Procedurálne programovanie

slido.com
3916580
PrPr – P6

Ján Zelenka

Ústav Informatiky

Slovenská akadémia vied

Obsah prednášky

slido slido.com
3916580
PrPr – P6

1. **Opakovanie**
2. **Dvojrozmerné polia**
3. **Reťazce**

Spätná väzba: *https://forms.gle/6q5D2G6UwrtimXEx9*

Opakovanie

slido.com
3916580
PrPr – P6

Statická a dynamická alokácia

slido slido.com
3916580
PrPr – P6

Statická alokácia vymedzuje pamäť na zásobníku

- automaticky sa uvoľní po dokončení bloku kódu (koniec funkcie, cyklu...)
- Výrazne rýchlejšia
  - Na zásobníku nevzniká fragmentácia, ľahké uvoľnenie
  - Vrchol zásobníka je v cache
- Krátkodobé premenné

Dynamická alokácia vymedzuje pamäť na halde

- existuje do explicitného uvolnenia (alebo do skončenia programu)
- vymedzenie pamäte v halde (heap)
- za behu programu dynamicky prideliť (alokovať) oblasť pamäte určitej veľkosti
- pristupuje sa do nej prostredníctvom ukazovateľov

Opakovanie

Dynamická alokácia pamäte:

```
char *p_c;

p_c = (char *) malloc(1000 * sizeof(char));
...
free(p_c);
p_c = NULL;
```

- nepotrebnú pamäť je vhodné ihneď vrátiť operačnému systému

Ukazateľová aritmetika

slido.com
3916580
PrPr – P6

Ukazateľová aritmetika

- aritmetické operácie nad ukazovateľmi

  `pole[X]` je definované ako `*(pole + X)`

- realizované na úrovni prvkov
  - nie bytov
  - majme `pole` typu `int*`
    - pripočíta sa **`X * sizeof(int) bytov`**
- adresa začiatku pola je priradená do ukazovateľa

Ukazateľová aritmetika

- S ukazovateľmi je možné vykonávať nasledovné aritmetické operácie:
  - súčet ukazovateľa a celého čísla
  - rozdiel ukazovateľa a celého čísla
  - porovnávanie ukazovateľov rovnakého typu
  - rozdiel dvoch ukazovateľov rovnakého typu
- Majú zmysel len v rámci bloku dynamicky vytvorenej pamäte (POLIA)
- Ostatné operácie nedávajú zmysel
  - OS nezaručí, že neskôr alokovaný blok bude na vyššej adrese

Ukazateľová aritmetika
- Operátor sizeof

slido slido.com
3916580

Vieme:

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

```
   char c, *p_c=&c; //&c 10
    int i, *p_i=&i; //&i 20
double f, *p_f =&f;//&f 30
```

Potom platí:

```
p_c + 1 == 11
p_i + 1 == 24
p_f + 1 == 38
```

Polia a ukazovatele

```
int *p;
int A[10]={0};
p = (int *) malloc(4 * sizeof(int));
```

`p` je dynamické pole, ktoré vzniká v čase behu programu

A je konštantný smerník jeho hodnota sa nedá meniť, adresa začiatku bloku pamäti alokovaného pre statické pole

Platí:

- `A[0] == *A`
- `A[1] == *(A + 1)`
- `A[2] == *(A + 2)`
- `A[3] == *(A + 3)`

…

- `p[0] == *p`
- `p[1] == *(p + 1)`
- `p[2] == *(p + 2)`
- `p[3] == *(p + 3)`

…

Rozdiel medzi statickými a dynamickými poliami je v definícii a v spôsobe prideľovania pamäte.

Porovnávanie ukazovateľov s konštantou NULL

- bez explicitného pretypovávania
- **`p = NULL`**
  - neukazuje na žiadne zmysluplné miesto v pamäti

```
int n, *p;
...
if (n >= 0)
    p = alokuj(n);
else
   p = NULL;
...
if (p != NULL)
  ...
```

Polia a ukazovatele

slido.com
3916580
PrPr – P6

Veľkosť poľa

- uchovávať v (celočíselnej) premennej
  - pri odovzdaní poľa ako argument funkcie treba dve premenné: smerník na adresu začiatku poľa a veľkosť poľa
- Dynamické pole
  - **`sizeof(*pole)`** vráti koľko bytov treba na uloženie jedného prvku poľa (napr. 4 byty v prípade poľa celých čísiel).
  - **`sizeof(pole)`** vráti koľko bytov treba na uloženie samotného ukazovateľa `pole` -> to je 8 bytov (respektíve 4 byty na 32-bitovom OS).
- Statické pole
  - **`sizeof(pole)`** vráti veľkosť poľa v bytoch
  - po predaní poľa cez argument funkcie informácia o jeho veľkosti sa stráca

Viacrozmerné polia

slido.com
3916580
PrPr – P6

Dvojrozmerné pole

slido slido.com
3916580
PrPr – P6

- **statické dvojrozmerné pole**
- pole ukazovateľov
- ukazovateľ na pole
- ukazovateľ na ukazovateľ

