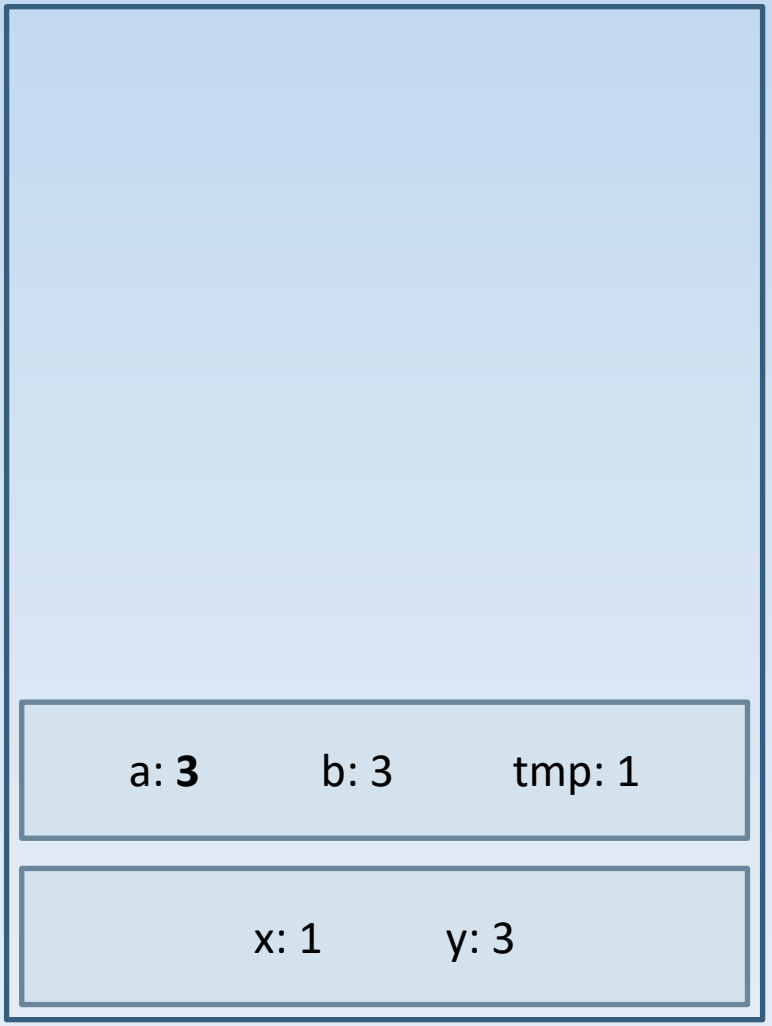
```c
#include <stdio.h>

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(x, y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

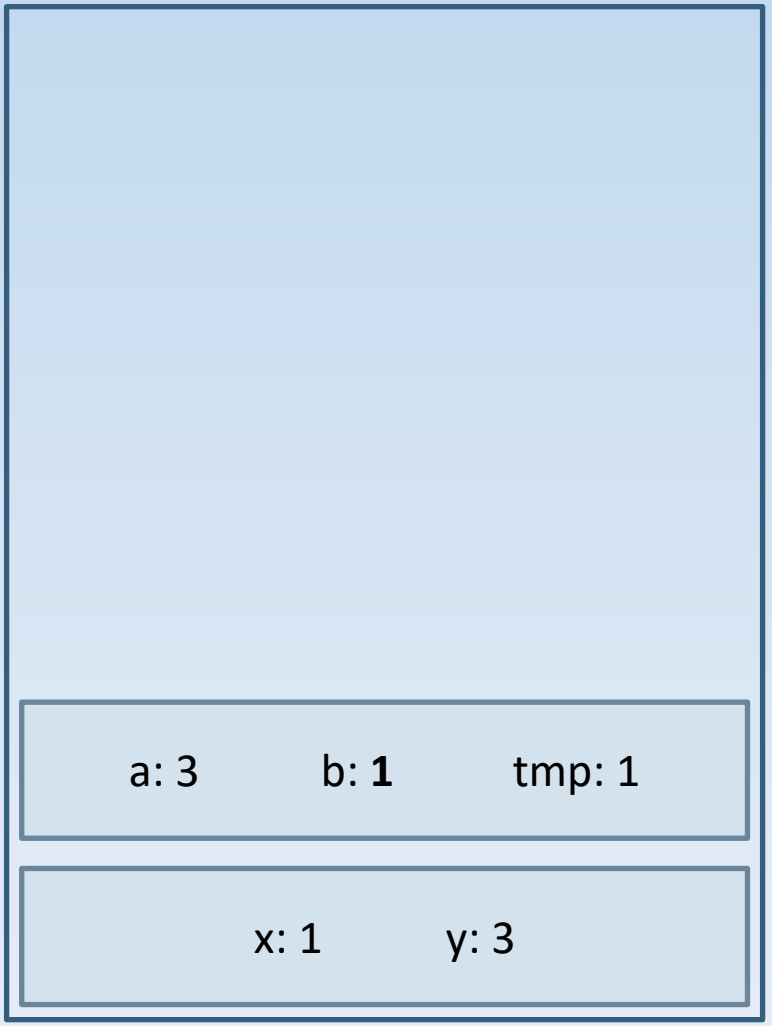
```c
#include <stdio.h>

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(x, y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

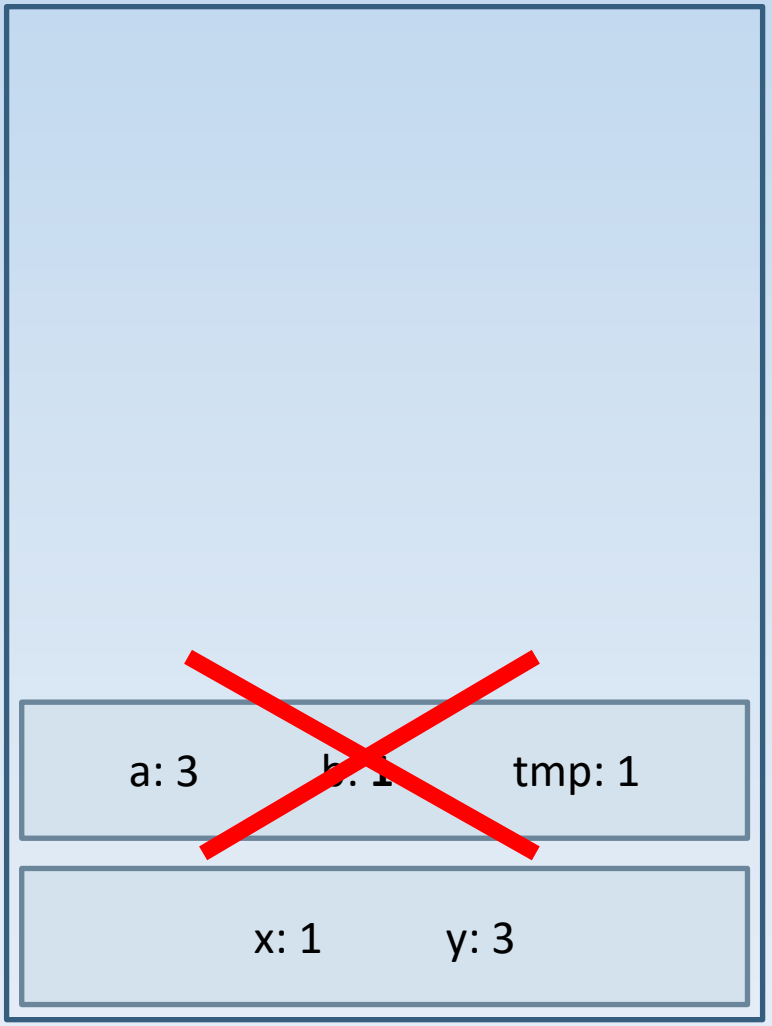
```c
#include <stdio.h>

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(x, y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

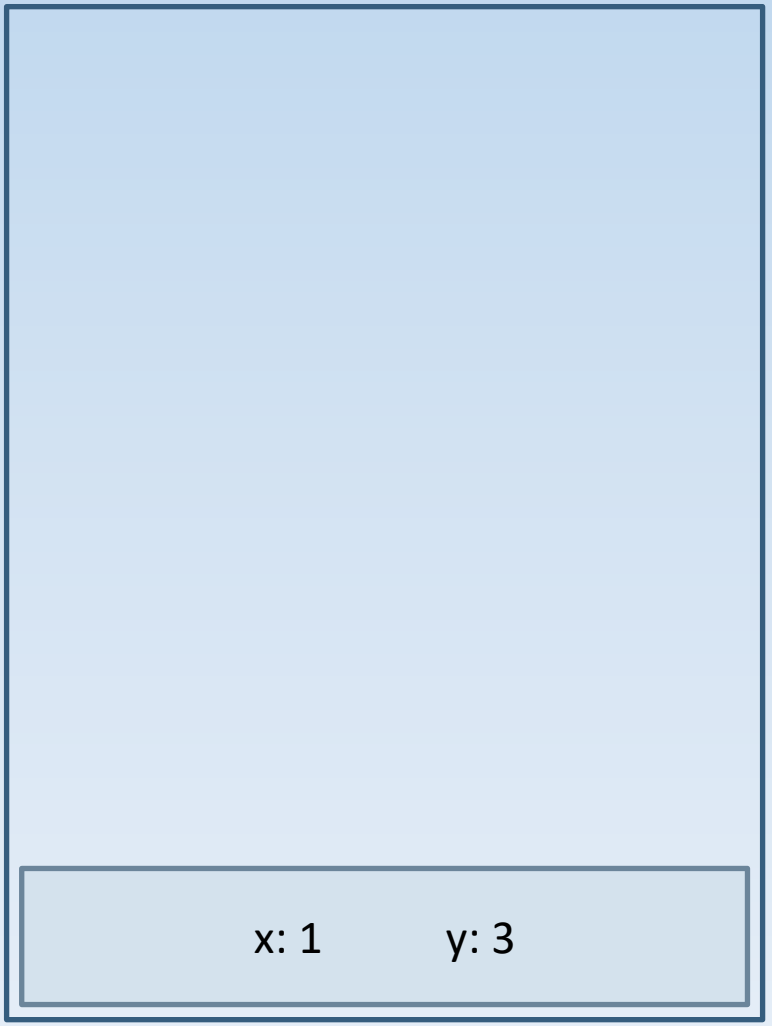
```c
#include <stdio.h>

