Imperatív programozás

Hatókör

Kozsik Tamás és mások

Eötvös Loránd Tudományegyetem

2022. szeptember 2.

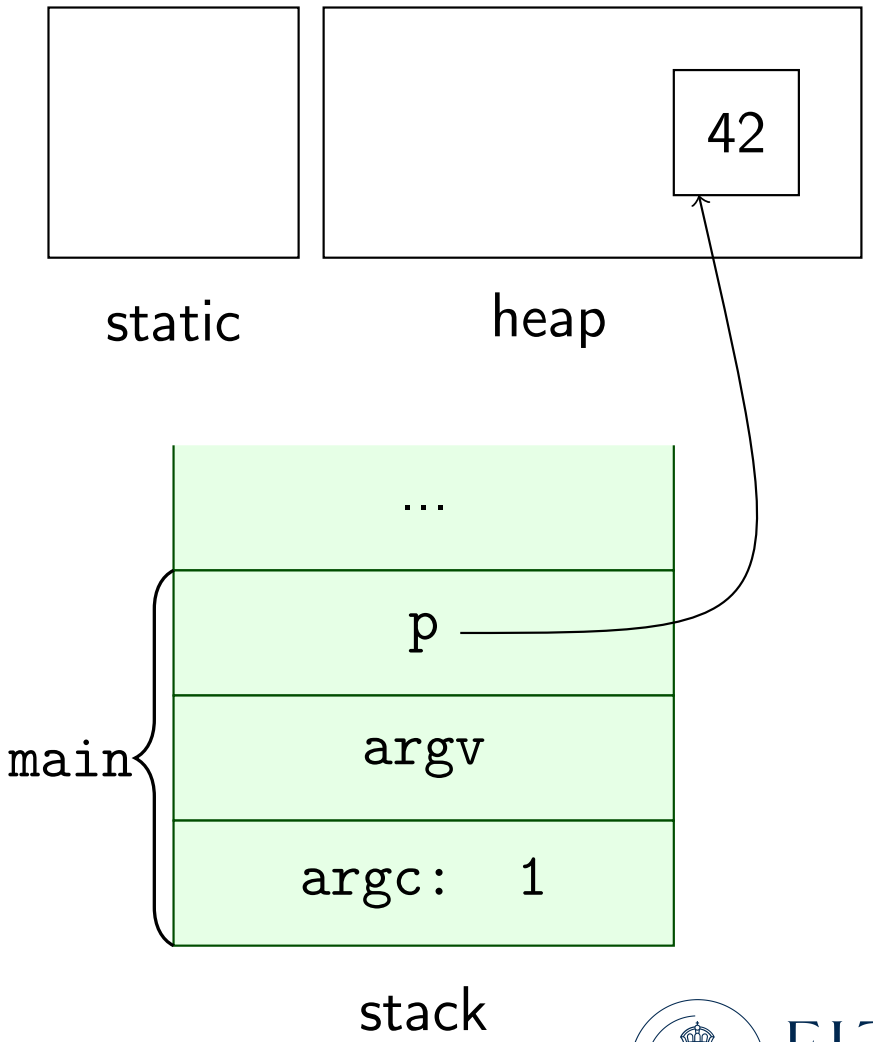
Tartalomjegyzék

Dinamikus memóriakezelés

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

Mutatók C-ben

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

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



Összetevők