Dvojrozmerné pole

Statické dvojrozmerné pole

```
int tabulka[5][4];
```

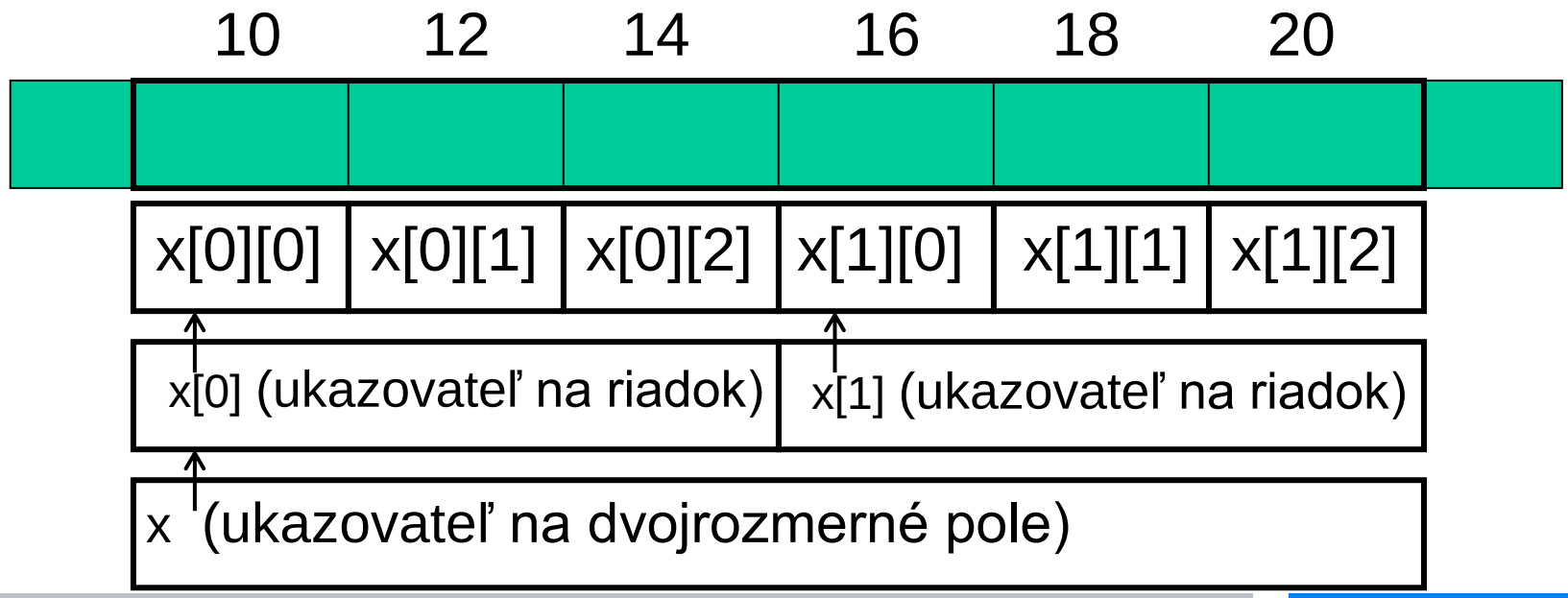
[0][0]	[0][1]	[0][2]	[0][3]
[1][0]	[1][1]	[1][2]	[1][3]
[2][0]	[2][1]	[2][2]	[2][3]
[3][0]	[3][1]	[3][2]	[3][3]
[4][0]	[4][1]	[4][2]	[4][3]

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20

Statické dvojrozmerné pole

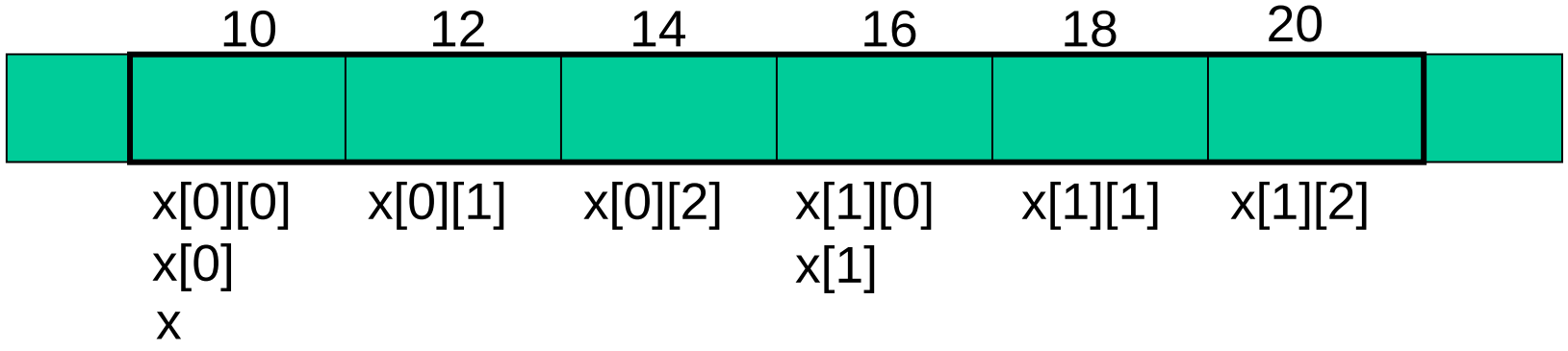
```
int xa[2][3];
```

- pole **xa**:
  - alokované pri preklade
  - súvislý blok 6 prvkov
  - uložené po riadkoch
  - konštantný ukazovateľ



Uloženie viacrozmerného poľa v pamäti

- `x` a `x[0]` - tá istá adresa, len iného typu
- `x+1` a `x[0]+1` - predstavujú odlišné adresy
- `x` - ukazovateľ na dvojrozmerné pole
- `x[i]` - ukazovateľ na i-ty riadok
- `*(x + 1) == x[1] ==16` - adresa prvého riadku
- `x[i][j]` - hodnota prvku dvojrozmerného poľa

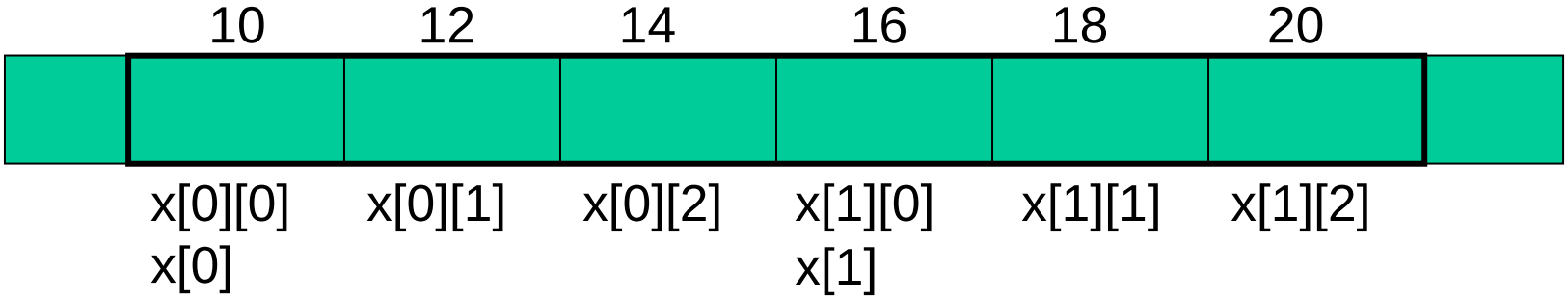


Uloženie viacrozmerného poľa v pamäti

- `x[i] == *(x + i)` - adresa `i`-teho riadku

- `&x[i][j] == x[i] + j == *(x + i) + j`
  - adresa (i,j) pozicie prvku

- `x[i][j] == *(x[i] + j) == *(*(x + i) + j)`
  - hodnota (i,j) pozicie prvku



Dvojrozmerné pole

slido.com
3916580
PrPr – P6

- statické dvojrozmerné pole
- **pole ukazovateľov**
- ukazovateľ na pole
- ukazovateľ na ukazovateľ

