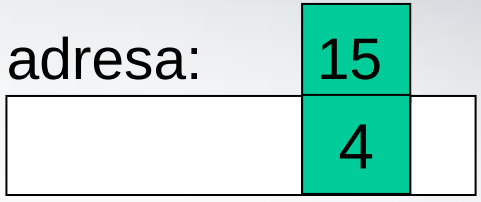
Algoritmizácia a programovanie: Let’s talk about s.. smerníky ☺

Ján Grman

Premenná

```
#include <stdio.h>

int main() {
   int i;
   i = 4;
}
```

dátová oblasť

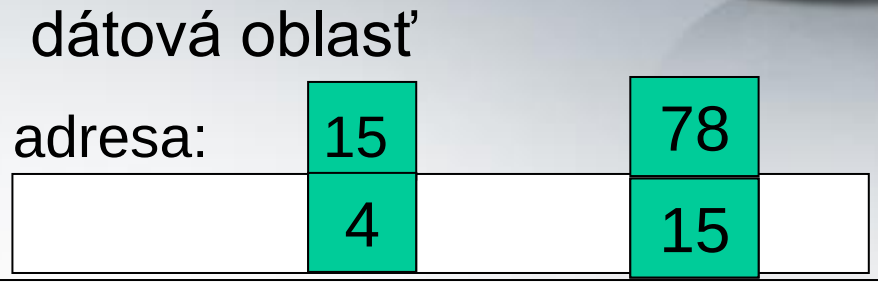


premenná **i** sa v pamäti vytvorila na adrese 15 s nejakou hodnotou

premenná **i** dostala hodnotu 4

Premenná a pointer na ňu (alebo pointrové ňuňuňu)

```
#include <stdio.h>

int main() {
  int i;

  int *p;

  p = &i;
  i = 4;


  printf("%d,%d,%d",
   i, p, *p);
}
```



premenná **i** sa v pamäti vytvorila na adrese 15 s nejakou hodnotou

premenná **p** sa v pamäti vytvorila na adrese 78 s nejakou hodnotou

premenná **p** bola naplnená ADRESOU premennej i (ukazuje teda na rovnakú pamäť ako prezentuje **i).Jéeej POINTER.**

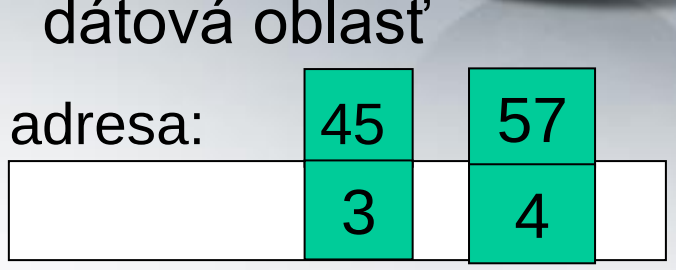
No, koľko to ukáže?

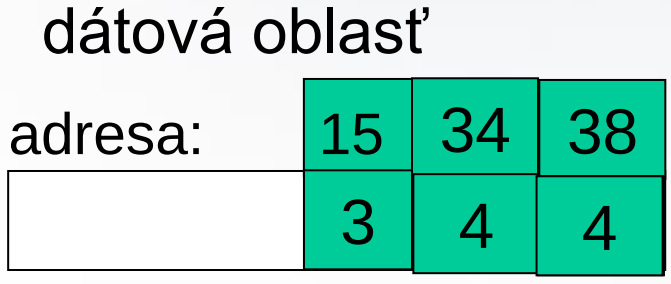
4,15,4 (premenná, jej adresa a hodnota na tej adrese = presne rovnaké miesto a hodnota

Premenná ako parameter hodnotou

```
int max(int a, int b)
{
   return (a > b ? a : b);
}
```

```
int main {
 int x = 3, y = 4, z;
 z = max(x,y);
}
```





Spustí sa main a vyhradí premenné **x,y,z**. **x** a **y** naplní

Spustí sa max a vyhradí premenné **a,b** a naplní ich (sú to kópie hodnôt **x** a **y**)

Return vyhodnotený výraz vráti a priradí do **z**

Premenná ako parameter odkazom

```
int max(int *a, int *b)
{
   return (*a > *b ? *a : *b);
}
```

```
int main {
 int x = 3, y = 4, z;
 z = max(&x,&y);
}
```

dátová oblasť

adresa:	45	57
	15	34

dátová oblasť

adresa:	15	34	38
	3	4	4

Spustí sa main a vyhradí premenné **x,y,z**. **x** a **y** naplní

Spustí sa max a vyhradí premenné **a,b** a naplní ich (sú to ADRESY **x** a **y**)

