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

Přehled témat

- Část 1 – Struktury a uniony

  Struktury – `struct`

  Proměnné se sdílenou pamětí – `union`

  S. G. Kochan: kapitola 9 a 17

  P. Herout: kapitola 14

- Část 2 – Přesnost výpočetů 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)

  P. Herout: kapitola 7 (typové konverze)

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

Část I

Část 1 – Struktury a uniony

Obsah

Struktury – struct

Proměnné se sdílenou pamětí – union

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 **přistupujeme tečkovou notací**
- K prvkům můžeme přistupovat přes ukazatel operátorem `->`
- Pro struktury stejného typu je definována operace přiřazení

```
struct1 = struct2;
```

Pro proměnné typu pole není přímé přiřazení definováno, přiřazení pole je tak nutné realizovat po prvcích.

- 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

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

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

```
record r; /* THIS 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 můžeme používat typ struktury již bez uvádění klíčového slova struct

lec06/struct.c

Definice jména struktury a typu struktury

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

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

  - Definujeme idenfikátor `record` ve jmeném prostoru struktur

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

```
typedef struct record record;
```

  - Definujeme globální idenfikátor `record` jako jméno typu `struct record`

Jedná se vlastně o „alias“

- Obojí můžeme kombinovat v jediné definici typu a struktury

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

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

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 ukazatelem

```
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
  - **ukazatelem** se kopíruje pouze hodnota ukazatele (adresa) a pracujeme tak s původní strukturou

lec06/struct.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; /* THIS IS NOT ALLOWED! */
print_record(rec1); /* number(5), value(13.100000) */
```

lec06/struct.c

Příklad struct – Přímá kopie paměti

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

Například funkcí 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

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 1/2

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

  Např. 8 bytů v případě 64-bitové architektury.

- Můžeme explicitně předepsat kompaktní paměťovou reprezentaci, např. direktivou `__attribute__((packed))` pro překladače clang a gcc

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

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

Obsah

Struktury – struct

Proměnné se sdílenou pamětí – union

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`

```c
union Nums {
   char c;
   int i;
};
Nums nums; /* THIS IS NOT ALLOWED! Type Nums is not known! */
union Nums nums;
```

Příklad union 1/2

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

```
int main(int argc, char *argv[])
{
    union Numbers {
        char c;
        int i;
        double d;
    };
    printf("size of char %lu\n", sizeof(char));
    printf("size of int %lu\n", sizeof(int ));
    printf("size of double %lu\n", sizeof(double));
    printf("size of Numbers %lu\n", sizeof(union Numbers));

    union Numbers numbers;

    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

```
numbers.c = 'a';
printf("\nSet the numbers.c to 'a'\n");
printf("Numbers c: %d i: %d d: %lf\n", numbers.c, numbers.i,
    numbers.d);

numbers.i = 5;
printf("\nSet the numbers.i to 5\n");
printf("Numbers c: %d i: %d d: %lf\n", numbers.c, numbers.i,
    numbers.d);

numbers.d = 3.14;
printf("\nSet the numbers.d to 3.14\n");
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

Část II

Část 2 – Vnitřní reprezentace číselných typů

Obsah

Přesnost výpočtu 1/2

- Ztráta přesnosti při aritmetických operacích.

Příklad sčítání dvou čísel

```c
#include <stdio.h>

int main(void)
{
   double a = 1e+10;
   double b = 1e-10;

   printf("a  : %24.12lf\n", a);
   printf("b  : %24.12lf\n", b);
   printf("a+b: %24.12lf\n", a + b);

   return 0;
}

clang sum.c && ./a.out
a  : 10000000000.000000000000
b  :           0.000000000100
a+b: 10000000000.000000000000
```

lec06/sum.c

Přesnost výpočtu 2/2

Příklad dělení dvou čísel