Pole ukazovateľov

```
int *xb[2];
```

- pole **xb**:
  - jednorozmerné pole dvoch ukazovateľov na **int**
  - ukazovatele sa využijú na riadky poľa, pre ktoré musíme alokovať pamäť pre 3 stĺpce

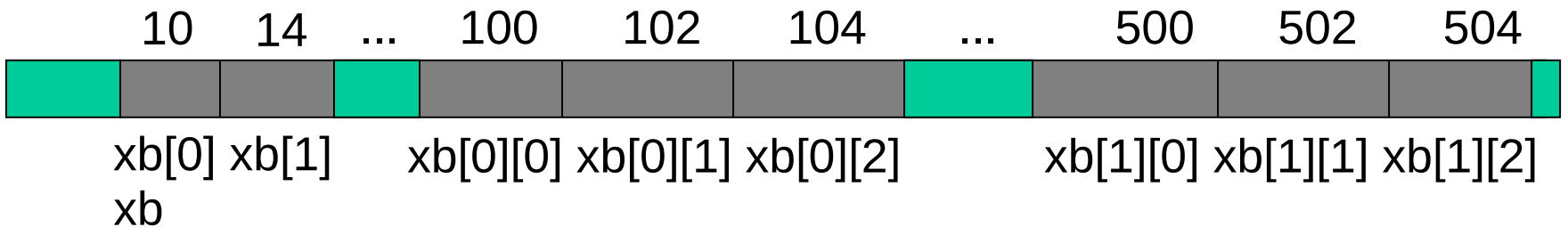
```
xb[0] = (int *) malloc(3* sizeof(int));
xb[1] = (int *) malloc(3* sizeof(int));
```

potom sa dá použiť: `xb[0][2] = 5;`

Pole ukazovateľov

```
int *xb[2];
```

- pole **xb**:
  - jednotlivé riadky nemusia nasledovať v pamäti bezprostredne za sebou
  - ak statickému poľu **int xa[2][3]** priradíme **xa[0][3] = 5**, potom sa hodnota priradí **xa[1][0]**, u poľa **xb** to **nemusí platiť**



Dvojrozmerné pole

slido slido.com
3916580
PrPr – P6

- statické dvojrozmerné pole
- pole ukazovateľov
- **ukazovateľ na pole**
- ukazovateľ na ukazovateľ

Ukazovateľ na pole

```
int (*xc)[3];
```

- pole **xc**:
  - **xc** je ukazovateľ na pole troch **int**-ov
  - ak alokujeme dostatok pamäte - ako dvojrozmerné pole

```
xc = (int *) malloc(2 * 3 * sizeof(int));
```

- **xc** ukazuje na pole 6 prvkov združených po troch
- obdoba statického poľa

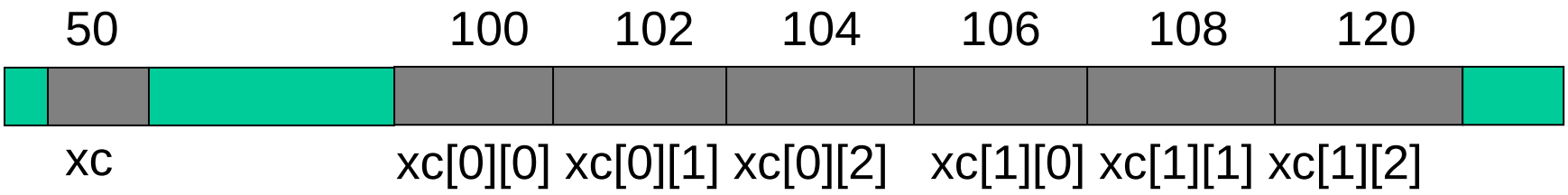
dá sa použiť:

```
xc[0][2] = 5;
```

Ukazovateľ na pole

```
int (*xc)[3];
```

- pole **xc**:
  - jednotlivé riadky nasledujú v pamäti bezprostredne za sebou



Dvojrozmerné pole

slido slido.com
3916580
PrPr – P6

- statické dvojrozmerné pole
- pole ukazovateľov
- ukazovateľ na pole
- **ukazovateľ na ukazovateľ**

Ukazovateľ na ukazovateľ

```
int **xd;
```

- pole **xd**:

  (1) alokujeme ukazovatele na riadky

```
xd = (int **) malloc(2 * sizeof(int *));
```

  (2) alokujeme jednotlivé riadky

```
xd[0] = (int *) malloc(3 * sizeof(int));
xd[1] = (int *) malloc(3 * sizeof(int));
```

dá sa použiť:

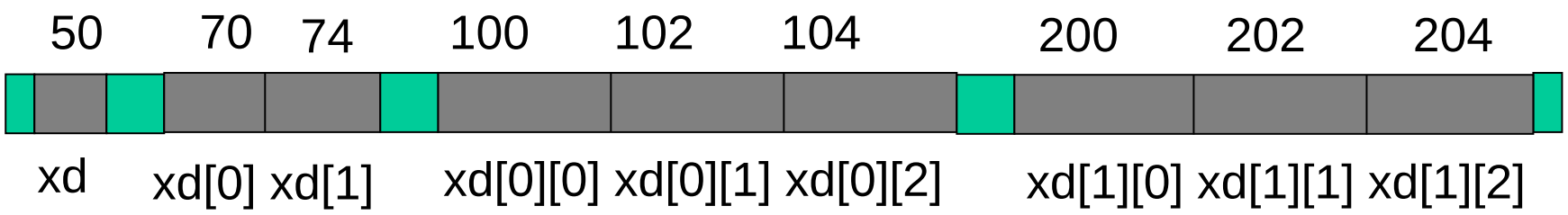
```
xd[0][2] = 5;
```

Ukazovateľ na ukazovateľ

```
int **xd;
```

- pole **xd**:

xd	je ukazovateľ na ukazovateľ na typ **int**
***xd**	je ukazovateľ na typ **int**
****xd**	je prvok typu **int**

