

Polia, pretypovanie, ukazovatele

Základy procedurálneho programovania 1, 2020

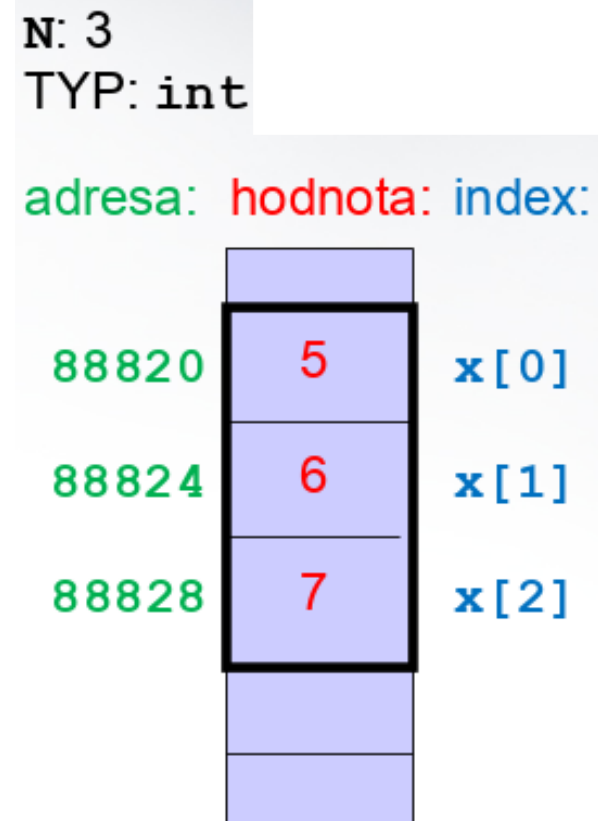
Ing. Marek Galinski, PhD.

Jednorozmerné polia

Základy práce s poliami

- Pole – štruktúra zložená z niekoľkých prvkov rovnakého typu
- Pri statickom poli veľkosť pola musí byť známa v čase prekladu
- Hodnoty nie sú inicializované

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



Definícia statického poľa

```
#define N 10  
  
int x[N], y[N+1], z[N*2];
```

x má 10 prvkov poľa, od indexu 0 po index 9

y má 11 prvkov poľa, od indexu 0 po index 10

z má 20 prvkov poľa, od indexu 0 po index 19

Zistenie veľkosti poľa

```
int i, n;  
int pole[] = { 3, 6, 9, 12, 15 };  
n = sizeof(pole)/sizeof(int);
```

- **Nikdy nepristupujte k prvkom poľa mimo stanovené hranice.**
- Program môže spadnúť, môže dávať nesprávne výsledky

Pole ako parameter funkcie

- Funkcii sa odovzdáva v parametri iba adresa poľa, nie veľkosť
- Veľkosť preto treba oznámiť samostatným parametrom

```
int maximum(int pole[], int n)
```

Zmena prvkov poľa vo funkcii

- Pole sa prenáša prostredníctvom adresy, nevytvára sa lokálna kópia
- Zmeny hodnôt prvkov sa prejavia aj mimo funkcie, kde sa zmena udiala

Príkaz switch

Príkaz switch

- **Výraz, podľa ktorého sa rozhoduje**
- Dátový typ výrazu musí byť typu int (alebo char?)
- Každá vetva sa ukončuje príkazom break
- V každej vetve môže byť viac príkazov, zátvorky nie sú nutné
- Vetva default – vtedy, ak sa nevykonala žiadna iná.
 - Nemusí byť na konci, ale je to zvyk
- Break ukončuje vykonávanie switch bloku
 - Poroz na switch v cykle a cyklus v bloku switch

Príkaz switch

- Výraz, podľa ktorého sa rozhoduje

```
switch (vyraz) {  
    case hodnota_1 : prikaz_1; break;  
    ...  
    case hodnota_n : prikaz_n; break;  
    default : prikaz_def; break;  
}
```

Príkaz switch

- Viacero hodnôt – tie isté príkazy

```
switch (vyraz) {  
    case h_1 :  
    case h_2 :  
    case h_3 : prikaz_123; break;  
    case h_4 : prikaz_4; break;  
    default : prikaz_def; break;  
}
```

Typová konverzia

Typová konverzia

- **Prevod premennej určitého typu na iný typ**
 - Napr. int na double
- **Dva druhy typových konverzií**
 - Implicitná – deje sa sama od seba, automaticky
 - Explicitná – vynútená, požadovaná