```
#include <stdio.h>

int main(void)
{
   const int number = 100;
   double dV = 0.0;
   float fV = 0.0f;

   for (int i = 0; i < number; ++i) {
      dV += 1.0 / 10.0;
      fV += 1.0 / 10.0;
   }

   printf("double value: %lf ", dV);
   printf(" float value: %lf ", fV);

   return 0;
}

clang division.c && ./a.out
double value: 10.000000  float value: 10.000002
```

lec06/division.c

Přesnost výpočtu - strojová přesnost

- Strojová přesnost ϵ_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 v Javě (test na ekvivalenci).

- Zaokrouhlovací chyba - nejméně ϵ_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 $N \cdot \epsilon_m$.

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 $RE(x) = \frac{\hat{x}-x}{x}$.

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 $\sim$ *2008.57, 92.76*

Informativní

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.

  http://www.esa.int/esaCP/Pr_33_1996_p_EN.html

- Systém Patriot - 25.2.1991
  Systémový čas v desetinách sekundy, převod na sekundy realizován dělením 10, registry pouze 24 bitů.

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

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

Obsah

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

Příklad číselných typů a vnitřní reprezentace

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

```c
int i; //deklarace promenne typu int
int decI = 120;  //deklarace spolu s prirazenim
int hexI = 0x78; //pocatecni hodnota v 16-kove soustave

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`

Reprezentace dat v počítači

- V počítači není u datové položky určeno jaký konkrétní datový typ je v paměti uložen
- Proto musíme přidělení paměti **deklarovat** s jakými typy dat budeme pracovat
- Překladač pak tuto deklaraci hlídá a volí odpovídající strojové instrukce pro práci s datovými položkami například jako s odpovídajícími číselnými typy

Např. neceločíselné (float) typy a využití tzv. FPU

Příklad ekvivalentních reprezentací v paměti počítače

- $(0100\ 0001)_2$ – binární zápis jednoho bajtu (8-mi bitů);
- $(65)_{10}$ – odpovídající číslo v dekadické soustavě;
- $(41)_{16}$ – odpovídající číslo v šestnáctkové soustavě;
- znak A – tentýž obsah paměťového místa $(0100\ 0001)_2$ o velikosti 1 byte může být interpretován také jako znak A.

Reprezentace dat v počítači

- V počítači není u datové položky určeno jaký konkrétní datový typ je v paměti uložen
- Proto musíme přidělení paměti **deklarovat** s jakými typy dat budeme pracovat
- Překladač pak tuto deklaraci hlídá a volí odpovídající strojové instrukce pro práci s datovými položkami například jako s odpovídajícími číselnými typy

Např. neceločíselné (float) typy a využití tzv. FPU

Příklad ekvivalentních reprezentací v paměti počítače

- $(0100\ 0001)_2$ – binární zápis jednoho bajtu (8-mi bitů);
- $(65)_{10}$ – odpovídající číslo v dekadické soustavě;
- $(41)_{16}$ – odpovídající číslo v šestnáctkové soustavě;
- znak A – tentýž obsah paměťového místa $(0100\ 0001)_2$ o velikosti 1 byte může být interpretován také jako znak A.

Obsah

Reprezentace celých čísel

- Číselné soustavy – poziční číselné soustavy (polyadické) jsou charakterizovány bází udávající kolik číslic lze maximálně použít $x_d = \sum_{i=-n}^{i=m} a_i \cdot z^i$, kde a_i je číslice a z je základ soustavy
- Unární – např. počet vypitých půllitrů
- Binární soustava (bin) – 2 číslice 0 nebo 1

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

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

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

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

$$\begin{aligned} 0x7D_h &= 7\cdot 16^1 + D\cdot 16^0 \\ &= 112 + 13 \\ &= 125 \end{aligned}$$

Reprezentace celých čísel

- Číselné soustavy – poziční číselné soustavy (polyadické) jsou charakterizovány bází udávající kolik číslic lze maximálně použít
  $x_d = \sum_{i=-n}^{i=m} a_i \cdot z^i$, kde a_i je číslice a z je základ soustavy
- Unární – např. počet vypitých půllitrů
- Binární soustava (bin) – 2 číslice 0 nebo 1

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

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

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

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

$$\begin{aligned} 0x7D_h &= 7\cdot 16^1 + D\cdot 16^0 \\ &= 112 + 13 \\ &= 125 \end{aligned}$$

