

# Programozási nyelvek 1

Szathmáry László  
Debreceni Egyetem  
Informatikai Kar

## 9. előadás

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

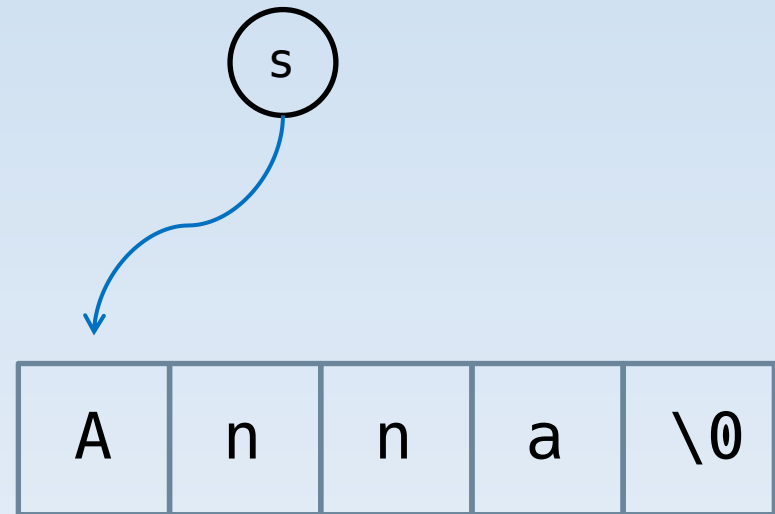
(utolsó módosítás: 2024. jan. 31.)

2023-2024, 2. félév



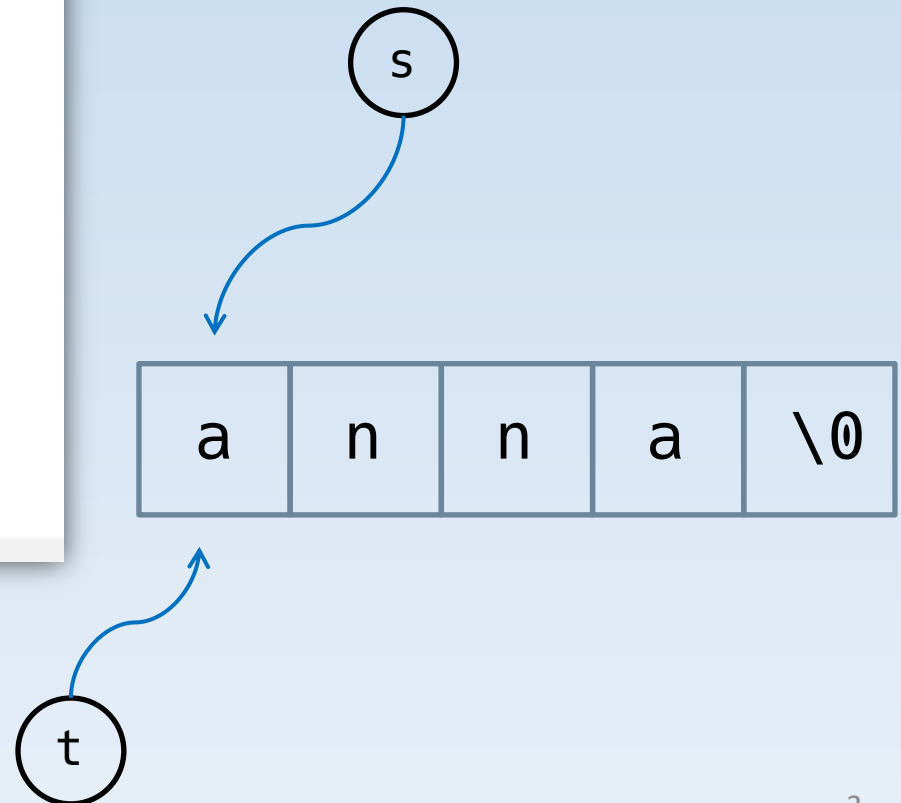
# Mutatók és sztringek (folyt.)

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main()
5  {
6      char* s = "Anna";
7
8      printf("%s\n", s);
9
10     return 0;
11 }
12
```



# Mutatók és sztringek

```
1  #include "prog1.h"
2  #include <stdio.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  int main()
7  {
8      char *s = get_string("Név: ");
9
10     char *t = s;
11
12     t[0] = toupper(t[0]);
13
14     printf("%s\n", s);
15     printf("%s\n", t);
16
17     return 0;
18 }
19
```



# Dinamikus memórfoglalás

```
12
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 memórfoglalás.  
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!

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     char *p = malloc(20 * sizeof(char));
7
8     return 0;
9 }
```

```
[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 változói**. A lokális változók az alprogramon kívülről nem láthatók, azokat az alprogram elrejt a külvilág elől.

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

Egy alprogram **környezete** alatt a globális változók együttesét értjük.

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

```
1 #include <stdio.h>
2
3 #define MAX 10
4
5 const double PI = 3.14159;
6
7 int counter = 0;
8
9 void f2()
10 {
11     counter = 42;
12 }
13
14 void f1()
15 {
16     int counter = 5;
17     printf("f1: %d\n", counter);
18 }
19
20 void f0()
21 {
22     printf("f0: %d\n", counter);
23 }
24
25 int main()
26 {
27     printf("MAX értéke: %d\n", MAX);
28     printf("PI értéke: %lf\n", PI);
29     f0();
30     f1();
31     f2();
32     printf("main: %d\n", counter);
33
34     return 0;
35 }
36
```

Kérdés:

Mi a program kimenete?

```
1 #include <stdio.h>
2
3 #define MAX 10
4
5 const double PI = 3.14159;
6
7 int counter = 0;
8
9 void f2()
10 {
11     counter = 42;
12 }
13
14 void f1()
15 {
16     int counter = 5;
17     printf("f1: %d\n", counter);
18 }
19
20 void f0()
21 {
22     printf("f0: %d\n", counter);
23 }
24
25 int main()
26 {
27     printf("MAX értéke: %d\n", MAX);
28     printf("PI értéke: %lf\n", PI);
29     f0();
30     f1();
31     f2();
32     printf("main: %d\n", counter);
33
34     return 0;
35 }
36
```

nevesített konstans

nevesített konstansként  
használt globális változó

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

a globális counter értékét módosítjuk

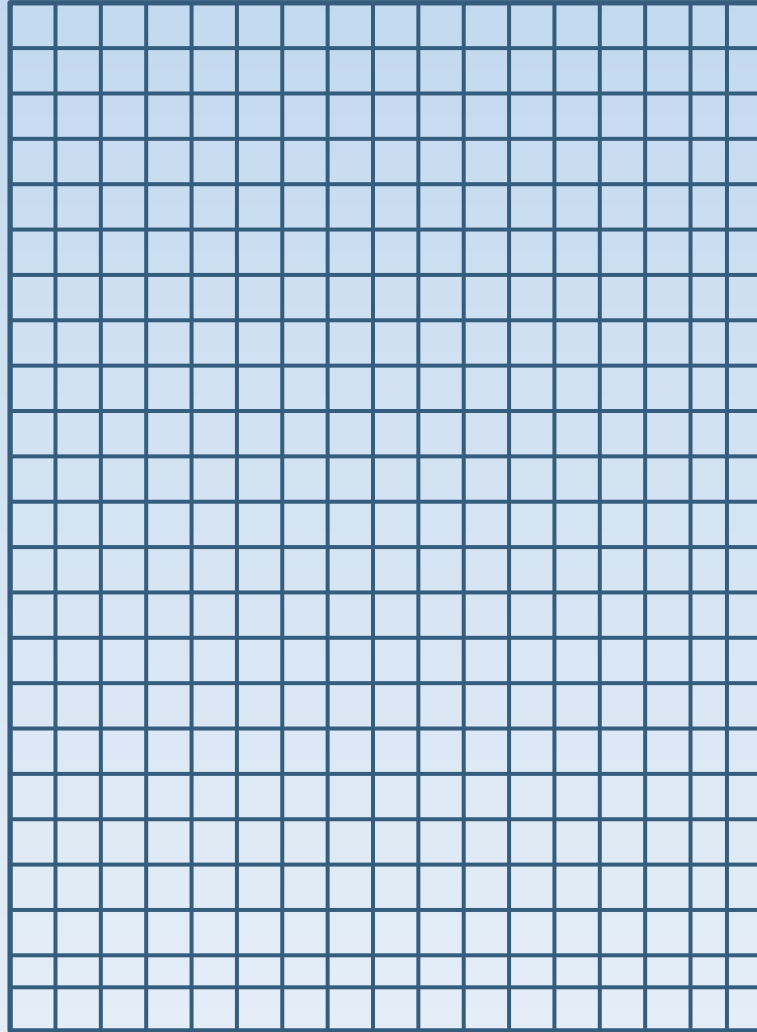
counter itt egy lokális változó  
(deklaráltuk); a globális counter  
változót elrejtí