Implicitná konverzia

- Pred vykonaním operácie sa jednotlivé operandy konvertujú
 - Napr. char alebo short int sa konvertujú na int
- Ak majú dva operandy jednej operácie rôzny typ, nižšia priorita je konvertovaná na vyššiu, nasledovne:

```
int      ⇒ unsigned int
unsigned int ⇒ long
long     ⇒ unsigned long
unsigned long ⇒ float
float    ⇒ double
double  ⇒ long double
```

Implicitná konverzia

- Pri operácii priradenia sa pravá strana prispôsobí ľavej strane

```
double x;  
  
x = 5;
```

```
int i;  
  
i = 'A';
```

```
int i;  
  
i = 5.0;
```

Explicitná konverzia

- Jazyk C dovoľuje takmer ľubovoľnú konverziu, nie vždy to však má zmysel, nemusí to byť vhodné

<code>(int) char_vyraz</code>	- prevod znaku na ordinálne číslo
<code>(char) int_vyraz</code>	- prevod ordinálneho čísla na znak
<code>(int) double_vyraz</code>	- odrezanie desatinnej časti
<code>(double) int_vyraz</code>	- prevod celého čísla na reálne
<code>(double) float_vyraz</code>	- zväčšenie presnosti

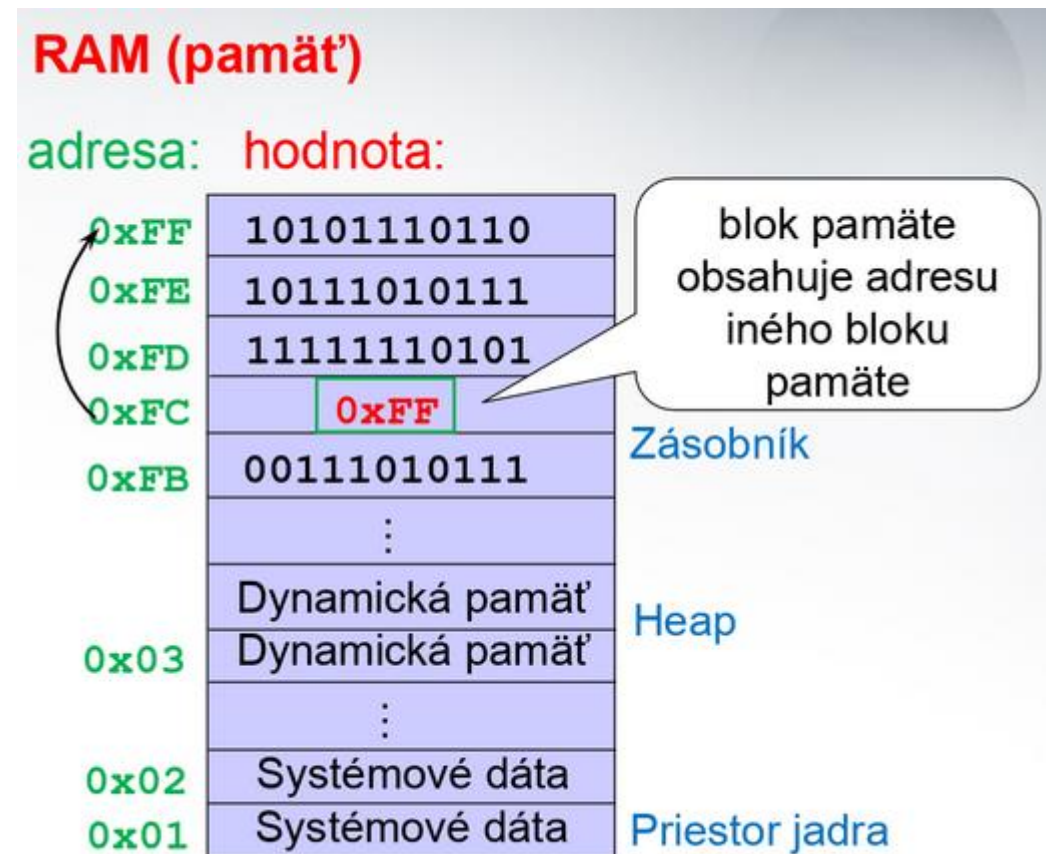
Ukazovatele

Ukazovatele

- **Ukazovatele = pointer, smerníky**
- Ukazovateľ je premenná, jeho hodnota je adresa v pamäti
- Objekt typu ukazovateľ obsahuje informáciu o tom, kde je umiestnený iný údajový objekt
 - Ukazovateľ na premennú
 - Ukazovateľ na funkciu
 - Ukazovateľ na štruktúru

Ukazovatele

- Ukazovatele = ukazujú na špecifickú adresu v pamäti
- Program môže skočiť na túto adresu v pamäti a prečítať z nej hodnotu
- V pamäti môžeme uchovávať aj inú adresu



