

## 8. Vnitřní reprezentace datových typů

B0B99PRPA – Procedurální programování

Stanislav Vítek

Katedra radioelektroniky  
Fakulta elektrotechnická  
České vysoké učení v Praze

# Přehled témat

---

- Část 1 – Pole v dynamické paměti
  - 2D pole v dynamické paměti
  - Struktura s ukazatelem na pole
- Část 2 – Vnitřní reprezentace datových typů
  - Přesnost výpočtů
  - Základní číselné typy a jejich reprezentace v počítači
  - Reprezentace celých čísel
  - Reprezentace reálných čísel
- Část 3 – Standardní knihovny
  - Standardní knihovny
  - Matematické funkce
  - Práce se soubory
  - Zpracování chyb
- Část 4 – Zadání 7. domácího úkolu

# Část I

## Pole v dynamické paměti

# I. Pole v dynamické paměti

---

2D pole v dynamické paměti

Struktura s ukazatelem na pole

## 2D pole v dynamické paměti

---

- Pole ukazatelů na jednotlivé řádky pole
  - nelze měnit počet řádků
  - délka řádků může být různá

```
1 | int *p[2];  
2 | /* prvni radek */  
3 | p[0] = (int *)malloc (3*sizeof(int));  
4 | /* druhy radek */  
5 | p[1] = (int *)malloc (3*sizeof(int));
```

## 2D pole v dynamické paměti

---

- Ukazatel na ukazatel
  - lze měnit počet řádků délka řádků může být různá

```
1 | int ** p;  
2 | p = (int **)malloc (2*sizeof(int));  
3 | p[0] = (int *)malloc (3*sizeof(int));  
4 | p[1] = (int *)malloc (3*sizeof(int));
```

- Ukazatel na N-prvkové pole
  - ekvivalent pole ve statické paměti

```
1 | int (*p)[3];  
2 | /* alokujeme souvisly blok 6 prvku */  
3 | p = (int (*)[3])malloc (6*sizeof(int));
```

# I. Pole v dynamické paměti

---

2D pole v dynamické paměti

Struktura s ukazatelem na pole

# Struktura s ukazatelem na pole

---

- Téměř vždy při práci s polem potřebujeme informaci o alokované velikosti
- Velmi často tuto informaci z různých důvodů nemáme
- Řešením může být struktura, která obsahuje metadata pole

```
1 typedef struct
2 {
3     int delka;
4     int *data;
5 } pole;
```

```
1 void init (pole * x, int y) {
2     x->delka = velikost;
3     x->data = malloc(y*sizeof(int));
4 }
5
6 int main() {
7     pole a;
8     init (&a, 5);
9 }
```

Co zde chybí?



## Část II

### Vnitřní reprezentace datových typů

## II. Vnitřní reprezentace datových typů

---

Přesnost výpočtů

Základní číselné typy a jejich reprezentace v počítači

Reprezentace celých čísel

Reprezentace reálných čísel

```
1  #include <stdio.h>
2  int main(void)
3  {
4      double a = 1e+10;
5      double b = 1e-10;
7      printf("a   : %24.12lf\n", a);
8      printf("b   : %24.12lf\n", b);
9      printf("a+b: %24.12lf\n", a + b);
11     return 0;
12 }
```

lec08/sum.c

```
a   : 10000000000.000000000000
b   :           0.000000000100
a+b: 10000000000.000000000000
```

```
1  int main(void)
2  {
3      const int number = 100;
4      double dV = 0.0;
5      float fV = 0.0f;
7      for (int i = 0; i < number; ++i) {
8          dV += 1.0 / 10.0;
9          fV += 1.0 / 10.0;
10     }
12     printf("double: %lf float: %lf", dV, fV);
14     return 0;
15 }
```

lec08/div.c

```
double: 10.000000 float: 10.000002
```

# Strojová přesnost

---

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

$$v + 1.0 == 1.0$$

- Zaokrouhlovací chyba – nejméně  $\epsilon_m$ .
- Přesnost výpočtu
  - aditivní chyba roste s počtem operací v řádu  $\sqrt{N}\epsilon_m$
  - často se však kumuluje preferabilně v jednom směru v řádu  $N\epsilon_m$
- Absolutní chyba aproximace  $E(x) = \hat{x} - x$ 
  - $\hat{x}$  přesná hodnota,  $x$  aproximace
- Relativní chyba  $RE(x) = (\hat{x} - x)/x$

