

# Struktury a uniony, přesnost výpočtů a vnitřní reprezentace číselných typů

Jan Faigl

Katedra počítačů  
Fakulta elektrotechnická  
České vysoké učení technické v Praze

Přednáška 06

B0B36PRP – Procedurální programování

## Část I

### Část 1 – Struktury a uniony

## Přehled témat

### ■ Část 1 – Struktury a uniony

Struktury – `struct`

Proměnné se sdílenou pamětí – `union`

Příklad

*S. G. Kochan: kapitola 9 a 17*

### ■ Část 2 – Přesnost výpočtů a vnitřní reprezentace číselných typů

Přesnost výpočtů a numerická stability

Základní číselné typy a jejich reprezentace v počítači

Reprezentace celých čísel

Reprezentace reálných čísel

Typové konverze

Matematické funkce

*S. G. Kochan: kapitola 14 (typové konverze)*

*Appendix B (matematické funkce)*

### ■ Část 3 – Zadání 6. domácího úkolu (HW06)

## Struktura – `struct`

- Struktura je konečná množina prvků (proměnných), které nemusí být stejného typu.
- Skladba struktury je definovaná uživatelem jako nový typ sestavený z již definovaných typů.
- K prvkům struktury **přístupujeme tečkovou notací**, např. `struct_proměnná.prvek`.
- K prvkům můžeme přistupovat přes ukazatel operátorem `->`, např.  
`proměnná_typu_ukazatel_na_struct->prvek`.
- **Pro struktury stejného typu je definován operátor přiřazení.**  
`var_struct1 = var_struct2;`
- Struktury (jako celek) **nelze** porovnávat relačním operátorem `==`.
- Struktura může být funkci předávána hodnotou i ukazatelem.
- Struktura může být návratovou hodnotou funkce.

## Příklad struct – Definice

- Bez zavedení nového typu (`typedef`) je nutné před identifikátor jména struktury uvádět klíčové slovo `struct`.
- Jméno struktury je ve jmenném prostoru složených typů (struktur).

```
struct record {
    int number;
    double value;
};

typedef struct {
    int n;
    double v;
} item;
```

```
record r; /* IT IS NOT ALLOWED! */
        /* Type record is not known */
```

```
struct record r; /* Keyword struct is required */
item i; /* type item defined using typedef */
```

- Zavedením nového typu `typedef` používáme definovaný typ a nemusíme používat (a ani definovat) jméno struktury.

lec06/struct.c

## Příklad struct – Inicializace

- Struktury:

```
struct record {
    int number;
    double value;
};

typedef struct {
    int n;
    double v;
} item;
```

- Proměnné typu struktura můžeme inicializovat prvek po prvku.

```
struct record r;
r.value = 21.4;
r.number = 7;
```

- Podobně jako pole lze inicializovat přímo při definici

```
item i = { 1, 2.3 };
```

- nebo pouze konkrétní položky (ostatní jsou nulovány).

```
struct record r2 = { .value = 10.4 };
```

lec06/struct.c

## Definice jména struktury a typu struktury

- Uvedením `struct record` zavádíme nové jméno struktury `record`.

```
struct record {
    int number;
    double value;
};
```

- Definujeme identifikátor `record` ve jmenném prostoru struktur.

- Definicí typu `typedef` zavádíme nové jméno typu `record`.

```
typedef struct record record;
```

- Definujeme globální identifikátor `record` jako jméno typu `struct record`.

- Obojí můžeme kombinovat v jediné definici jména a typu struktury.

```
typedef struct record {
    int number;
    double value;
} record;

typedef struct record_struct_name {
    int number;
    double value;
} record_type;
```

## Příklad struct jako parametr funkce

- Struktury můžeme předávat jako parametry funkcí hodnotou.

```
void print_record(struct record rec) {
    printf("record: number(%d), value(%lf)\n",
           rec.number, rec.value);
}
```

- Nebo hodnotou ukazatele

```
void print_item(item *v) {
    printf("item: n(%d), v(%lf)\n", v->n, v->v);
}
```

- Při předávání parametru

- **hodnotou** se vytváří nová proměnná a původní obsah předávané struktury se kopíruje na zásobník (pro složený typ je definován operátor přiřazení);
- **hodnotou ukazatele** se kopíruje pouze hodnota ukazatele (adresa) a pracujeme tak s původní strukturou.

lec06/struct.c

## Složený typ, operátor přiřazení a pole jako prvek složeného typu 1/2

- Velikost složeného typu musí být známa během překladač, proto můžeme mít definovaný operátor přiřazení.  
*Nebo naopak, abychom mohli jednoduše přiřazovat, tak potřebujeme znát velikost typu.*
- Prvek složeného typu může být pole (definované velikosti) nebo ukazatel.

```

1 void print(const char *str, int n, int *a);
2
3 #define N 10 // We need named literal.
4
5 int main(void)
6 {
7     struct { // Anonymous struct
8         int a[N]; // Defined size, no VLA
9     } s1, s2; // Two struct variables
10
11     printf("s1 %p; s2 %p\n", &s1, &s2);
12     for (int i = 0; i < n; ++i) {
13         s1.a[i] = i;
14     }
15     print("s1.a", n, s1.a);
16     s2 = s1; // Assignment
17     print("s2.a", n, s2.a);
18     for (int i = 0; i < n; ++i) {
19         s1.a[i] = n - i;
20     }
21     print("s1.a", n, s1.a);
22     print("s2.a", n, s2.a);
23     return 0;
24 } // end main()
25
26 void print(const char *str, int n, int *a) {
27     printf("%s %p: ", str, a);
28     for (int i = 0; i < n; ++i) {
29         printf("%d%s", a[i], i < (n-1) ? ", " : "\n");
30     }
31 }