Return vyhodnotený výraz vráti a priradí do **z**

Premenná ako výstupný parameter

```
void max(int *a, int *b, int *c)
{
   *c = (*a > *b ? *a : *b);
}
```

dátová oblasť

adresa:	45	57	77
	15	34	38

```
int main {
 int x = 3, y = 4, z;
 max(&x,&y,&z);
}
```

dátová oblasť

adresa:	15	34	38
	3	4	4

Spustí sa main a vyhradí premenné **x,y,z**. **x** a **y** naplní

Spustí sa max a vyhradí premenné **a,b,c** a naplní ich (sú to ADRESY **x,y** a **z**)

výraz použije adresy – ukazujú na hodnoty 3 a 4 a číslo 4 sa priradí kam ukazuje **c**

Premenná ako výstupný parameter a mierne inak

```
void max(int a, int b, int *c)
{
   *c = (a > b ? a : b);
}
```

dátová oblasť

adresa:	45	57	77
	3	4	38

```
int main {
 int x = 3, y = 4, z;
 max(x,y,&z);
}
```

dátová oblasť

adresa:	15	34	38
	3	4	4

Spustí sa main a vyhradí premenné **x,y,z**. **x** a **y** naplní

Spustí sa max a vyhradí premenné **a,b,c** a naplní ich (**a,b** sú kópie **x,y** a **c** je ADRESA premennej **z**)

výraz použije kópie hodnôt 3 a 4 a číslo 4 sa priradí kam ukazuje **c**

```c
#include <stdio.h>

#define PI 3.14
#define na_druhu(i) ((i) * (i))

void kruh(int r, float *o, float *s)
{
   *o = 2 * PI * r;
   *s = PI * na_druhu(r);
}

int main()
{
   int polomer;
   float obvod, obsah;

   printf("Zadaj polomer kruhu: ");
   scanf("%d", &polomer);

   kruh(polomer, &obvod, &obsah);
   printf("obvod: %.2f, obsah: %.2f\n", obvod, obsah);
   return 0;
}
```

Funkcia vypočíta obvod a obsah kruhu vo funkcii `kruh()`. Volanie odkazom.

Jednorozmerné polia a smerníky

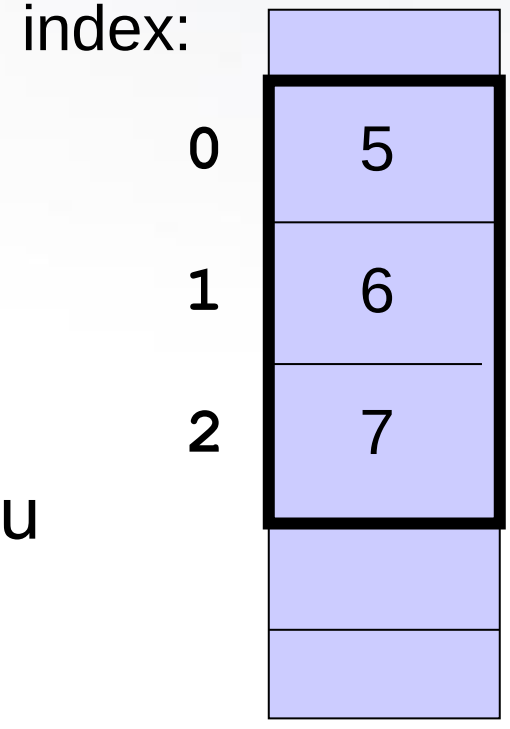
Základy práce s poliami

- pole je štruktúra zložená z niekoľkých prvkov rovnakého typu (blok prvkov)

```
TYP x[N];
```

statická definícia poľa

N: 3
TYP: **int**



- pole obsahuje **N** prvkov
- dolná hranica je vždy **0**
  ⇒ horná hranica je **N-1**
- číslo **N** musí byť známe v čase prekladu
- hodnoty nie sú inicializované na 0

Prístup k prvkom poľa

```
#define N 10

...
   int x[N], i;

   x[0] = 1;

   for (i = 0; i < N; i++)
      x[i] = i+1;

   for (i = 0; i < N; i++)
      pritnf("x[%d]: %d\n", i, x[i]);
```