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

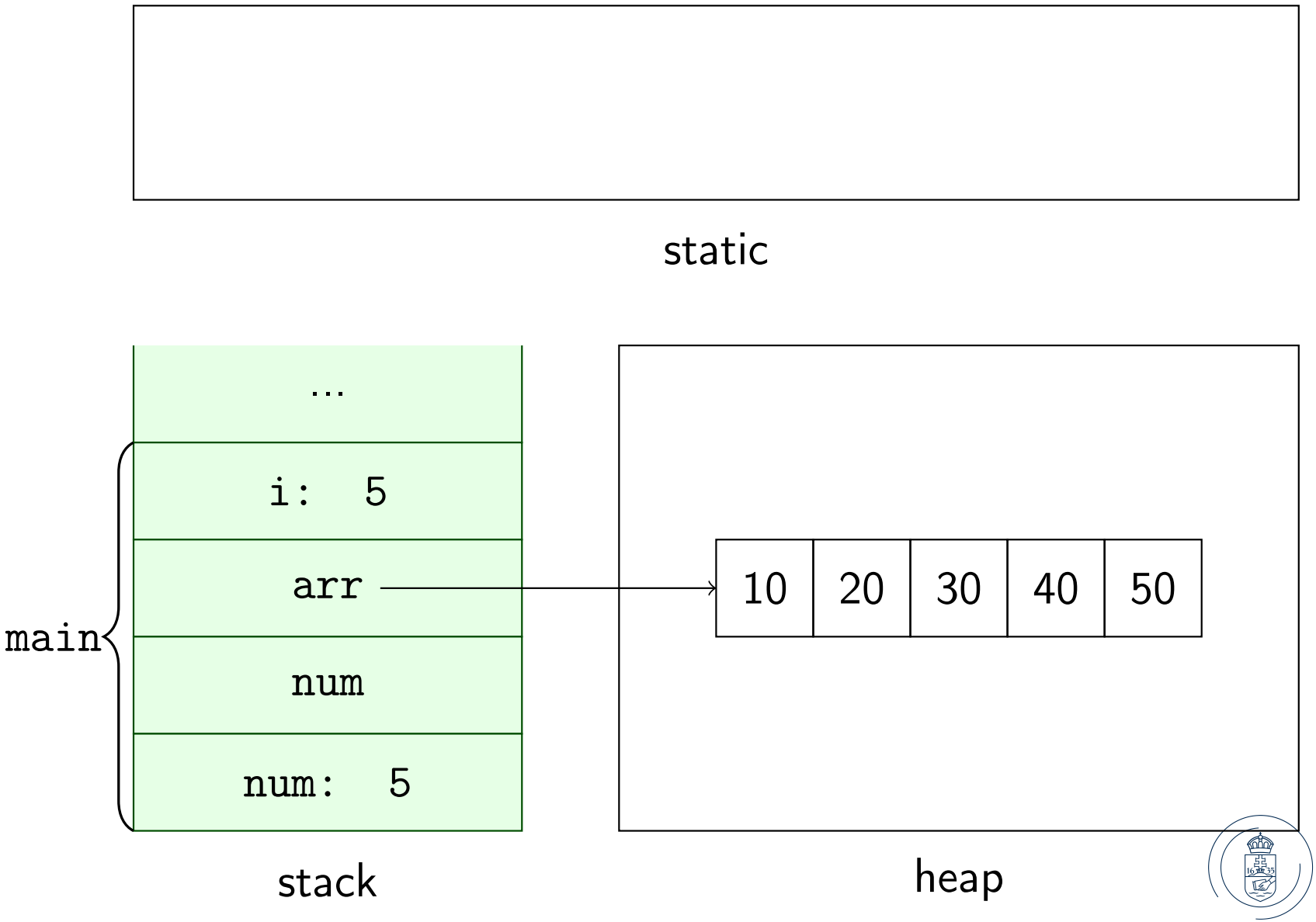
Mire jó?

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

Dinamikus méretű adatszerkezet

```c
#include <stdlib.h>

int getNum() { ... }

int main()
{
  int num = getNum();
  int* arr = (int*)malloc(num * sizeof(int));

  if (arr == NULL)
    return 1;

  for (int i = 0; i < num; ++i)
    arr[i] = (i + 1) * 10;

  free(arr);
}
```