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



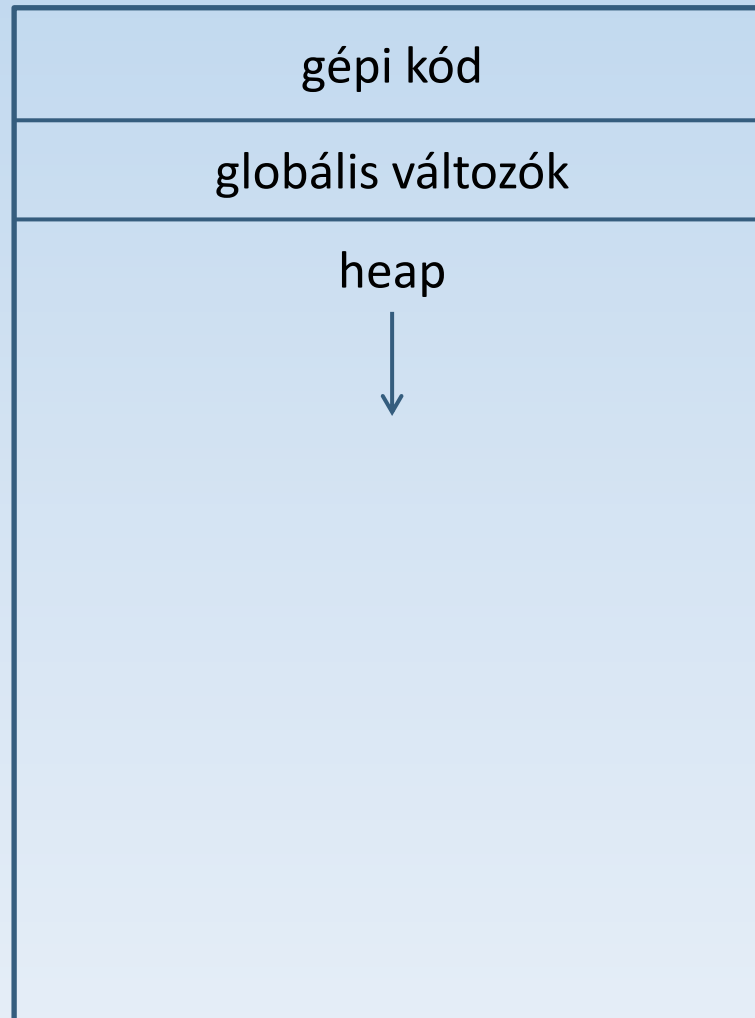
# A memória (RAM) kiosztása

gépi kód
globális változók

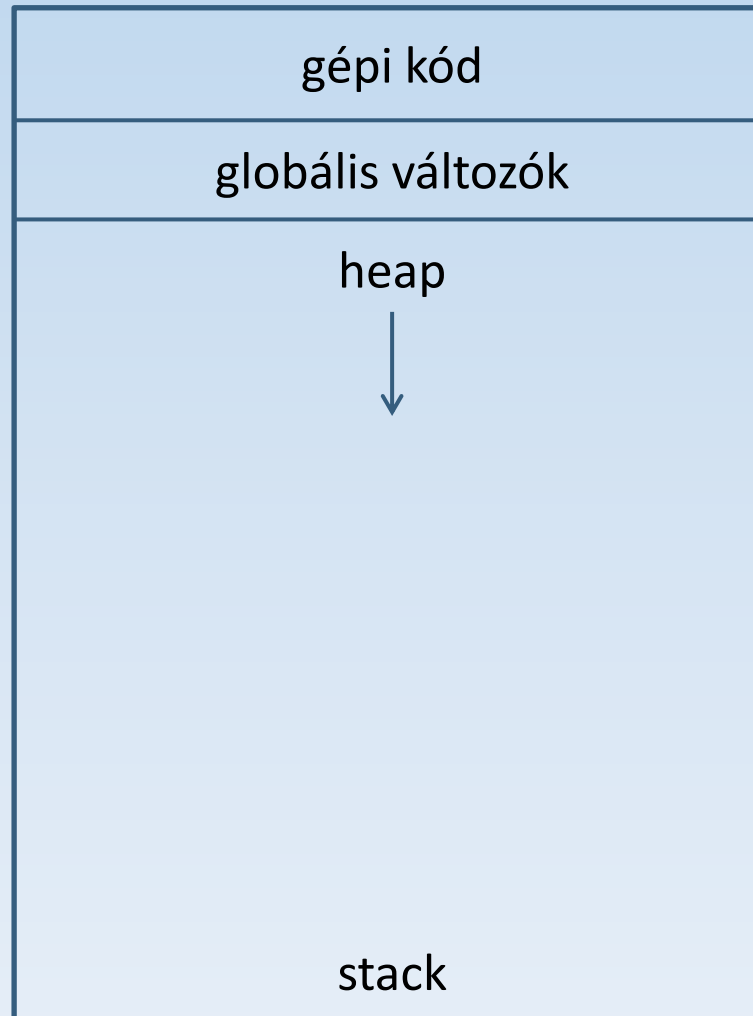
# A memória (RAM) kiosztása



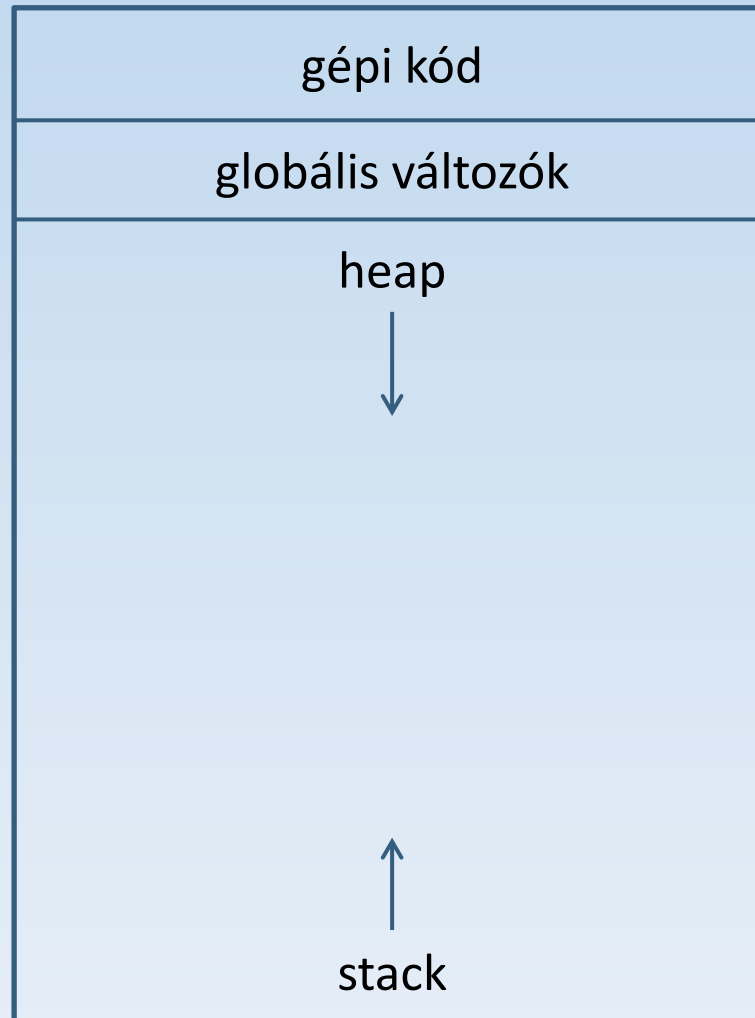
# A memória (RAM) kiosztása



# A memória (RAM) kiosztása



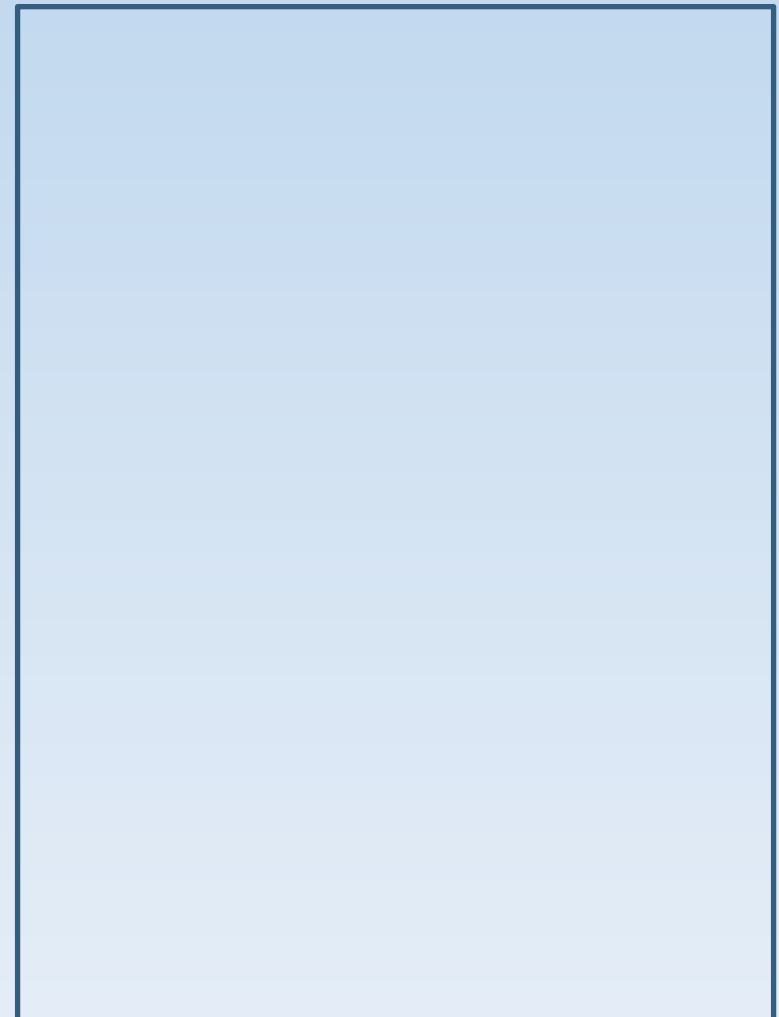
# A memória (RAM) kiosztása





# Stack

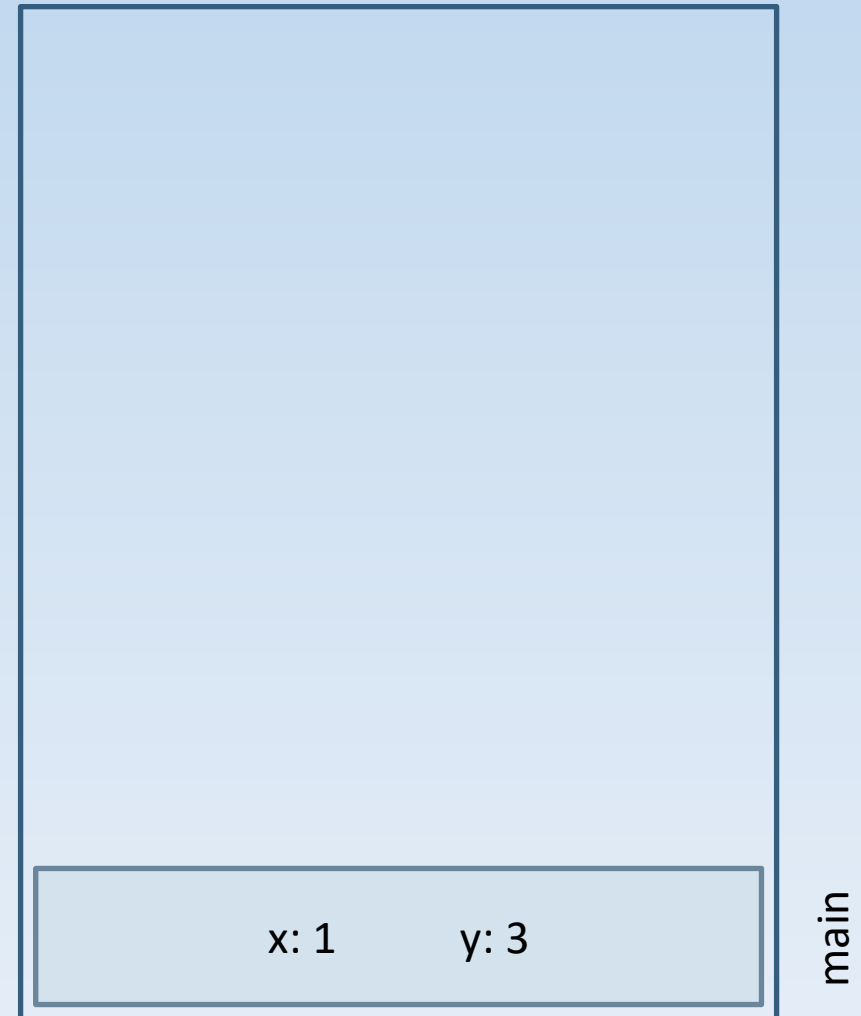
```
1  #include <stdio.h>
2
3  void swap(int a, int b)
4  {
5      int tmp = a;
6      a = b;
7      b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(x, y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

# Stack

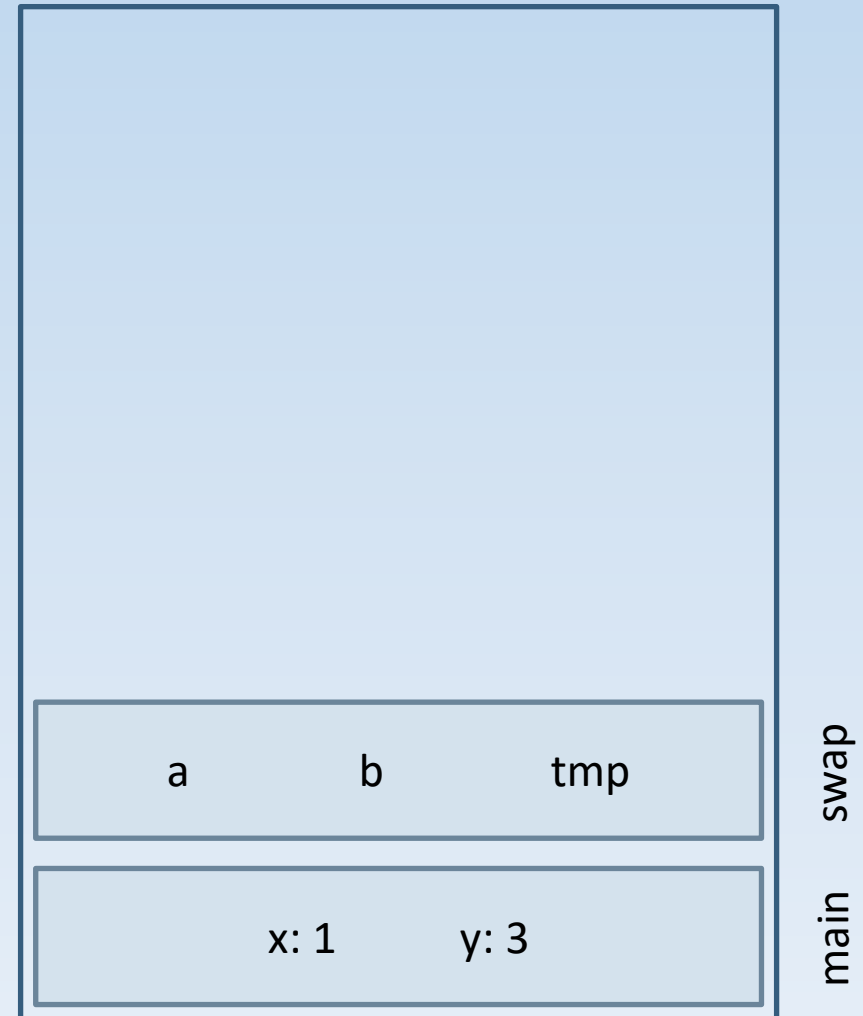
```
1  #include <stdio.h>
2
3  void swap(int a, int b)
4  {
5      int tmp = a;
6      a = b;
7      b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(x, y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