priradenie hodnoty do prvého prvku poľa

v cykle priradenie hodnoty postupne všetkým prvkom poľa

výpis prvkov poľa

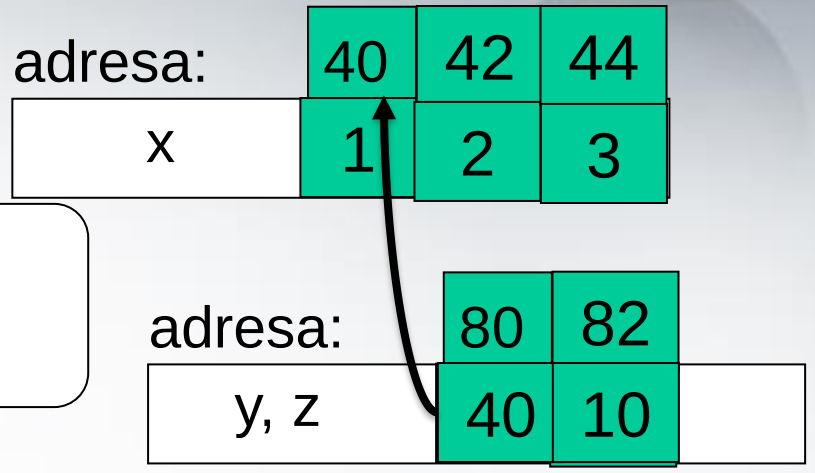
Dynamické pole

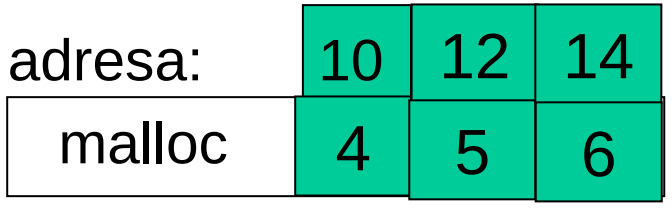
```
int x[3] = { 1, 2, 3 };

int *y;
int *z;
...


y=x;


// teraz sa y[1] rovná x[1]
// alebo *(y+2) sa rovná x[2]

z=(int*)malloc(3*sizeof(int));
z[0]=4; z[1]=5; z[2]=6;
```

Statické pole

Smerníky na **int**
Čo znamenajú? NIČ!

Teraz už **y** ukazuje presne na **x** (x je pole, teda x je smerník, len x[i] je int)





Pole ako parameter funkcie

- identifikátor nasledovaný zátvorkami:

```
int pole[]
```

```
int maximum(int p
{
    int i, max =
    for (i = 1; i
        if (pole[
            max =
    }
    return max;
}
```

```
int maximum(int pole[], int n)
{
    int i, max = *pole;
    for (i = 1; i < n; i++) {
        if (*(pole+i) > max)
            max = *(pole+i);
    }
    return max;
}
```

Pole ako parameter funkcie inak

- pole ako adresa:

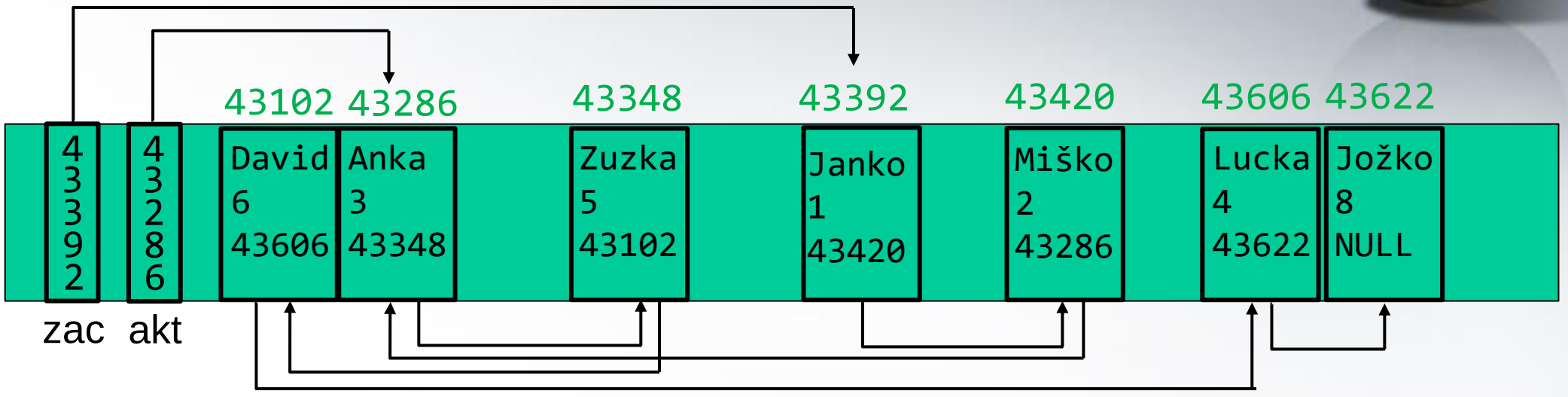
```
int *pole
```

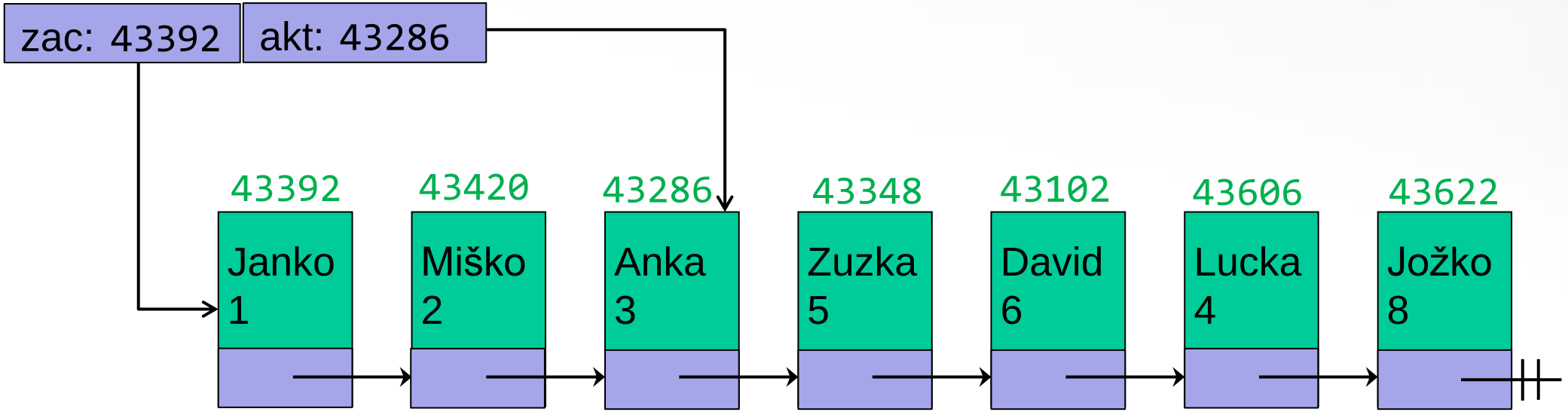
```
int maximum(int *pole, int n)
{
     int i, max = *pole;
     for (i = 1; i < n; i++) {
          if (*(pole+i) > max)
               max = *(pole+i);
     }
     return max;
}
```

```
int maximum(int *
{
     int i, max =
     for (i = 1; i
          if (pole
               max =
     }
     return max;
}
```

Spájané zoznamy

Spájaný zoznam: záznamy v pamäti





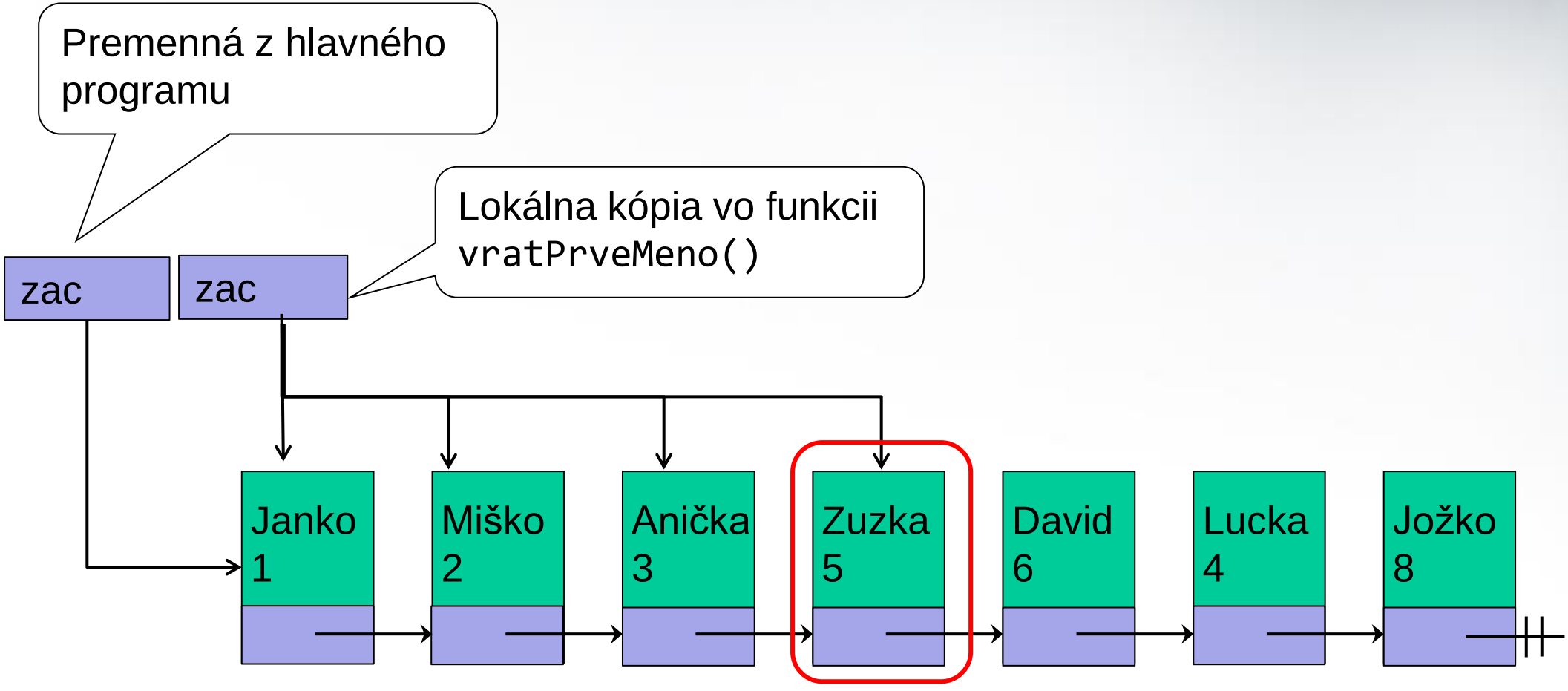
Vrátenie ukazovateľa na prvý záznam spĺňajúci podmienku

