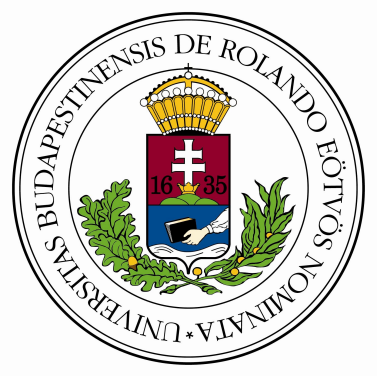
Imperatív programozás

Dinamikus memóriakezelés



Kozsik Tamás és mások

ELTE Eötvös Loránd Tudományegyetem

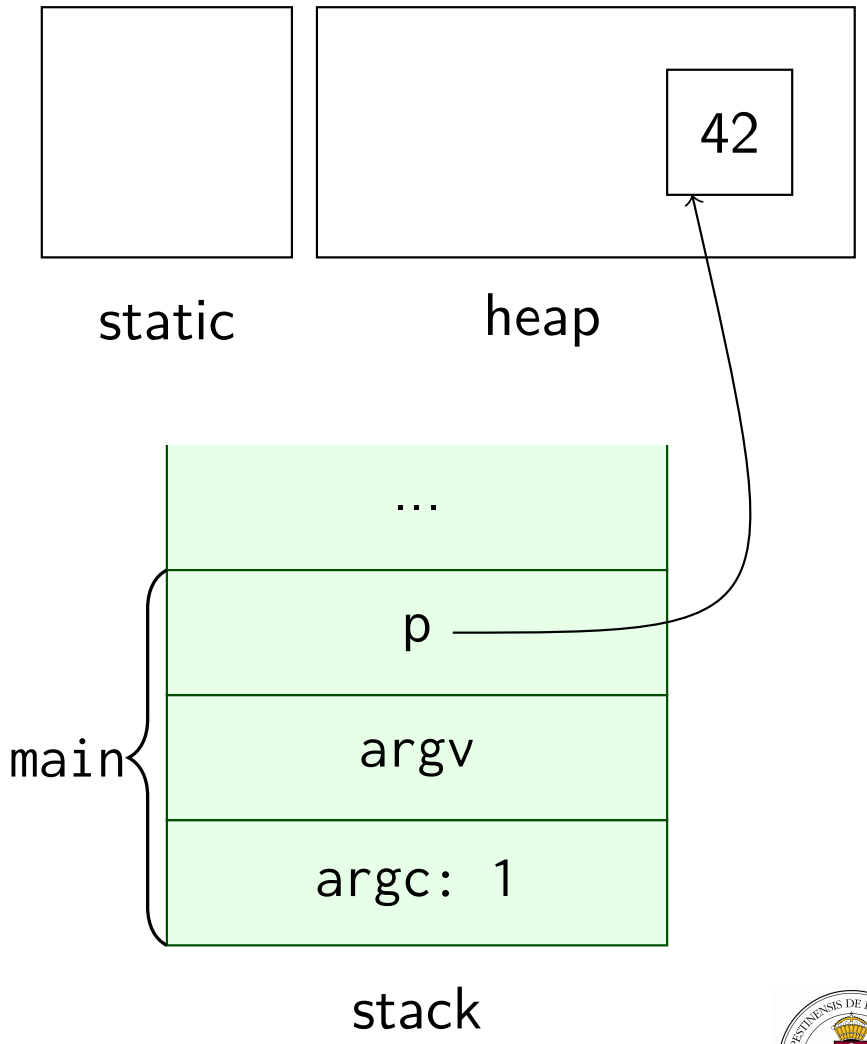
Dinamikus memóriakezelés

- Dinamikus tárolású „változók”
  - Heap (dinamikus tárhely)
- Élettartam: programozható
  - Létrehozás: allokáló utasítással
  - Felszabadítás
    - Felszabadító utasítás (C)
    - Szemétgyűjtés – garbage collection (Python)
- Használat: indirekció
  - Mutató – pointer (C)
  - Referencia – reference (Python)