# Stack

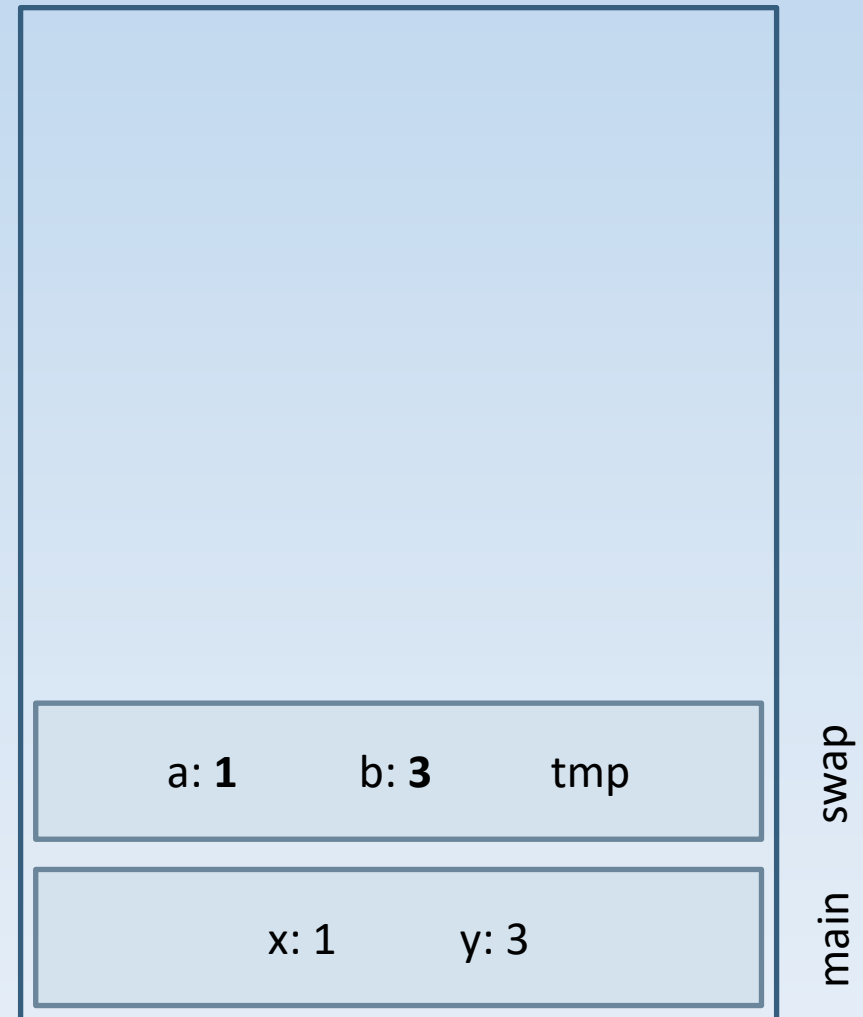
```
1  #include <stdio.h>
2
3  void swap(int a, int b)
4  {
5      int tmp = a;
6      a = b;
7      b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(x, y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

# Stack

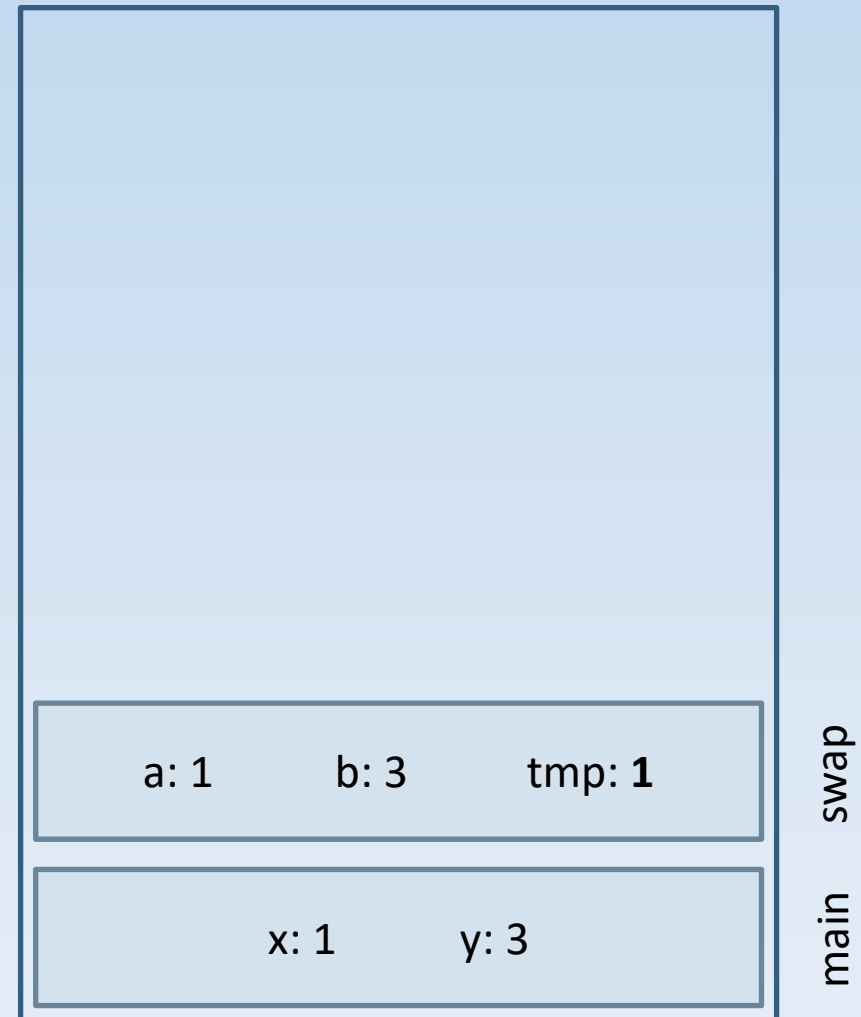
```
1  #include <stdio.h>
2
3  void swap(int a, int b)
4  {
5      int tmp = a;
6      a = b;
7      b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(x, y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

# Stack

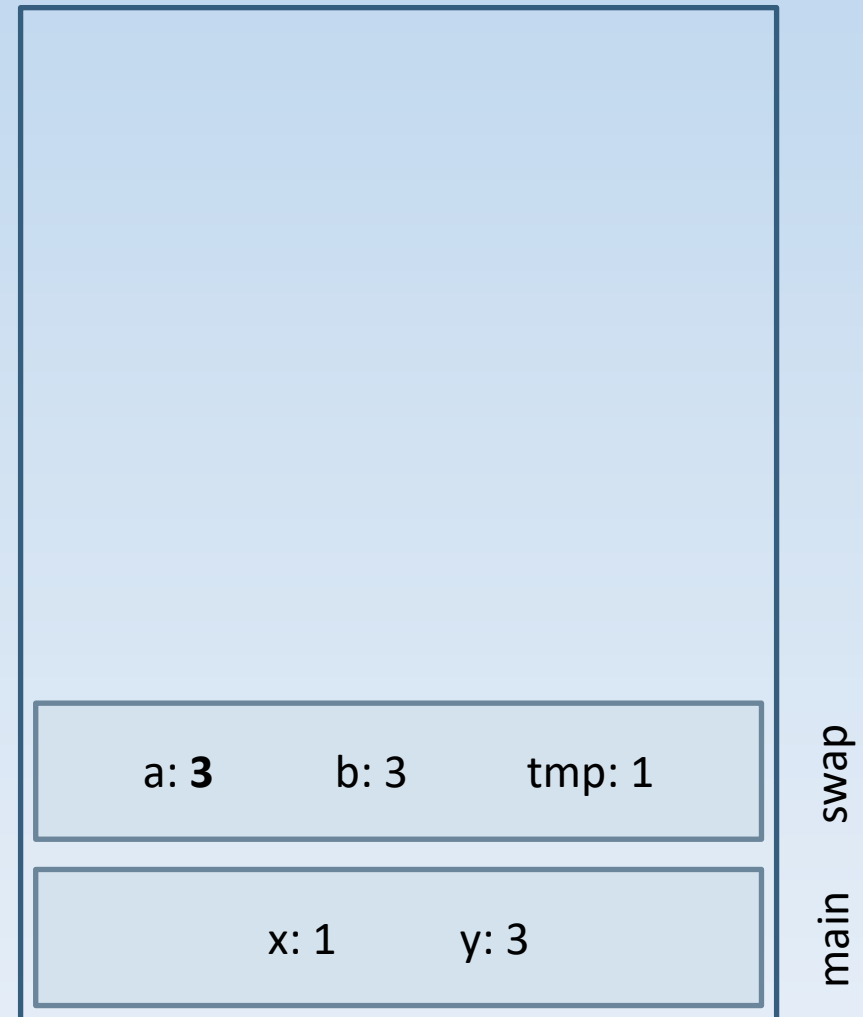
```
1  #include <stdio.h>
2
3  void swap(int a, int b)
4  {
5      int tmp = a;
6      a = b;
7      b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(x, y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

# Stack

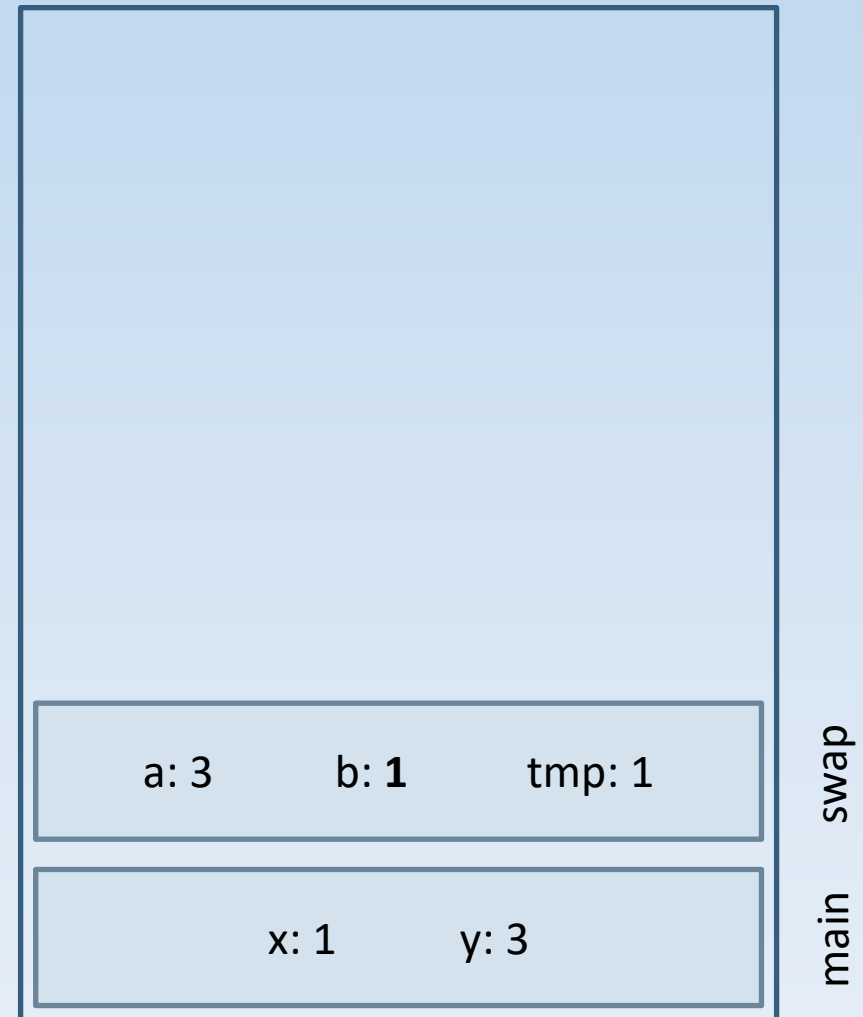
```
1  #include <stdio.h>
2
3  void swap(int a, int b)
4  {
5      int tmp = a;
6      a = b;
7      b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(x, y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