- Úlohafunkcia `vratPrveMeno()`: vráti ukazovateľ na prvý záznam s daným menom

```
CLOVEK *vratPrveMeno(CLOVEK *zac, char meno[]) {

    while(zac != NULL) {
        if(!strcmp(zac->meno, meno))
            return zac;
        zac = zac->dalsi;
    }
    return NULL;
}
```

Vrátenie ukazovateľa na prvý záznam spĺňajúci podmienku

- Ak nájdeme hľadaný prvok – vrátime `akt`, inak `NULL`



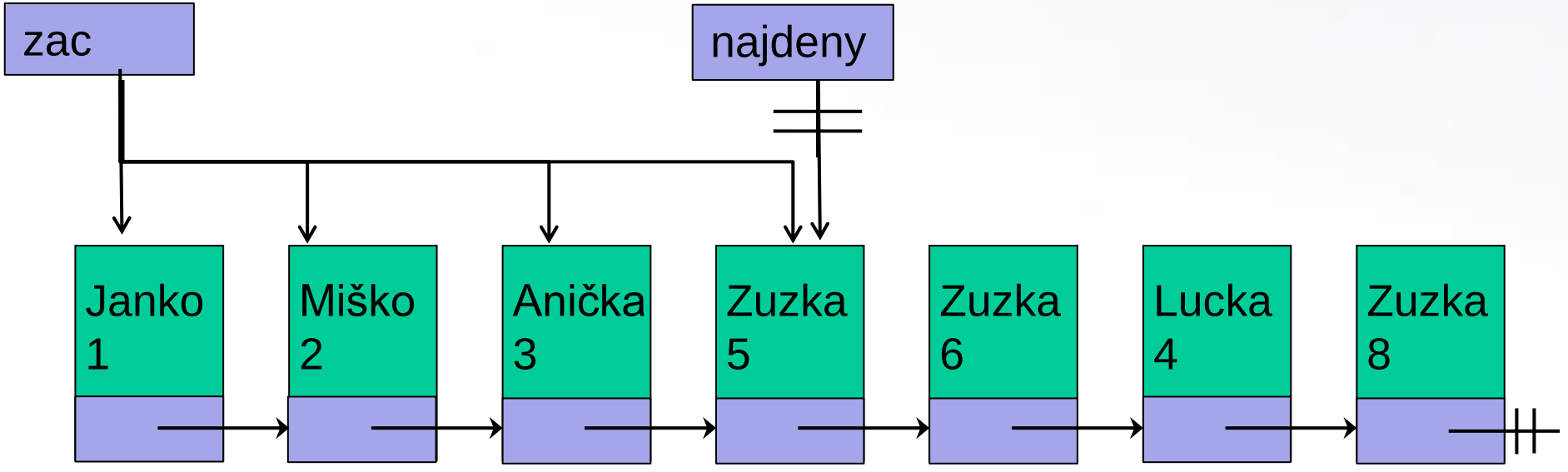
Vrátenie ukazovateľa na posledný záznam spĺňajúci podmienku

- funkcia `vratPosledneMeno()`: vráti ukazovateľ na prvý záznam s daným menom

```
CLOVEK *vratPosledneMeno(CLOVEK *zac, char meno[]) {
    CLOVEK *najdeny = NULL;

    while(zac != NULL) {
        if(!strcmp(zac->meno, meno))
            najdeny = zac;
        zac = zac->dalsi;
    }
    return najdeny;
}
```