Mutatók C-ben

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

int main()
{
    int *p;
    p = (int*)malloc(sizeof(int));
    if( NULL != p )
    {
        *p = 42;
        printf("%d\n", *p);
        free(p);
        return 0;
    }
    else return 1;
}
```



Összetevők

- Mutató (típusú) változó: `int *p;`
  - Vigyázat: `int* p, v;`
  - Hasonlóan: `int v, t[10];`
- Dereferálás (hova mutat?): `*p`
- „Sehova sem mutat”: `NULL`
- Allokálás és felszabadítás: `malloc` és `free` (`stdlib.h`)
  - Típuskényszerítés: `void*` $\rightarrow$ pl. `int*`

Mire jó?

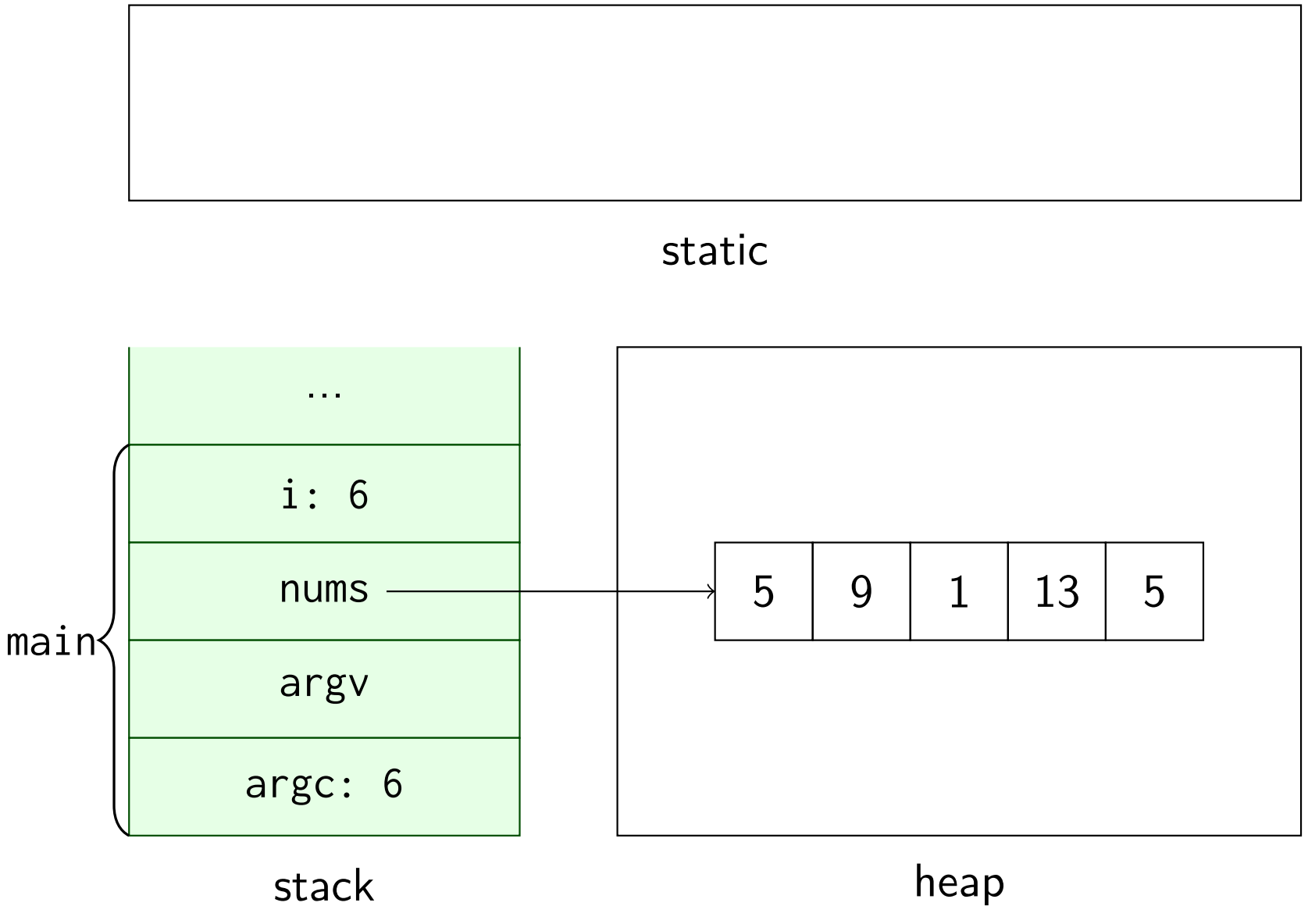
- Dinamikus méretű adat(-struktúra)
- Láncolt adatszerkezet
- Kimenő szemantikájú paraméterátadás
- ...

Dinamikus méretű adatszerkezet

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

int main( int argc, char* argv[] ){
    int *nums = (int*)malloc((argc-1)*sizeof(int));
    if( NULL != nums ){
        int i;
        for( i=1; i<argc; ++i ) nums[i-1] = atoi(argv[i]);
        /* TODO: sort nums */
        for( i=1; i<argc; ++i ) printf("%d\n", nums[i-1]);
        free(nums);
        return 0;
    } else return 1;
}
```