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(x, y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

```c
#include <stdio.h>

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(x, y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

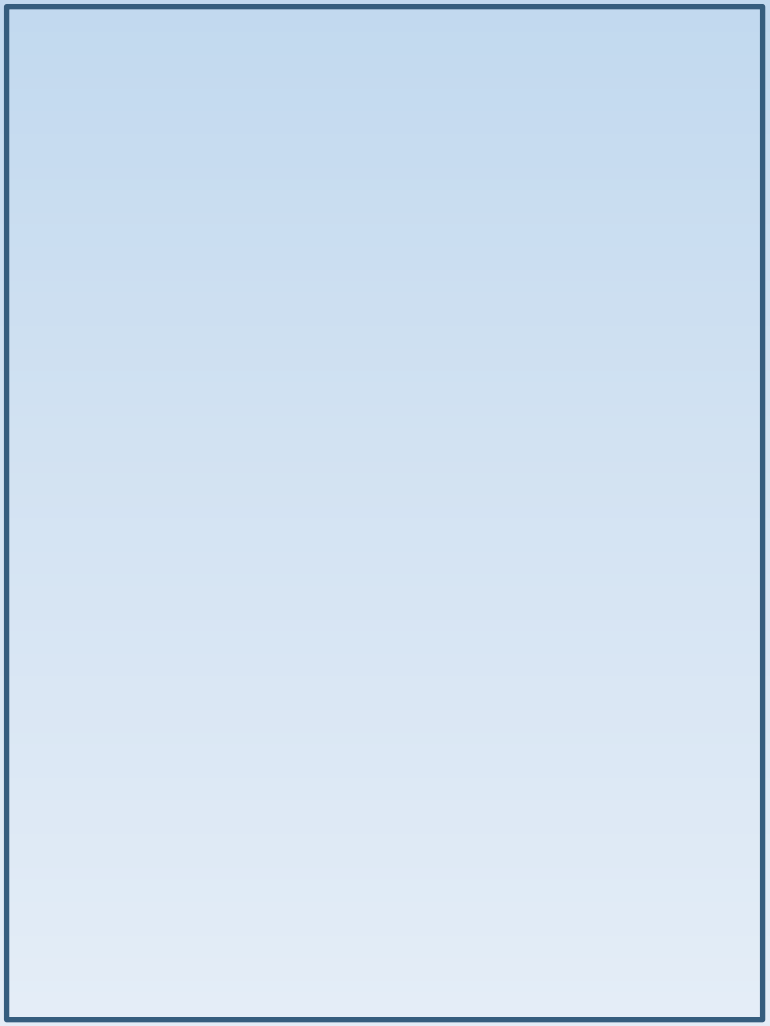
Stack

Az előző módszerrel nem tudtuk felcserélni a main() függvényben lévő két változót.

Nézzük meg, hogy mutatók segítségével ezt hogyan tudjuk megtenni.

Stack

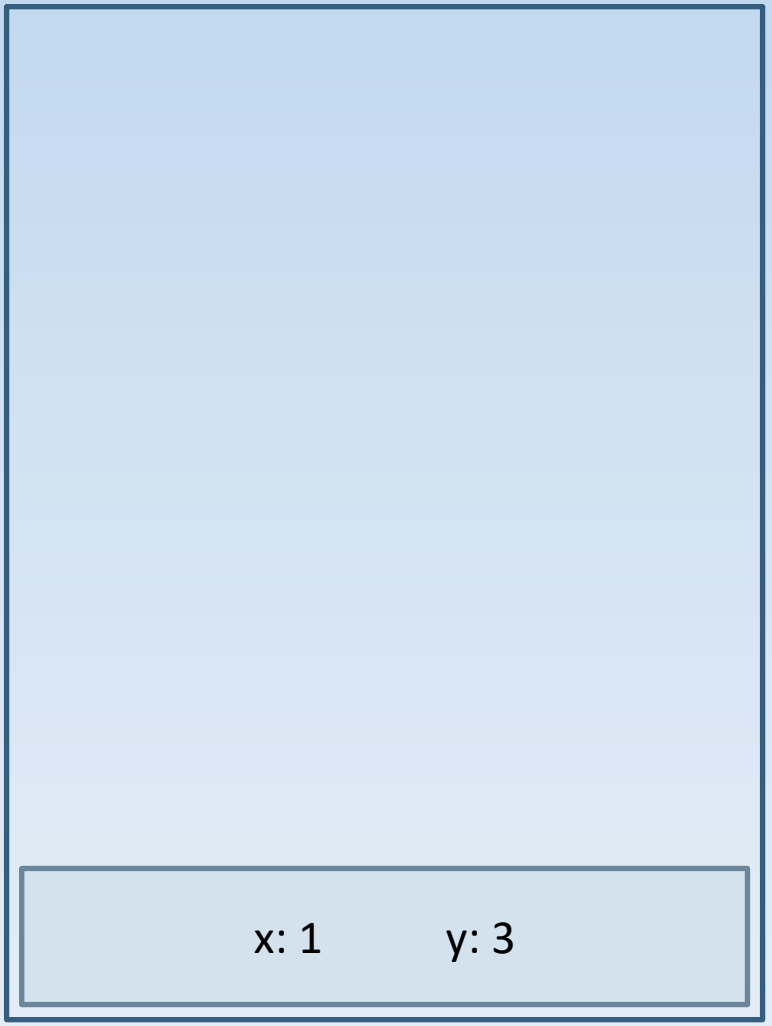
```c
#include <stdio.h>

void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(&x, &y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