# Stack

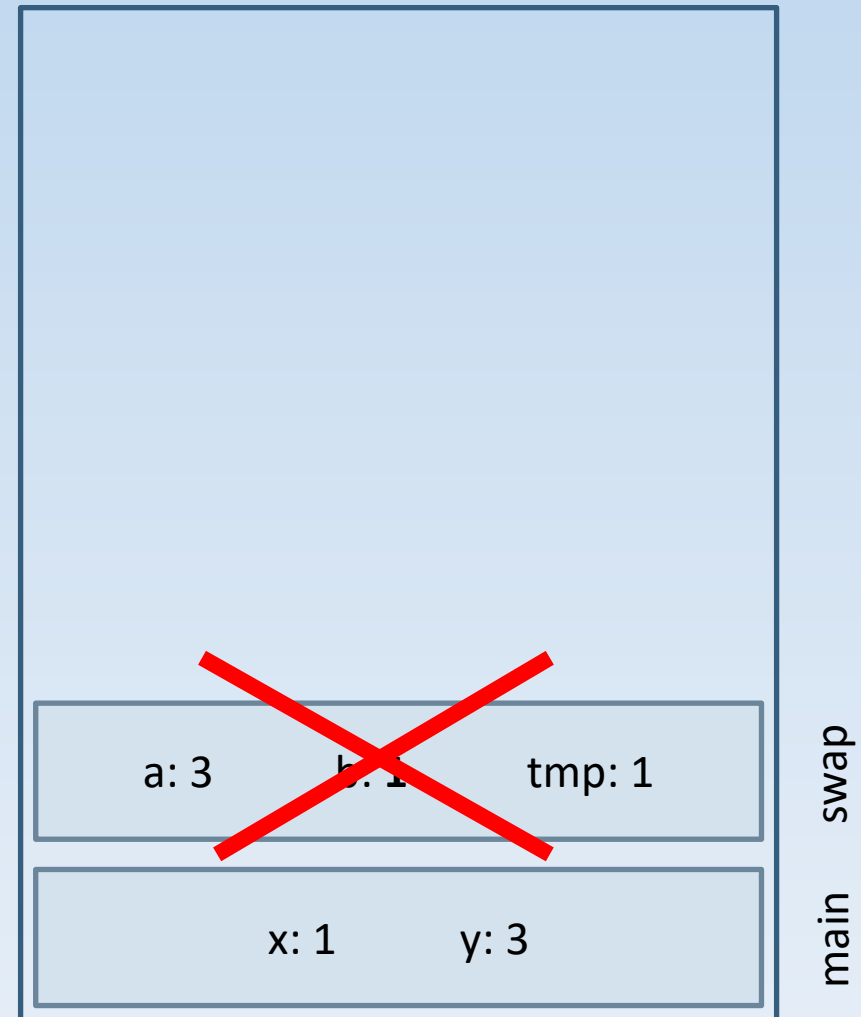
```
1  #include <stdio.h>
2
3  void swap(int a, int b)
4  {
5      int tmp = a;
6      a = b;
7      b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(x, y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

# Stack

```
1  #include <stdio.h>
2
3  void swap(int a, int b)
4  {
5      int tmp = a;
6      a = b;
7      b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(x, y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```