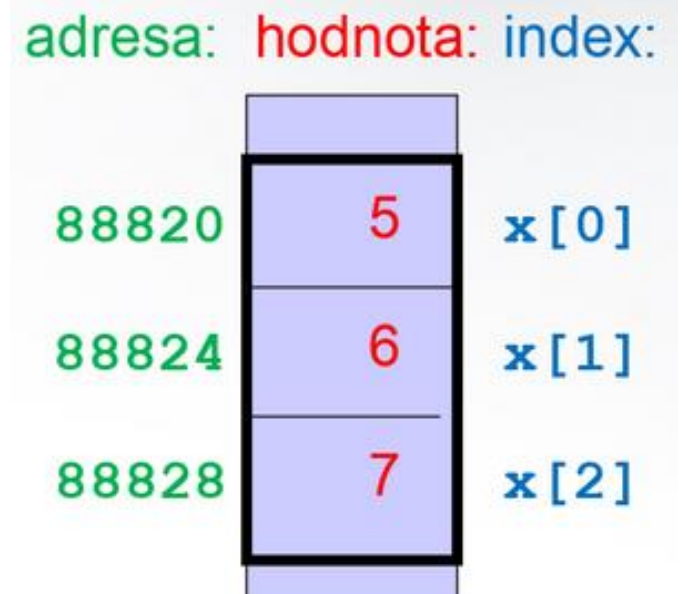
Ukazovatele

- Už sme sa s tým trochu stretli
- Ukazovateľ – ukazovateľ na súbor

```
FILE *f
```

- Adresa – načítanie hodnoty do premennej

```
scanf ("%d", &x)
```



Ukazovatele

- Ukazovateľ je definovaný pomocou *
- Klasická celočíselná premenná `int i`
- Ukazovateľ na celočíselnú premennú `int *p_i`

- Definícia

```
int i;
int *p_i;
```

```
int i, *p_i;
```

```
int *p1, p2, p3;
```

```
int *p1, *p2, *p3;
```

Ukazovatele

- Definícia `int *p;`
- Zatiaľ je však ukazovateľ nepoužiteľný – má iba vyhradené miesto, nemá priradenú adresu.
- Ukazuje na náhodné miesto v pamäti, to nikdy nedopadne dobre.
 - Prístup do pamäte, kde program nemá povolené pristupovať

Ukazovatele

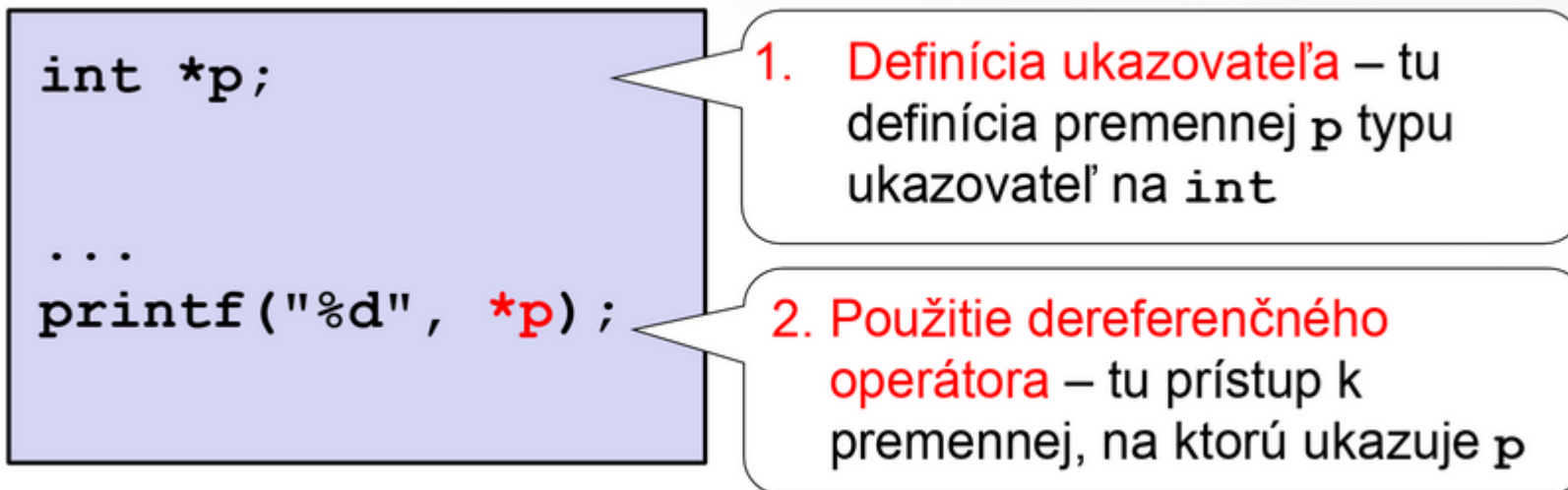
- Priradenie hodnoty do ukazovateľa – referenčný operator &
- Úlohou ukazovateľa je ukazovať na nejakú premennú
- Každá premenná je na nejakej adrese
- Adresu musíme poznať, ak ju chceme nastaviť do ukazovateľa
- Referenčný operátor

```
p = &i;
```

```
int i, *p_i;  
p_i = &i;
```