Dinamikus méretű adatszerkezet



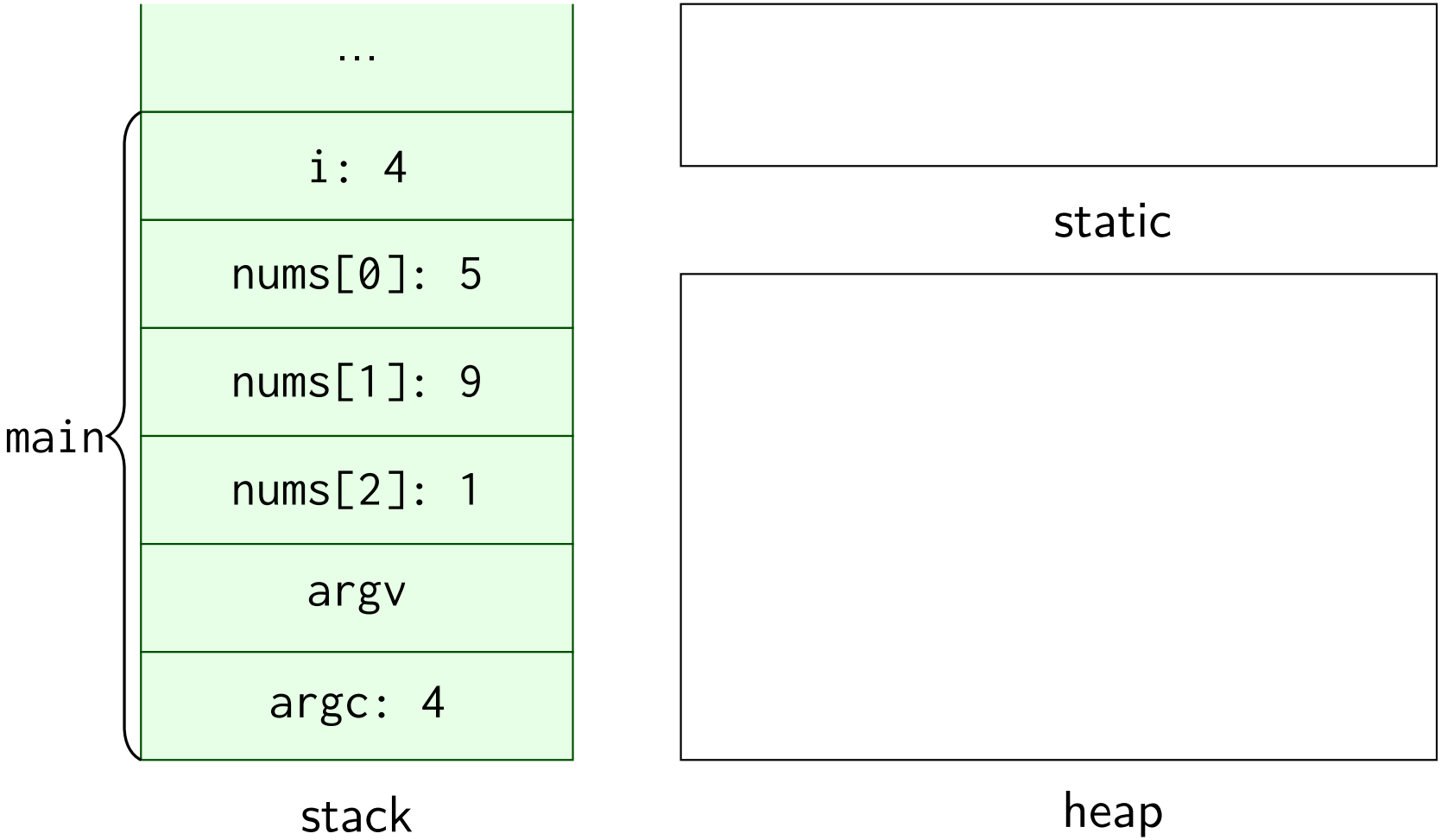
Kerülendő megoldás

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

int main( int argc, char* argv[] ){
    int nums[argc-1];
    int i;
    for( i=1; i<argc; ++i ) nums[i-1] = atoi(argv[i]);
    /* TODO: sort nums */
    for( i=1; i<argc; ++i ) printf("%d\n", nums[i-1]);
    return 0;
}
```

- C99: Variable Length Array (VLA)
- Nincs az ANSI C és C++ szabványokban

Kerülendő: VLA



