VYBRANÉ PARTIE Z JAZYKA C

Aritmetické dátové typy

- Základné aritmetické typy: očakávaná veľkosť v bytoch
  - char, alebo signed char 1
  - unsigned char 1
  - short, alebo short int, signed short, signed short int 2
  - unsigned short, alebo unsigned short int 2
  - int, alebo signed int 4
  - unsigned, alebo unsigned int 4
  - long, alebo long int, signed long int 4
  - unsigned long, alebo unsigned long int 4
  - long long int, alebo signed long long int 4
  - unsigned long long, alebo unsigned long long int 8
  - float 4
  - double 8
  - long double 16

Reálne číselné typy

float	4
double	8
long double	8 /16 (10)

Všeobecne sa predpokladá, že aritmetické operácie nad reálnymi číslami sú výrazne pomalšie ako nad celočíselnými typmi.

Niektoré reálne čísla sa nedajú v počítači presne reprezentovať, treba si pri nich dávať pozor.

Odvodené dátové typy

- Smerníky.
- Polia.
- Štruktúry.
- Uniony.

Smerníková aritmetika

- Ak **`p`** je smerník, tak **`p+1`** znamená zväčšenie adresy o **`sizeof(*p)`**.
- Ak **`p1`** a **`p2`** sú smerníky na rovnaký typ, tak **`p1-p2`** je rozdiel adries vydelený **`sizeof(*p)`**.
- **Ak p je smerník, tak výraz p[i] je ekvivalentný výrazu *(p + (i))**

Polia

- súvislá oblasť pamäte vyplnená prvkami s daným typom

```
int a[10];        // pole desiatich prvkov typu int
double b[20];     // pole 20 realnych čísel
int x[10][20];    // pole 10 polí o 20 prvkoch int
int y[2][3][4]; // pole 2 polí 3 polí 4 prvkov ...
```

Viacrozmerné polia (polia polí)

declarácia:

```
int A[3][5];
```

uloženie v pamäti:

A == A[0]	A [0] [0]	A [0] [1]	A [0] [2]	A [0] [3]	A [0] [4]
A[1]	A [1] [0]	A [1] [1]	A [1] [2]	A [1] [3]	A [0] [4]
A[2]	A [2] [0]	A [2] [1]	A [2] [2]	A [2] [3]	A [2] [4]

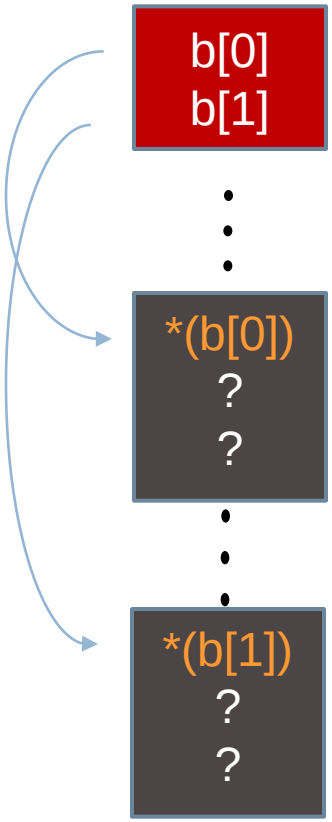
Veľkosť viacrozmerných polí

```
#include <stdio.h>

int main() {
        char a[3][5];
        printf("%d\n", sizeof(a));
        printf("%d\n", sizeof(a[0]));
        printf("%d\n", sizeof(a[0][0]));
}
```

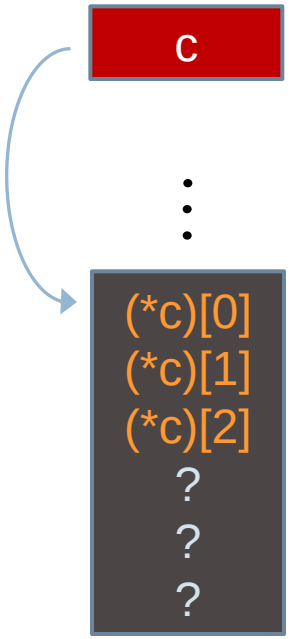
Pole, pointer na pole a pole pointrov

int a[2][3];