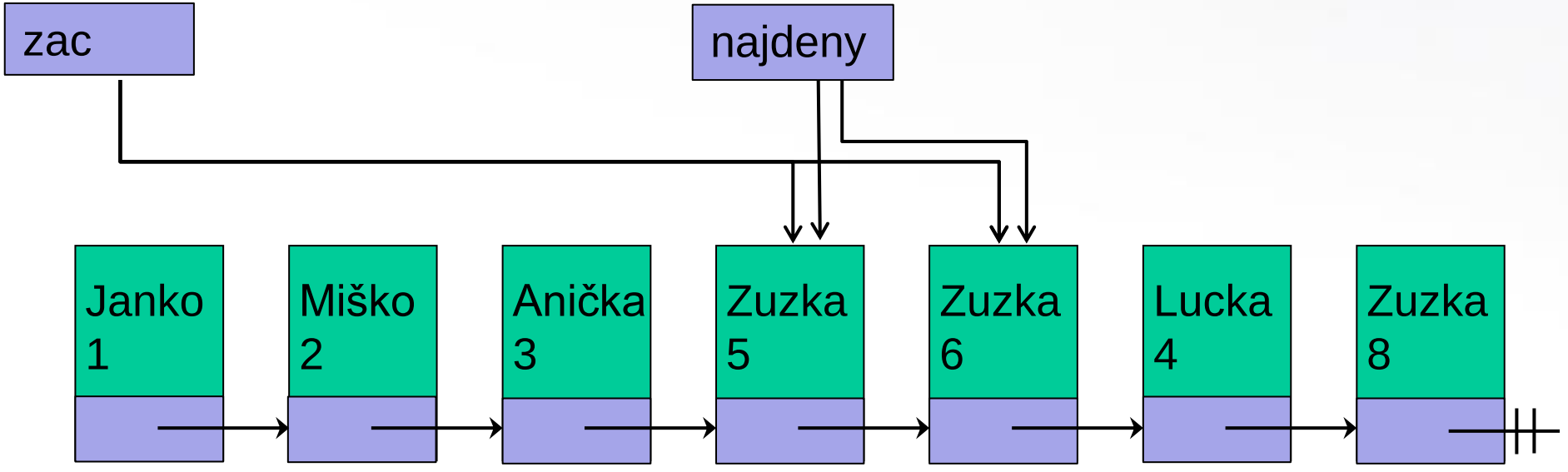
Vrátenie ukazovateľa na posledný záznam spĺňajúci podmienku

- Pomocný ukazovateľ `najdeny`:
  - Bude ukazovať na posledný doteraz záznam spĺňajúci podmienku



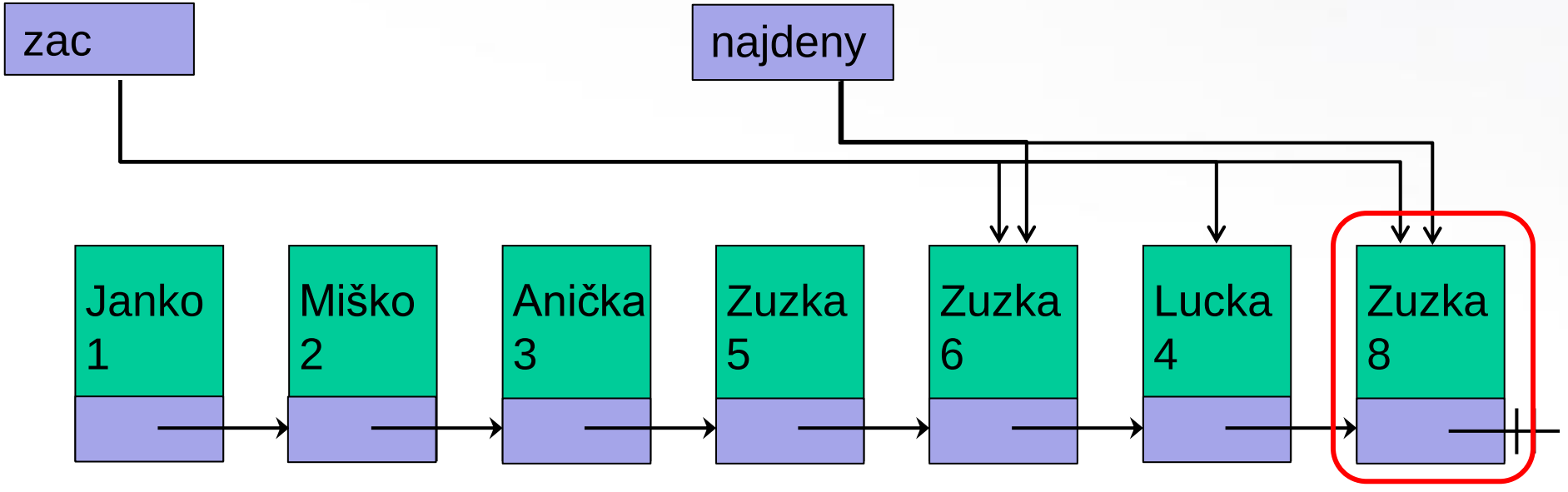
Vrátenie ukazovateľa na posledný záznam spĺňajúci podmienku

- Pomocný ukazovateľ `najdeny`:
  - Bude ukazovať na posledný doteraz záznam spĺňajúci podmienku



Vrátenie ukazovateľa na posledný záznam spĺňajúci podmienku

- Pomocný ukazovateľ `najdeny`:
  - Bude ukazovať na posledný doteraz záznam spĺňajúci podmienku