```

lec06/demo-struct\_array.c

## Příklad struct – Přiřazení

- Hodnoty proměnné stejného typu struktury můžeme přiřadit operátorem =.

```

struct record {
    int number;
    double value;
};

typedef struct {
    int n;
    double v;
} item;

struct record rec1 = { 10, 7.12 };
struct record rec2 = { 5, 13.1 };
item i;
print_record(rec1); /* number(10), value(7.120000) */
print_record(rec2); /* number(5), value(13.100000) */
rec1 = rec2;
i = rec1; /* IT IS NOT ALLOWED! */
// Different types, albeit with the same memory representation.
print_record(rec1); /* number(5), value(13.100000) */

```

lec06/struct.c

## Složený typ, operátor přiřazení a pole jako prvek složeného typu 2/2

Příklad `lec06/demo-struct_array.c`

- Používáme anonymní složený typ - definice strukturu přímo v definici proměnných `s1` a `s2`.
- Musíme použít textový literál pro definici velikosti položky `a` jako pole definované délky.
- Ve funkci `print()` tiskneme hodnotu adresy, kde je alokované pole.

*V našem případě se shoduje s adresou, kde je struktura uložena. Struktura je „organizovaný“ pohled na blok paměti důležitý zejména pro zpřehlednění programu. Při běhu programu vlastně není nutné mít v paměti dílčí jména prvku složeného typu.*

```

s1 0x7fffffff840; s2 0x7fffffff818
s1.a 0x7fffffff840: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
s2.a 0x7fffffff818: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
s1.a 0x7fffffff840: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
s2.a 0x7fffffff818: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

```

- V příkladu si vyzkoušejte chování překladač a programu v případě použití VLA nebo konstantní proměnné definující velikost pole.
- Pole definované velikosti nahraďte dynamicky alokovaným polem.

## Příklad struct – Přímá kopie paměti

- Jsou-li dvě struktury stejně veliké, můžeme přímo kopírovat obsah příslušné paměťové oblasti.

*Například funkci `memcpy()` z knihovny `string.h`*

```

struct record r = { 7, 21.4 };
item i = { 1, 2.3 };
print_record(r); /* number(7), value(21.400000) */
print_item(&i); /* n(1), v(2.300000) */
if (sizeof(i) == sizeof(r)) {
    printf("i and r are of the same size\n");
    memcpy(&i, &r, sizeof(i));
    print_item(&i); /* n(7), v(21.400000) */
}

```

- V tomto případě je interpretace hodnot v obou strukturách identická, obecně tomu však být nemusí. Například v případě změny pořadí prvků typu `int` a `double`.

lec06/struct.c

## Struktura struct a velikost

- Vnitřní reprezentace struktury nutně nemusí odpovídat součtu velikostí jednotlivých prvků.

```

struct record {
    int number;
    double value;
};

typedef struct {
    int n;
    double v;
} item;

printf("Size of int: %lu size of double: %lu\n", sizeof(int), sizeof(
double));
printf("Size of record: %lu\n", sizeof(struct record));
printf("Size of item: %lu\n", sizeof(item));

```

```

Size of int: 4 size of double: 8
Size of record: 16
Size of item: 16

```

lec06/struct.c

## Struktura struct a velikost 2/2

- Nebo

```

typedef struct __attribute__((packed)) {
    int n;
    double v;
} item_packed;

```

- Příklad výstupu:

```

printf("Size of int: %lu size of double: %lu\n", sizeof(int), sizeof(double));
printf("record_packed: %lu\n", sizeof(struct record_packed));
printf("item_packed: %lu\n", sizeof(item_packed));

```

```

Size of int: 4 size of double: 8
Size of record_packed: 12
Size of item_packed: 12

```

lec06/struct.c

- Zarovnání zpravidla přináší rychlejší přístup do paměti, ale zvyšuje paměťové nároky.

<http://www.catb.org/esr/structure-packing>

<https://stackoverflow.com/questions/4306186/structure-padding-and-packing>

## Struktura struct a velikost 1/2

- Při kompilaci zpravidla dochází k zarovnání prvků na velikost slova příslušné architektury.

*Např. 8 bytů v případě 64-bitové architektury.*

*Jednotlivé prvky jsou na adrese v násobNapř. 8 bytů v případě 64-bitové architektury.*

- Můžeme explicitně předeepsat kompaktní paměťovou reprezentaci, např. direktivou `__attribute__((packed))` překladačů `clang` a `gcc`.

```

struct record_packed {
    int n;
    double v;
} __attribute__((packed));

```

lec06/struct.c

## Proměnné se sdílenou pamětí – union

- **Union** je množina prvků (proměnných), které nemusí být stejného typu.
- Prvky unionu sdílejí společně stejná paměťová místa.

*Překrývají se*

- Velikost unionu je dána velikostí největšího z jeho prvků.
- Skladba unionu je definována uživatelem jako nový typ sestavený z již definovaných typů.
- K prvkům unionu se přistupuje tečkovou notací.
- Pokud nedefinujeme nový typ je nutné k identifikátoru proměnné unionu uvádět klíčové slovo `union`.