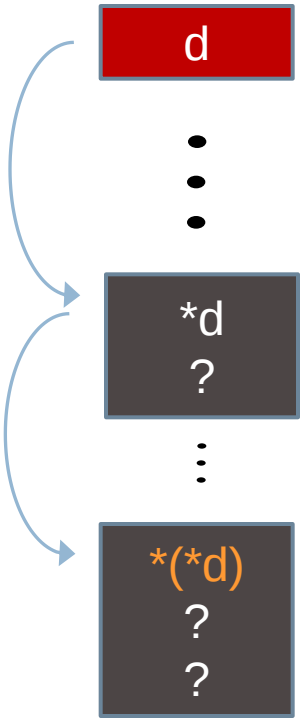
int *b[2];



int (*c)[3];



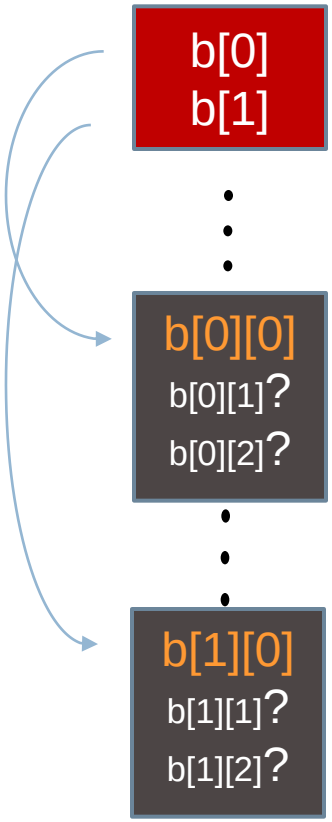
int **d;



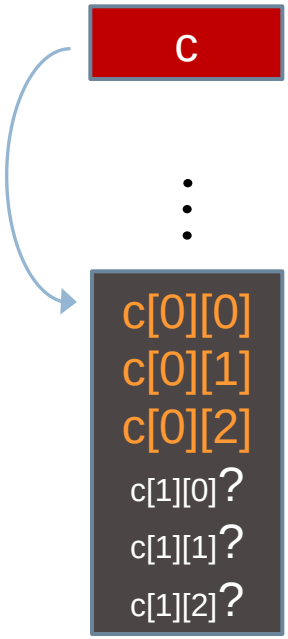
Pole, pointer na pole a pole pointrov

`int a[2][3];`

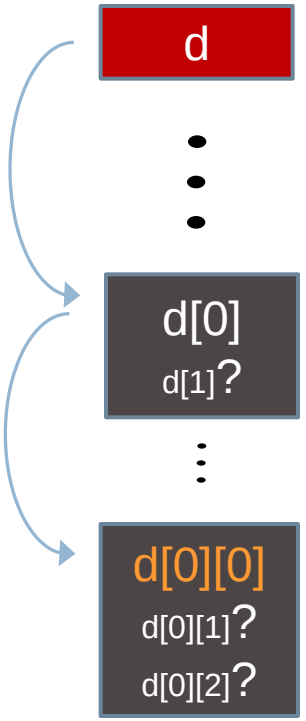
`int *b[2];`



`int (*c)[3];`



`int **d;`



Viacrozmerných polia a smerníková aritmetika

```
#include <stdio.h>

int main() {
        char a[3][5];
        printf("%p %p\n", a, a[0]);
        printf("%p %p\n", a, a[0]+1);
        printf("%p %p\n", a, a+1);
}
```

Viacrozmerné pole ako parameter funkcie

```
#include <stdio.h>

void fun(char (*b)[5]) {
     printf("%d %d %d\n", b, b+1, sizeof(b));
}

int main() {
     char a[3][5];
     printf("%d %d %d\n", a, a+1, sizeof(a));
     fun(p);
}
```

Viacrozmerné pole ako parameter funkcie

```
#include <stdio.h>

void fun(char b[3][5]) {
     printf("%d %d\n", b, sizeof(b));
}

int main() {
     char a[3][5];
     printf("%d %d\n", a, sizeof(a));
     fun(p);
}
```

Viacrozmerné pole ako parameter funkcie

```
#include <stdio.h>

void fun(char b[][5]) {
     printf("%d %d\n", b, sizeof(b));
}

int main() {
     char a[3][5];
     printf("%d %d\n", a, sizeof(a));
     fun(p);
}
```