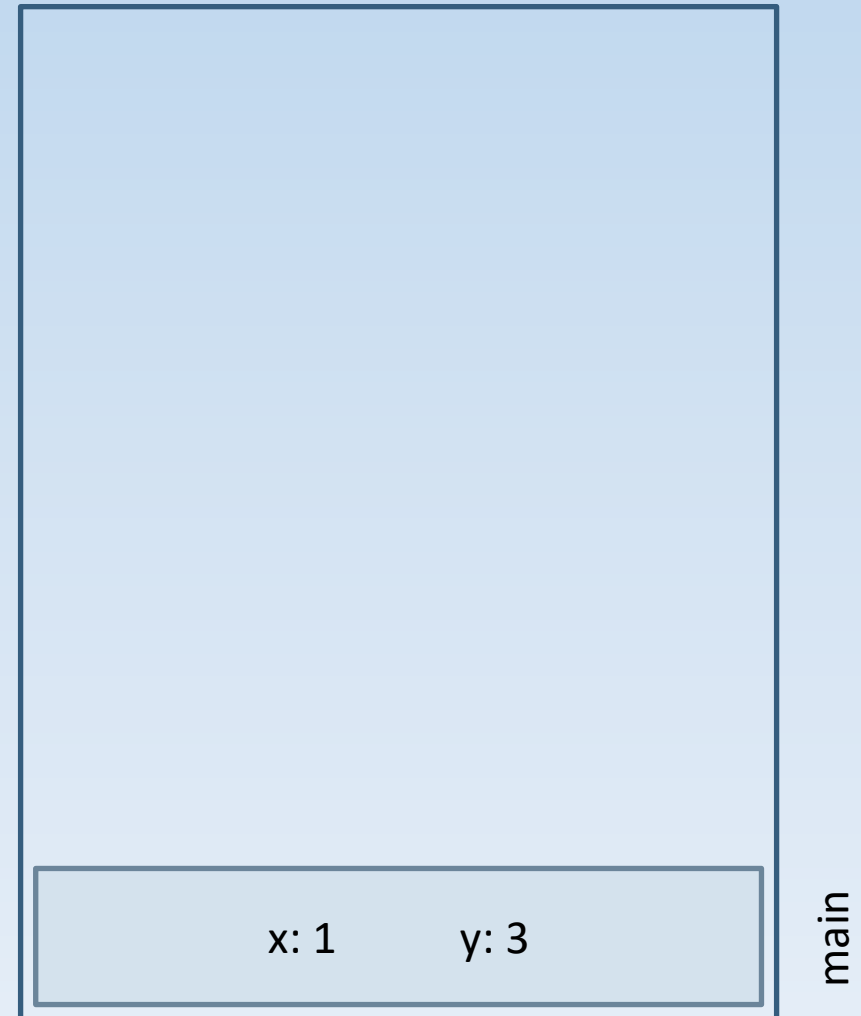


tekintsük csak a **stack**-et



# Stack

```
1  #include <stdio.h>
2
3  void swap(int a, int b)
4  {
5      int tmp = a;
6      a = b;
7      b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(x, y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

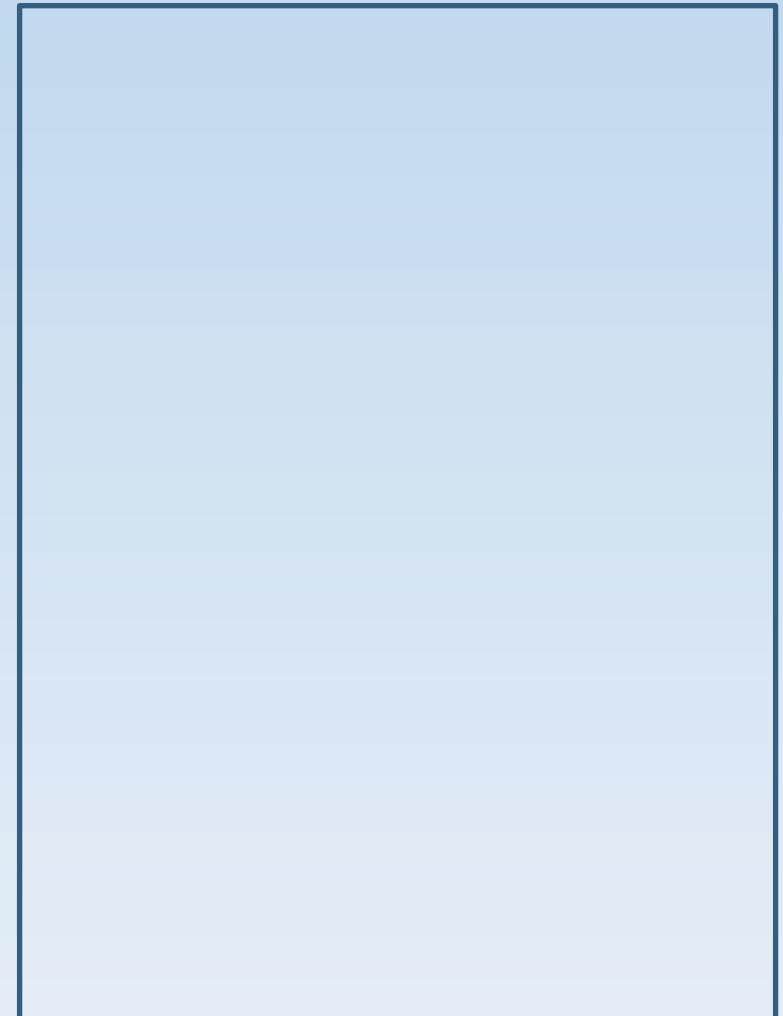
# Stack

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

Nézzük meg, hogy mutatók segítségével ezt hogyan tudjuk megtenni.

# Stack

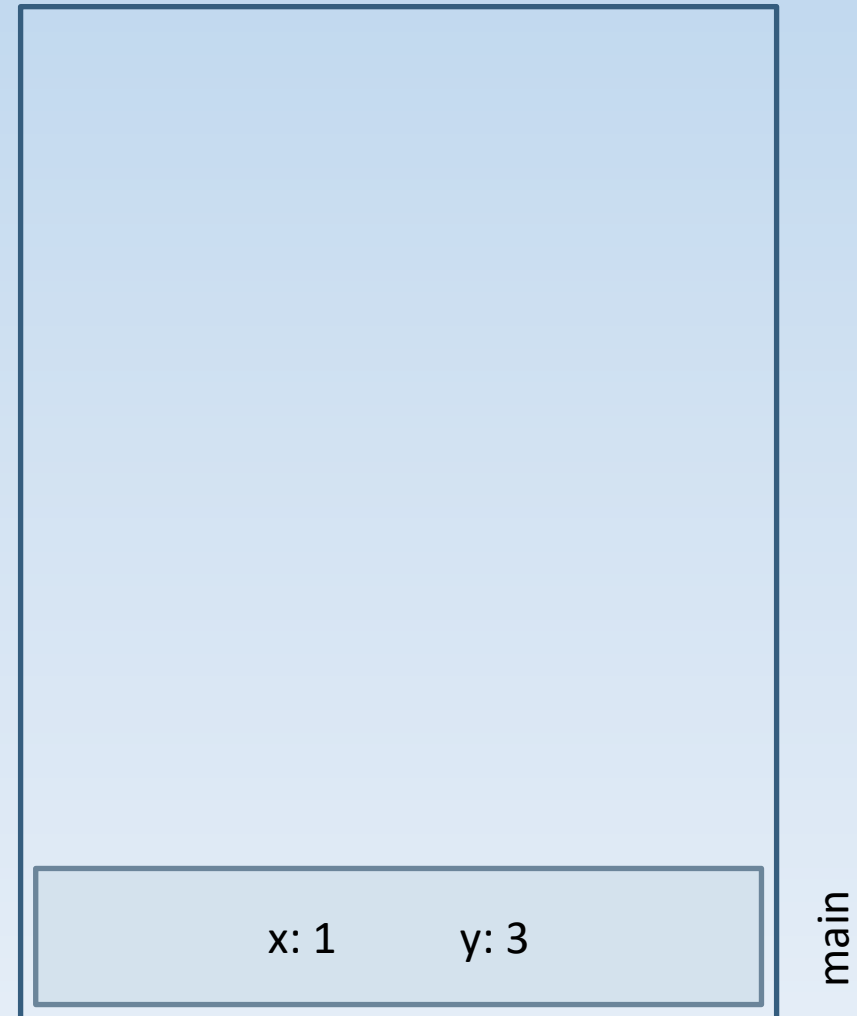
```
1  #include <stdio.h>
2
3  void swap(int *a, int *b)
4  {
5      int tmp = *a;
6      *a = *b;
7      *b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(&x, &y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
21
```



tekintsük csak a **stack**-et

# Stack

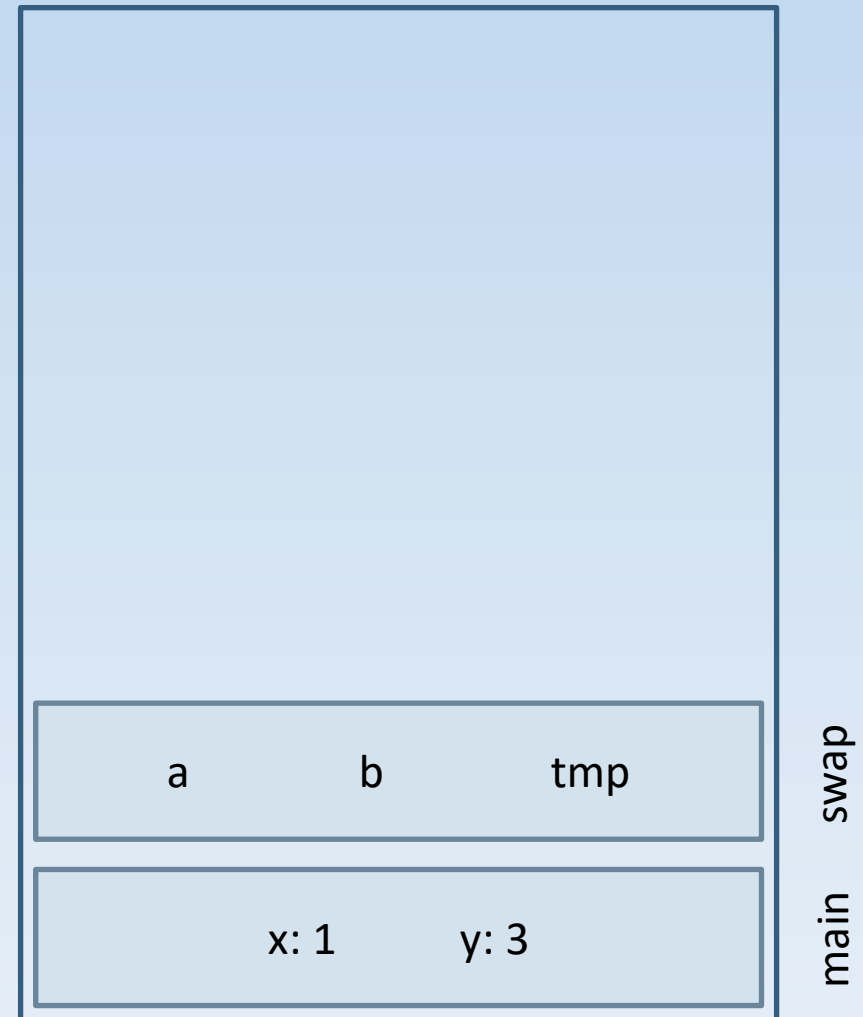
```
1  #include <stdio.h>
2
3  void swap(int *a, int *b)
4  {
5      int tmp = *a;
6      *a = *b;
7      *b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(&x, &y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

# Stack

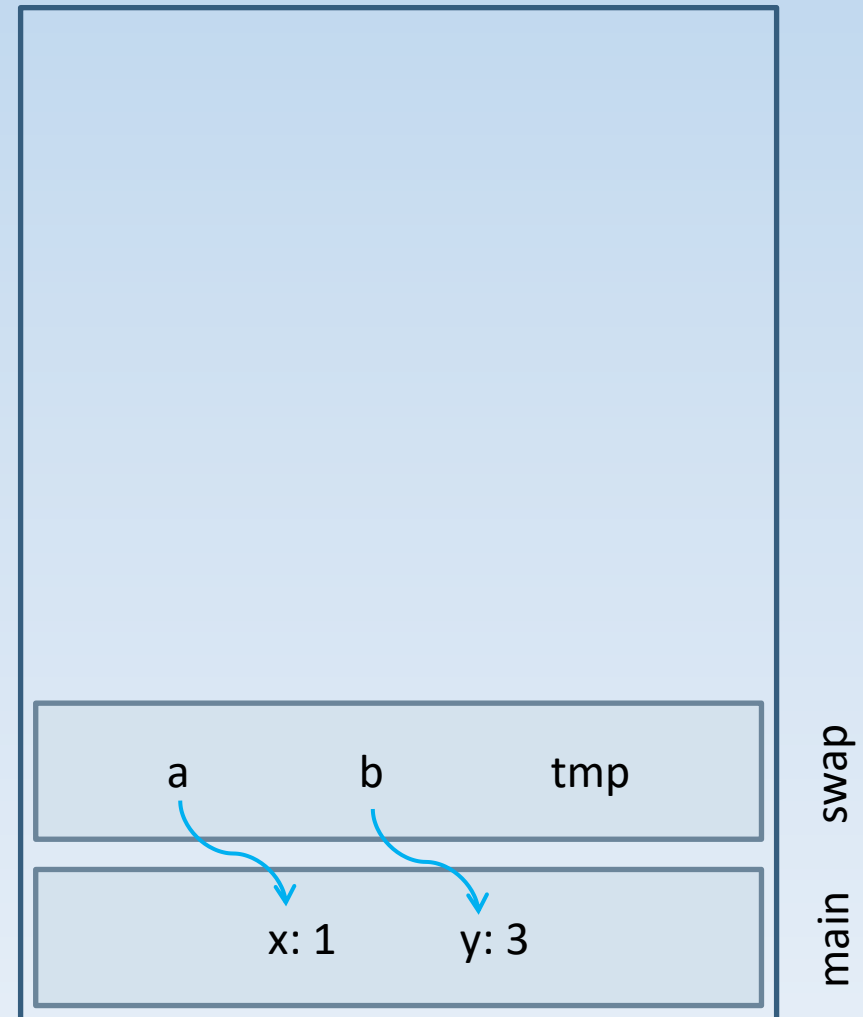
```
1  #include <stdio.h>
2
3  void swap(int *a, int *b)
4  {
5      int tmp = *a;
6      *a = *b;
7      *b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(&x, &y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

# Stack

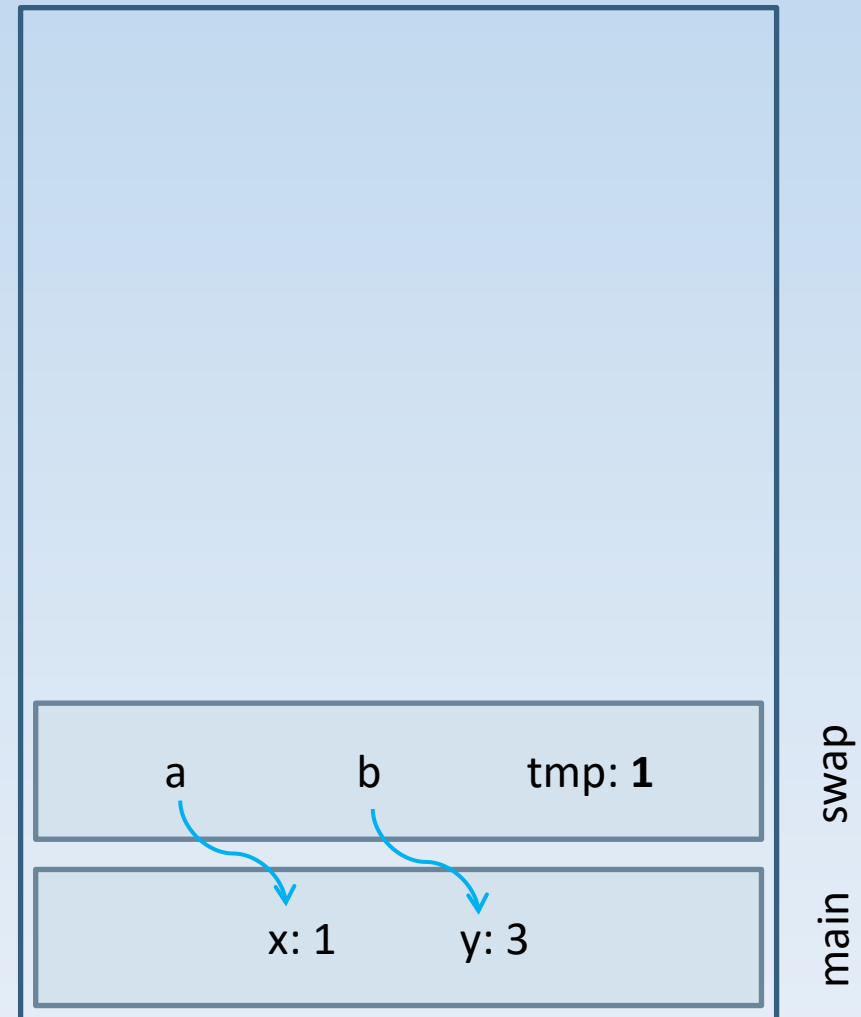
```
1  #include <stdio.h>
2
3  void swap(int *a, int *b)
4  {
5      int tmp = *a;
6      *a = *b;
7      *b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(&x, &y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