*Podobně jako u struktury `struct`.*

```

1 union Nums {
2     char c;
3     int i;
4 };
5 Nums nums; /* THIS IS NOT ALLOWED! Type Nums is not known! */
6 union Nums nums;

```

## Příklad union 1/2

- Union složený z proměnných typu: `char`, `int` a `double`.

```

1 int main(int argc, char *argv[])
2 {
3     union Numbers {
4         char c;
5         int i;
6         double d;
7     };
8     printf("size of char %lu\n", sizeof(char));
9     printf("size of int %lu\n", sizeof(int ));
10    printf("size of double %lu\n", sizeof(double));
11    printf("size of Numbers %lu\n", sizeof(union Numbers));
12
13    union Numbers numbers;
14
15    printf("Numbers c: %d i: %d d: %lf\n", numbers.c, numbers.i, numbers.d);

```

- Příklad výstupu.

```

size of char 1
size of int 4
size of double 8
size of Numbers 8
Numbers c: 48 i: 740313136 d: 0.000000

```

lec06/union.c

## Příklad union 2/2

- Proměnné sdílejí paměťový prostor.

```

1 numbers.c = 'a';
2 printf("\nSet the numbers.c to 'a'\n");
3 printf("Numbers c: %d i: %d d: %lf\n", numbers.c, numbers.i, numbers.d);
4
5 numbers.i = 5;
6 printf("\nSet the numbers.i to 5\n");
7 printf("Numbers c: %d i: %d d: %lf\n", numbers.c, numbers.i, numbers.d);
8
9 numbers.d = 3.14;
10 printf("\nSet the numbers.d to 3.14\n");
11 printf("Numbers c: %d i: %d d: %lf\n", numbers.c, numbers.i, numbers.d);

```

- Příklad výstupu

```

Set the numbers.c to 'a'
Numbers c: 97 i: 1374389601 d: 3.140000

```

```

Set the numbers.i to 5
Numbers c: 5 i: 5 d: 3.139999

```

```

Set the numbers.d to 3.14
Numbers c: 31 i: 1374389535 d: 3.140000

```

lec06/union.c

## Inicializace union

- Proměnnou typu `union` můžeme inicializovat při definici.

```

1 union {
2     char c;
3     int i;
4     double d;
5 } numbers = { 'a' };

```

*Pouze první položka (proměnná) může být inicializována.*

- V C99 můžeme inicializovat konkrétní položku (proměnnou).

```

1 union {
2     char c;
3     int i;
4     double d;
5 } numbers = { .d = 10.3 };

```

## Příklad struktura, pole a výčtový typ 1/3

- Hodnoty (konstanty) výčtového typu jsou celá čísla, která mohou být použita jako indexy (pole).
- Také je můžeme použít pro inicializaci pole struktur.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 enum weekdays { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY };
6
7 typedef struct {
8     char *name;
9     char *abbr; // abbreviation
10 } week_day_s;
11
12 const week_day_s days_en[] = {
13     [MONDAY] = { "Monday", "mon" },
14     [TUESDAY] = { "Tuesday", "tue" },
15     [WEDNESDAY] = { "Wednesday", "wed" },
16     [THURSDAY] = { "Thursday", "thr" },
17     [FRIDAY] = { "Friday", "fri" },
18 };

```

lec06/demo-struct.c

## Příklad struktura, pole a výčtový typ 2/3

- Připravíme si pole struktur pro konkrétní jazyk (angličtina a čeština).
- Program vytiskne jméno a zkratku dne v týdnu dle čísla dne v týdnu.  
V programu používáme jednotné číslo dne bez ohledu na jazykovou mutaci.

```

19 const week_day_s days_cs[] = {
20     [MONDAY] = { "Pondeli", "po" },
21     [TUESDAY] = { "Utery", "ut" },
22     [WEDNESDAY] = { "Streda", "st" },
23     [THURSDAY] = { "Ctvrtek", "ct" },
24     [FRIDAY] = { "Patek", "pa" },
25 };
26
27 int main(int argc, char *argv[], char **envp)
28 {
29     int day_of_week = argc > 1 ? atoi(argv[1]) : 1;
30     if (day_of_week < 1 || day_of_week > 5) {
31         fprintf(stderr, "(EE) File: '%s' Line: %d -- Given day of week out of range\n",
32             __FILE__, __LINE__);
33         return 101;
34     }
35     day_of_week -= 1; // start from 0
36
37     lec06/demo-struct.c

```

## Část II

## Část 2 – Vnitřní reprezentace číselných typů

## Příklad struktura, pole a výčtový typ 3/3

- Detekci národního prostředí provedeme podle hodnoty proměnné prostředí.  
Pro jednoduchost detekujeme češtinu na základě výskytu řetězce "cs" v hodnotě proměnné prostředí LC\_CTYPE.