# Podmíněnost numerických úloh

---

$$C_p = \frac{\text{relativní chyba výstupních údajů}}{\text{relativní chyba vstupních údajů}}$$

- 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.

# 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.
- Mars Climate Orbiter – 23. 10. 1999  
Chyba při výměně dat při výpočtu impulsu motorů. Jeden systém počítal v librách/s, ale druhý to interpretoval jako N/s.

## II. Vnitřní reprezentace datových typů

---

Přesnost výpočtů

Základní číselné typy a jejich reprezentace v počítači

Reprezentace celých čísel

Reprezentace reálných čísel



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

Záleží na způsobu reprezentace

# Reprezentace 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 pak tuto definici hlídá a volí odpovídající strojové instrukce pro práci s datovými položkami například jako s odpovídajícími číselnými typy
- Příklad ekvivalentních reprezentací v paměti počítače
  - $01000001_{(2)}$  – 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 o velikosti 1B může být interpretován také jako znak A.

## II. Vnitřní reprezentace datových typů

---

Přesnost výpočtů

Základní číselné typy a jejich reprezentace v počítači

Reprezentace celých čísel

Reprezentace reálných čísel

# Číselné soustavy

---

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

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

- Dvojková soustava (bin) – 2 číslice 0 nebo 1

$$11010,01_{(2)} = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}$$

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

$$138,24_{(10)} = 1 \cdot 10^2 + 3 \cdot 10^1 + 8 \cdot 10^0 + 2 \cdot 10^{-1} + 4 \cdot 10^{-2}$$

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

$$0x7D_{(16)} = 7 \cdot 16^1 + D \cdot 16^0$$

# Kódování záporných čísel

---

- **Přímý kód** – znaménko je určeno 1. bitem (zleva), snadné stanovení absolutní hodnoty, dvě nuly, příklad reprezentace:
  - $121_{(10)}$  0111 1001<sub>(2)</sub>
  - $-121_{(10)}$  1111 1001<sub>(2)</sub>
  - $0_{(10)}$  0000 0000<sub>(2)</sub>
  - $-0_{(10)}$  1111 1111<sub>(2)</sub>
- **Inverzní kód** – záporné číslo odpovídá bitové negaci kladné hodnoty čísla; dvě nuly; příklad reprezentace:
  - $121_{(10)}$  0111 1001<sub>(2)</sub>
  - $-121_{(10)}$  1000 0111<sub>(2)</sub>
  - $0_{(10)}$  0000 0000<sub>(2)</sub>
  - $-0_{(10)}$  1111 1111<sub>(2)</sub>
- **Doplňkový kód** – záporné číslo je uloženo jako hodnota kladného čísla po bitové negaci zvětšená o 1; jediná reprezentace nuly
  - $121_{(10)}$  0111 1001<sub>(2)</sub>
  - $-121_{(10)}$  1000 0111<sub>(2)</sub>
  - $127_{(10)}$  0111 1111<sub>(2)</sub>
  - $-128_{(10)}$  1000 0000<sub>(2)</sub>

# Více-bajtová reprezentace a pořadí bajtů

---

- Číselné typy s více-bajtovou reprezentací mohou mít bajty uloženy v různém pořadí
  - **little-endian** – nejméně významný bajt (LSB) na nejnižší adrese
    - x86, ARM
  - **big-endian** – nejvíce významný bajt (MSB) na nejnižší adrese
    - 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
  - hodnoty z paměti jsou ukládány a přenášeny v tomto pořadí bajtů a na cílové stanici pak zpětně zapsány do konkrétního nativního pořadí
    - vždy big-endian

