

Ukazovatele – pokračovanie Reťazce

Základy procedurálneho programovania 1, 2020

Ing. Marek Galinski, PhD.

Opakovanie

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] ;
```

N: 3
TYP: int

adresa: hodnota: index:

88820	5	x[0]
88824	6	x[1]
88828	7	x[2]

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

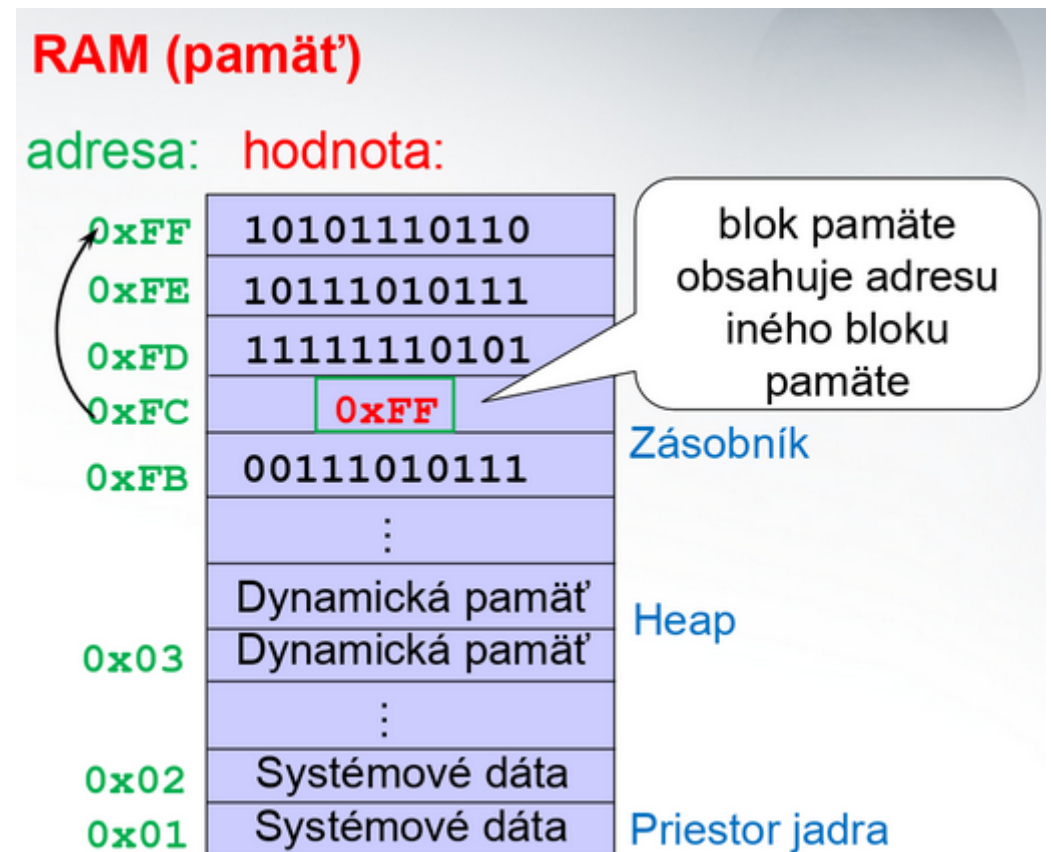
```
int maximum(int pole[], int n)
```

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

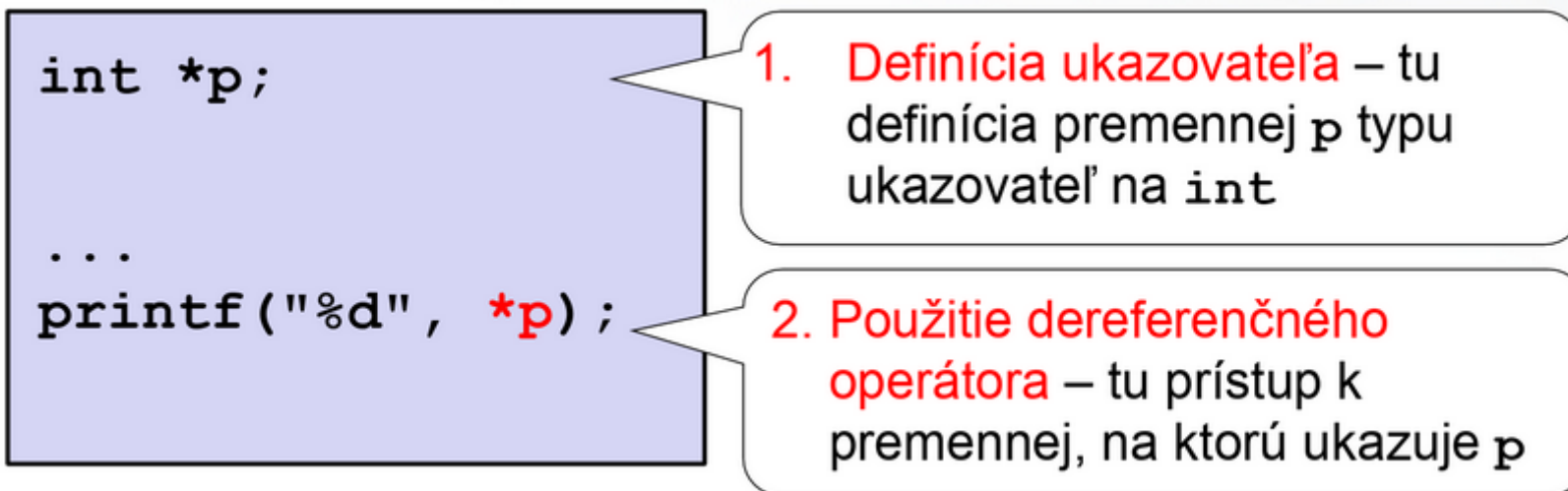
Ukazovatele

- **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

- 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`

Ukazovatele - pokračovanie

Typ ukazovateľa

- **Všetky ukazovatele majú rovnakú veľkosť – adresa v pamäti**
- Ak nasmerujeme ukazovateľ na nejakú adresu, potrebujeme vedieť, koľko bytov chceme čítať, alebo do koľkých bytov chceme zapisovať

Konverzia ukazovateľov

- **Ked' netreba, nepoužívať!**
- Ked' treba, tak potom explicitne pretypovať