```

35     _Bool cz = 0;
36     while (*envp != NULL) {
37         if (strstr(*envp, "LC_CTYPE") && strstr(*envp, "cs")) {
38             cz = 1;
39             break;
40         }
41         envp++;
42     }
43     const week_day_s *days = cz ? days_cs : days_en;
44
45     printf("%d %s %s\n", day_of_week,
46         days[day_of_week].name,
47         days[day_of_week].abbr
48     );
49     return 0;
50 }
51
52 lec06/demo-struct.c

```

V programu jsme využili koncept definování datových struktur, které následně programově přepínáme a využíváme. Alternativně můžeme data načítat ze souboru. V programu se snažíme obecně pracovat s datovými strukturami.

## Přesnost výpočtu 1/2

- Ztráta přesnosti při aritmetických operacích.

## Příklad sčítání dvou čísel

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     double a = 1e+10;
6     double b = 1e-10;
7
8     printf("a : %24.121f\n", a);
9     printf("b : %24.121f\n", b);
10    printf("a+b: %24.121f\n", a + b);
11
12    return 0;
13 }
14
15 clang sum.c && ./a.out
16 a : 10000000000.000000000000
17 b : 0.00000000000100
18 a+b: 10000000000.000000000000

```

lec06/sum.c

## Přesnost výpočtu 2/2

### Příklad dělení dvou čísel

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     const int number = 100;
6     double dV = 0.0;
7     float fV = 0.0f;
8
9     for (int i = 0; i < number; ++i) {
10        dV += 1.0 / 10.0;
11        fV += 1.0 / 10.0;
12    }
13
14    printf("double value: %lf ", dV);
15    printf(" float value: %lf ", fV);
16
17    return 0;
18 }
19
20 clang division.c && ./a.out
21 double value: 10.000000 float value: 10.000002          lec06/division.c
    
```

## Zdroje a typy chyby

- Chyby matematického modelu - matematická aproximace fyzikální situace.
- Chyby vstupních dat.
- Chyby numerické metody.
- Chyby zaokrouhlovací.
  
- Absolutní chyba aproximace  
 $E(x) = \hat{x} - x$ ,  $\hat{x}$  přesná hodnota,  $x$  aproximace.
- Relativní chyba  $RE(x) = \frac{\hat{x}-x}{x}$ .

## Přesnost výpočtu - strojová přesnost

- Strojová přesnost  $\epsilon_m$  - nejmenší desetinné číslo, které přičtením k 1.0 dává výsledek různý od 1, pro  $|v| < \epsilon_m$ , platí

$$v + 1.0 == 1.0.$$

*Symbol == odpovídá porovnání dvou hodnot (test na ekvivalenci).*

- Zaokrouhlovací chyba - nejméně  $\epsilon_m$ .
- Přesnost výpočtu - aditivní chyba roste s počtem operací v řádu  $\sqrt{N} \cdot \epsilon_m$ .
  - Často se však kumuluje preferabilně v jedno směru v řádu  $N \cdot \epsilon_m$ .

## Podmíněnost numerických úloh

- Podmíněnost úlohy  $C_p = \frac{\text{relativní chyba výstupních údajů}}{\text{relativní chyba vstupních údajů}}$ .
- Dobře podmíněná úloha  $C_p \approx 1$ .
- Výpočet je dobře podmíněný, je-li málo citlivý na poruchy ve vstupních datech.
- Numericky stabilní výpočet - vliv zaokrouhlovacích chyb na výsledek je malý.
- Výpočet je stabilní, je-li dobře podmíněný a numericky stabilní.

## Možnosti zvýšení přesnosti

- Reprezentace racionálních čísel - podíl dvou celočíselných hodnot, např. *Homogenní souřadnice*.
- „Libovolná přesnost“ - speciální knihovny, např. `gmp` až do výše volné paměti.  
*Souřadnice  $x,y$  - 7511164176768 346868669952 3739567104 ~ 2008,57; 92,76.*

*Informativní*

## Datové typy

- Při návrhu algoritmu abstrahujeme od binární podoby paměti počítače.
- S daty pracujeme jako s hodnotami různých datových typů, které jsou uloženy v paměti předepsaným způsobem.
- Datový typ specifikuje
  - Množinu hodnot, které je možné v počítači uložit;

*Záleží na způsobu reprezentace.*

  - Množinu operací, které lze s hodnotami typu provádět.
- **Jednoduchý typ** je takový typ, jehož hodnoty jsou atomické, tj. z hlediska operací dále nedělitelné.

## Příklady chyb

- Ariane 5 - 4.6.1996  
40 sekund po startu explodovala. Datová konverze z 64-bitového desetinné reprezentace na 16-ti bitový znaménkový integer.
- Systém Patriot - 25.2.1991  
Systémový čas v desetinách sekundy, převod na sekundy realizován dělením 10, registry pouze 24 bitů.

[http://www.esa.int/esaCP/Pr\\_33\\_1996\\_p\\_EN.html](http://www.esa.int/esaCP/Pr_33_1996_p_EN.html)

<http://www.ima.umn.edu/~arnold/disasters/patriot.html>

<http://www5.informatik.tu-muenchen.de/~huckle/bugse.html>

## Příklad číselných typů a vnitřní reprezentace

- 32-bitový typ `int` umožňuje uložit celá čísla v intervalu  $(-2147483648, 2147483647)$ , pro která můžeme použít:
    - aritmetické operace `+`, `-`, `*`, `/` s výsledkem hodnota typu `int`;
    - relační operace `==`, `!=`, `>`, `<`, `>=`, `<=`.
  - Inicializovat hodnotou dekadického nebo hexadecimálního literálu.
- ```