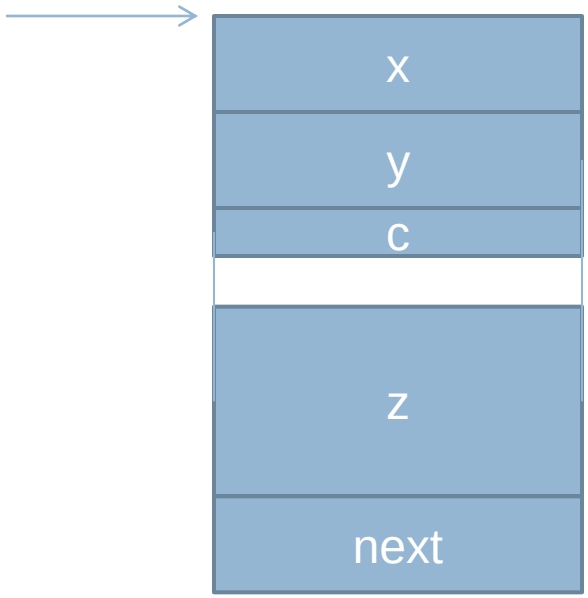
Zakázané konštrukcie

```
void fun(char **b) {
}

void gun(char b[][]) { // chyba
}

int main() {
   char a[3][5];
   fun(a);                    // chyba
}
```

Štruktúry

- postupnosť objektov s možno odlišnými typmi.

```
struct sss {
    int             x,y;
    char            c;
    double          z;
    struct sss      *next;
};

struct sss *p;
```



Bitové polia

- Bitové polia sa definujú ako špecialny prípad štruktúry, kde celočíselné typy majú veľkosť s presne daným počtom bitov.

```
struct cas {
  unsigned minuta:6;
  unsigned hodina:5;
  unsigned den:5;
  unsigned mesiac:4;
  unsigned rok:12;
};
```

Zarovnanie v bitovom poli

- Pole bez mena s dĺžkou 0 bitov spôsobí zarovnanie nasledovných častí na ďalšie slovo v pamäti.

```
struct cas {
  unsigned minuta:6;
  unsigned hodina:5;
  unsigned den:5;
  unsigned mesiac:4;
  unsigned :0;
  unsigned rok:12;
};
```

Maximálny počet bitov

- Počet bitov v jednej položke nesmie presiahnuť počet bitov daného celočíselného typu.

```
struct X {
  unsigned a:33;                    // chyba
};

struct Y {
  long long b:33;                   // O.K.
};

struct Z {
  unsigned c:30;
  unsigned d:30;
};
```

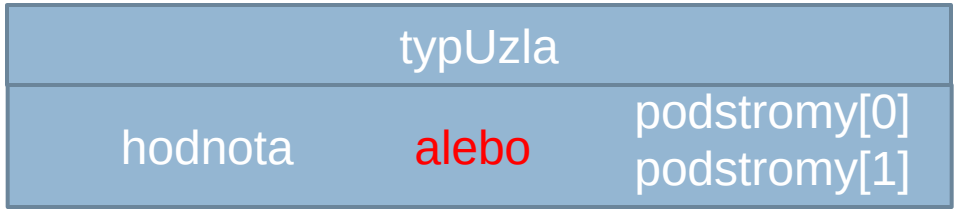
Uniony

- Syntax pred definovanie unionu je rovnaká ako pre štruktúru, ale všetky členy sú uložené na tej istej adrese na začiatku unionu. Slúži to na ušetrenie zmeny typov (castov). Veľkosť unionu je určená veľkosťou najväčšieho člena.

```
struct strom {
    int     somList;
    union {
        double          hodnota;
        struct strom podstromy[2];
    } u;
};
```

Uniony

```
struct strom {
    int     typUzla;
    union {
        double          hodnota;
        struct strom *podstromy[2];
    } u;
};
```



Deklarácie a definície

- Deklarácia sa skladá zo základného typu a z "výrazu" obsahujúceho novú premennú.
- Myšlienka je taká, že nová premenná má taký typ aby po vyhodnotení "výrazu" sme dostali ten základný typ.
- Ak použijeme špecifikátor typedef, tak nedefinujeme novú premennú, ale nový základný typ.

deklaracia:

specifikator+ zoznam-init-deklaratorov

specifikator:

(typedef | extern | static | auto | register
| void | char | short | int | long | float | double | signed | unsigned
| (struct | union) struct-union-specifikator | enum enum-specifikator
| typedef)

zoznam-init-deklaratorov: deklarator [initializacia] [, zoznam-init-deklaratorov]

deklarator:

** priamy-deklarator

priamy-deklarator:

identifikator

(deklarator)

priamy-deklarator [[konstantny-vyraz]]

priamy-deklarator (deklaracia-parametrov)