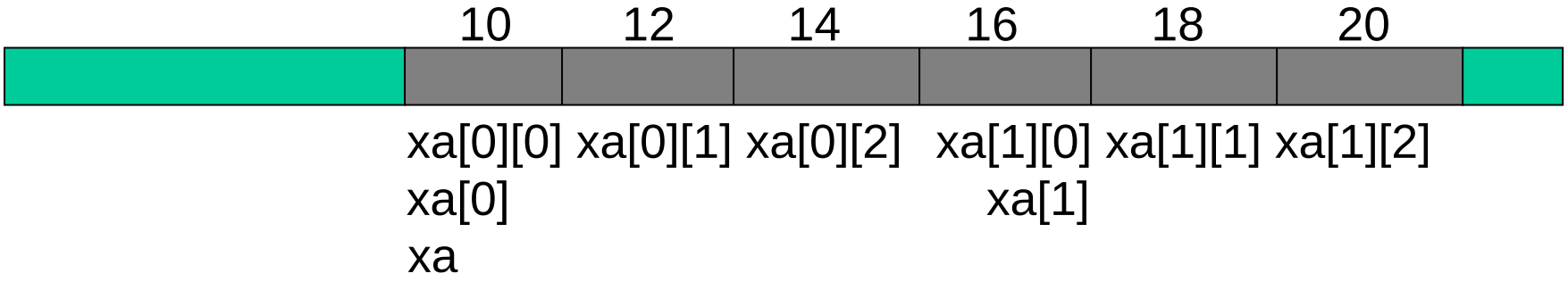


Porovnanie: typ poľa

- definícia **`xa`** (**`int xa[2][3]`**) predstavuje statické pole
- definícia **`xb`** (**`int *xb[2]`**), **`xc`** (**`int (*xc)[3]`**) a **`xd`** (**`int **xd`**) predstavujú po alokácii dynamické polia

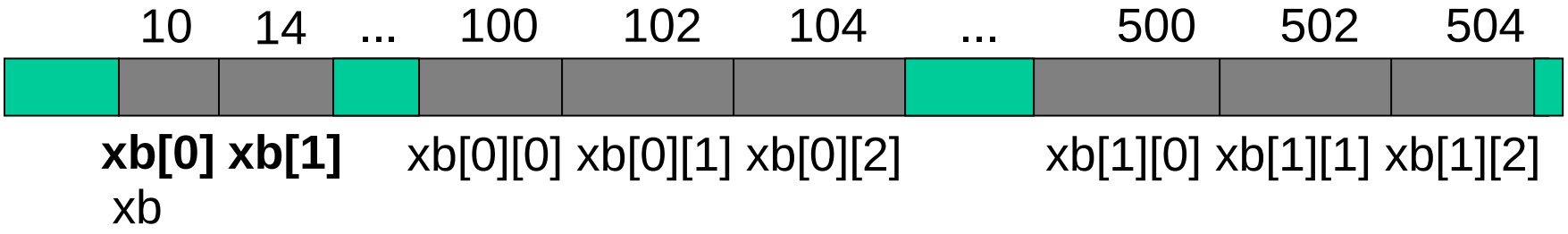
Porovnanie: pamäťové nároky

- **`xa`** (**`int xa[2][3]`**): pamäťovo najvýhodnejšia
  - Výhoda - nie je potrebná alokácia ďalších smerníkov
  - Nevýhoda – potreba veľkého bloku pamäte, problém v prípade poľa veľkého rozsahu



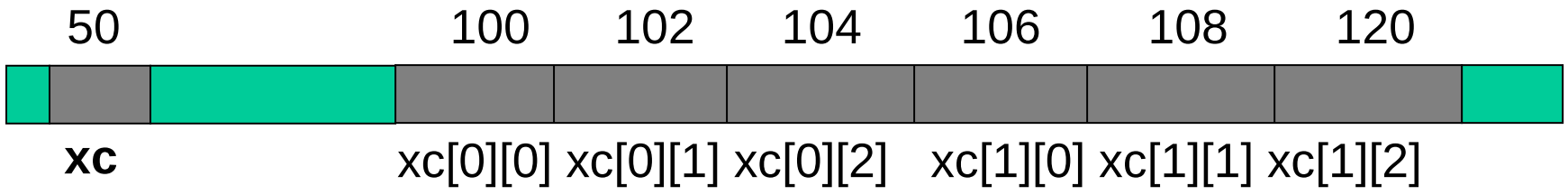
Porovnanie: pamäťové nároky

- **`xa`** (**`int xa[2][3]`**): pamäťovo najvýhodnejšia
- **`xb`** (**`int *xb[2]`**): naviac pamäť pre 2 ukazovatele (počet riadkov **`xb[0]`**, **`xb[1]`**)
  - riadky nie sú v pamäti uložené v celku



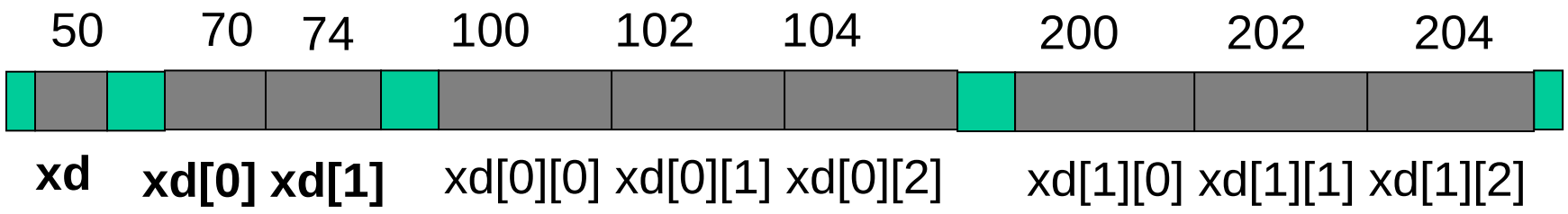
Porovnanie: pamäťové nároky

- **`xa`** (**`int xa[2][3]`**): pamäťovo najvýhodnejšia
- **`xb`** (**`int *xb[2]`**): naviac pamäť pre 2 ukazovatele (počet riadkov **`xb[0]`**, **`xb[1]`**)
- **`xc`** (**`int (*xc)[3]`**): naviac pamäť pre 1 ukazovateľ na typ **`int`**