1 int i; // definice promenne typu int
2 int decI = 120; // definice spolu s prirazenim
3 int hexI = 0x78; //pocatecni hodnota v 16-kove soustave
4
5 int sum = 10 + decI + 0x13; //pocatecni hodnota je vyraz
```
- Vnitřní reprezentace typů (např. `int`, `short`, `double`) umožňuje uložit čísla z definovaného rozsahu s různou přesností.
  - Číselné datové typy lze vzájemně převádět implicitní nebo explicitní typovou konverzí.
  - Při konverzi nemusí být hodnota zachována – viz `lec06/demo-types.c`.



## Repräsentace dat v počítači

- V počítači není u datové položky určeno jaký konkrétní datový typ je v paměti uložen.
- Proto musíme přidělení paměti **definovat** s jakými typy dat budeme pracovat.
- Překladač tuto definici hlídá a volí odpovídající strojové instrukce pro práci s daty, např. jako s odpovídajícími číselnými typy.

*Např. neceločíselné (float) typy a využití tzv. FPU (Floating Point Unit).*

### Příklad zápisů stejného čísla v různých soustavách.

- $0100\ 0001_{(2)}$  – binární zápis jednoho bajtu (8-mi bitů);
- $65_{(10)}$  – odpovídající číslo v dekadické soustavě;
- $41_{(16)}$  – odpovídající číslo v šestnáctkové soustavě;
- Obsah paměťového místa  $0100\ 0001_{(2)}$  o velikosti 1 byte může být interpretován jako znak A.

## Kódování záporných čísel

- |                                                                                                                                        |                                    |
|----------------------------------------------------------------------------------------------------------------------------------------|------------------------------------|
| ■ <b>Přímý kód</b> – znaménko je určeno prvním bitem (zleva), snadné stanovení absolutní hodnoty. Repräsentace má dvě nuly.            | ■ $121_{(10)}$ $0111\ 1001_{(2)}$  |
|                                                                                                                                        | ■ $-121_{(10)}$ $1111\ 1001_{(2)}$ |
|                                                                                                                                        | ■ $0_{(10)}$ $0000\ 0000_{(2)}$    |
|                                                                                                                                        | ■ $-0_{(10)}$ $1000\ 0000_{(2)}$   |
| ■ <b>Inverzní kód</b> – záporné číslo odpovídá bitové negaci kladné hodnoty čísla. Repräsentace má dvě nuly.                           | ■ $121_{(10)}$ $0111\ 1001_{(2)}$  |
|                                                                                                                                        | ■ $-121_{(10)}$ $1000\ 0110_{(2)}$ |
|                                                                                                                                        | ■ $0_{(10)}$ $0000\ 0000_{(2)}$    |
|                                                                                                                                        | ■ $-0_{(10)}$ $1111\ 1111_{(2)}$   |
| ■ <b>Doplňkový kód</b> – záporné číslo je uloženo jako hodnota kladného čísla po bitové negaci zvětšená o 1. Jediná repräsentace nuly. | ■ $121_{(10)}$ $0111\ 1001_{(2)}$  |
|                                                                                                                                        | ■ $-121_{(10)}$ $1000\ 0110_{(2)}$ |
|                                                                                                                                        | ■ $-121_{(10)}$ $1000\ 0111_{(2)}$ |
|                                                                                                                                        | ■ $127_{(10)}$ $0111\ 1111_{(2)}$  |
|                                                                                                                                        | ■ $-128_{(10)}$ $1000\ 0000_{(2)}$ |
|                                                                                                                                        | ■ $-1_{(10)}$ $1111\ 1111_{(2)}$   |

## Číselné soustavy

- Číselné soustavy – poziční číselné soustavy (polyadické) jsou charakterizovány bází udávající kolik číslic lze maximálně použít.

$$x_d = \sum_{i=-n}^{i=m} a_i \cdot z^i, \text{ kde } a_i \text{ je číslice a } z \text{ je základ soustavy.}$$

- Unární – např. počet vypitých půllitrů.
- Binární soustava (bin) – 2 číslice 0 nebo 1.

$$\begin{aligned} 11010,01_{(2)} &= 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} \\ &= 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 + 0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{4} \\ &= 26,25 \end{aligned}$$

- Desítková soustava (dec) – 10 číslic, znaky 0 až 9.

$$\begin{aligned} 138,24_{(10)} &= 1 \cdot 10^2 + 3 \cdot 10^1 + 8 \cdot 10^0 + 2 \cdot 10^{-1} + 4 \cdot 10^{-2} \\ &= 1 \cdot 100 + 3 \cdot 10 + 8 \cdot 1 + 2 \cdot 0,1 + 4 \cdot 0,01 \end{aligned}$$

- Šestnáctková soustava (hex) – 16 číslic, znaky 0 až 9 a A až F.

$$\begin{aligned} 0x7D_{(16)} &= 7 \cdot 16^1 + D \cdot 16^0 \\ &= 112 + 13 \\ &= 125 \end{aligned}$$

## Více-bajtová repräsentace a pořadí bajtů

- Číselné typy s více-bajtovou repräsentací mohou mít bajty uloženy v různém pořadí.
  - *little-endian* – **nejméně** významný bajt se ukládá na nejnižší adresu. *x86, ARM*
  - *big-endian* – **nejvíce** významný bajt se ukládá na nejnižší adresu. *Motorola, ARM*

- Pořadí je důležité při přenosu hodnot z paměti jako posloupnosti bajtů a jejich následné interpretaci.

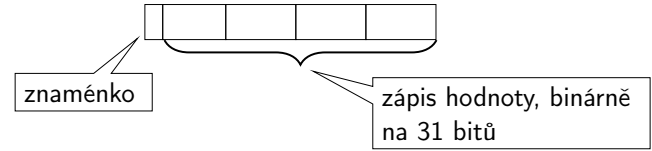