Reprezentace celých čísel

- Číselné soustavy – poziční číselné soustavy (polyadické) jsou charakterizovány bází udávající kolik číslic lze maximálně použít $x_d = \sum_{i=-n}^{i=m} a_i \cdot z^i$, kde a_i je číslice a z je základ soustavy
- Unární – např. počet vypitých půllitrů
- Binární soustava (bin) – 2 číslice 0 nebo 1

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

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

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

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

$$\begin{aligned} 0x7D_h &= 7\cdot 16^1 + D\cdot 16^0 \\ &= 112 + 13 \\ &= 125 \end{aligned}$$

Reprezentace celých čísel

- Číselné soustavy – poziční číselné soustavy (polyadické) jsou charakterizovány bází udávající kolik číslic lze maximálně použít $x_d = \sum_{i=-n}^{i=m} a_i \cdot z^i$, kde a_i je číslice a z je základ soustavy
- Unární – např. počet vypitých půllitrů
- Binární soustava (bin) – 2 číslice 0 nebo 1

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

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

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

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

$$
\begin{aligned}
0x7D_h &= 7\cdot 16^1 + D\cdot 16^0\\
&= 112 + 13\\
&= 125
\end{aligned}
$$

Reprezentace celých čísel

- Číselné soustavy – poziční číselné soustavy (polyadické) jsou charakterizovány bází udávající kolik číslic lze maximálně použít $x_d = \sum_{i=-n}^{i=m} a_i \cdot z^i$, kde a_i je číslice a z je základ soustavy
- Unární – např. počet vypitých půllitrů
- Binární soustava (bin) – 2 číslice 0 nebo 1

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

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

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

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

$$\begin{aligned} 0x7D_h &= 7\cdot 16^1 + D\cdot 16^0 \\ &= 112 + 13 \\ &= 125 \end{aligned}$$

Více-bajtová reprezentace a pořadí bajtů

- Číselné typy s více-bajtovou reprezentací 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

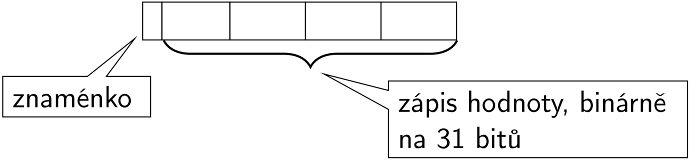
    Motorola, ARM
- Pořadí je důležité při přenosu hodnot z paměti jako posloupnosti bajtů a jejich následné interpretaci
- **Network byte order** – je definován pro síťový přenos a není tak nutné řešit konkrétní architekturu
  - Tj. hodnoty z paměti jsou ukládány a přenášeny v tomto pořadí bajtů a na cílové 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 $0111\ldots111 = 2^{31}-1 = 2147483647$

    Nezapomínat na 0
  - Nejmenší číslo je $-2^{31}{=}{-}2147483648$

    0 už je zahrnuta
- Pro zobrazení záporných čísel je použit tzv. doplňkový kód
  Nejmenší číslo v doplňkovém kódu $1000\ldots000$ je -2^{31}

Reprezentace záporných celých čísel

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

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

- 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

Obsah

Reprezentace reálných čísel

- Pro uložení čísla vyhrazujeme omezený paměťový prostor

  Příklad – zápis čísla $\frac{1}{3}$ v dekadické soustavě

  - $= 33333333\ldots3333$
  - $= 0,3\overline{3}$
  - $\approx 0,33333333333333333333$
  - $\approx 0,333$

  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, π, $\sqrt{2}$
  - Čísla, která mají v dané soustavě periodický rozvoj, např. $\frac{1}{3}$
  - Čísla, která mají příliš dlouhý zápis

Model reprezentace reálných čísel

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

$$\begin{aligned} x &= \textbf{mantisa} \cdot \textbf{základ}^{\textbf{exponent}} \\ x &= m \cdot z^{exponent} \end{aligned}$$