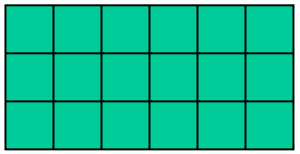
Porovnanie: pamäťové nároky

- **`xa`** (**`int xa[2][3]`**): pamäťovo najvýhodnejšia
- **`xb`** (**`int *xb[2]`**): naviac pamäť pre 2 ukazovatele (počet riadkov **`xb[0]`**, **`xb[1]`**)
- **`xc`** (**`int (*xc)[3]`**): naviac pamäť pre 1 ukazovateľ na typ **`int`**
- **`xd`** (**`int **xd`**): naviac 3 ukazovatele (pre **`xd`** a riadky), najpomalší prístup k prvkom
  - riadky nie sú v pamäti uložené v celku

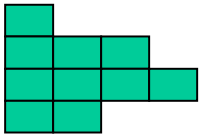


Porovnanie: charakteristika

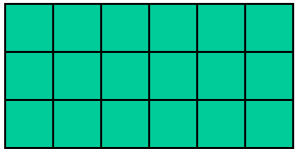
- `xa` (`int xa[2][3]`): pravoúhle pole



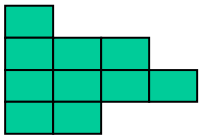
- `xb` (`int *xb[2]`): "zubaté" pole



- `xc` (`int (*xc)[3]`): pravoúhle pole



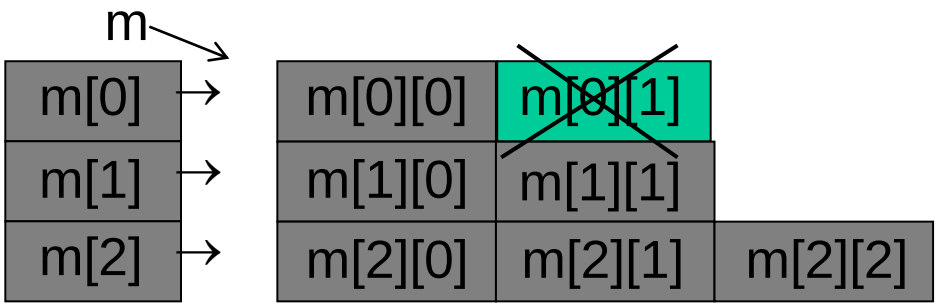
- `xd` (`int **xd`): "zubaté" pole



Zubaté pole

dvojrozmerné pole s rôznou dĺžkou riadkov - časť matice pod diagonálou (vrátane) - **`int *m[3]`**

```
int *m[3], i;
for(i = 0; i < 3; i++)
    m[i] = (int *) malloc((i+1) * sizeof(int));
```



Pravouhlé pole

```
int **alokuj2D(int riadky, int stlpce)
{
  int **p, i;

  p = (int **) malloc(riadky * sizeof(int *));
  for(i = 0; i < riadky; i++)
    p[i] = (int *) malloc(stlpce * sizeof(int));

  return p;
}
```

```
int **a, **b;
a = alokuj2D(3, 5);
b = alokuj2D(10, 20);
```

príklad volania funkcie:

Dvojrozmerné pole ako parameter funkcie

Podobne ako jednorozmerné pole

- odlišnosť:
  - prvá dimenzia - prázdna []
  - druhá dimenzia musí byť uvedená, napr. [10]
- preto
  - je potrebné preniesť do funkcie aj počet riadkov
  - skutočný parameter: len pravouhlé polia (**`xa`**, **`xc`**)

```
double x[5][6];
```

```
double x[][5]
```

alebo

```
double (*x)[5]
```

~~`double *x[5]`~~

Inicializácia polí

```
double f[3] = {1.5, 3.0, 7.6 };
```

```
double f[] = {1.5, 3.0, 7.6 };
```

ak nie je uvedený počet prvkov, určí sa podľa počtu hodnôt.