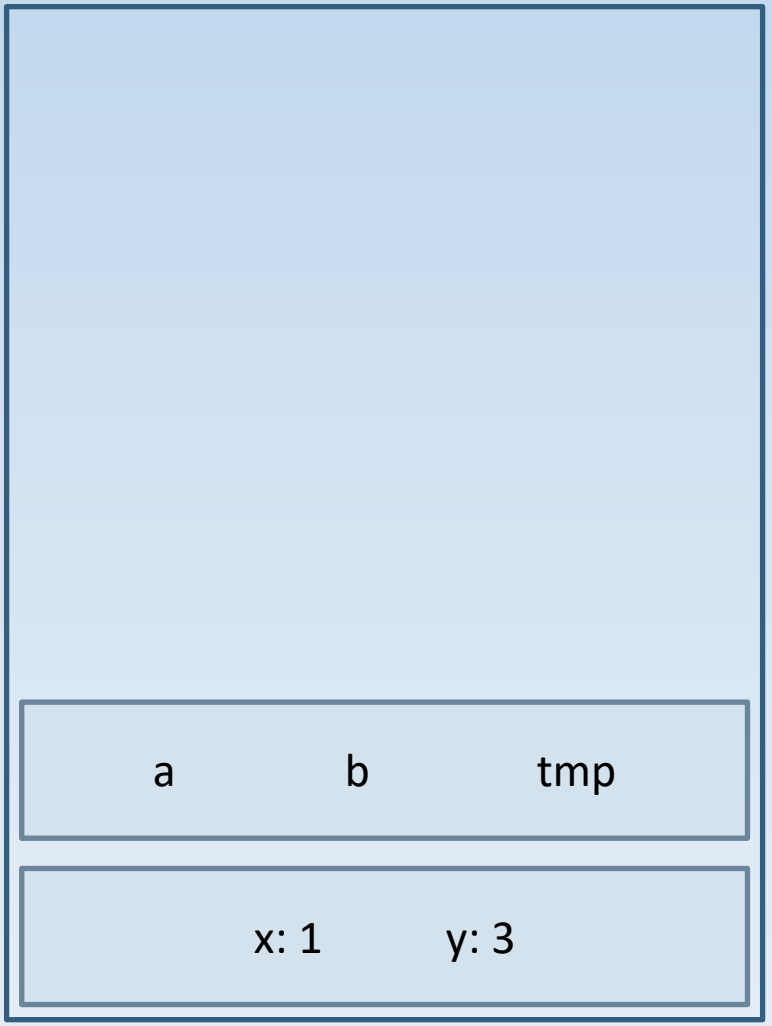
```c
#include <stdio.h>

void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(&x, &y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

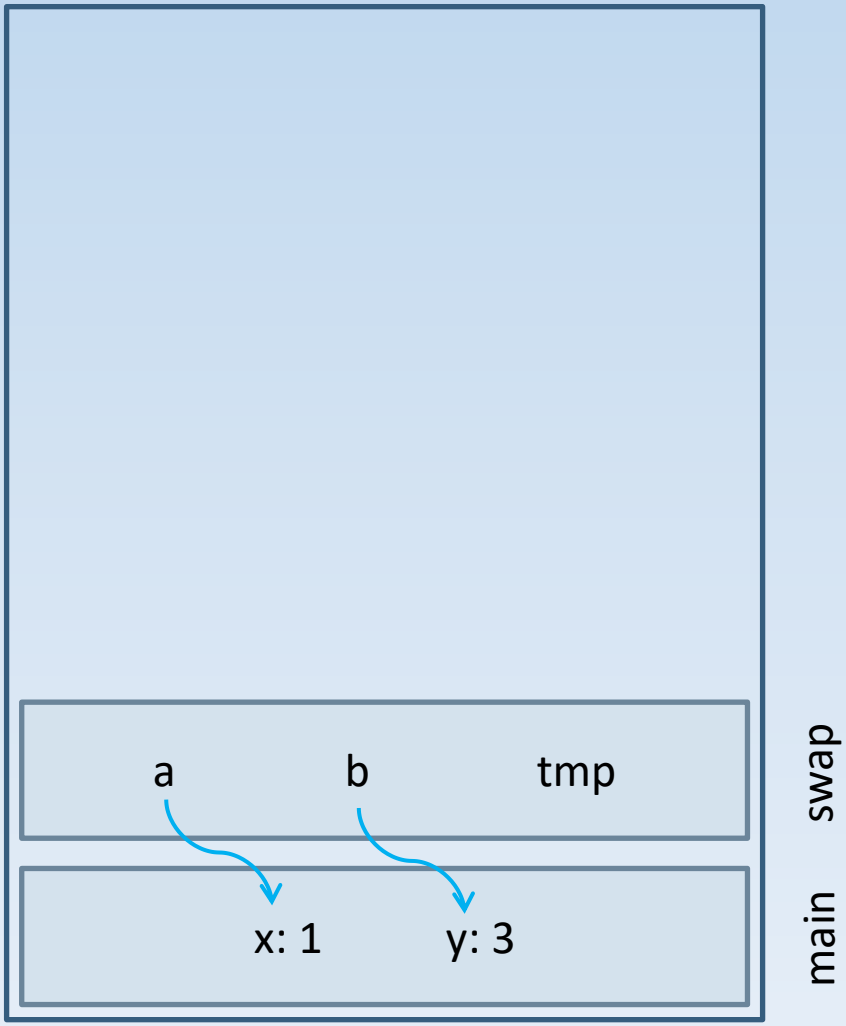
```c
#include <stdio.h>

void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(&x, &y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