```
int *p_i;  
char *p_c;
```

```
p_c = p_i;
```

```
p_c = (char *)p_i;
```

Ukazovateľ typu void

- Keď nevieme dopredu, na aký dátový typ bude ukazovateľ ukazovať
- Pred použitím je nevyhnutné explicitne pretypovať

```
int i;  
float f;  
void *p_void = &i;
```

```
*(float *) p_void = 3.5;
```

Ukazovateľ nikam - NULL

- Nulový ukazovateľ, konštanta NULL (stdio.h)

```
#define NULL 0  
#define NULL ((void *) 0)
```

- Je možné použiť pre akýkoľvek typ ukazovateľa

```
if (p_i == NULL)  
    ...
```

Ukazovateľ a statické pole

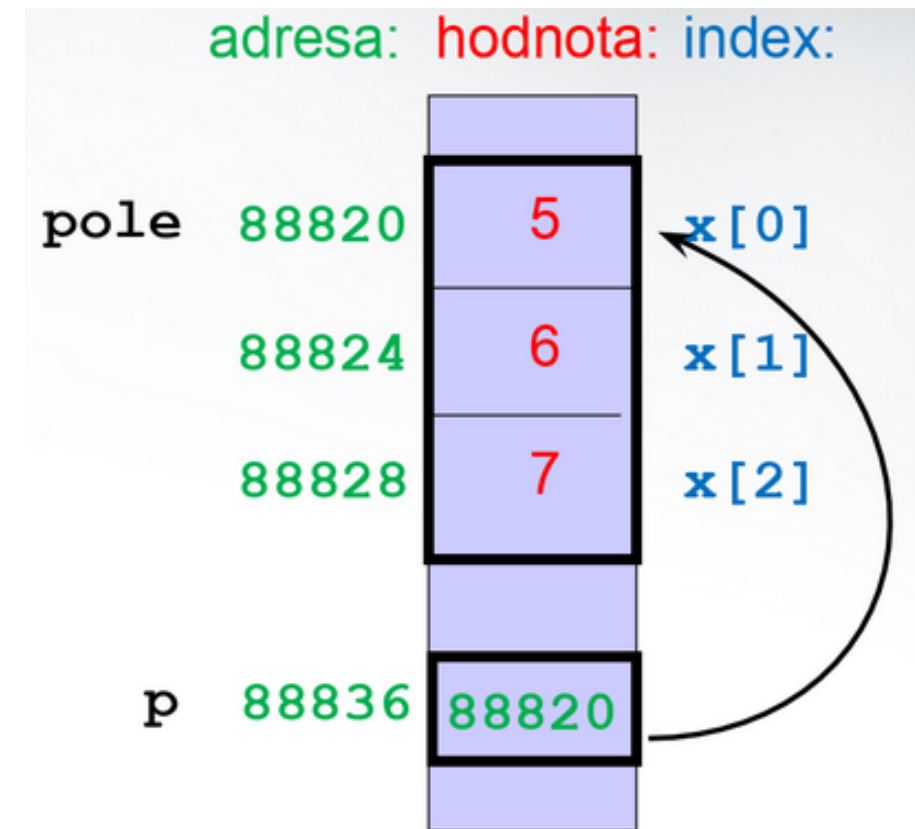
- Definícia poľa

```
int pole[N];
```

- Premenná pole je statický ukazovateľ – ukazuje na začiatok poľa.
- K prvkom poľa je možné pristupovať aj pomocným ukazovateľom

```
int pole[N];
int *p;

p = pole;
```



Aritmetika s ukazovateľmi

Ukazovateľová aritmetika

- **S ukazovateľmi sa dajú robiť niektoré aritmetické operácie**
- Súčet / Rozdiel ukazovateľa a celého čísla
- Porovnávanie ukazovateľov rovnakého typu
- Rozdiel ukazovateľov rovnakého typu
- Má to zmysel iba pri poliach, inak nie

Operátor sizeof

- Zisťuje veľkosť dátového typu v bytoch, vyhodnocuje sa v čase prekladu programu

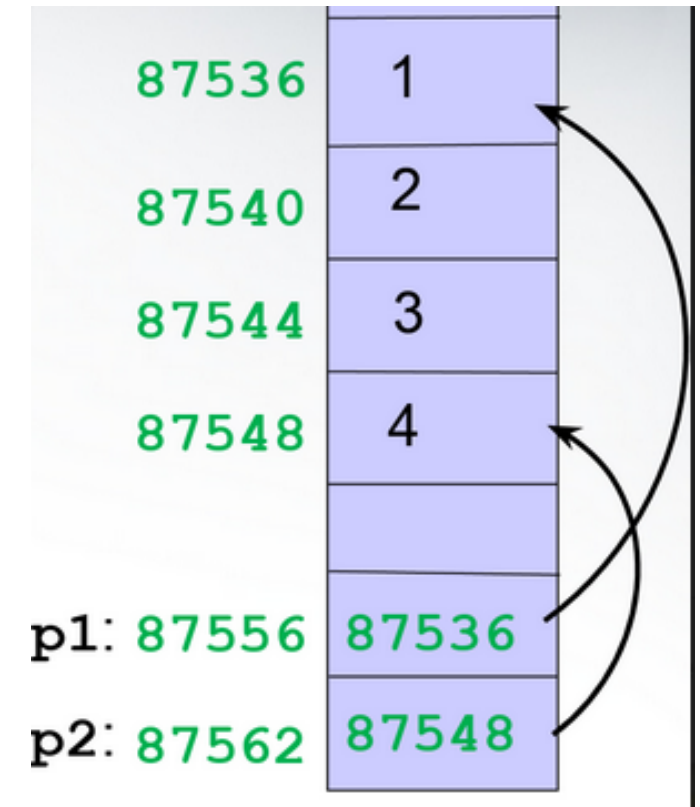
```
int i, *p_i;  
i = sizeof(p_i);
```