- Pro jednoznačnost zobrazení musí být mantisa normalizována

$$0,1 \leq m < 1$$

- Ve vyhrazeném paměťovém prostoru je pro zvolený základ uložen exponent a mantisa jako dvě celá čísla

exponent	mantisa

Příklad modelu reprezentace reálných čísel 1/2

Reprezentace na 7 bajtů

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

?	??	+	000

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

+	02	+	775

Příklad modelu reprezentace reálných čísel 2/2

Limitní zobrazitelná čísla

- Maximální zobrazitelné kladné číslo $0,999z^{99}$

+	99	+	999

- Maximální zobrazitelné záporné číslo $-0,100z^{-99}$

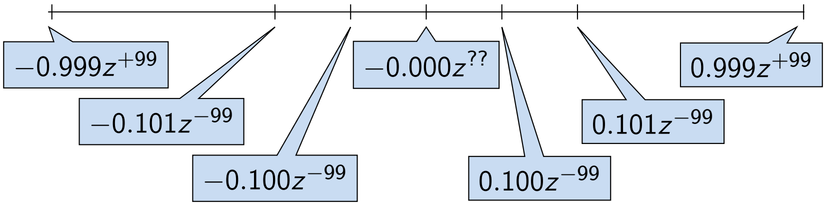
−	99	−	100

- Minimální zobrazitelné kladné číslo $0,100z^{-99}$

−	99	+	100

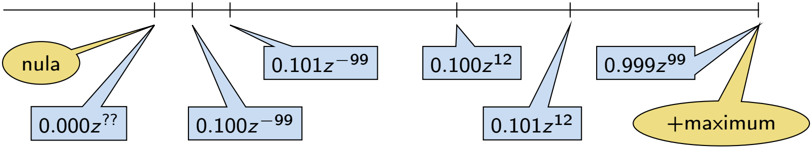
- Minimální zobrazitelné záporné číslo $-0,999z^{+99}$

+	99	−	999



Model reprezentace reálných čísel a vzdálenost mezi aproximacemi

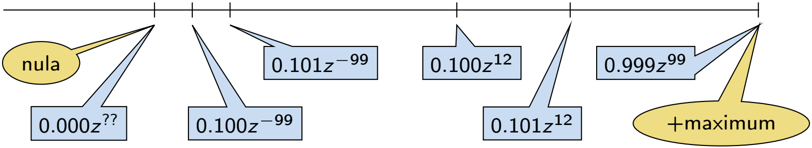
- Rozsah hodnot pro konkrétní exponent je dán velikostí mantisy
- Absolutní vzdálenost dvou aproximací tak záleží na exponentu
  - Mezi hodnotou 0 a 1,0 je využít celý rozsah mantisy pro exponenty $\{-99, -98, \ldots, 0\}$



- Aproximace reálných čísel nejsou na číselné ose rovnoměrně rozloženy

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 $\{-99, -98, \ldots, 0\}$



- Aproximace reálných čísel nejsou na číselné ose rovnoměrně rozloženy

Typ **double** – reprezentace necelých čísel

- **double** – 64 bitů (8 bajtů), norma IEEE 754

ISO/IEC/IEEE 60559:2011

- s – 1 bit znaménko ($+$ nebo $-$)
- **exponent** – 11 bitů, tj. 2048 možností
- **mantisa** – 52 bitů $\approx$ 4.5 biliardy možností

4 503 599 627 370 496

  - Neumožňuje přesně uložit čísla se zápisem delším než 52 bitů
  - Čím větší exponent, tím větší „mezery" mezi sousedními aproximacemi čísel
- Reálné číslo x se zobrazuje ve tvaru

$$x = (-1)^s \mathbf{mantisa} \cdot 2^{\mathbf{exponent}-\mathbf{bias}}$$

- **bias** umožňuje reprezentovat exponent vždy jako kladné číslo

Lze zvolit, např. 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

Obsah

Přiřazovací operátor a příkaz

- Slouží pro nastavení hodnoty proměnné

  Uložení číselné hodnoty do paměti, kterou proměnná reprezentuje
- Tvar přiřazovacího operátoru