# Stack

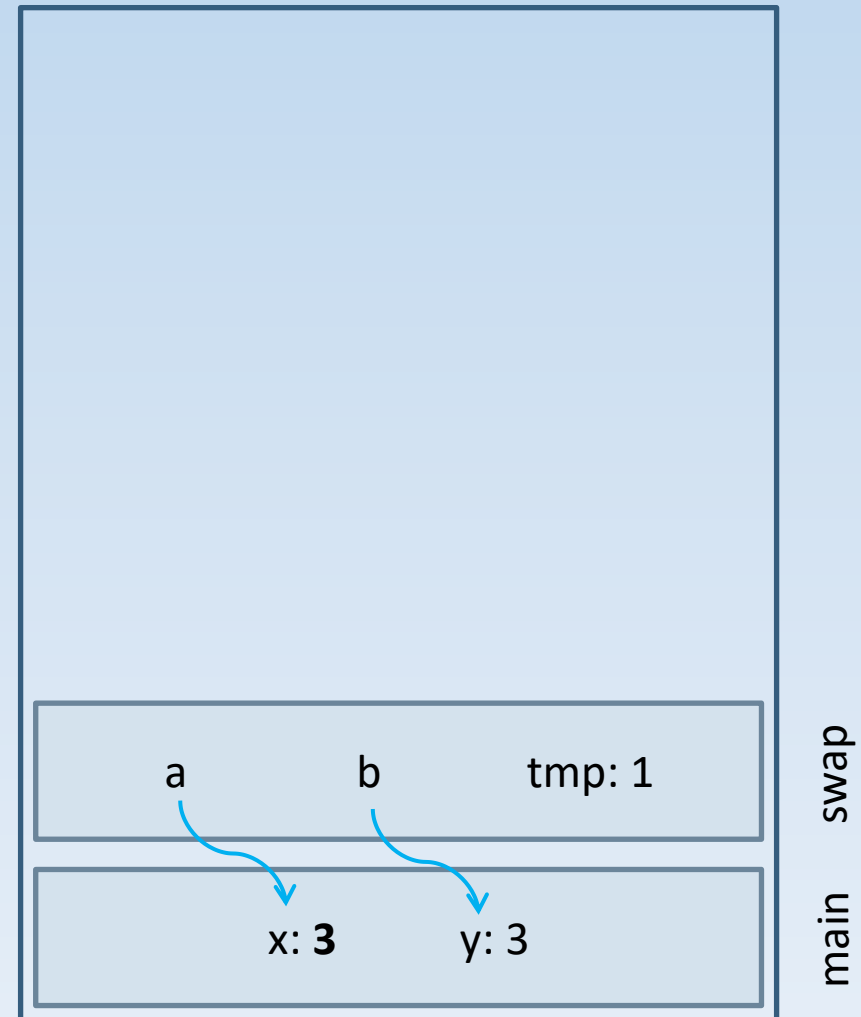
```
1  #include <stdio.h>
2
3  void swap(int *a, int *b)
4  {
5      int tmp = *a;
6      *a = *b;
7      *b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(&x, &y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

# Stack

```
1  #include <stdio.h>
2
3  void swap(int *a, int *b)
4  {
5      int tmp = *a;
6      *a = *b;
7      *b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(&x, &y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```

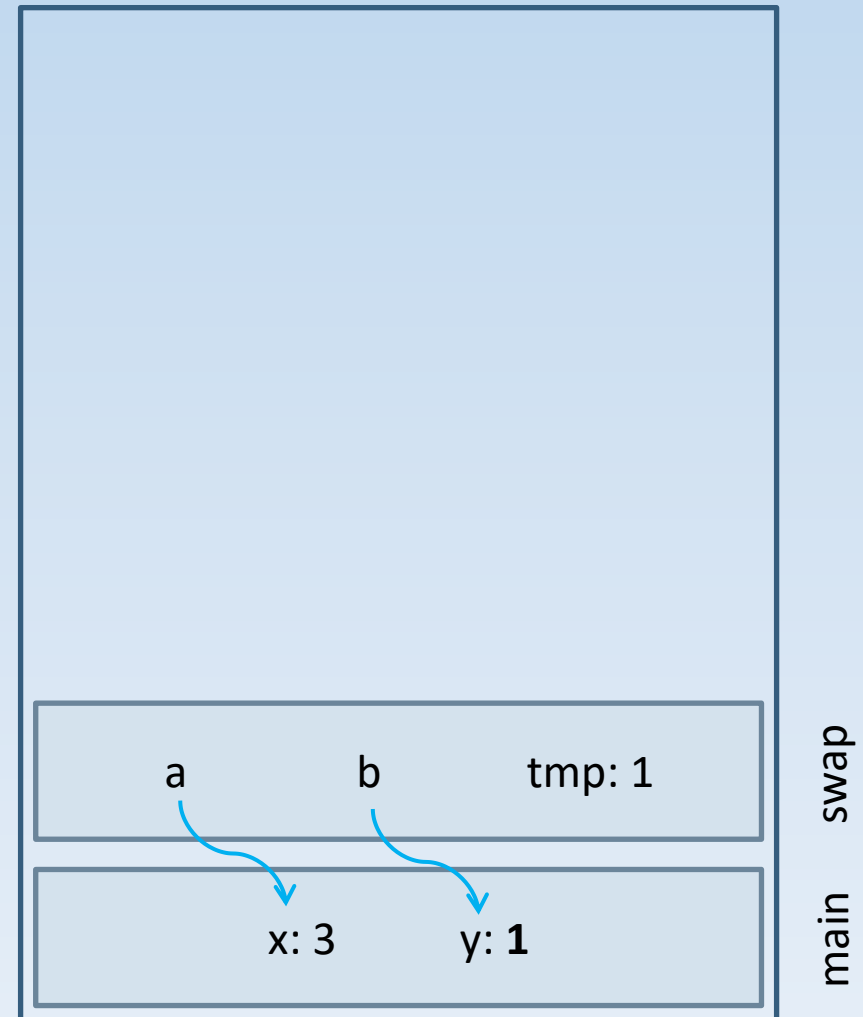


tekintsük csak a **stack**-et



# Stack

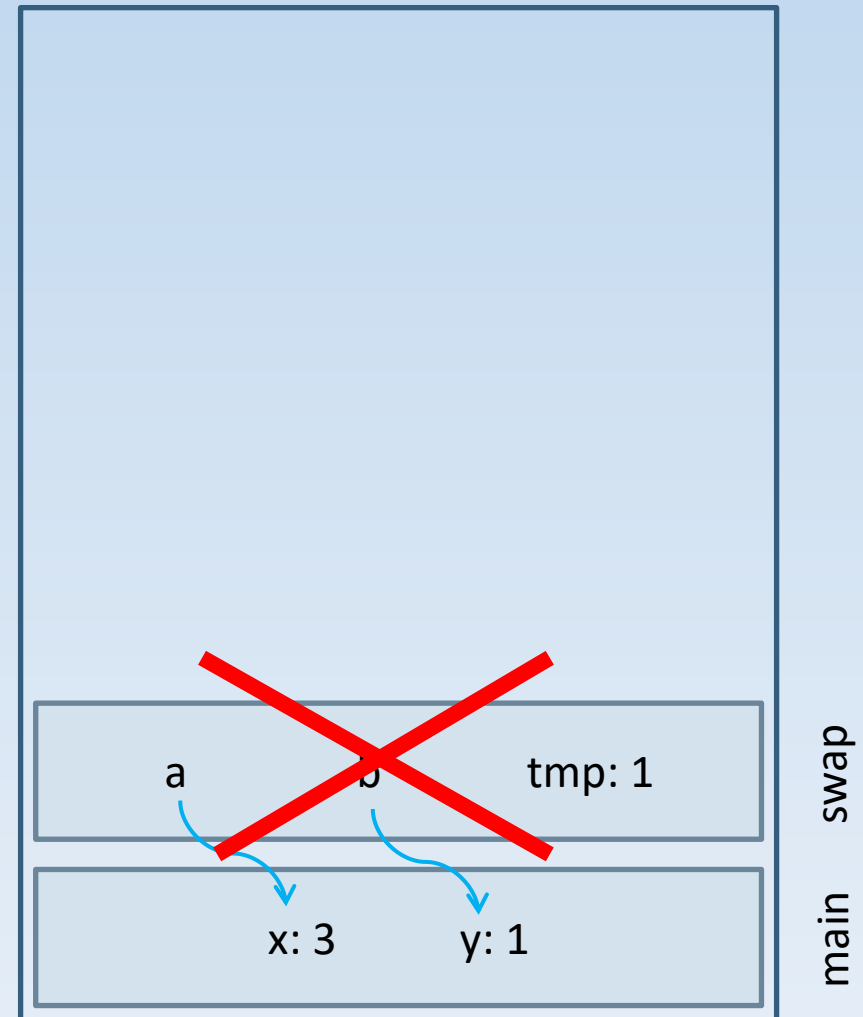
```
1  #include <stdio.h>
2
3  void swap(int *a, int *b)
4  {
5      int tmp = *a;
6      *a = *b;
7      *b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(&x, &y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

# Stack

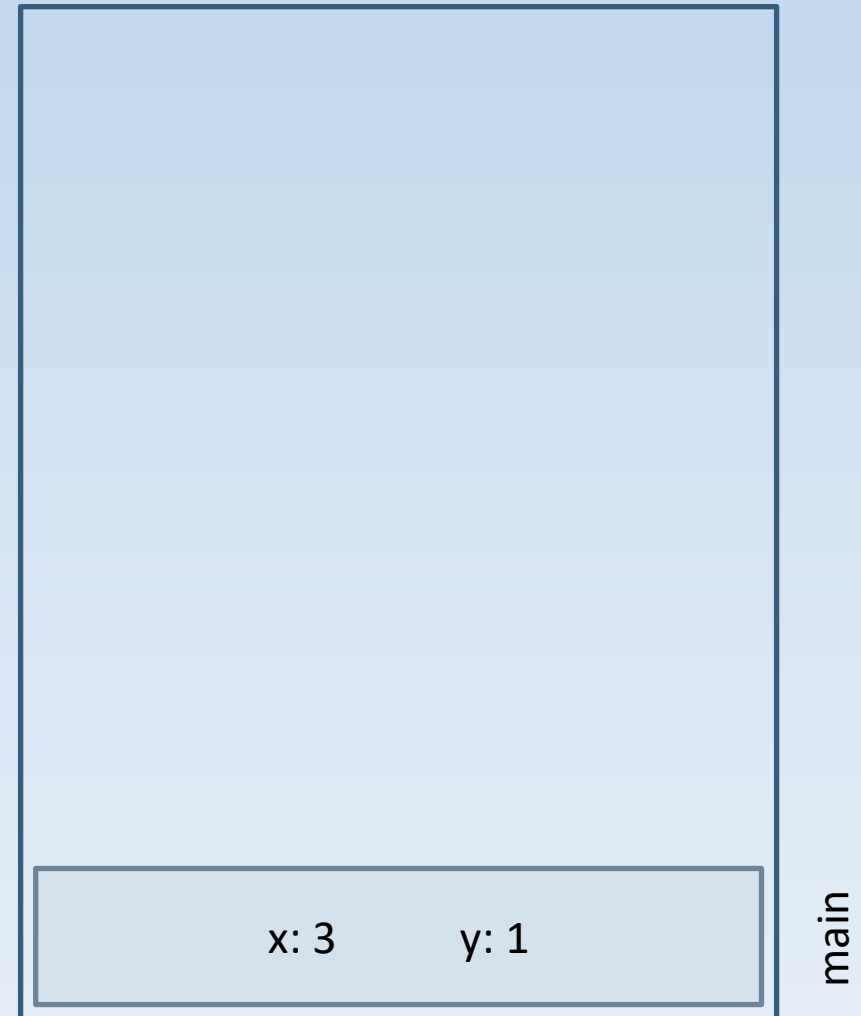
```
1  #include <stdio.h>
2
3  void swap(int *a, int *b)
4  {
5      int tmp = *a;
6      *a = *b;
7      *b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(&x, &y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

# Stack

```
1  #include <stdio.h>
2
3  void swap(int *a, int *b)
4  {
5      int tmp = *a;
6      *a = *b;
7      *b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("%d, %d\n", x, y);
16     swap(&x, &y);
17     printf("%d, %d\n", x, y);
18
19     return 0;
20 }
```



tekintsük csak a **stack**-et

# Stack

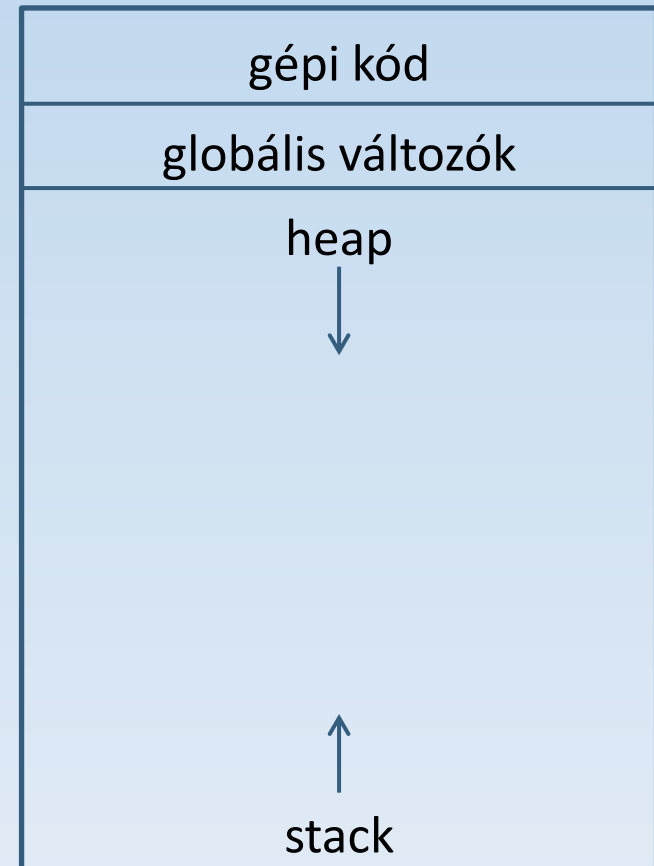
Mekkora a stack mérete?