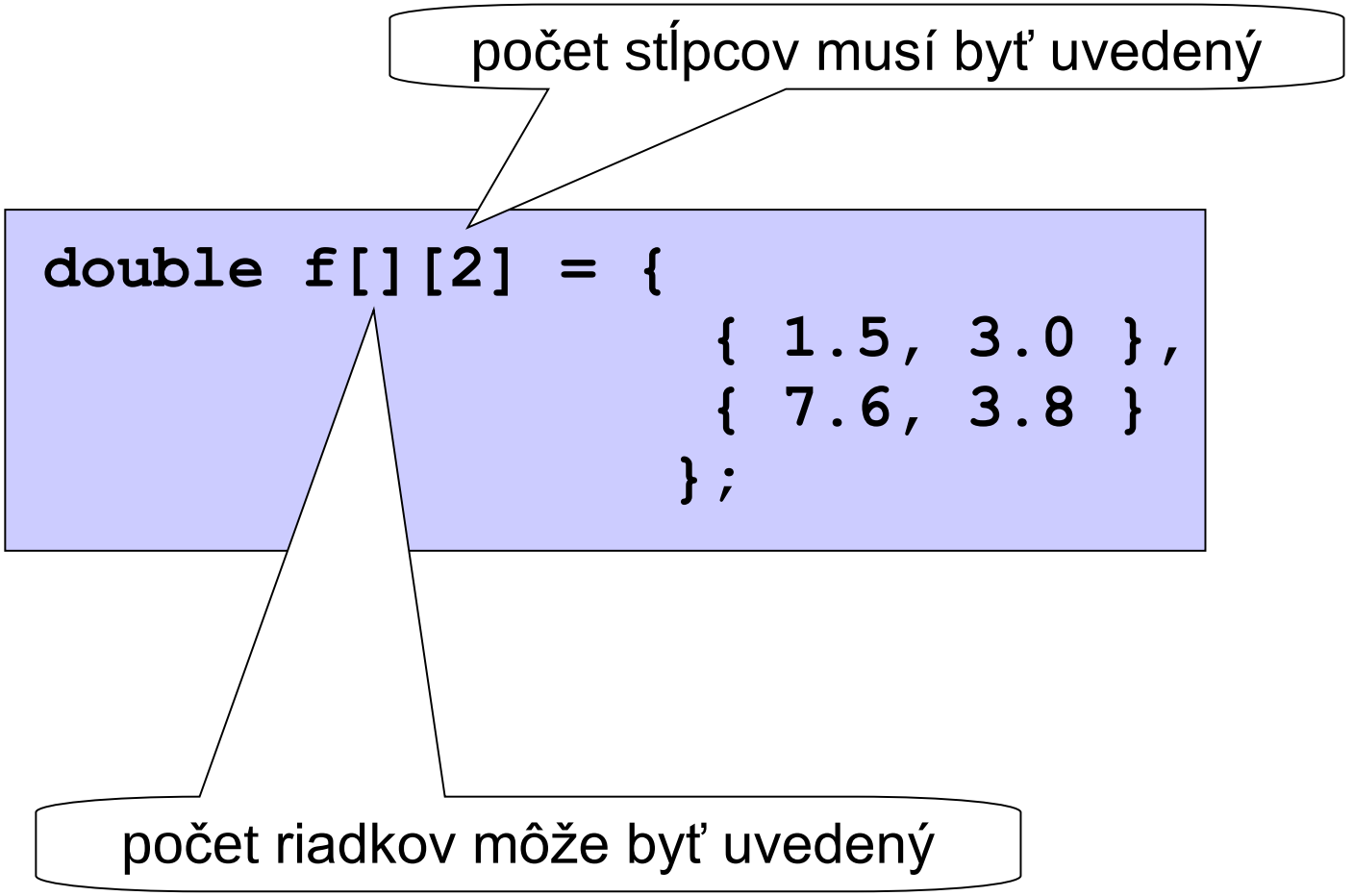
```
double f[3] = {1.5, 3.0 };
```

ak je hodnôt menej, doplní sa hodnotami 0.0

```
double f[3] = {1.5, 3.0, 7.6, 3.8 };
```

ak je hodnôt viac → chyba

Inicializácia dvojrozmerných polí



Korektné uvoľnenie pamäte

slido slido.com
3916580
PrPr – P6

```
int n;
scanf("%d", &n);
int **pole_2D = malloc(n*sizeof(int*));
for(int i=0; i<n; i++){
    pole_2D[i]= malloc((i+1)*10*sizeof(int));
}
...
for(int i=0;i<n;i++){
    free(pole_2D[i]);
    pole_2D[i] = NULL;
}

free(pole_2D);
pole_2D= NULL;
```

Reťazce

slido.com
3916580
PrPr – P6

Reťazce

- znaková premenná uchováva kód znaku
  - celé číslo
- reťazce sú jednorozmerné **polia typu char**
  - z celkovej pamäte je aktívna len časť od začiatku poľa po znak '\0' ukončovací znak
  - ak nie je reťazec ukončený znakom '\0', považuje sa za reťazec celá nasledujúca oblasť v pamäti až do najbližšieho znaku '\0'
  - dĺžka reťazca je pozícia znaku '\0'
    - nemusíme si ju pamätať v premennej
    - skrátenie pomocou posunu null

Reťazce

reťazec s najviac **5 znakmi**:

- staticky:

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

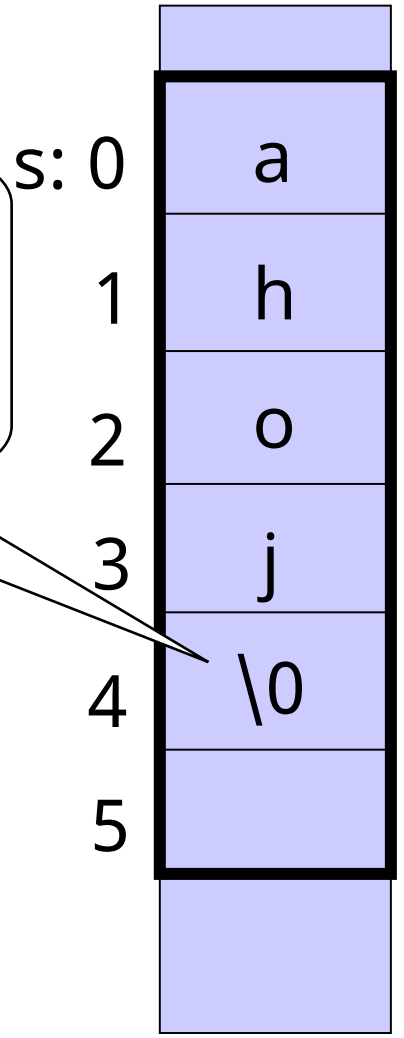
```
char s[] = "abrakadabra";
```

- "dynamicky":

```
char *s;
s = (char *) malloc(6);
```

znak null alebo 0
(ASCII kód 0)
nepliesť si s cifrou 0
(jej kód je 48)

inicializuje sa miesto
práve pre daný text



Reťazce

```
char s[] =
{'A','h','o','j'};
```

neukončený `'\0'`

```
char s[] =
{'A','h','o','j',0};
```

ukončený `'\0'`

pole inicializované na "ahoj"

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

```
char *s = "ahoj";
```

s nie je dynamický reťazec, ale ukazovateľ na typ **char** a je inicializovaný adresou reťazcovej konštanty (ktorá má obsah ahoj)

Reťazce

```
char *retazec = "Jan";
retazec = "Karol";
retazec[3] = 's';
```

nie je možné meniť reťazcovú konštantu

```
char s[10] = "Hallo";
s = "ahoj";
s[3] = 'r';
```

nie je možné priradiť statickému reťazcu konštantu

Reťazce

- 'z' vs "retazcova_konstanta"
- statická časť programu
- Inicializácia reťazca (== pole znakov ukončené '\0'):
- ako pole:

```
char s[] = {'A','h','o','j','\0'};
```

- pomocou konštanty:

```
char s[] = "Ahoj";
```

Reťazce

- Pracujeme s ním ako s jednorozmerným poľom
  - operátor []

```
char s[10];

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

dôležité: ukončiť reťazec!

Reťazce

- Môžeme využiť ukazateľovú aritmetiku

```
char s[] = "Ahoj svet";

*(s + 4) = 0; //s[4]

char* s2 = s + 5;
```

s je adresa na prvý znak reťazca

zmena konca reťazca

ukazovateľ na časť iného reťazca

- Reťazce nemenia svoju veľkosť automaticky
  - automatické zväčšovanie
  - zápis za koniec poľa
- operátor + sčíta ukazovatele na reťazce nie je to zreťazenia (Java, C++)

Reťazce

- dynamicky vytvorený reťazec sa nedá inicializovať
- (inicializácia sa vykonáva pri preklade, kedy ešte pole nie je vytvorené)