Láncolt adatszerkezet

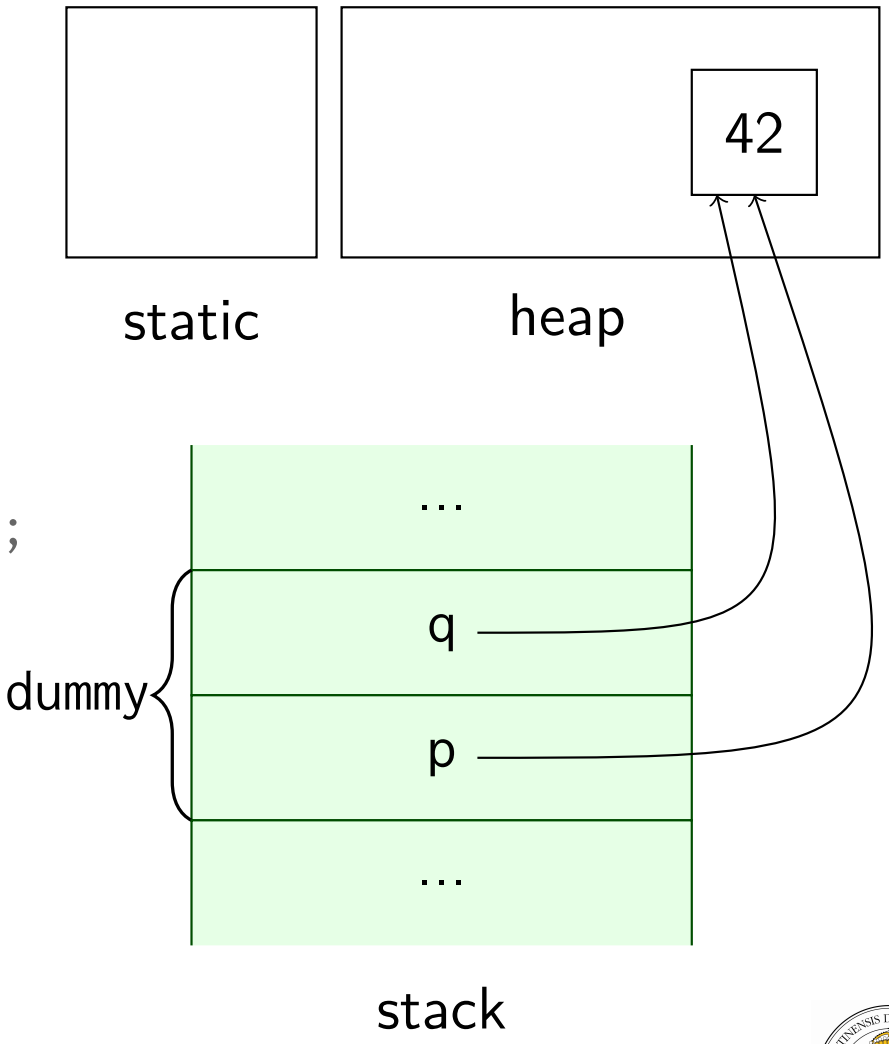
- Sorozat típus
- Bináris fa típus
- Gráf típus
- ...

Bejárás közben konstans idejű törlés/beszúrás

Aliasing

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

void dummy(void)
{
    int *p, *q;
    p = (int*)malloc(sizeof(int));
    if( NULL != p ){
        q = p;
        *p = 42;
        printf("%d\n", *q);
        free(p);
    }
}
```



Felszabadítás

Minden dinamikusan létrehozott változót pontosan egyszer!