Ukazovatele

- Sprístupnenie hodnoty, kam ukazovateľ ukazuje
– dereferenčný operátor *
- Hviezdička má dva významy



```
int *p;

...
printf("%d", *p);
```

1. **Definícia ukazovateľa** – tu definícia premennej `p` typu ukazovateľ na `int`
2. **Použitie dereferenčného operátora** – tu prístup k premennej, na ktorú ukazuje `p`

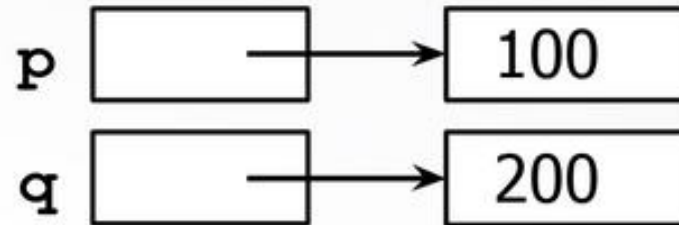
Základy práce s ukazovateľmi

<code>p_i = &i;</code>	- správne
<code>p_i = &(i + 3);</code>	- chyba: <code>(i + 3)</code> nie je premenná
<code>p_i = &15;</code>	- chyba: konštanta nemá adresu
<code>p_i = 15;</code>	- chyba: priradovanie absolútnej adresy
<code>i = p_i;</code>	- chyba: priradovanie adresy
<code>i = &p_i;</code>	- chyba: priradovanie adresy
<code>*p_i = 4;</code>	- správne, ak <code>p_i</code> bol inicializovaný

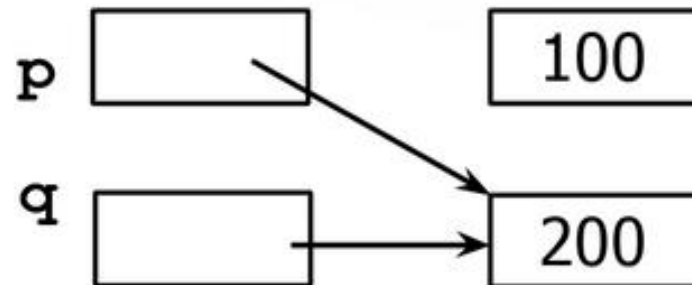
Základy práce s ukazovateli

Príkaz: `p = q;` kde p , q sú smerníky rovnakého typu

pred priradením:



po priradení:

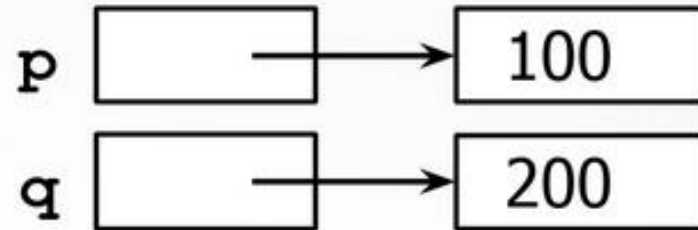


p ukazuje na rovnakú premennú ako q

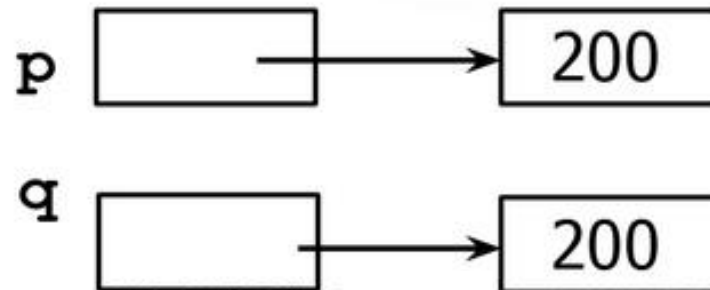
Základy práce s ukazovateli

Príkaz: `*p = *q;` kde p , q sú smerníky rovnakého typu

pred priradením:



po priradení:



Footnotes

- Prednáška je dostupná na YouTube:
https://www.youtube.com/watch?v=C4Dg767US_g

V prednáške boli použité materiály zo slidov prednášok ZPrPr1 od Gabriely Grmanovej.

Q?