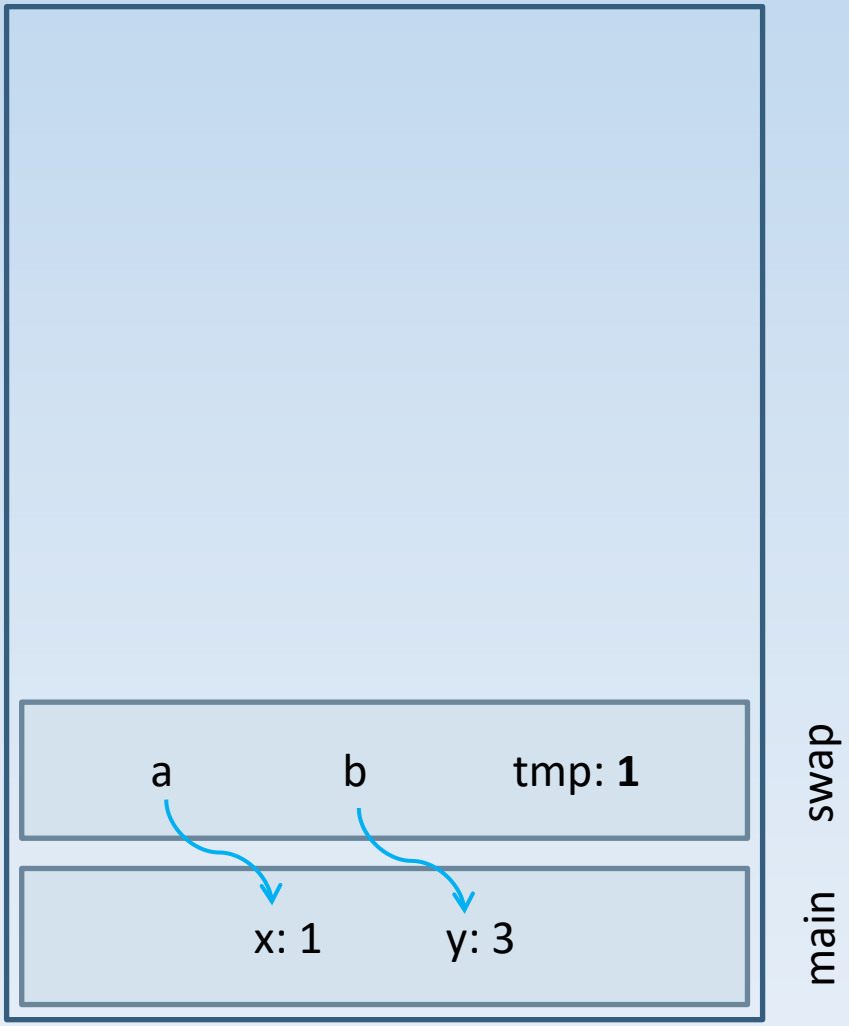
```c
#include <stdio.h>

void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(&x, &y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

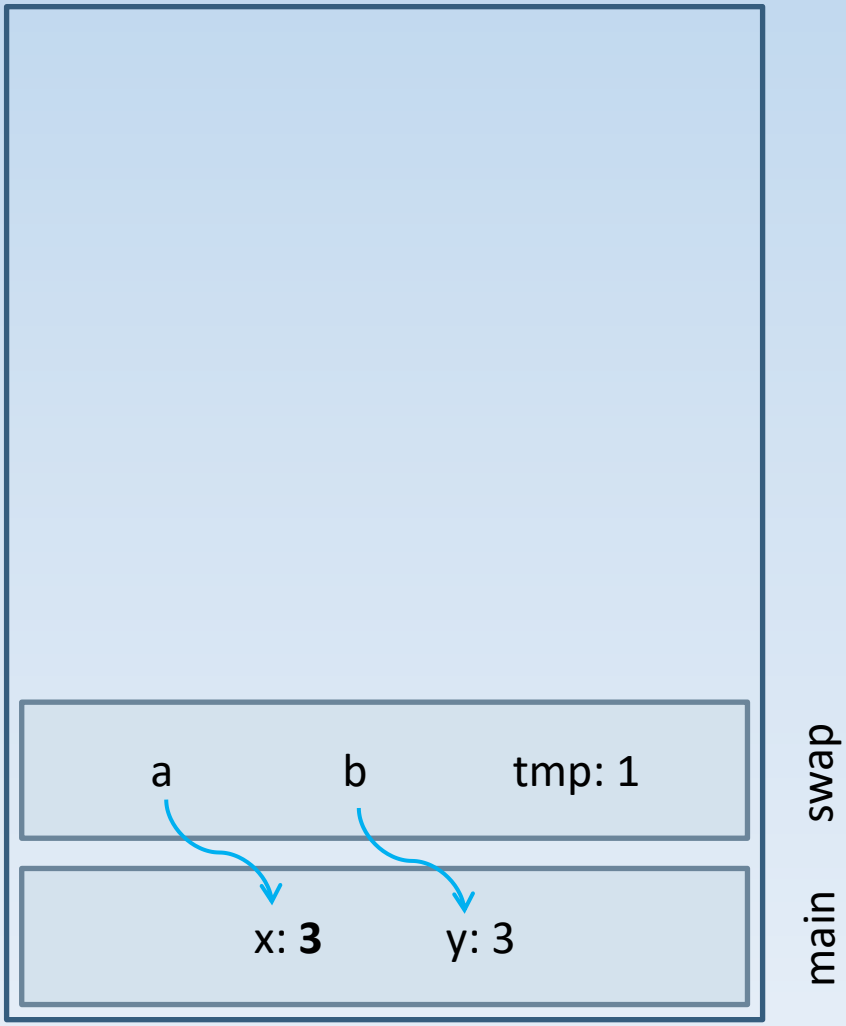
```c
#include <stdio.h>

void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(&x, &y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

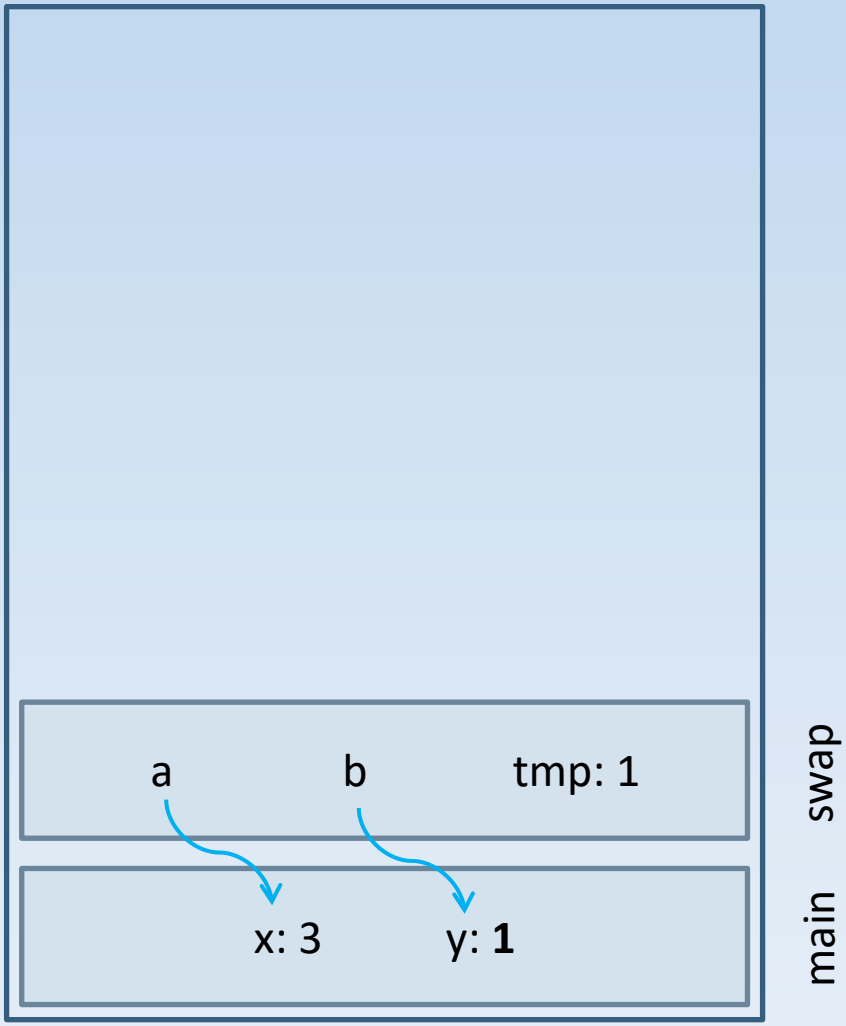
```c
#include <stdio.h>

void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(&x, &y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

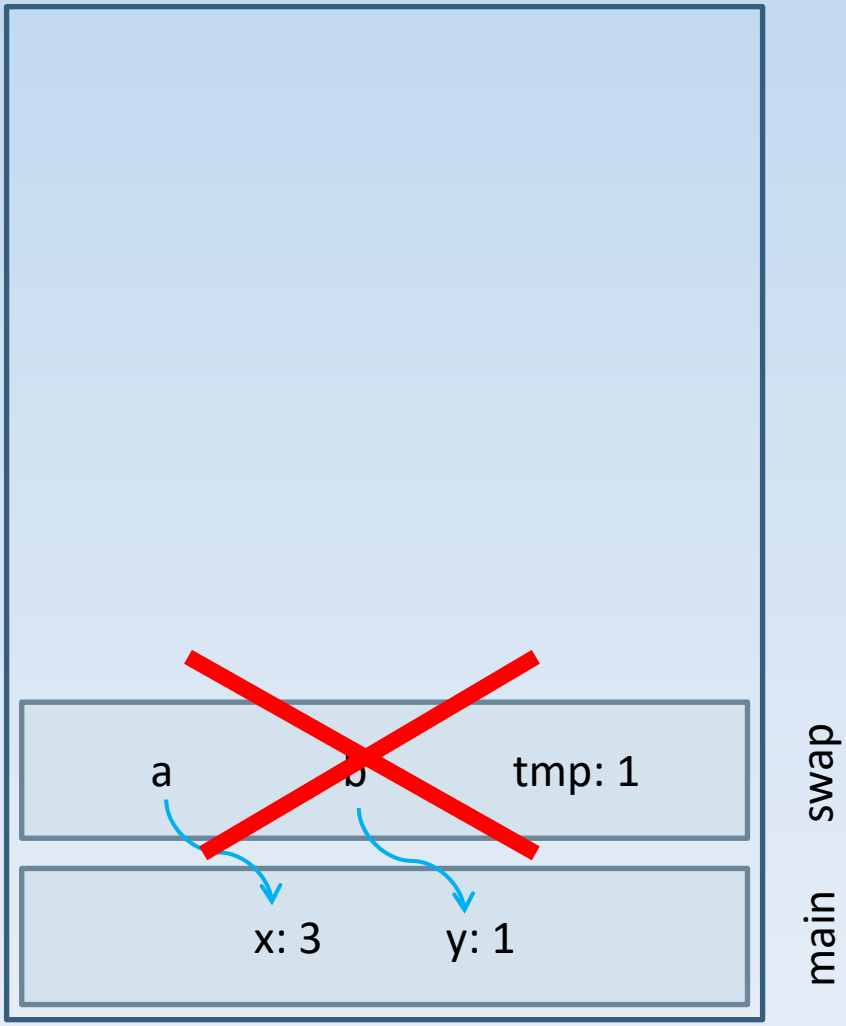
```c
#include <stdio.h>