- **Network byte order** – je definován pro síťový přenos a není tak nutné řešit konkrétní architekturu.
  - Tj. hodnoty z paměti jsou ukládány a přenášeny v tomto pořadí bajtů a na cílové stanici pak zpětně zapsány do konkrétního nativního pořadí.

*big-endian*

*Informativní*

### Příklad reprezentace celých čísel **int**

- Na 32-bitových a 64-bitových strojích je celočíselný typ **int** zpravidla reprezentován 32 bity (4 byty).



- Typ **int** je znaménkový typ.
- Znaménko je zakódováno v 1 bitu a vlastní číselná hodnota pak ve zbývajících 31 bitech.
  - Největší číslo je  $0111 \dots 111 = 2^{31} - 1 = 2\,147\,483\,647$ . *Reprezentujeme i nulu.*
  - Nejmenší číslo je  $-2^{31} = -2\,147\,483\,648$ . *0 už je zahrnuta.*
- Pro zobrazení záporných čísel je použit **doplňkový kód**.  
Nejmenší číslo v doplňkovém kódu 1000...000 je  $-2^{31}$ .

### Repräsentace reálných čísel

- Pro uložení čísla vyhradzujeme omezený paměťový prostor.
- Příklad – zápis čísla  $\frac{1}{3}$  v dekadické soustavě**
  - $= 33333333 \dots 3333$
  - $= 0,3\bar{3}$
  - $\approx 0,33333333333333333333$
  - $\approx 0,333$

*V trojkové soustavě:  $0 \cdot 3^1 + 0 \cdot 3^0 + 1 \cdot 3^{-1} = (0,1)_3$*
- Nepřesnosti v zobrazení reálných čísel v konečné posloupnosti bitů způsobují
  - Iracionální čísla, např.  $e, \pi, \sqrt{2}$ ;
  - Čísla, která mají v dané soustavě periodický rozvoj, např.  $\frac{1}{3}$ ;
  - Čísla, která mají příliš dlouhý zápis.

### Repräsentace záporných celých čísel

- Doplňkový kód –  $D(x)$ .
- Pro 8-mi bitovou reprezentací čísel.
  - Můžeme reprezentovat  $2^8=256$  čísel.
  - Rozsah  $r = 256$ .

$$D(x) = \begin{cases} x & \text{pro } 0 \leq x < \frac{r}{2} \\ r + x & \text{pro } -\frac{r}{2} \leq x < 0 \end{cases} \quad (1)$$

- Příklady

| Desítkově | Doplňkový kód                                  |
|-----------|------------------------------------------------|
| 0–127     | 0000 0000 – 0111 1111                          |
| 128       | nelze zobrazit na 8 bitů v doplňkovém kódu     |
| -128      | $D(-128) = 256 + (-128) = 128$ to je 1000 0000 |
| -1        | $D(-1) = 256 + (-1) = 255$ to je 1111 1111     |
| -4        | $D(-4) = 256 + (-4) = 252$ to je 1111 1100     |

*Informativní*

### Model repräsentace reálných čísel

- Reálná čísla se zobrazují jako aproximace daným rozsahem paměťového místa.
- Reálné číslo  $x$  se zobrazuje ve tvaru

$$x = \text{mantisa} \cdot \text{základ}^{\text{exponent}}$$

- Pro jednoznačnost zobrazení musí být mantisa normalizována, např.  $0,1 \leq m < 1$  nebo ve tvaru  $\pm 1.[\text{mantisa}] \cdot 2^{\text{exponent}}$
- Ve vyhrazeném paměťovém prostoru je pro zvolený základ uložen exponent a mantisa jako dvě celá čísla.



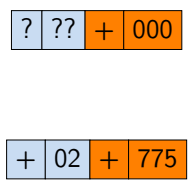
## Příklad modelu reprezentace reálných čísel 1/2

### Repräsentace na 7 bajtů se základem 10

- Délka mantisy 3 pozice (bajtů) plus znaménko.
- Délka exponentu 2 pozice plus znaménko.
- Základ  $z = 10$ .
- Repräsentace nuly.

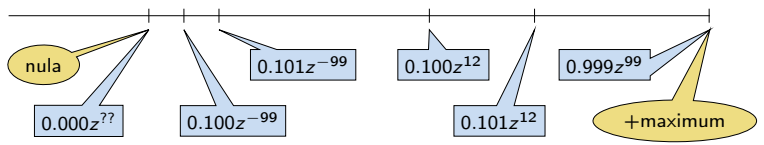
Repräsentace dle IEEE-754 používá dvojkový základ

- Příklad  $x = 77,5 = 0,775 \cdot z^{+02}$



## Model reprezentace reálných čísel a vzdálenost mezi aproximacemi

- Rozsah hodnot pro konkrétní exponent je dán velikostí mantisy.
- Absolutní vzdálenost dvou aproximací tak záleží na exponentu.
  - Mezi hodnotou 0 a 1,0 je využít celý rozsah mantisy pro exponenty  $\{-99, -98, \dots, 0\}$ .



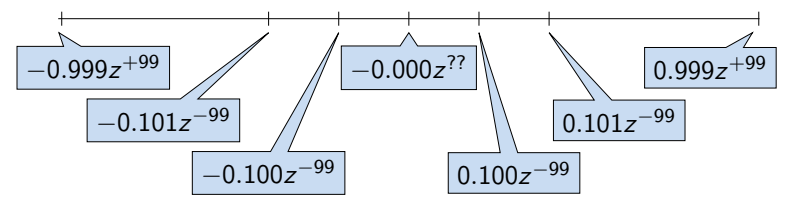
- Aproximace reálných čísel nejsou na číselné ose rovnoměrně rozloženy.



Čím větší exponent, tím větší „mezery“ mezi sousedními aproximacemi čísel.

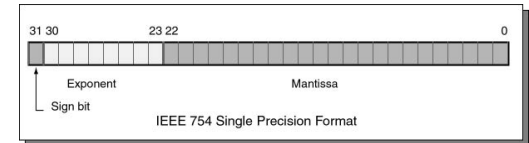
## Příklad modelu reprezentace reálných čísel 2/2 – Limitní zobrazitelná čísla

- Maximální zobrazitelné kladné číslo  $0,999z^{99}$ .
- Maximální zobrazitelné záporné číslo  $-0,100z^{-99}$ .
- Minimální zobrazitelné kladné číslo  $0,100z^{-99}$ .
- Minimální zobrazitelné záporné číslo  $-0,999z^{+99}$ .



## Repräsentace necelých čísel – IEEE 754

- Reálné číslo  $x$  se zobrazuje ve tvaru  $x = (-1)^s \text{mantisa} \cdot 2^{\text{exponent} - \text{bias}}$ . Základ 2.  
IEEE 754, ISO/IEC/IEEE 60559:2011
- Mantisa je **normalizována** na první jedničku vlevo (v soustavě o dvojkovém základu).
- **float** – 32 bitů (4 bajty): **s** – 1 bit znaménko (+ nebo –), **exponent** – 8 bitů, tj. 256 možností. **mantisa** – 23 bitů  $\approx 16,7$  milionu možností.



- **double** – 64 bitů (8 bajtů).
  - **s** – 1 bit znaménko (+ nebo –).
  - **exponent** – 11 bitů, tj. 2048 možností.
  - **mantisa** – 52 bitů  $\approx 4,5$  bilióny možností (4 503 599 627 370 495).
- **bias** umožňuje reprezentovat exponent vždy jako kladné číslo.

Lze zvolit, např.  $\text{bias} = 2^{eb-1} - 1$ , kde  $eb$  je počet bitů exponentu.  
<http://www.root.cz/clanky/norma-ieee-754-a-pribuzni-formaty-plovouci-radove-tecky>

## Příklad reprezentace float hodnot dle IEEE 754

- Chyba reprezentace -256.75 vs -256.74.
- Infinity (0x7f800000), -Infinity (0xff800000), a NaN (0x7fffffff).

<https://www.h-schmidt.net/FloatConverter/IEEE754.html>  
 B0B36PRP – Přednáška 06: Struktury, unieony a číselné typy

## Přřazovací operátor a příkaz

- Slouží pro nastavení hodnoty proměnné.
- Tvar přřazovacího operátoru.

*Uložení číselné hodnoty do paměti, kterou proměnná reprezentuje.*

$$\langle \text{proměnná} \rangle = \langle \text{výraz} \rangle$$

*Výraz je literál, proměnná, volání funkce, ...*

- Zkrácený zápis

$$\langle \text{proměnná} \rangle \langle \text{operátor} \rangle = \langle \text{výraz} \rangle$$

- Přřazení je výraz asociativní zprava.

- Přřazovací příkaz – výraz zakončený středníkem ;