- Ha többször próbálom: hiba
- Ha egyszer sem: „elszivárog a memória” (memory leak)

Felszabadított változóra hivatkozni hiba!

Hivatkozás felszabadított változóra

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

void dummy(void)
{
    int *p, *q;
    p = (int*)malloc(sizeof(int));
    if( NULL != p ){
        q = p;
        *p = 42;
        free(p);
        printf("%d\n", *q);   /* hiba */
    }
}
```

Többszörösen felszabadított változó

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

void dummy(void)
{
    int *p, *q;
    p = (int*)malloc(sizeof(int));
    if( NULL != p ){
        q = p;
        *p = 42;
        printf("%d\n", *q);
        free(p);
        free(q);       /* hiba */
    }
}
```

Fel nem szabadított változó

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

void dummy(void)
{
    int *p, *q;
    p = (int*)malloc(sizeof(int));
    if( NULL != p ){
        q = p;
        *p = 42;
        printf("%d\n", *q);
    }       /* hiba */
}
```

Tulajdonos?

```
void dummy(void)
{
    int *q;
    {
        int *p = (int*)malloc(sizeof(int));
        q = p;
        if( NULL != p ){
            *p = 42;
        }
    }
    if( NULL != q ){
        printf("%d\n", *q);
        free(q);
    }
}
```

Könnyű elrontani!

```
int *produce( int argc, char* argv[] ){
    int *nums = (int*)malloc((argc-1)*sizeof(int));
    if( NULL != nums ){
        for( int i=1; i<argc; ++i ) nums[i-1] = atoi(argv[i]);
    }
    return nums;
}
void consume( int n, int *nums ){
    for( int i=0; i<n; ++i ) printf("%d\n", nums[i]);
    free(nums);
}
int main( int argc, char* argv[] ){
    int *nums = produce(argc,argv);
    if( NULL != nums ){ /* TODO: sort nums */ consume(argc-1,nums); }
    return (NULL == nums);
}
```

Alias C-ben és Pythonban

Alias: ugyanarra a tárterületre többféle névvel hivatkozhatunk

C: mutatók

```c
int xs[] = {1,2,3};
int *ys = xs;
xs[2] = 4;
printf("%d\n", ys[2]);
```

Python: referenciák

```python
xs = [1,2,3]
ys = xs
xs[2] = 4
print(ys[2])
```

Mutató gyűjtőtípusa

```c
int *p = (int *)malloc(sizeof(int));
if( NULL != p )
{
    *p = 123;
    *p = 12.3;  /* automatikus típuskonverzió */
    printf("%d\n", *p);
    free(p);
}
```

Mutató gyűjtőtípusa: típuskényszerítés

```c
float *q = (float *)malloc(sizeof(float));
if( NULL != q )
{
    int *p = (int *)q;
    *q = 12.3;
    printf("%d\n",*p);
    free(q);
}
```

Dinamikus tárhely elérése

C

- Explicit (mutató)
- Statikus típusellenőrzés
- Erősen típusos
- Felszabadítás

Python

- Implicit
- Dinamikus típusellenőrzés
- Erősen típusos
- Szemétgyűjtés

A del utasítás

C: dinamikus változó felszabadítása

```c
int *p = ...
int *q = p;
free(p);
printf("%d",*q);
```

Python: hivatkozás törlése

```python
v = [1,2,3]
u = v
del v   # v becomes undefined
print(u)
```

„Módosítható” és „nem módosítható” típusok

Mutable: list

```
v = [1,2,3]
print(v[2]) # 3
v[2] = 4
print(v)
```

Immutable: tuple

```
v = (1,2,3)
print(v[2]) # 3
v[2] = 4  # TypeError: 'tuple' object
   # does not support item assignment
```

„Módosító” értékadás

Mutable: list

```
v = [1,2,3]
u = v
v += [4,5]
print(v)    # [1,2,3,4,5]
print(u)    # [1,2,3,4,5]
```

Immutable: tuple