void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(&x, &y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

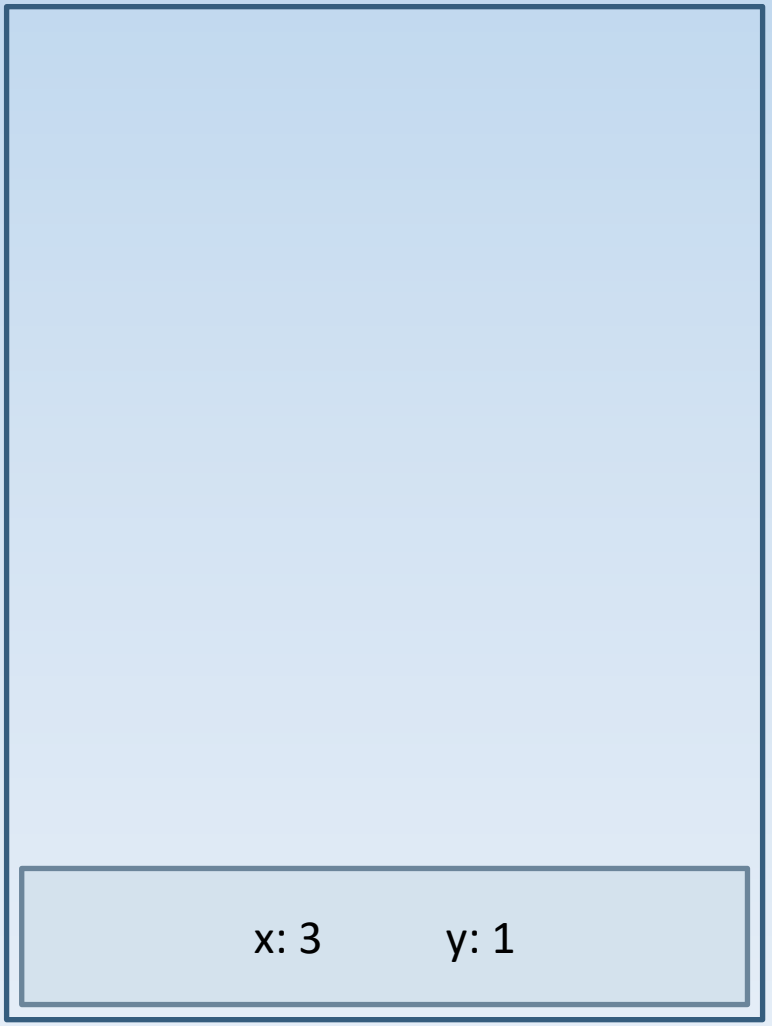
```c
#include <stdio.h>

void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(&x, &y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

```c
#include <stdio.h>

void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int x = 1;
    int y = 3;

    printf("%d, %d\n", x, y);
    swap(&x, &y);
    printf("%d, %d\n", x, y);

    return 0;
}
```



tekintsük csak a **stack**-et

Stack

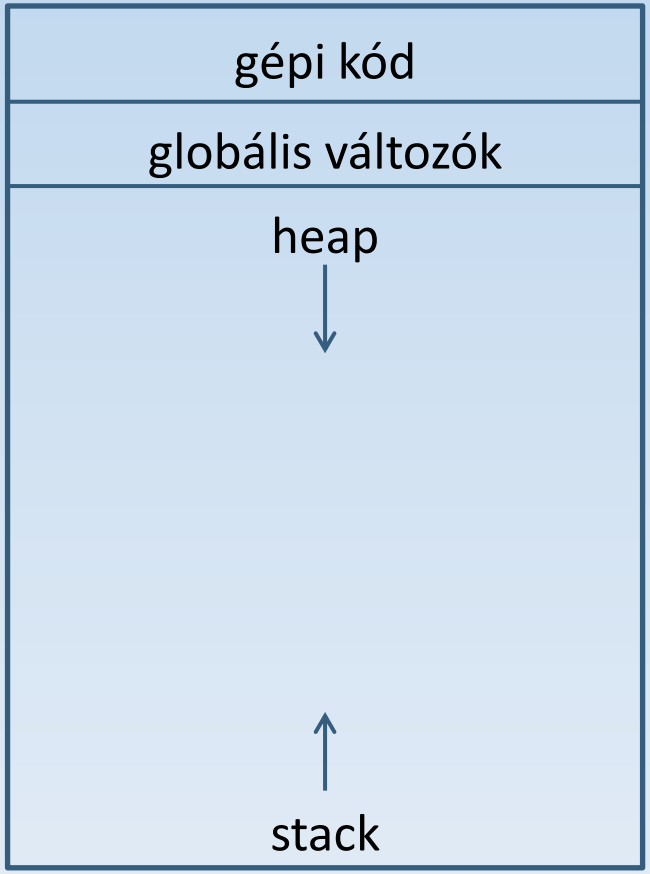
Mekkora a stack mérete?

Nem túl nagy; néhány MB méretű.

```
$ ulimit -s
8192
```

Az adott platformtól függ. Linux operációs rendszeren pl. 8 MB, de van, ahol csak 1 MB a stack mérete.

Ha túllépjük ezt a limitet, akkor verem túlcsordulás (*stack overflow*) történik, aminek a hatására hibaüzenettel leáll a programunk.



Heap

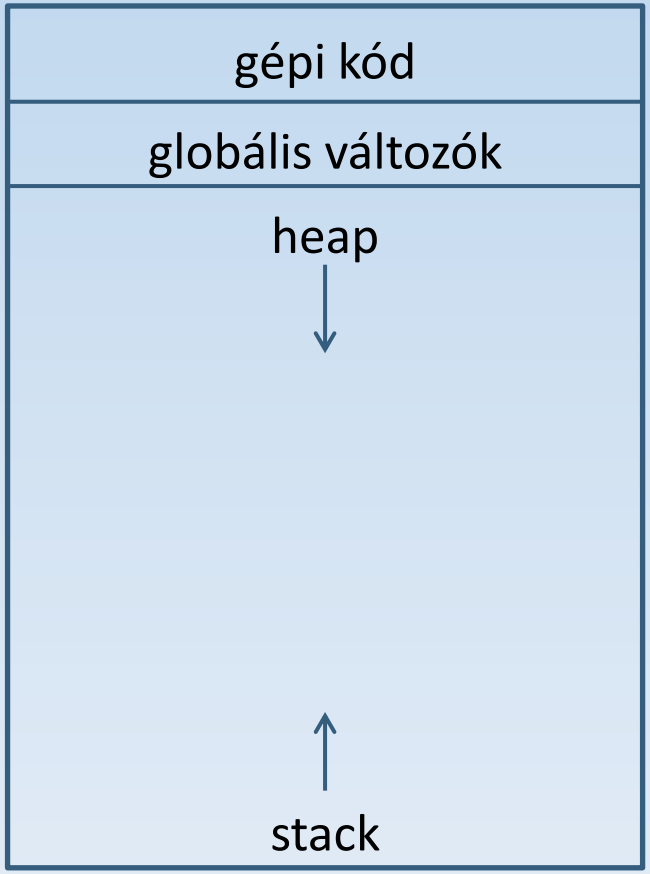
Mekkora a heap mérete?

Nagy. A pontos mérete a memória méretétől és a memória kihasználtságától függ.