Dinamikus méretű adatszerkezet

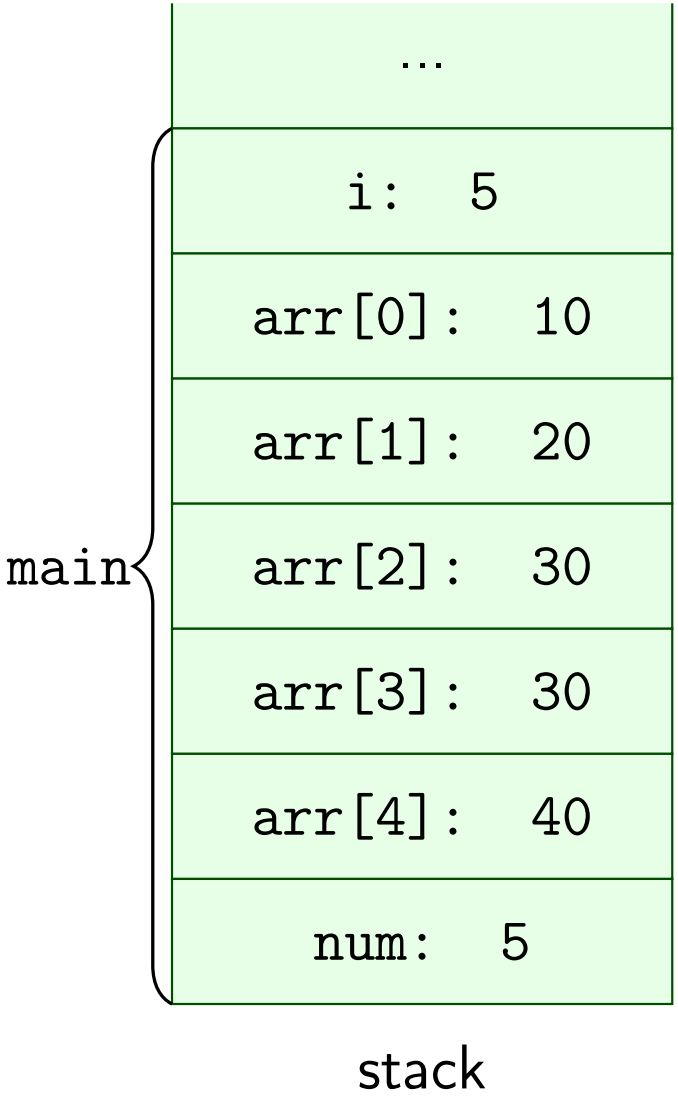


Kerülendő megoldás

```
#include <stdlib.h>

int getNum() { ... }

int main()
{
  int num = getNum();
  int arr[num];

  if (arr == NULL)
    return 1;

  for (int i = 0; i < num; ++i)
    arr[i] = (i + 1) * 10;
}
```

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

Kerülendő: VLA

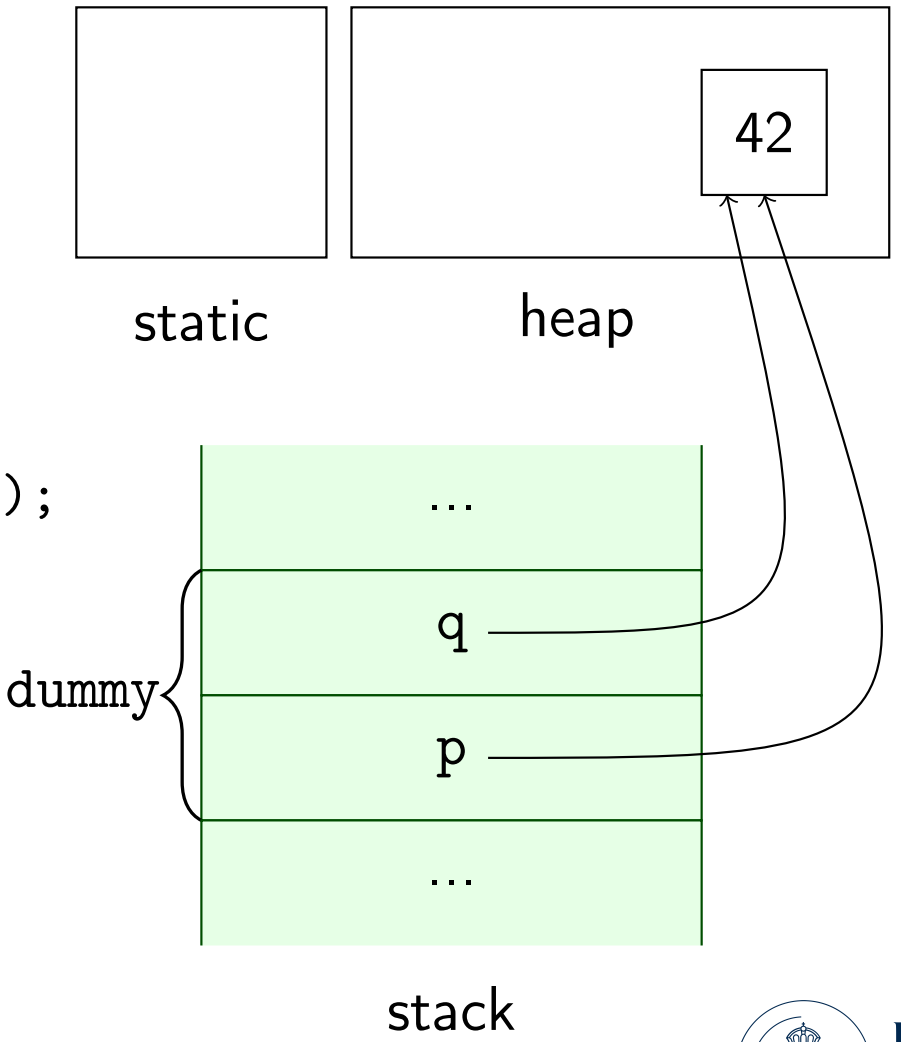


Láncolt adatszerkezet

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

Aliasing

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

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



Felszabadítás

Minden dinamikusan létrehozott változót pontosan egyszer!

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

Hivatkozás felszabadított változóra

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

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

Többszörösen felszabadított változó

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

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

Tulajdonos?

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

```
int global = 42;
int* make_ptr() {
  return &global;
}
```

```
int main()
{
  int* ptr = make_ptr();
  printf("%d\n", *ptr);
  free(ptr);   /* fel kell szabadítani? */
}
```