Pridanie prvku na pozíciu

```
CLOVEK *vlozNaPoziciu(CLOVEK *zac, CLOVEK *vloz, int p){
   CLOVEK *akt = zac;
   int i = 1;

   if (zac == NULL) return vloz;
   if (vloz == NULL) return zac;
   if (p == 1) {
      vloz->dalsi = zac;
      return vloz;
   }
   while (akt->dalsi != NULL && i < p-1) {
      akt = akt->dalsi;
      i++;
   }
   vloz->dalsi = akt->dalsi;
   akt->dalsi = vloz;
   return zac;
}
```

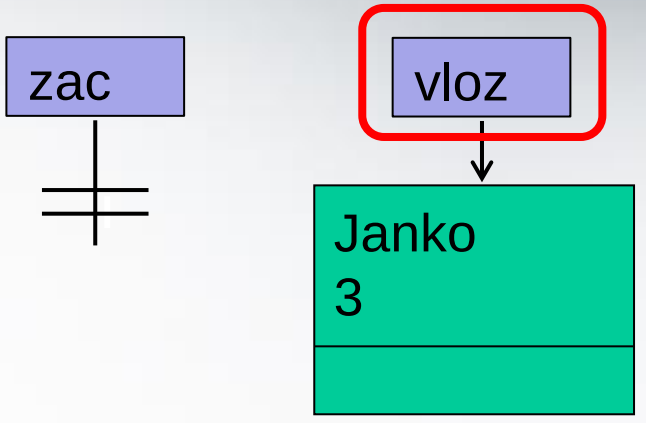
prázdny zoznam

nie je čo vkladať

Pridanie prvku na pozíciu

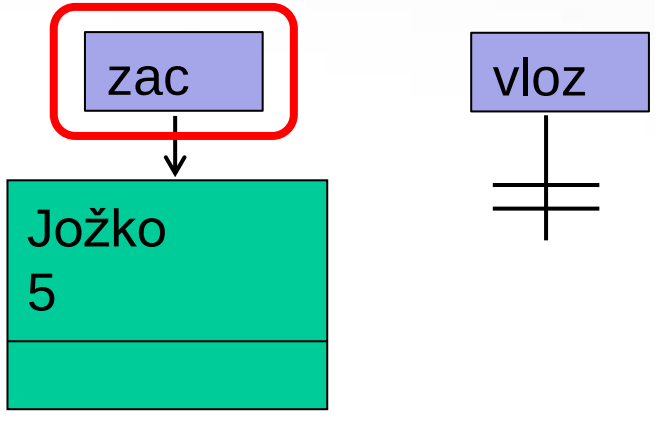
- Pridanie prvku do prázdneho zoznamu

```
if (zac == NULL) return vloz;
```



- Pridanie prázdneho prvku

```
if (vloz == NULL) return zac;
```



Pridanie prvku na pozíciu

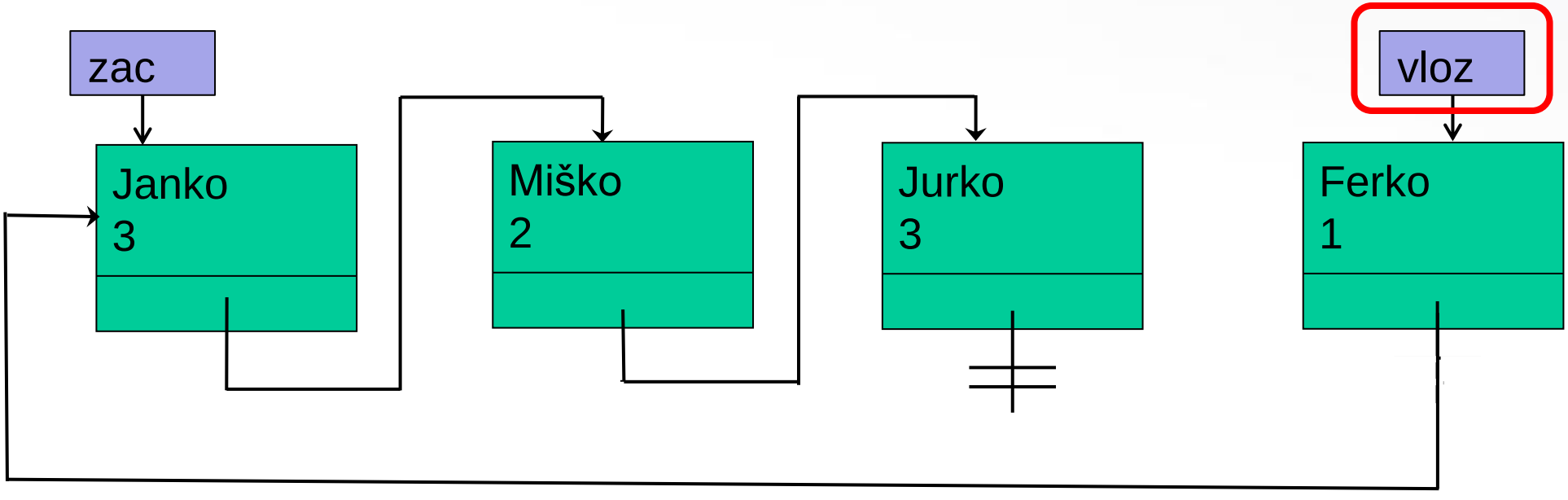
```
CLOVEK *vlozNaPoziciu(CLOVEK *zac, CLOVEK *vloz, int p){
   CLOVEK *akt = zac;
   int i = 1;

   if (zac == NULL) return vloz;
   if (vloz == NULL) return zac;
   if (p == 1) {
      vloz->dalsi = zac;
      return vloz;
   }
   while (akt->dalsi != NULL && i < p-1) {
      akt = akt->dalsi;
      i++;
   }
   vloz->dalsi = akt->dalsi;
   akt->dalsi = vloz;
   return zac;
}
```