```
malloc()
+
free()
```

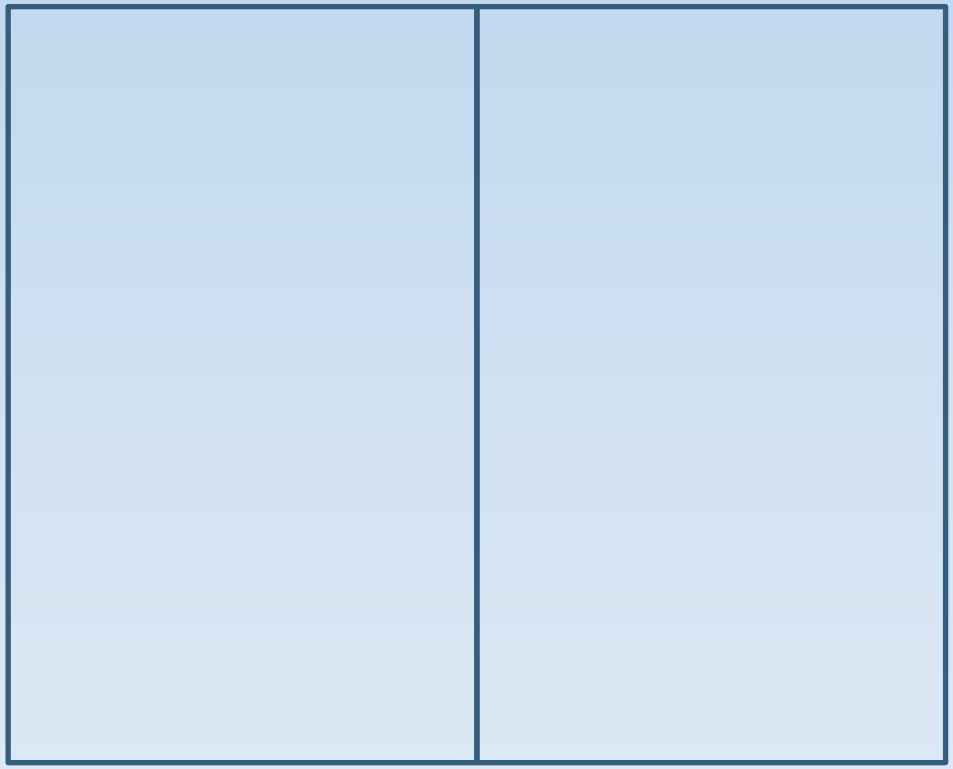
A heap-ben lefoglalt területeket nekünk kell felszabadítani!

Itt is előfordulhat, hogy megtelik a heap, ekkor *heap overflow* hibáról beszélünk.



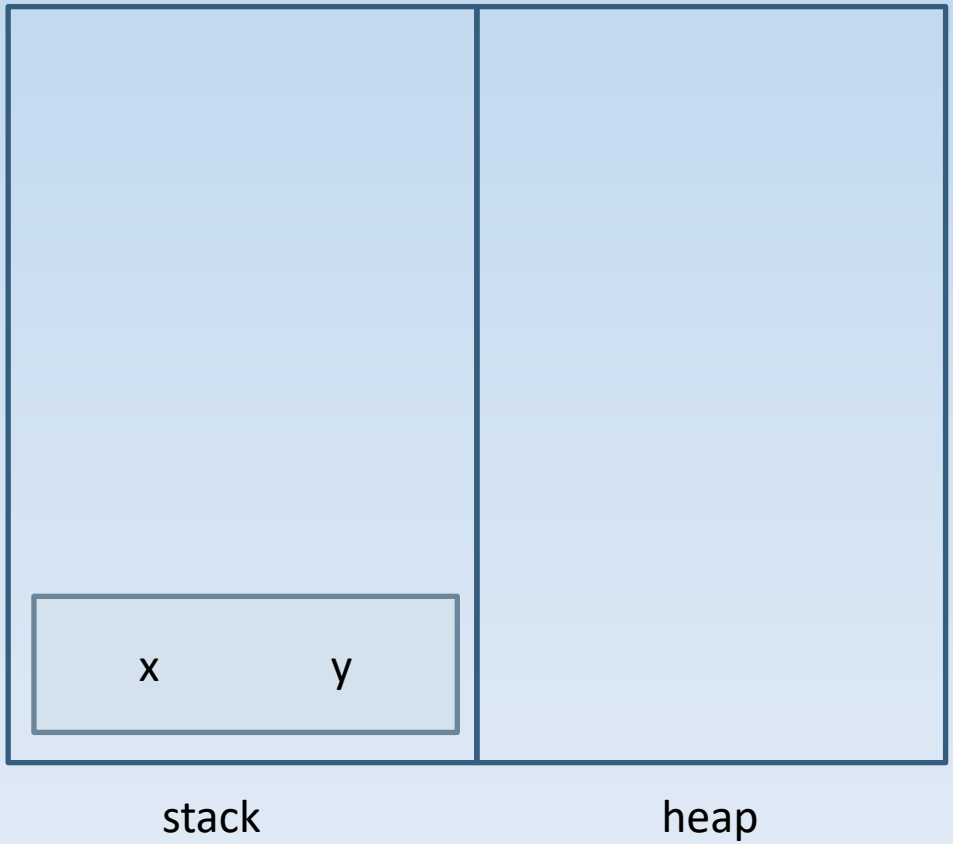
Heap

```c
#include <stdio.h>
#include <stdlib.h>

const long BYTE = 1;
const long KB = 1024 * BYTE;
const long MB = 1024 * KB;

void f0()
{
    char *p = malloc(100 * MB);
}

int main()
{
    int x, y;

    f0();