```
char *s;
s = (char *) malloc(10);
s = "ahoj";

char c;
int i=0;
while ((c=getchar()) != '\n' && i < 9)
    s[i++] = c;
s[i] = '\0';
```

alokovanie 10 znakov do **s**

načítanie aj pomocou **scanf**

Reťazce

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

sem nepartrí **&**, pretože **s** je adresa

- **scanf()** vynecháva biele znaky a číta po prvý biely znak
  - Výhoda nemusíme ošetrovať prázdne znaky na začiatku riadku
  - Nevýhoda načítavanie sa ukončí pri prvom prázdnom znaku
- ak je na vstupe **" ahoj Eva!"**, **scanf()** prečíta iba **"ahoj"** a zvyšok zostáva v bufferi klávesnice

Formátovaný vstup a výstup z a do reťazca

- použitie výhod formátovaného vstupu a výstupu, ale nevypísať nič na obrazovku ani nenačítavať

```
int sprintf(char *s, char *format, ...);
```

pracuje ako **fprintf**, ale zapisuje do reťazca **s**

```
int sscanf(char *s, char *format, ...);
```

pracuje ako **fscanf**, ale číta z reťazca **s**

Riadkovo orientovaný vstup a výstup z terminálu

- okrem **`scanf()`**, **`printf()`** aj:

```
char *gets(char *s);
```

číta celý riadok do **`s`**: na koniec nezapíše **`\n`**, ale **`\0`**, vracia ukazovateľ na **`s`**, ak je riadok prázdny dáva do **`s`** **`\0`** a vráti **`NULL`**

```
int puts(char *s);
```

vypíše reťazec a odriadkuje (**`\n`**), vráti nezáporné číslo ak sa podarilo vypísať, inak **`EOF`**

Riadkovo orientovaný vstup a výstup z terminálu

- okrem **`fscanf()`**, **`fprintf()`** aj:

```
char *fgets(char *s, int max, FILE *fr);
```

číta riadok zo súboru do konca riadku ale maximálne **`max`** znakov, načítané zapíše do **`s`** (aj s **`\n`**), vracia ukazovateľ na **`s`**, ak je koniec súboru tak **`NULL`**

```
int fputs(char *s, FILE *fw);
```

do súboru **`fw`** vypíše reťazec **`s`**, neodriadkuje ani neukončuje pomocou **`\0`**, vráti nezáporné číslo ak sa podarilo vypísať, inak **`EOF`**

Reťazce

Definícia typu pre reťazce

```
typedef char *STRING;
```

Treba rozlišovať:

- nulový ukazovateľ **`NULL`** a nulový reťazec `'\0'`:
  - nulový ukazovateľ neukazuje na žiadne miesto v pamäti,
  - nulový reťazec má v 0-tom prvku znak `'\0'`
- `"x"` a `'x'`:
  - `"x"` je reťazec s jedným znakom ukončený `'\0'` (2 Byty)
  - `'x'` je jeden znak (1 Byte)

Štandardné funkcie pre prácu s reťazcami

slido slido.com
3916580
PrPr – P6

- Reťazec sa nedá kopírovať priradením
- Reťazce sa nedajú porovnávať (==, !=, >, < atď.)

Funkcionality si treba naprogramovať, alebo použiť funkcie z knižnice

- nie sú súčasťou samotného jazyka C

- Využitie knižnice **string.h**
  - dokumentácia http://www.cplusplus.com/reference/cstring/
  - prehľad: https://en.cppreference.com/w/c/string/byte
- Základné pravidlá
  - reťazec je ukončený znakom '\0'
  - pri modifikácii reťazca má výstupný reťazec dostatočnú veľkosť

Štandardné funkcie pre prácu s reťazcami

slido slido.com
3916580
PrPr – P6

```
int strlen(char *s);
```

vracia dĺžku reťazca (bez `\0`)

```
char *strcpy(char *kam, char *co);
```

kopírovanie reťazca **`co`** do **`kam`**, vracia ukazovateľ na **`kam`**
(reťazec **`kam`** musí byť dosť dlhý)

Štandardné funkcie pre prácu s reťazcami

```
char *strcat(char *kam, char *co);
```

pripojí reťazec **co** ku **kam**, vracia ukazovateľ na **kam**
(reťazec **kam** musí byť dosť dlhý
-> strlen(kam) strlen(co) + 1)

```
int strcmp(char *s1, char *s2);
```

vracia **0**, ak sú reťazce rovnaké, záporné číslo, ak **s1** je skôr (abecedne), inak kladné číslo
(porovnáva sa na základe kódov znakov -> 'Z' je skôr ako 'a'

Štandardné funkcie pre prácu s reťazcami

```
char *strchr(char *s, char c);
```

nájdenie znaku **c** v reťazci **s**, vracia prvý výskyt znaku, ak sa v **s** nenachádza, vráti **NULL**

```
char *strstr(char *s1, char *s2);
```

vracia ukazovateľ na prvý výskyt reťazca **s2** v reťazci **s1**, v prípade neúspechu **NULL**

Práca s časťou reťazca

- podobne ako uvedené funkcie,
- v názve je **`n`** (zo slova *number*), napr. **`strncpy()`**:

```
char *strncpy(char *s1, char *s2, int max);
```

kopírovanie najviac max znakov z reťazca **`s2`** do **`s1`**, vracia ukazovateľ na **`s1`**

Práca s reťazcom naopak

- podobne ako uvedené funkcie,
- v názve je r (zo slova *reverse*), napr. **`strrchr()`**:

```
char *strrchr(char *s, char c);
```

nájdenie znaku **`c`** v reťazci **`s`**, vracia posledný výskyt znaku, ak sa v **`s`** nenachádza, vráti NULL

Prevody reťazcov na čísla

- konvertovanie reťazca číslic na číslo (funkcie definované v **`stdlib.h`**)