Príklady

- int *p;
- int *q = NULL;
- void *r;
- int **s;
- int (*(*t));
- int *a[10];
- int (*a)[10];
- int *((*a)[10]);
- int **a[10];

Príklady

- int *p; // smerník na int
- int *q = NULL;
- void *r; // smerník na neznámy typ
- int **s; // smerník na smerník na int
- int (*(*t)); // detto
- int *a[10]; // pole 10-tich smerníkov na int
- int (*a)[10]; // smerník na pole 10-tich int-ov
- int *((*a)[10]); // smerník na pole 10-tich smerníkov na int
- int **a[10]; // pole 10-tich smerníkov na smerník na int

Príklady 2

- int *p();
- double *q(int);
- double (*r)(int);
- char *(*s)(double, int);
- typedef void (*T)(int x);
- T (*w)[10];

Príklady 2

- int *p(); // funkcia bez parametrov vracajúca smerník na int
- double *q(int); // funkcia s parametrom int vracajúca smerník na double
- double (*r)(int); // smerník na funkciu s parametrom int a vracajúcu double
- char *(*s)(double, int); // smerník na funkciu s parametrami double a int a vracajúcu smerník na char
- typedef void (*T)(int x); // T je typ "smerník na procedúru s parametrom int
- T (*w)[10]; // smerník na pole 10-tich objektov typu T.

Čo je lepšie?

```
int  *p;
```

alebo

```
int*  p;
```

Čo je lepšie?

```
int  *p;
```

alebo

```
int*  p;
```

Kernighan, Ritchie

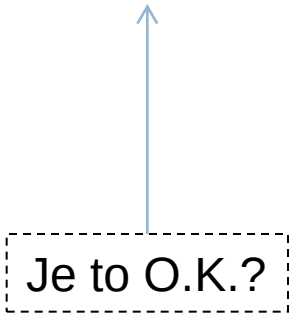
Stroustrup

Čo ak chcem definovať dva smerníky v jednej definícii?

```
int  *p, *q;
```

alebo

```
int*  p, q;
```



Cvičenie

- Definujte premennú:
  - x typu pole 10 smerníkov na integer.
  - z typu smerník na pole 20 znakov.
  - f typu smerník na funkciu vracajúcu integer s jedným parametrom typu double.
  - g typu smerník na funkciu vracajúcu smerník na integer s jedným parametrom typu smerník na double.
  - a pole smerníkov na funkcie vracajúce integer s jedným parametrom typu double.

Operátory (priorita)

- () [] -> .
- ! ~ ++ -- - (cast)
  * & sizeof
- * / %
- + -
- << >>
- < <= > >=
- == !=
- &
- ^
- |
- &&
- ||
- ?:
- = += -= *= /= %= ...
- ,

Bude to fungovať?

```
#include <stdio.h>

#define SWAP(a, b) b = (a+b) - (a=b)

int main() {
        int i = 10;
        int j = 20;
        SWAP(i, j);
        printf("%d %d \n", i, j);
}
```

Zradný príklad

```
#include <stdio.h>

void main() {
        int i;
        i = 0;
        printf("%d \n", i++ + i++);
}
```

Sequence point

- Sequence point (sekvenčný bod) je miesto v programe, kde je isté, že všetky predchádzajúce výrazy boli vypočítané a ich vedľajšie efekty uskutočnené.

Sequence point

- Pred vyvolaním funkcie po vyhodnotení argumentov
- Po vyhodnotení prvého argumentu operátorov && || ?: ,
- Po úplnom vyhodnotení celého výrazu v príkaze.

Poznámka: Priradenie nie je sekvenčný bod !

Príklad

```
int a[10];

int najdiNulu() {
        int i;
        i = 0;
        while (i<10 && a[i] != 0) i ++;
        return(i);
}
```

Cvičenie

- Napíšte funkciu ktorá vypočíta n modulo 8 iba s použitím bitových operácii.
- Napíšte funkciu s jedným parametrom typu unsigned, ktorá vráti prvý nenulový bit sprava. T.j. napríklad pre číslo 0x34 vráti 4, pre číslo 0xff0000 vráti 0x10000. atď.
- Podobne funkciu. ktorá vráti prvý nenulový bit zľava.
- Funkciu, ktorá v každom byte v celom čísle vymení prvý a siedmy bit. Tj. 0x1 --> 0x40, 0x1234 -> 0x1234, 0x7654 -> 0x3715.