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

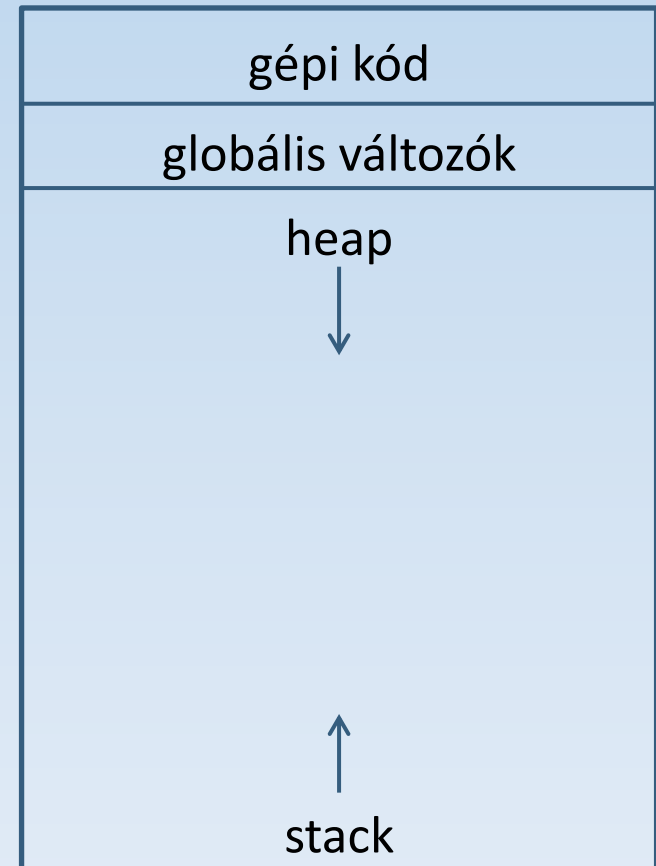
Mekkora a heap mérete?

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

`malloc( )`  
+  
`free( )`

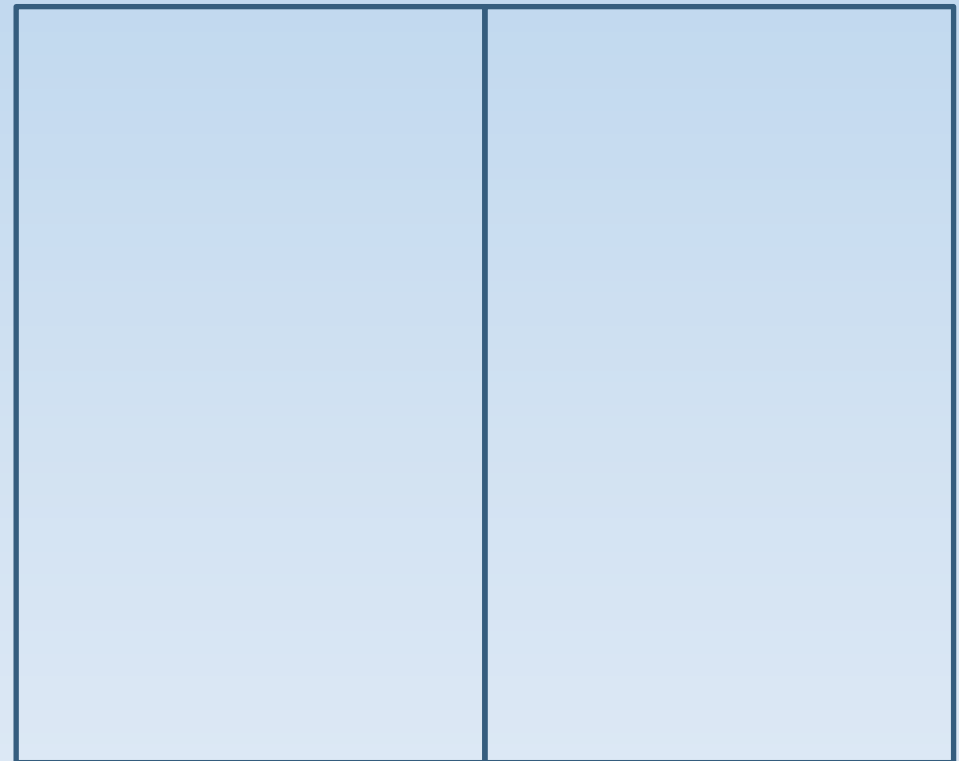
A heap-ben lefoglalt területeket nekünk kell felszabadítani!

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



# Heap

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  const long BYTE = 1;
5  const long KB = 1024 * BYTE;
6  const long MB = 1024 * KB;
7
8  void f0()
9  {
10     char *p = malloc(100 * MB);
11 }
12
13 int main()
14 {
15     int x, y;
16
17     f0();
18
19     return 0;
20 }
21
```

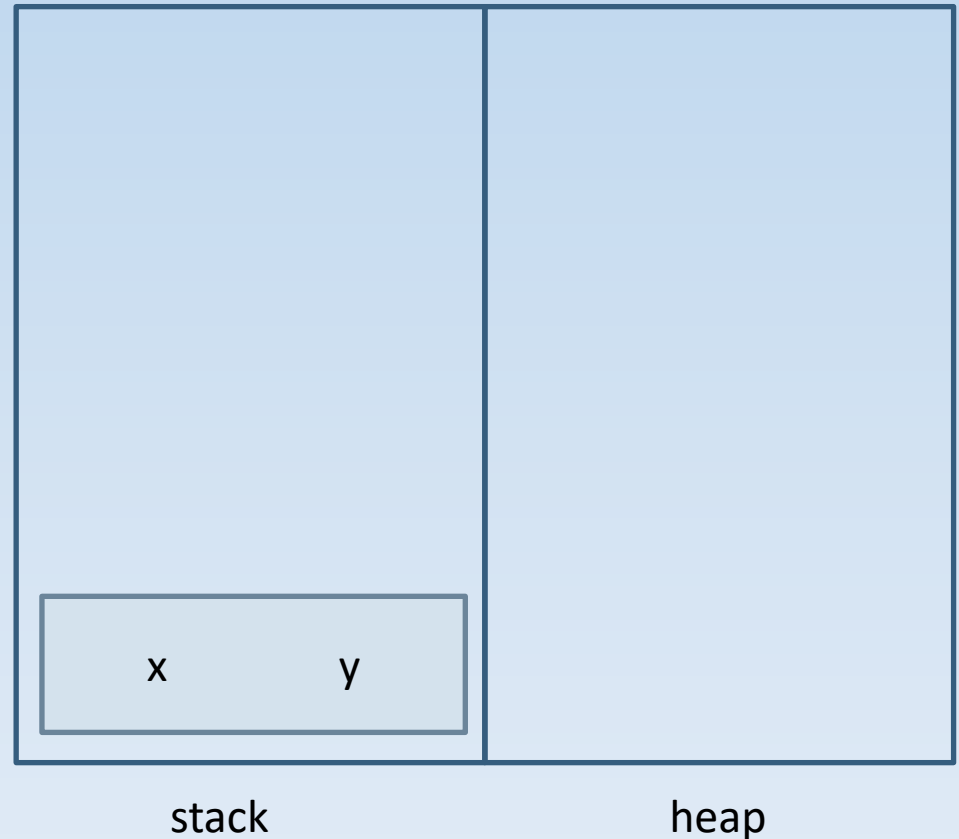


stack

heap

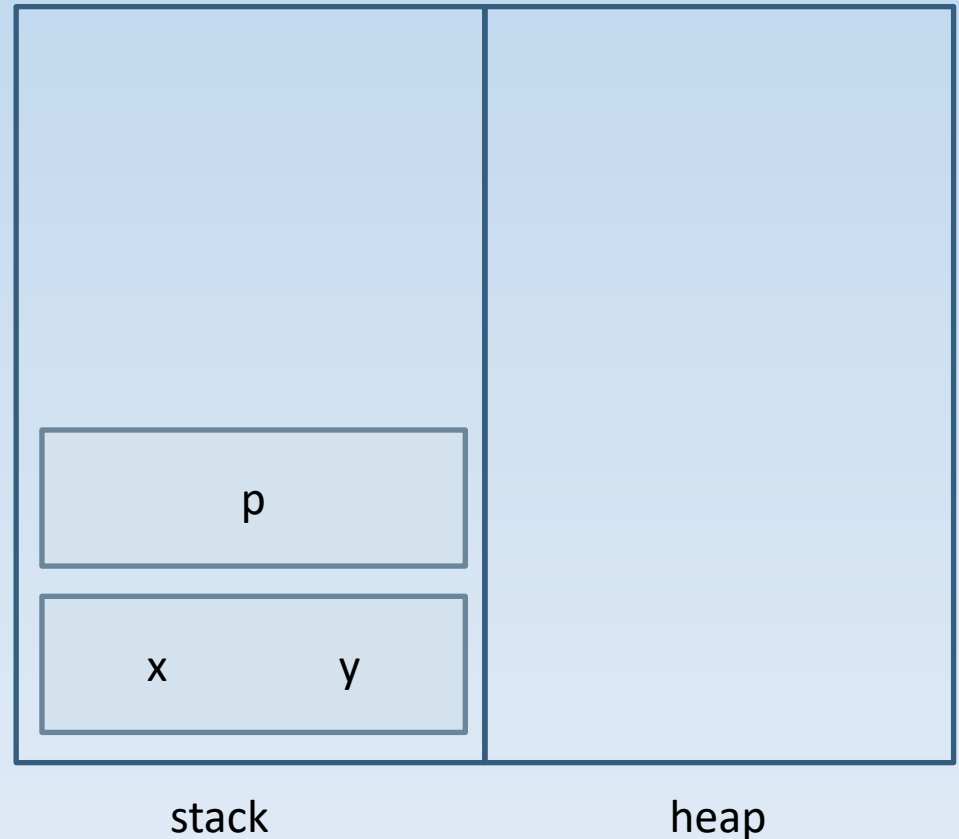
# Heap

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  const long BYTE = 1;
5  const long KB = 1024 * BYTE;
6  const long MB = 1024 * KB;
7
8  void f0()
9  {
10     char *p = malloc(100 * MB);
11 }
12
13 int main()
14 {
15     int x, y;
16
17     f0();
18
19     return 0;
20 }
21
```