```
int i, *p_i;  
i = sizeof(*p_i);
```

Súčet ukazovateľa a celého čísla

```
int n, *p1, *p2;    n=3
...
p2 = p1 + n;
```

```
p2 = (int *) p1 + sizeof(*p1)*n;
```



Súčet ukazovateľa a celého čísla

```
char *p_c;  
int *p_i;  
double *p_d;
```

```
sizeof(char) == 1  
sizeof(int) == 4  
sizeof(double) == 8
```

Ak `p_c` obsahuje adresu 10, `p_c + 1 == 11`
 `p_i` obsahuje adresu 20, `p_i + 1 == 24`
 `p_d` obsahuje adresu 30, potom platí: `p_d + 1 == 38`

Porovnávanie ukazovateľov

- Porovnávacie operátory `<` `<=` `>` `>=` `==` `!=`
- Porovnávanie má zmysel iba keď sú ukazovatele rovnakého typu
- Výsledok porovnania
 - 1, ak podmienka platí
 - 0, ak podmienka neplatí
- **Pozor na to, či sa pohybujeme v tom istom úseku pamäte**

Dynamické polia - základy

Dynamické polia - základy

- Využívanie ukazovateľov a správy pamäte

```
ptr = (float*) malloc(100 * sizeof(float));
```

- Ukazovateľ ptr ukazuje na prvú pozíciu v poli
- Pamäť sa dá uvoľniť pomocou free()

```
free(ptr);
```

Dynamické polia - základy

```
int main()
{
    int n, i, *ptr, sum = 0;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    ptr = (int*) malloc(n * sizeof(int));

    printf("Enter elements: ");
    for(i = 0; i < n; ++i)
    {
        scanf("%d", ptr + i);
        sum += *(ptr + i);
    }

    free(ptr);

    return 0;
}
```

```
ptr = realloc(ptr, x);
```

Ret'azce

Reťazce

- **Jednorozmerné polia typu char ukončené znakom ‘\0’**
- Dĺžka reťazca 0 – veľkosť pol'a-1
- Reťazec musí byť ukončený ukončovacím znakom
- Pole znakov nemusí byť ukončené ukončovacím znakom

Definícia a inicializácia reťazca

- Inicializácia – len v definícii
- Ukončovací znak implicitne

```
char s[6] = "ahoj";
```

- Nedá sa to pre “statické pole”

```
char s[10];  
s = "ahoj";
```

```
char s[10];  
...  
printf("%s", s);
```

```
char s[10];  
int i;  
  
for (i = 0; i < 10-1; i++)  
    s[i] = '*';  
s[10-1] = '\\0';
```

Funkcie pre prácu s reťazcom

- Nie sú súčasťou jazyka C samotného
- Sú definované v <string.h>
- Zistenie dĺžky reťazca `int strlen(char *s);`
- Kopírovanie reťazca znakov `char *strcpy(char *s1, char *s2);`
- Spojenie reťazcov `char *strcat(char *s1, char *s2);`
- Porovnávanie reťazcov `int strcmp(char *s1, char *s2);`
- Nájdenie prvého výskytu znaku `char *strchr(char *s, char c);`
- Nájdenie prvého výskytu reťazca `char *strstr(char *s1, char *s2);`

Footnotes

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

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

Q?