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

  Výraz je literál, proměnná, volání funkce, ...
- Zkrácený zápis

  $$\langle \textbf{proměnná} \rangle \; \langle \textbf{operátor} \rangle = \langle \textbf{výraz} \rangle$$
- Přiřazení je výraz
  - Asociativní zprava
- Přiřazovací příkaz – výraz zakončený středníkem ;

```
int x; //deklarace
    promenne x
int y; //deklarace
    promenne y

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

```
int x, y; //deklarace
    promennych x a y

x = 10;
y = 7;

y += x + 10;
```

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`.

  Příklad

  ```
  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* předepsat
- Dojde k „odseknutí" necelé části hodnoty `int`

Příklad

```
double x = 1.2; // deklarace proměnné typu double
int i;          // deklarace 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;
int i = (int)d;

// i je -2147483648
// to je asi -2e9 misto 1e30
```

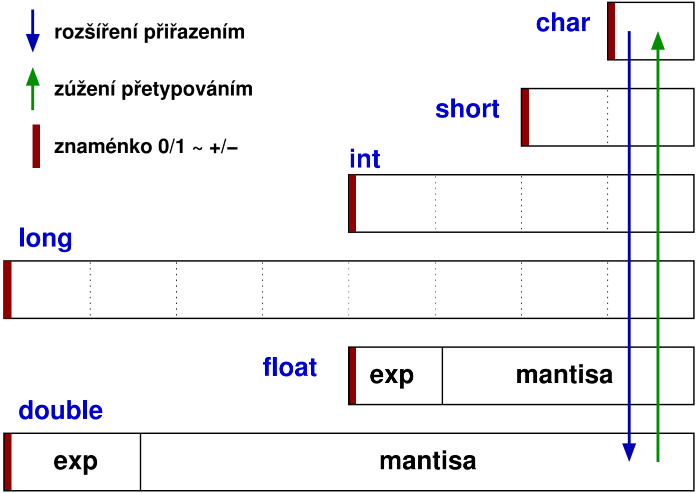
```
long l = 5000000000L;
int i = (int)l;

// i je 705032704
// (oříznuté 4 bajty)
```

lec06/demo-type_conversion.c

Konverze primitivních číselných typů

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



Obsah

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.

Část III

Část 3 – Zadání 6. domácího úkolu (HW06)

Zadání 6. domácího úkolu HW06

Téma: Caesarova šifra

Povinné zadání: **4b**; Volitelné zadání: **4b**; Bonusové zadání: není

- **Motivace**: Získat zkušenosti s dynamickou alokací paměti. Implementovat výpočetní úlohu optimalizačního typu.
- **Cíl**: Osvojit si práci s dynamickou alokací paměti
- **Zadání**: https://cw.fel.cvut.cz/wiki/courses/b0b36prp/hw/hw06
  - Načtení dvou vstupních textu a tisk dekódované zprávy na výstup
  - Zakódovaný text i (špatně) odposlechnutý text mají stejné délky
  - Nalezení největší shody dekódovaného a odposlechnutého textu na základě hodnoty posunu v Caesarově šifře
  - Optimalizace hodnoty Hammingovy vzdálenosti

    https://en.wikipedia.org/wiki/Hamming_distance
  - Volitelné zadání rozšiřuje úlohu o uvažování chybějících znaků v odposlechnutém textu, což vede na využití Levenštejnovy vzdálenosti.

    https://en.wikipedia.org/wiki/Levenshtein_distance
- **Termín odevzdání**: 26.11.2016, 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 celočíslených i neceločíselných číselných typů
- Knihovna `math.h`

- Příště:
  - Přednáška x67 - Vyzvané téma (přednáška pouze v úterý 15.11.2016)
  - Přednáška 7 - Standarní knihovny C. Rekurze.

Diskutovaná témata

- Struktury, způsoby definování, inicializace a paměťové reprezentace
- Uniony
- Přesnost výpočtu
- Vnitřní paměťová reprezentace celočíslených i neceločíselných číselných typů
- Knihovna `math.h`

- Příště:
  - Přednáška x67 - Vyzvané téma (přednáška pouze v úterý 15.11.2016)
  - Přednáška 7 - Standarní knihovny C. Rekurze.