# Heap

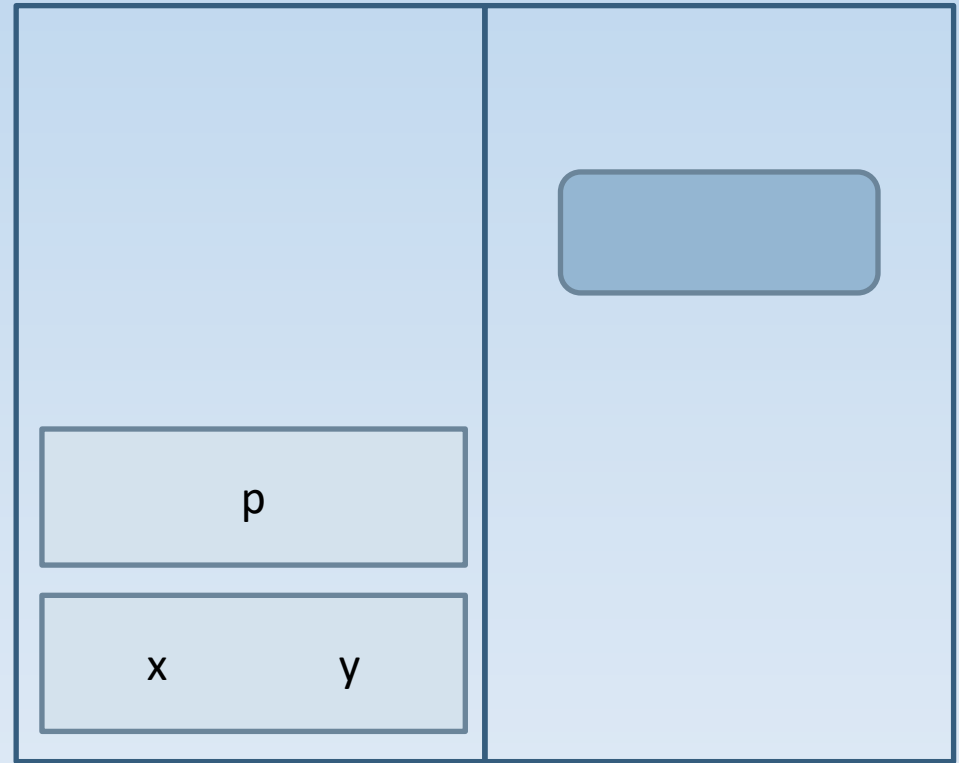
```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  const long BYTE = 1;
5  const long KB = 1024 * BYTE;
6  const long MB = 1024 * KB;
7
8  void f0()
9  {
10     char *p = malloc(100 * MB);
11 }
12
13 int main()
14 {
15     int x, y;
16
17     f0();
18
19     return 0;
20 }
21
```





# Heap

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  const long BYTE = 1;
5  const long KB = 1024 * BYTE;
6  const long MB = 1024 * KB;
7
8  void f0()
9  {
10     char *p = malloc(100 * MB);
11 }
12
13 int main()
14 {
15     int x, y;
16
17     f0();
18
19     return 0;
20 }
21
```

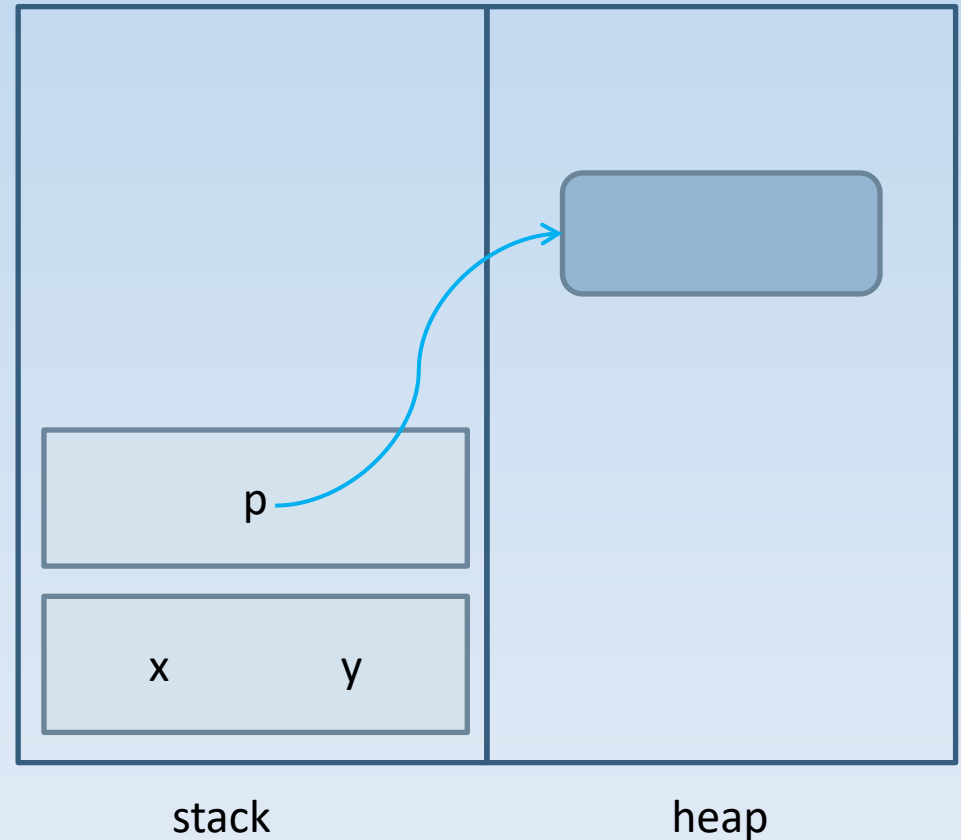


stack

heap

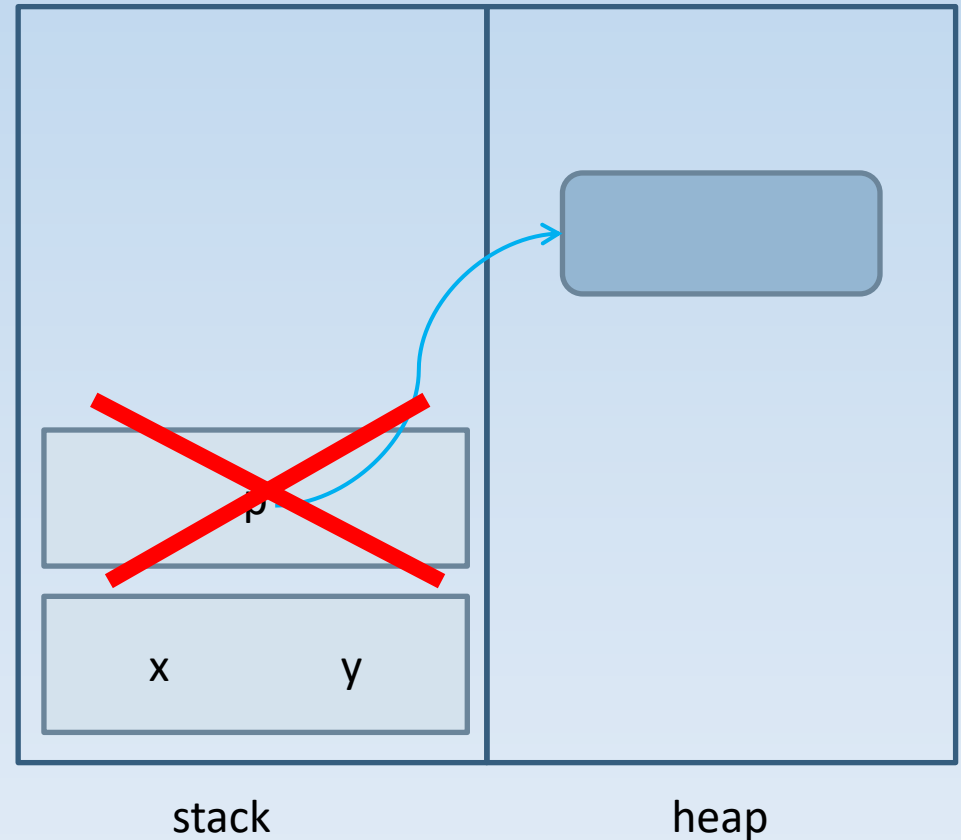
# Heap

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 const long BYTE = 1;
5 const long KB = 1024 * BYTE;
6 const long MB = 1024 * KB;
7
8 void f0()
9 {
10     char *p = malloc(100 * MB);
11 }
12
13 int main()
14 {
15     int x, y;
16
17     f0();
18
19     return 0;
20 }
21
```



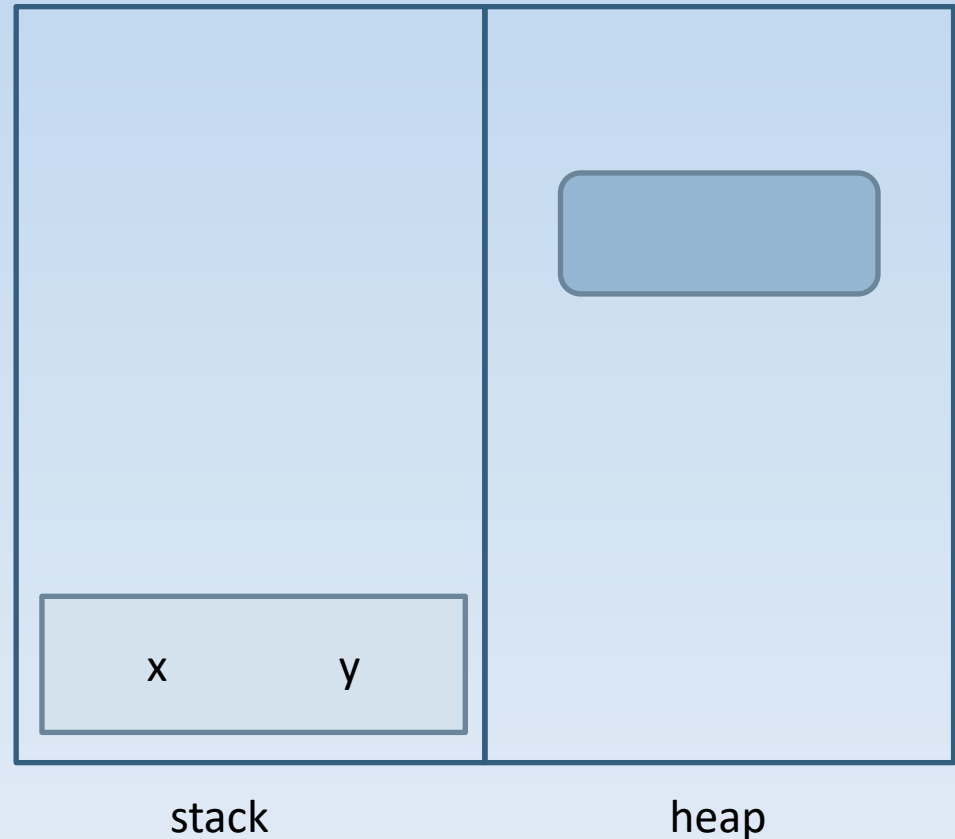
# Heap

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  const long BYTE = 1;
5  const long KB = 1024 * BYTE;
6  const long MB = 1024 * KB;
7
8  void f0()
9  {
10     char *p = malloc(100 * MB);
11 }
12
13 int main()
14 {
15     int x, y;
16
17     f0();
18
19     return 0;
20 }
21
```



# Heap

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 const long BYTE = 1;
5 const long KB = 1024 * BYTE;
6 const long MB = 1024 * KB;
7
8 void f0()
9 {
10     char *p = malloc(100 * MB);
11 }
12
13 int main()
14 {
15     int x, y;
16
17     f0();
18
19     return 0;
20 }
21
```



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

```
4
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

```
5
6 typedef struct {
7     int x;
8     int y;
9 } Pont;
10
11 Pont * origo()
12 {
13     Pont *p = malloc(sizeof(Pont));
14     p->x = 0;
15     p->y = 0;
16
17     return p;
18 }
19
20 int main()
21 {
22     Pont *kozeppont = origo();
23
24     printf("P(%d, %d)\n", kozeppont->x, kozeppont->y);
25
26     free(kozeppont);
27
28     return 0;
29 }
30
```

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](#)