vloženie na začiatok zoznamu

Pridanie prvku na pozíciu

- Pridanie na začiatok zoznamu

```
if(p == 1) {
   vloz->dalsi = zac;
   return vloz;
}
```



Pridanie prvku na pozíciu

```
CLOVEK *vlozNaPoziciu(CLOVEK *zac, CLOVEK *vloz, int p){
   CLOVEK *akt = zac;
   int i = 1;

   if (zac == NULL) return vloz;
   if (vloz == NULL) return zac;
   if (p == 1) {
      vloz->dalsi = zac;
      return vloz;
   }
   while (akt->dalsi != NULL && i < p-1) {
      akt = akt->dalsi;
      i++;
   }
   vloz->dalsi = akt->dalsi;
   akt->dalsi = vloz;
   return zac;
}
```

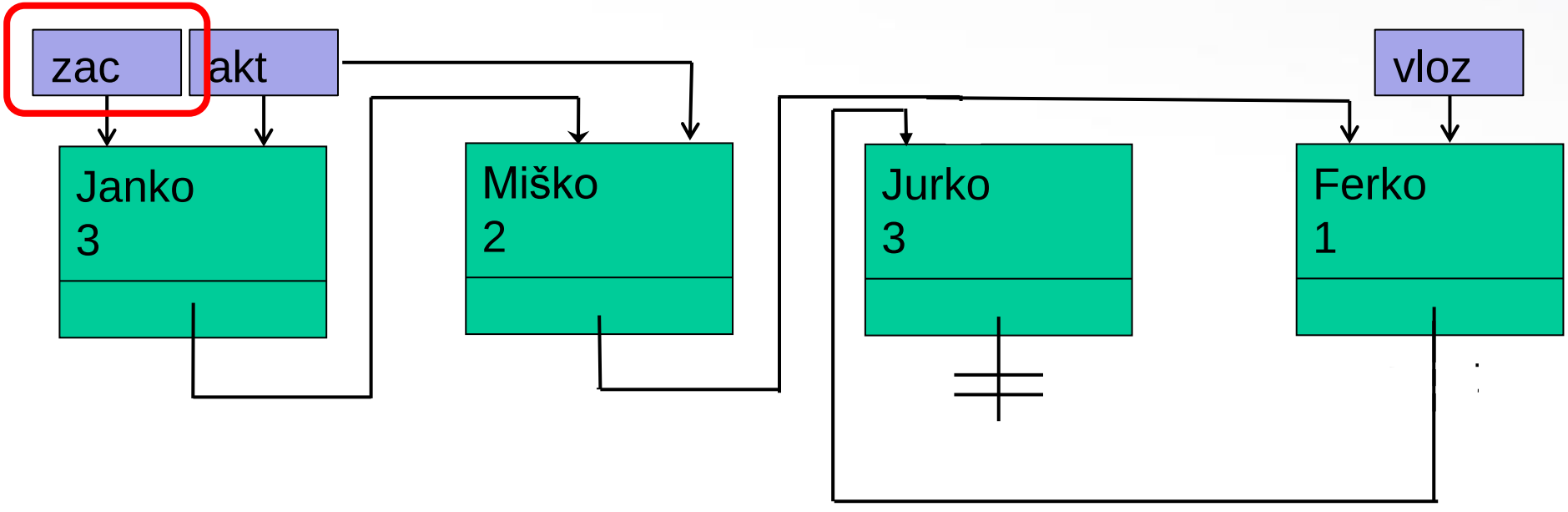
vloženie do zoznamu
(aj na koniec zoznamu)

Pridanie prvku na pozíciu

- Pridanie do stredu alebo zoznamu: p: 3

```
CLOVEK *akt = zac;
int i = 1;
while(akt->dalsi != NULL && i < p-1) {
    akt = akt->dalsi;
    i++;
}
vloz->dalsi = akt->dalsi;
akt->dalsi = vloz;
return zac;
```

i: 2



Toto je koniec