```
int x; //definice promenne x
int y; //definice promenne y

x = 6;
y = x = x + 6;

int x, y; //definice promennych x a y

x = 10;
y = 7;
y += x + 10;
```

## Přřklady reprezentace hodnot typu float

### Repräsentace čísla 85,125 (float)

- 85 odpovídá  $1010101_{(2)}$ .
- 0,125 odpovídá 001
  - $0,125/2^{-1} = 0,25 \quad | \quad 0$
  - $0,125/2^{-2} = 0,50 \quad | \quad 0$
  - $0,125/2^{-3} = 1,00 \quad | \quad 1$
- 85,125 odpovídá  $1010101,001_{(2)} = 1,010101001_{(2)} \times 2^6$ ,
- Bias pro float je 127.
- Exponet je  $127 + 6 = 133$
- 133 odpovídá  $1000101_{(2)}$ .
- Normalizovaná mantisa je  $010101001_{(2)}$ , kterou doplníme nulami na 23 bitů (zprava, je to desetinné číslo).
- 0 - 1000 0101 - 0101 0100 1000 0000 0000 000.
- 01000010 10101010 01000000 00000000.
- V šestnáctkové soustavě 0x42 0xaa 0x40 0x00, tedy 0x42aa4000.

### Repräsentace čísla 0,1 (float)

- 0,1 má periodický rozvoj
  - 0,1 \* 2 = 0,2 | 0
  - 0,2 \* 2 = 0,4 | 0
  - 0,4 \* 2 = 0,8 | 0
  - 0,8 \* 2 = 1,6 | 1
  - 0,6 \* 2 = 1,2 | 1
  - 0,2 \* 2 = 0,4 | 0
- Opakuje se 0011, 23-bitů tak reprezentuje menší hodnotu.
- $0,1_{(10)} \sim 0,000110011001100110011001100_{(2)} = 1,10011001100110011001100_{(2)} \times 2^{-4}$ .
- Exponet je  $127 - 4 = 123$  odpovídá  $01111011_{(2)}$ .
- Normalizovaná mantisa  $\pm 10011001100110011001100$ .
- 0 - 0111 1011 - 100 1100 1100 1100 1100 1100.
- 00111101 11001100 11001100 11001100.
- V šestnáctkové soustavě to je 0x3d 0xcc 0xcc 0xcc, tedy 0x3dcccc.
- Prakticky je 0,1 převedeno na o něco větší číslo 0x3dccccd, protože absolutní chyba je menší.

[lec06/floats.c](http://lec06/floats.c)

## Typové konverze

- Typová konverze je operace převedení hodnoty nějakého typu na hodnotu typu jiného.
- Typová konverze může být
  - implicitní – vyvolá se automaticky;
  - explicitní – je nutné v programu explicitně uvést.

- Konverze typu int na double je implicitní.

*Hodnota typu int může být použita ve výrazu, kde se očekává hodnota typu double, dojde k automatickému převodu na hodnotu typu double.*

### Přřklad