    return 0;
}
```



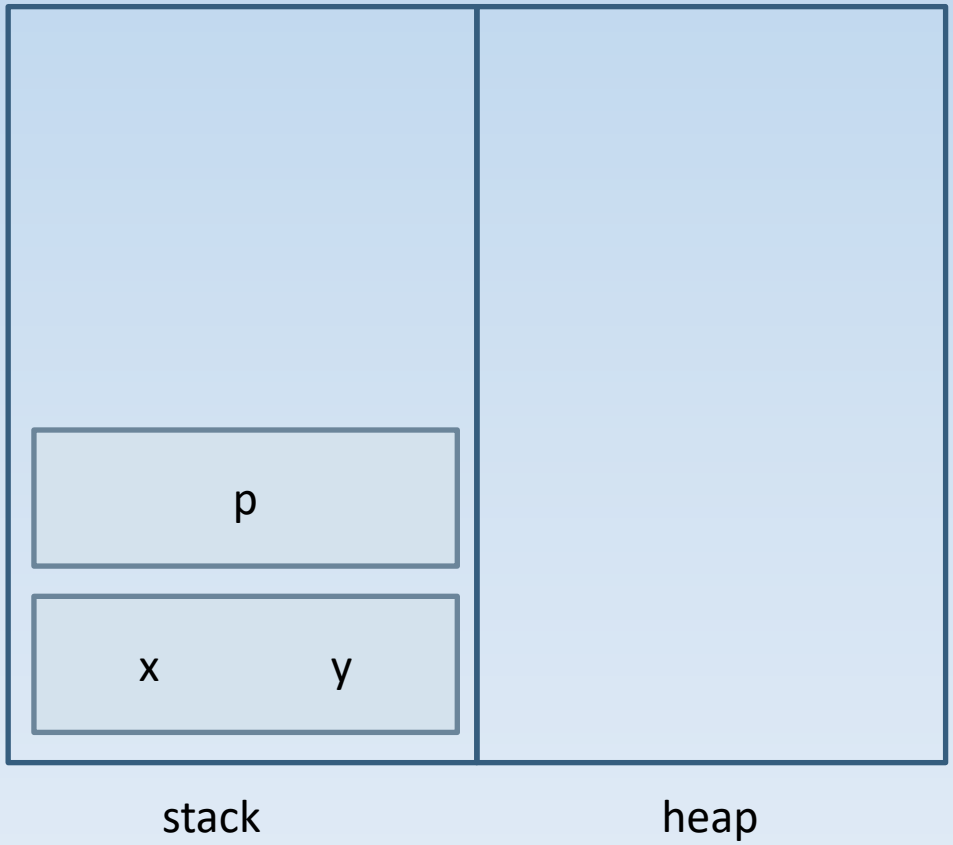
Heap

```c
#include <stdio.h>
#include <stdlib.h>

const long BYTE = 1;
const long KB = 1024 * BYTE;
const long MB = 1024 * KB;

void f0()
{
    char *p = malloc(100 * MB);
}

int main()
{
    int x, y;

    f0();

    return 0;
}
```



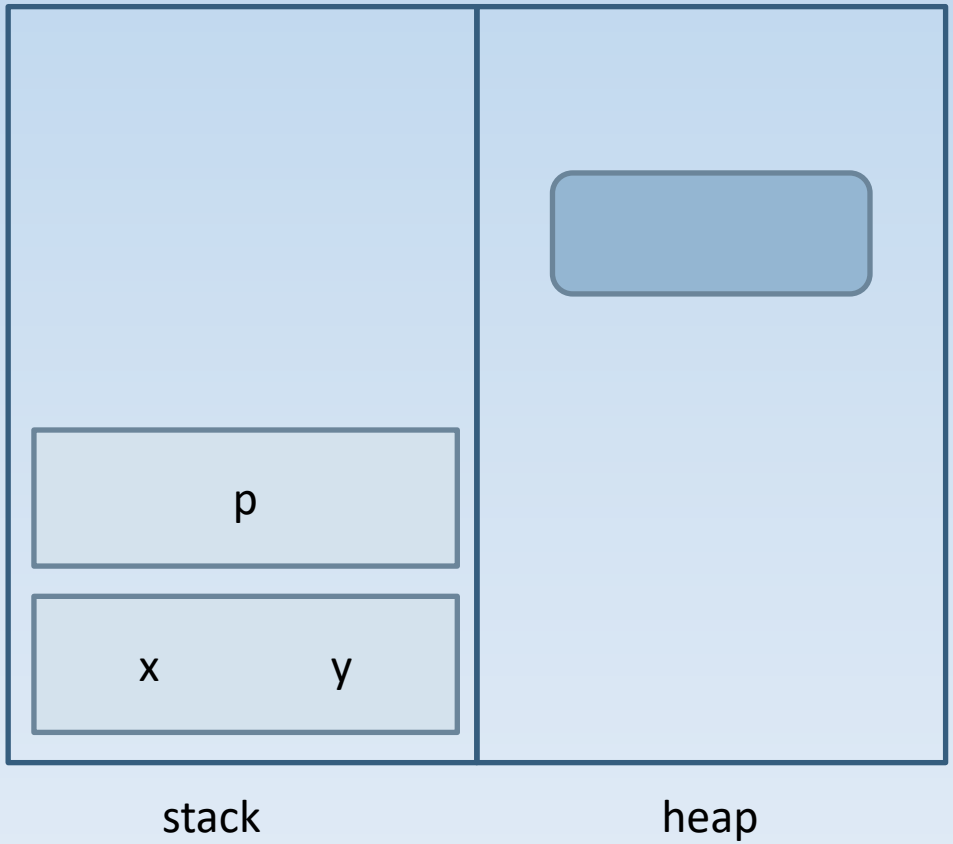
Heap

```c
#include <stdio.h>
#include <stdlib.h>

const long BYTE = 1;
const long KB = 1024 * BYTE;
const long MB = 1024 * KB;

void f0()
{
    char *p = malloc(100 * MB);
}

int main()
{
    int x, y;

    f0();

    return 0;
}
```



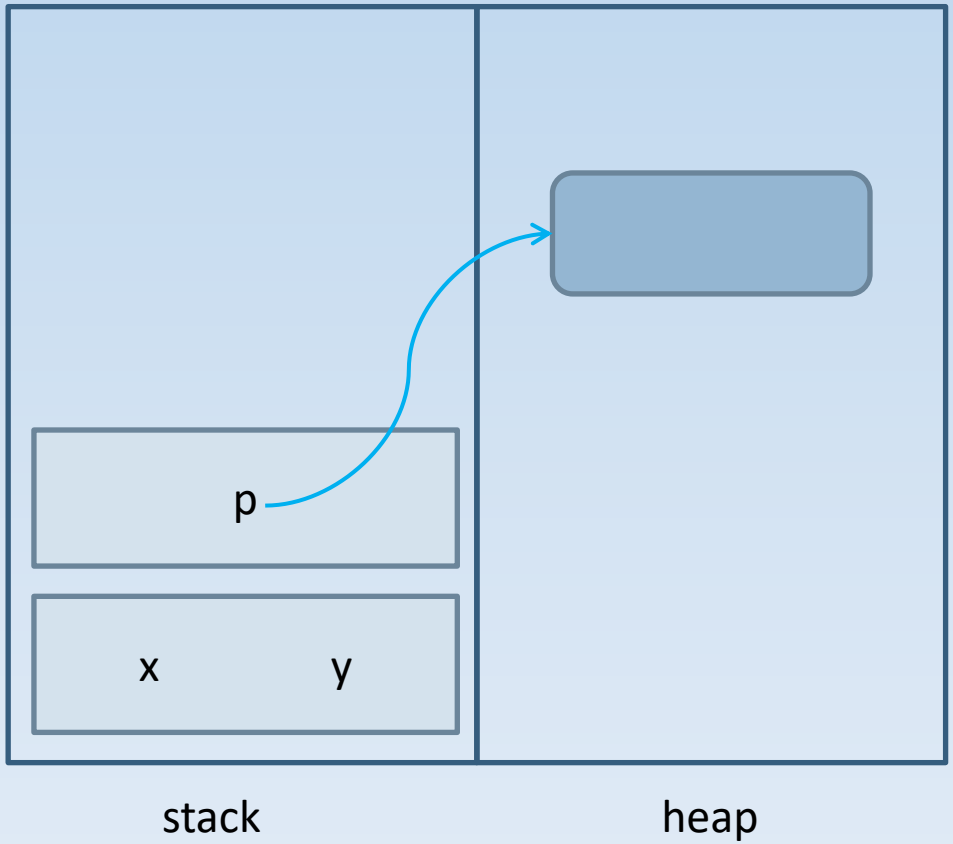
Heap

```c
#include <stdio.h>
#include <stdlib.h>

const long BYTE = 1;
const long KB = 1024 * BYTE;
const long MB = 1024 * KB;

void f0()
{
    char *p = malloc(100 * MB);
}

int main()
{
    int x, y;

    f0();