```
int atoi(char *s);
```

prekonvertuje reťazec znakov na **`int`**

```
long atol(char *s);
```

prekonvertuje reťazec znakov na **`long`**

```
float atof(char *s);
```

prekonvertuje reťazec znakov na **`float`**

Pri vstupe a výstupe nie je konverzia potrebná (**`scanf()`** a **`printf()`**)

Kontrola znakov v reťazci

slido slido.com
3916580
PrPr – P6

Knižnica **type.h** – http://www.cplusplus.com/reference/cctype/

- **isalnum(znak)** – je znak písmeno alebo číslica?
- **isdigit(znak)** – je znak desiatková číslica?
- **isxdigit(znak)** – je znak hexadecimálna číslica?
- **isalpha(znak)** – je znak písmeno (veľké/malé)?
- **islower(znak)** – je znak malé písmeno?
- **isupper(znak)** – je znak veľké písmeno?
- **ispunct(znak)** – je znak špeciálny (vypísateľný)?
- **isprint(znak)** – dá sa znak vypísať (medzera, písmeno...)?
- **isgraph(znak)** – má znak grafickú podobu (písmeno, číslica…)?
- **isspace(znak)** – je znak biely (nový riadok, medzera..)?
- **iscntrl(znak)** – je znak riadiaci?

Prevod znakov:

- **tolower(znak)** – veľké písmeno prevedie na malé písmeno
- **toupper(znak)** – malé písmeno prevedie na veľké písmeno

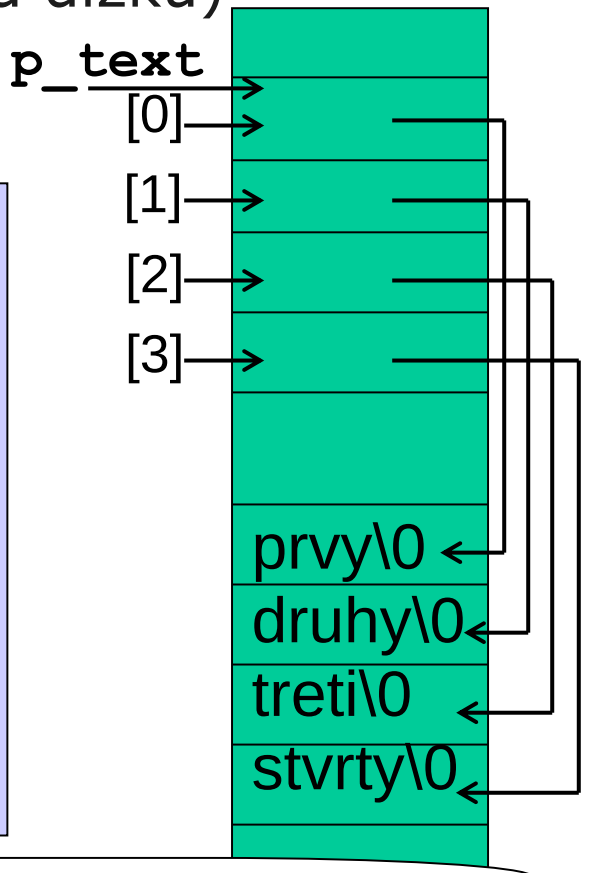
Pole reťazcov

- pole ukazovateľov na reťazce
- zvyčajne zubaté (reťazce/riadky majú rôznu dĺžku)

iba reťazec **p_text[2]** je alokovaný dynamicky, ostatné sú statické

```
char *p_text[4];

p_text[0] = "prvy";
p_text[1] = "druhy";
p_text[2] = (char *) malloc(6);
strcpy(p_text[2], "treti");
p_text[3] = "stvrty";
strcpy(p_text[3], "Stvrty");
```



chyba **p_text[3]** zápis do pamäti s konštantným reťazcom

Pole reťazcov

```
char *p_text[4], c, *p;
...
c = p_text[0][0];

p = &p_text[0][0];
while (*p != '\0')
    putchar(*p++);

printf("%s \n", p_text[1]);

puts(p_text[2]);
```

prístup k jednotlivým prvkom reťazca

vypísanie reťazca po znakoch

vypísanie reťazca pomocou **`printf()`**

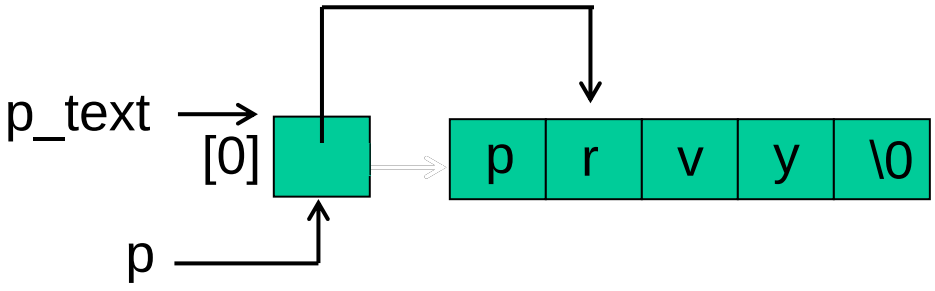
vypísanie reťazca pomocou **`puts()`**

Pole reťazcov

```
char *p_text[4], **p;
...
p = p_text;
puts(++*p);
```

- **p** ukazuja na **p_text**,
- ***p** ukazuje na **p_text[0]**
- príkaz **++*p** zväčší hodnotu na tej adrese o 1, teda zväčší **p_text[0]**

- vypíše sa "**rvy**" pretože ***p** ukazuje na nultý prvok poľa **p_text**
- **p_text[0]** potom ukazuje na "**rvy**" a táto zmena je trvalá



Pole reťazcov

```
char *p_text[4], **p;
...
p = p_text;
puts(*++p);
```

vypíše sa **"druhy"** pretože sme najprv zvýšili **p** o 1 (posunuli sme ho na druhý riadok a potom vypísali reťazec, kam ukazuje **p**)

```
char *p_text[4], **p;
...
p = p_text;
for (i = 0; i < 4; i++)
    puts(*p++);
```

++ má väčšiu prioritu ako ***,** riadok sa najprv vypíše a ukazovateľ **p** sa posunie na druhý riadok.

Časté chyby pri práci s reťazcami

slido slido.com
3916580
PrPr – P6

- nedostatočne veľký cieľový reťazec (napr. **`strcpy()`**)
- nekorektné ukončenie reťazca
  - funkcie z knižnice string sa správajú nekorektne
  - napr. zapisujeme mimo rozsah poľa
  - vzniká napr. pri prepísaní, nevložením... znaku `'\0'`
- dĺžka/veľkosť reťazca sa udáva bez znaku `'\0'` (**`strlen()`**)
- operátor `==` neporovnáva reťazce (**`strcmp()`**)
- operátor `+` nezreťazuje reťazce

Ďakujem vám za pozornosť!

slido.com
3916580
PrPr – P6

Spätná väzba: *https://forms.gle/6q5D2G6UwrtimXEx9*