## II. Vnitřní reprezentace datových typů

---

Přesnost výpočtů

Základní číselné typy a jejich reprezentace v počítači

Reprezentace celých čísel

Reprezentace reálných čísel

# Reprezentace reálných čísel

---

- Pro uložení čísla vyhradjeme omezený paměťový prostor
- Příklad – zápis čísla  $\frac{1}{3}$  v dekadické soustavě
  - = 33333333...3333
  - =  $0,3\bar{3}$
  - $\approx 0,33333333333333333333$
  - $\approx 0,333$
- V trojkové soustavě:  $0 \cdot 3^1 + 0 \cdot 3^0 + 1 \cdot 3^{-1}$
- 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



# Model reprezentace reálných čísel 1/2

---

- Reálná čísla se zobrazují jako aproximace daným rozsahem paměťového místa

- Reálné číslo  $x$  se zobrazuje ve tvaru

$$x = m \cdot z^e$$

- Pro jednoznačnost zobrazení musí být mantisa normalizována

$$0,1 \lesssim m < 1$$

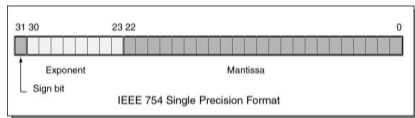
- Ve vyhrazeném paměťovém prostoru je pro zvolený základ uložen exponent a mantisa jako dvě celá čísla

## Model reprezentace reálných čísel 2/2

- Reálné číslo  $x$  se zobrazuje ve tvaru IEEE 754

$$x = (-1)^s \cdot m \cdot 2^{e-b}$$

- `float` – 32 bitů (4 bajty):  $s$  – 1 bit znaménko (+ nebo -), mantisa – 23 bitů  $\approx 16,7$  milionu možností; exponent – 8 bitů



- `double` – 64 bitů (8 bajtů):  $s$  – 1 bit znaménko (+ nebo -), mantisa – 52 bitů  $\approx 4,5$  biliardy možností, exponent – 11 bitů
- Čím větší exponent, tím větší mezery mezi sousedními aproximacemi čísel
- `bias` umožňuje reprezentovat exponent vždy jako kladné číslo

## Využití struktury, unionu a bitových operací

---

```
typedef union t_fp32 {
    float fp;           // float hodnota
    unsigned int uint; // 32b int na stejne adrese
    struct {           // rozklad 32b slova na slozky
        unsigned int mantisa:DELKA_MAN; // - 23b mantisa
        unsigned int exp:DELKA_EXP;     // - 8b exponent
        unsigned int signum:DELKA_SIG;  // - 1b znamenko
    } fp_deleny;
} t_fp32;
```

## Část III

### Standardní knihovny

# III. Standardní knihovny

---

Standardní knihovny

Matematické funkce

Práce se soubory

Zpracování chyb

# Standardní knihovny

---

- `stdio.h` – Vstup a výstup (formátovaný i neformátovaný)
- `stdlib.h` – Matematické funkce, alokace paměti, převod řetězců na čísla, řazení (qsort), vyhledávání (bsearch), generování náhodných čísel (rand)
- `limits.h` – Rozsahy číselných typů
- `math.h` – Matematické funkce
- `errno.h` – Definice chybových hodnot
- `assert.h` – Zpracování běhových chyb
- `ctype.h` – Klasifikace znaků (char)
- `string.h` – Řetězce, blokové přenosy dat v paměti (memcpy)
- `locale.h` – Internacionalizace
- `time.h` – Datum a čas

# Standardní knihovny (POSIX)

---

- Komunikace s operačním systémem (OS)
- **POSIX** – Portable Operating System Interface
- `stdlib.h` – Funkce využívají prostředků OS
- `signal.h` – Asynchronní události, vlákna
- `unistd.h` – Procesy, čtení/zápis souborů, ...
- `pthread.h` – Vlákna (POSIX Threads)
- `threads.h` – Standardní knihovna pro práci s vlákny (C11)