```
v = (1,2,3)
u = v
v += (4,5)  # v = v + (4,5)
print(v)    # (1,2,3,4,5)
print(u)    # (1,2,3)
```

Beépített Python típusok

Mutable

- list, pl. `[1,2,3]`
- set, pl. `{1,2,3}`
- dictionary, pl. `{'a':1, 'b':2, 1:'a'}`

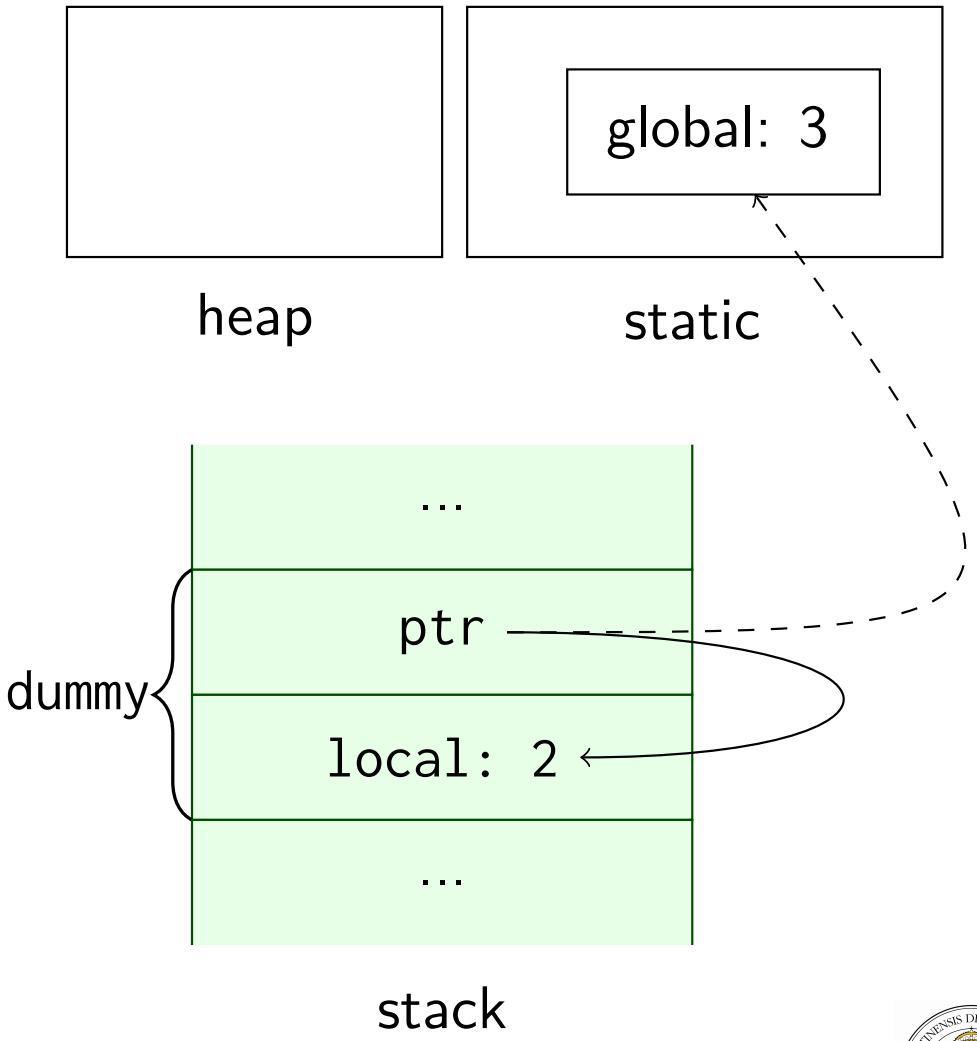
Immutable

- tuple, pl. `(1,2,3)`
- frozenset, pl. `frozenset({1,2,3})`
- range, pl. `range(1,23)`
- numeric: int, float, complex
- text, pl. `'123'`

Mutató nem dinamikus változóra

```
int global = 1;

void dummy(void)
{
    int local = 2;
    int *ptr;
    ptr = &global; *ptr = 3;
    ptr = &local;  *ptr = 4;
}
```



Érvénytelen mutató

Értelmetlen

```
int *make_ptr(void)
{

    int n = 42;
    return &n;
}
```

Értelmes

```
int *make_ptr(void)
{
    int *ptr = (int*)malloc(sizeof(int));
    *ptr = 42;
    return ptr;
}
```

```
    printf("%d\n", *make_ptr());
```