```
double x;
int i = 1;

x = i; // hodnota 1 typu int se automaticky převede
// na hodnotu 1.0 typu double
```

- Implicitní konverze je bezpečná.

## Explicitní typové konverze

- Převod hodnoty typu **double** na **int** je třeba **explicitně** předeepsat.
- Dojde k „odseknutí“ necelé části hodnoty int.

### Příklad

```
double x = 1.2; // definice proměnné typu double
int i; // definice proměnné typu int
int i = (int)x; // hodnota 1.2 typu double se převede
// na hodnotu 1 typu int
```

- Explicitní konverze je potenciálně nebezpečná.

### Příklady

```
double d = 1e30; // i je -2147483648 // to je asi -2e9 místo 1e30
int i = (int)d;

long l = 5000000000L; // i je 705032704 // (oříznuté 4 bajty)
int i = (int)l;
lec06/demo-type_conversion.c
```

## Matematické funkce

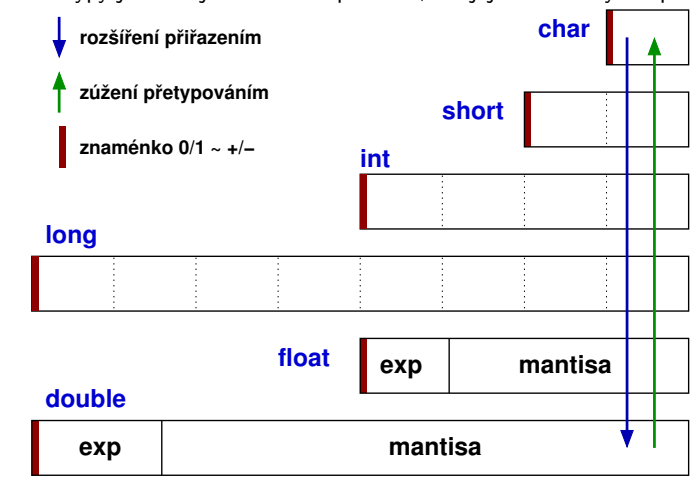
- `<math.h>` – základní funkce pro práci s „reálnými“ čísly.
  - Výpočet odmocniny necelého čísla  $x$ .
 

```
double sqrt(double x);, float sqrtf(float x);
```

*V C funkce nepřetěžujeme, proto jsou jména odlišena.*
  - `double pow(double x, double y);` – výpočet obecné mocniny.
  - `double atan2(double y, double x);` – výpočet  $\arctan y/x$  s určením kvadrantu.
  - Symbolické konstanty – `M_PI`, `M_PI_2`, `M_PI_4`, atd.
    - `#define M_PI 3.14159265358979323846`
    - `#define M_PI_2 1.57079632679489661923`
    - `#define M_PI_4 0.78539816339744830962`
  - `isfinite()`, `isnan()`, `isless()`, ... – makra pro porovnání reálných čísel.
  - `round()`, `ceil()`, `floor()` – zaokrouhlování, převod na celá čísla.
- `<complex.h>` – funkce pro počítání s komplexními čísly. ISO C99
- `<fenv.h>` – funkce pro řízení zaokrouhlování a reprezentaci dle IEEE 754. man math

## Konverze primitivních číselných typů

- Primitivní datové typy jsou vzájemně nekompatibilní, ale jejich hodnoty lze převádět.



## Část III

### Část 3 – Zadání 6. domácího úkolu (HW06)

## Zadání 6. domácího úkolu HW06

Téma: **Maticové počty**Povinné zadání: **2b**; Volitelné zadání: **3b**; Bonusové zadání: **3b**

- **Motivace:** Získání zkušenosti s dvojrozměrným polem.
- **Cíl:** Osvojit si práci s polem variabilní délky a předávání ukazatelů.
- **Zadání:** <https://cw.fel.cvut.cz/wiki/courses/b0b36prp/hw/hw06>
  - Načtení vstupních hodnot dvou matic a znaku operace (`'*'` – násobení).
  - **Volitelné zadání** rozšiřuje úlohu o další operace s maticemi sčítání (`'+'`) a odčítání (`'-'`), které mohou být zapsány ve výrazu.
  - **Bonusové zadání** pak řeší zpracování celého výrazu, ve kterém jsou však jednotlivé matice uvedeny jako symboly, které jsou nejdříve definovány načtením hodnot matic ze standardního vstupu.

Využití `struct` a dynamické alokace může být výhodnou, není však nutné.

- **Termín odevzdání:** 26.11.2022, 23:59:59 PST.
- **Bonusová úloha:** 14.01.2023, 23:59:59 PST.

PST – Pacific Standard Time

## Shrnutí přednášky

## Diskutovaná témata

- Struktury, způsoby definování, inicializace a paměťové reprezentace
- Uniony
- Přesnost výpočtu
- Vnitřní paměťová reprezentace celočíselných i neceločíselných číselných typů
- Knihovna `math.h`
- **Příště:** Standární knihovny C. Rekurze.