# III. Standardní knihovny

---

Standardní knihovny

Matematické funkce

Práce se soubory

Zpracování chyb



# Matematické funkce

---

- `<math.h>` – základní funkce pro práci s reálnými čísly
  - Výpočet odmocniny nečelého čísla  $x$   
`double sqrt(double x);, float sqrtf(float x);`
  - `double pow(double x, double y);` – výpočet obecné mocniny
  - `double atan2(double y, double x);` – výpočet arctan
  - Symbolické konstanty:
    - `#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

# III. Standardní knihovny

---

Standardní knihovny

Matematické funkce

Práce se soubory

Zpracování chyb

# Práce se soubory

---

- Knihovna `<stdio.h>`
- Přístup k souboru je prostřednictvím ukazatele `FILE*`
- Otevření souboru `FILE *fopen(char *filename, char *mode);`
- Práce s textovými a binárními (modifikátor "b") soubory
- Soubory jsou čteny/zapisovány sekvenčně
  - Se soubory se pracuje jako s proudem dat – postupné načítání/zápis
  - Aktuální pozici v souboru si můžeme představit jako kurzor
  - Při otevření souboru se kurzor nastavuje na začátek souboru
- Režim práce se souborem je dán hodnotou proměnné `mode`
  - "r" – režim čtení, ("r" – čtení textového souboru, "rb" – čtení binárního souboru)
  - "w" – režim zápisu (Vytvoří soubor, pokud neexistuje, jinak smaže obsah souboru)
  - "a" – režim přidávání do souboru (Kurzor je nastaven na konec souboru)

# Testování otevření / zavření souboru

---

- Otevření souboru

```
char *fname = "file.txt";

if ((f = fopen(fname, "r")) == NULL) {
    fprintf(stderr, "Error: open '%s'\n", fname);
}
```

- Zavření souboru

```
if (fclose(f) == EOF) {
    fprintf(stderr, "Error: close '%s'\n", fname);
}
```

- Konec souboru – `int feof(FILE *file);`

## Příklad – čtení souboru znak po znaku

---

- Čtení znaku: `int getc(FILE *file);`
- Hodnota znaku (unsigned char) je vrácena jako int

```
int count = 0;
while ((c = getc(f)) != EOF) {
    printf("Read character %d is '%c'\n", count, c);
    count++;
}
```

- Pokud nastane chyba nebo konec souboru vrátí funkce `getc()` hodnotu `EOF`
- Pro rozlišení chyby a konce souboru lze využít funkce `feof()` a `ferror()`

## Formátované čtení z textového souboru

---

```
int fscanf(FILE *file, const char *format, ...);
```

```
FILE *f = fopen("text.txt");
```

```
int r = fscanf(f, "%s %d %lf\n", str, &i, &d);
```

- Při čtení textového řetězce je nutné zajistit dostatečný paměťový prostor pro načítaný textový řetězec, např. omezením velikosti řetězce

```
char str[10];
```

```
int r = fscanf(f, "%9s %d %lf\n", str, &i, &d);
```

# Zápis do textového souboru

---

```
int main(int argc, char *argv[]) {
    char *fname = argc > 1 ? argv[1] : "out.txt";
    FILE *f;
    if ((f = fopen(fname, "w")) == NULL) {
        fprintf(stderr, "Error: Open '%s'\n", fname);
        return -1;
    }
    fprintf(f, "Program arguments argc: %d\n", argc);
    for (int i = 0; i < argc; ++i) {
        fprintf(f, "argv[%d]='%s'\n", i, argv[i]);
    }
    if (fclose(f) == EOF) {
        fprintf(stderr, "Error: Close '%s'\n", fname);
        return -1;
    }
    return 0;
}
```