    return 0;
}
```



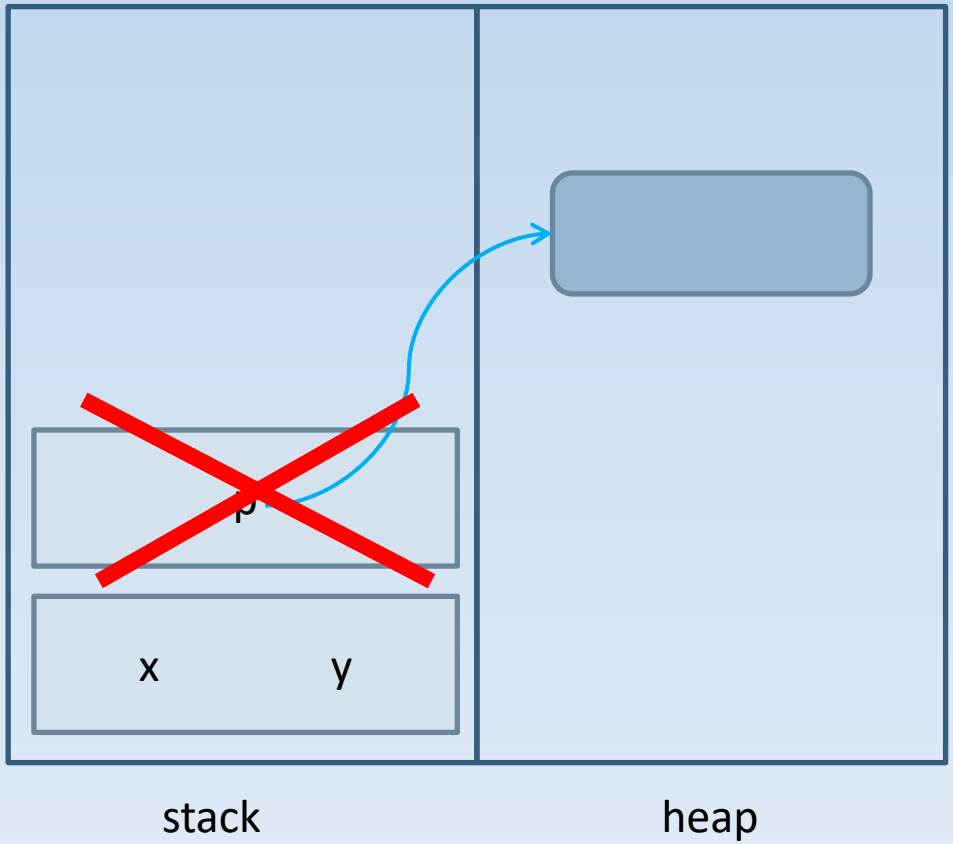
Heap

```c
#include <stdio.h>
#include <stdlib.h>

const long BYTE = 1;
const long KB = 1024 * BYTE;
const long MB = 1024 * KB;

void f0()
{
    char *p = malloc(100 * MB);
}

int main()
{
    int x, y;

    f0();

    return 0;
}
```



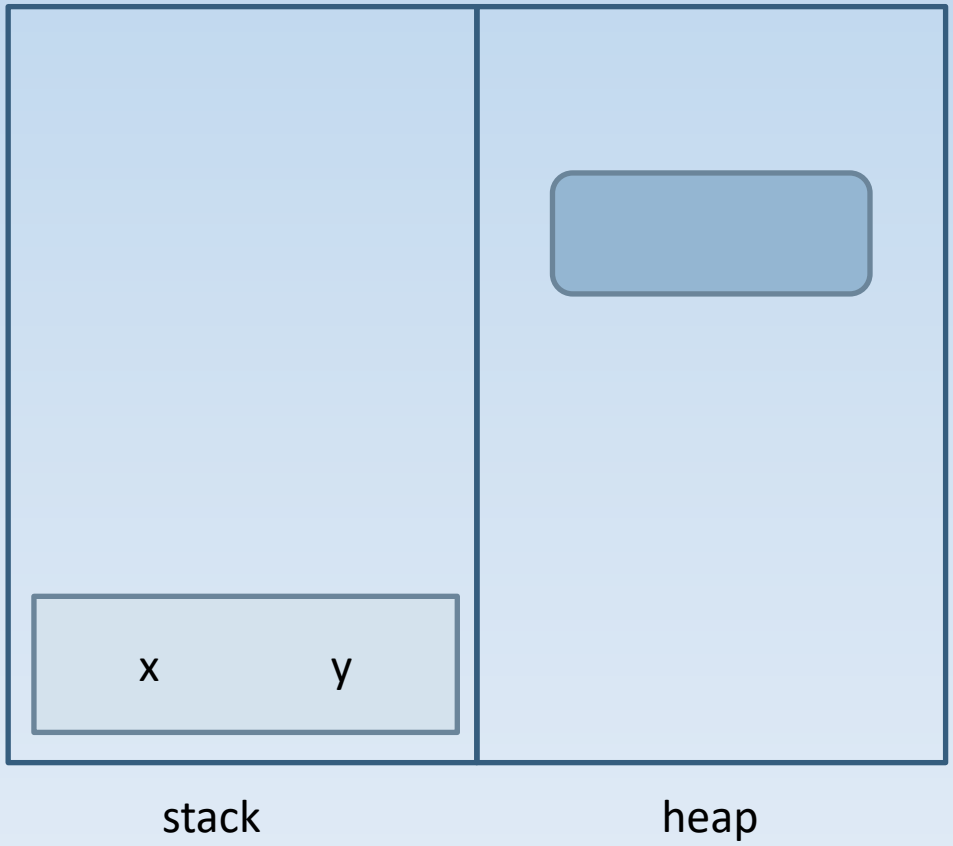
Heap

```c
#include <stdio.h>
#include <stdlib.h>

const long BYTE = 1;
const long KB = 1024 * BYTE;
const long MB = 1024 * KB;

void f0()
{
    char *p = malloc(100 * MB);
}

int main()
{
    int x, y;

    f0();

    return 0;
}
```



Heap

```c
#include <stdio.h>
#include <stdlib.h>

const long BYTE = 1;
const long KB = 1024 * BYTE;
const long MB = 1024 * KB;

void f0()
{
    char *p = malloc(100 * MB);
}

int main()
{
    int x, y;

    f0();

    return 0;
}
```



Eredmény? 100 MB beragadt memória ☹ (memóriaszivárgás)
Mi lenne a megoldás? Pl. az `f0()` -ban a `malloc()` után kellene egy `free(p);` hívás.

Struktúrák és mutatók

```
5  typedef struct {
6      int x;
7      int y;
8      char leiras[500];
9  } Pont;
10
11 void kiir(const Pont *p)
12 {
13     printf("kiir: P(%d, %d)\n", p->x, p->y);
14 }
15
16 int main()
17 {
18     Pont a;
19     a.x = 1;
20     a.y = 2;
21
22     kiir(&a);
23
24     return 0;
25 }
```

Nagy méretű struktúra átadása olcsón és biztonságosan.
Olcsón: csak egy mutatót adunk át.
Biztonságosan: a hívott fél nem tudja módosítani a struktúrát (lásd const kulcsszó).
Ha nem mutatóval adjuk át, akkor az egész struktúráról másolat készülne az érték szerinti paraméterátadás miatt.

Struktúrák és mutatók

```c
typedef struct {
    int x;
    int y;
} Pont;

Pont * origo()
{
    Pont *p = malloc(sizeof(Pont));
    p->x = 0;
    p->y = 0;

    return p;
}

int main()
{
    Pont *kozeppont = origo();

    printf("P(%d, %d)\n", kozeppont->x, kozeppont->y);

    free(kozeppont);

    return 0;
}
```

Objektum (struktúra) létrehozása dinamikus tárfoglalással, majd az objektum címének visszaadása.

A hívó fél megkapja az objektum címét. Feldolgozás után töröljük az objektumot (az általa lefoglalt tárterületet explicit módon felszabadítjuk).

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).

- puffertúlcsordulás (buffer overflow), link