## Náhodný přístup k souborům – fseek()

---

- Nastavení pozice kurzoru v souboru relativně vůči `whence` v bajtech

```
int fseek(FILE *stream, long offset, int whence);
```

- kde `whence`:

- `SEEK_SET` – nastavení pozice od začátku souboru
- `SEEK_CUR` – relativní hodnota vůči současné pozici v souboru
- `SEEK_END` – nastavení pozice od konce souboru

- `fseek()` vrací 0 v případě úspěšného nastavení pozice

- Nastavení pozice v souboru na začátek

```
void rewind(FILE *stream);
```

- Zjištění pozice kurzoru

```
long int ftell(FILE *stream);
```



# Binární čtení/zápis z/do souboru

---

- Otevření souboru s příznakem `b`

vliv na řetězce, řídicí znaky např. `\0`, `\n` nebo EOF – Ctr+Z

- Pro čtení a zápis bloku dat můžeme využít funkce `fread()` a `fwrite()` z knihovny `stdio.h`
- Načtení `nmemb` prvků, každý o velikosti `size` bajtů

```
size_t fread(void* ptr, size_t size, size_t nmemb, FILE *stream);
```

- Zápis `nmemb` prvků, každý o velikosti `size` bajtů

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

- Funkce vrací počet přečtených/zapsaných bajtů
- Pokud došlo k chybě nebo detekci konce souboru funkce vrací menší než očekávaný počet bajtů

# III. Standardní knihovny

---

Standardní knihovny

Matematické funkce

Práce se soubory

Zpracování chyb

# Zpracování chyb

---

- Základní chybové kódy jsou definovány v `<errno.h>`
- Tyto kódy jsou ve standardních C knihovnách používány jako příznaky nastavené v případě selhání volání funkce v globální proměnné `errno`
- Například otevření souboru `fopen()` vrací hodnotu `NULL`, pokud se soubor nepodařilo otevřít

Z této hodnoty, ale nepoznáme proč volání selhalo.

- Pro funkce, které nastavují `errno`, můžeme podle hodnoty identifikovat důvod chyby
- Textový popis číselných kódů pro standardní knihovnu C je definován v `<string.h>`
- Řetězec můžeme získat voláním funkce

```
char* strerror(int errnum);
```

## Příklad použití `errno`

---

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
int main(int argc, char *argv[])
{
    FILE *f = fopen("soubor.txt", "r");
    if (f == NULL) {
        int r = errno;
        printf("Open file failed errno %d\n", errno);
        printf("String error '%s'\n", strerror(r));
    }
    return 0;
}
```

## Testovací makro `assert()`

---

- Do kódu lze přidat podmínky na nutné hodnoty proměnných
- Testovat můžeme makrem `assert(expr)`
  - z knihovny `<assert.h>`
  - Pokud není `expr` `true` program se ukončí a vypíše jméno zdrojového souboru a číslo řádku.
- Makro vloží příslušný kód do programu
  - získáme tak relativně jednoduchý způsob indikace případné chyby, např. nevhodným argumentem funkce
- Vložení makra lze zabránit kompilací s definováním makra `NDEBUG`

## Testovací makro `assert()` – příklad

---

```
#include <stdio.h>
#include <assert.h>
int main(int argc, char *argv[])
{
    assert(argc > 1);
    printf("program argc: %d\n", argc);
    return 0;
}
```

## Část IV

### Zadání domácího úkolu

# Zadání 7. domácího úkolu (HW07)

---

## Téma: Zpracování číselné řady

- **Motivace:** Práce s polem v dynamické paměti
- **Cíl:** Zpracování teoreticky neomezeně dlouhé řady celých čísel
- **Zadání:** <https://cw.fel.cvut.cz/wiki/courses/b0b99prpa/hw/hw07>
  - Načtení předem neznámého počtu vstupních hodnot
  - Výpočet standardních statistických ukazatelů – medián, histogram
- **Termín odevzdání:** 23.11.2019, 23:59:59