Programování v jazyce C

9 Znaky a matematika

- Znakové a řetězcové konstanty
- Zpracování znaků
- Knihovna `ctype`
- Matematické funkce
- Knihovna `math`
- Generování náhodných čísel

Klasifikace a zpracování znaků v ANSI C
Obecné skutečnosti

- Funkce a datové typy pro práci se znaky (klasifikace, převody) jsou v knihovně **ctype** → připojit hlavičkový soubor příkazem preprocesoru: `#include <ctype.h>`

- V jazyce ANSI C je znak (ve smyslu ASCII) jako argument funkcí vždy předáván v datovém typu (`signed`) `int` – přičemž se ale bere z `int`u v potaz jen `unsigned char` (tj. pouze 8 bitů nejnižších řádů), je-li hodnota různá od -1. (důvodem je `EOF == -1`)

- Knihovna **ctype** v ANSI C nezvládá ani znaky národních abeced, ani široké znaky (Unicode, UTF-8, ap.).

Klasifikace a zpracování znaků v ANSI C
Obecné skutečnosti

- Funkce z knihovny **ctype** jsou dvojího druhu –
  (i) **klasifikační** a (ii) **převodní**:

- Jména **klasifikačních funkcí** začínají prefixem `is`… a vrací nulovou nebo nenulovou hodnotu (`int`) podle toho, zda předaný znak nepatří nebo patří do určité třídy znaků
  (např. `int isupper(int c)`).

- Jména **převodních funkcí** začínají prefixem `to`… a vrací hodnotu (`int`) odpovídající převedenému znaku (pokud bylo možné převod provést, jinak vrací nezměněný původní znak)
  (např. `int toupper(int c)`).

Znakové konstanty
Zápis pomocí výjimkových kódů

- Obecně existují 3 varianty zápisu znakové konstanty:

```
\<znak>
\<oktalová-konstanta>
\x<hexadecimální-konstanta>
```

Těchto tzv. **escape sekvencí** je celá řada (viz literaturu).

```
int c = '\r';
        '\n'
        '\t'

        '\\'
        '\''
        '\"'

        '\0'
        '\377'
        '\xA7'
```

CR (Carriage Return)
LF (Line Feed)
tabulátor

zpětné lomítko
apostrof
uvozovka

ASCII 0
ASCII 255
ASCII 167

ASCII/... kód znaku
– následují max. 3 osmičkové číslice

x musí být malé
– následují 2 šestnáctkové číslice

Znakové konstanty
Deklarace znakových konstant

- **Znakový literál** = jeden **nebo více** znaků (až 4) uzavřených v apostrofech:

```
const int c = 'A';
const int d = 'ABCD';
```

hodnota c je 65 (ASCII 'A')

hodnota c je buď `0x41424344` nebo `0x44434241` (podle endianu CPU)

- Implicitně je znakový literál typu **`int`**, ovšem lze překladač přimět, aby ho deklaroval jako **`long int`**, a to uvedením prefixu **`L`**: **`L'A'`**;
- Některé překladače (např. Watcom, GCC) znakové konstanty s prefixem **`L`** překládají jinak, než definuje norma ANSI C (třeba ho ignorují).
- Pokud se literálem zapsaná znaková konstanta nevejde do implicitního datového typu (**`int`**), hlásí překladač chybu či jen warning (pak ořezává na **`int`** od konce!).

Řetězcové konstanty
Syntax řetězcových literálů (odbočka)

- Posloupnost znaků uzavřená v uvozovkách, se kterou překladač pracuje jako s polem **`char`**ů (+ ukončovací znak).
- Lze použít výjimkové kódy pro spec. znaky.

Sekvence '\↵' → řetězec pokračuje na další řádce...

```
""
"\"Hello!\""
"Press any key.\n"
"This is an unbelievably very very very\
 very long string (K&R C)"
```

Text musí začínat na začátku řádky – případné mezery, tabulátory, apod. budou součástí řetězce.

```
char msg[] = "This is an unbelievably very"
           " very very long string (ANSI C)";
```

ANSI C dovoluje „sestavit" řetězcovou konstantu z více částí, za obsah řetězce se považuje pouze text uvnitř uvozovek, tj. mezery, tabulátory, apod. na začátku řádky teď nejsou součástí řetězce.

Funkce pro klasifikaci znaků
Alfanumerické znaky

Všechny funkce z knihovny **ctype** vrací vždy přímo výsledek klasifikace znaku nebo převodu → návratová hodnota nepředstavuje indikaci úspěšnosti operace, tzn. **nelze zjistit, zda funkce znak klasifikovala/převedla nebo ignorovala!**

```
int isalnum(int c);
```

- Testuje (= vrací nenulovou hodnotu, pokud ano), zda je předaný znak **číslice** nebo **velké či malé písmeno**.
- **POZOR:** To, zda je nějaký znak písmenem, může záviset na aktivním **locale**! (viz fce `setlocale(·)` z knihovny **locale**)

```
int isalpha(int c);
```

- Testuje, zda je předaný znak **velké či malé písmeno**.
- I tuto funkci ovlivňuje aktivní **locale**!

Funkce pro klasifikaci znaků
Číslice a bílé znaky

```
int isdigit(int c);
```

- Testuje, zda je předaný znak **dekadická číslice**, tj. znak z množiny C_{DEC} = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'}.

```
int isxdigit(int c);
```

- Testuje, zda je předaný znak **hexadecimální číslice**, tj. znak z množiny $C_{HEX} = C_{DEC} \cup$ {'A', 'a', 'B', 'b', 'C', 'c', 'D', 'd', 'E', 'e', 'F', 'f'}.

```
int isspace(int c);
```

- Testuje, zda je předaný znak tzv. **bílý**, tzn. mezera '␣' (0x20) nebo znaky z množiny C_{SP} = {'\t', '\n', '\v', '\f', '\r'}.

```
int isblank(int c);
```

C11

- Testuje, zda je předaný znak **oddělovačem slov v textu**.

Funkce pro klasifikaci znaků
Grafické a řídicí znaky

```
int iscntrl(int c);
```

- Testuje, zda je předaný znak tzv. **řídicí**, tj. znak, který se nevypisuje do konzole (v ASCII 0x00 až 0x1F a znak 0x7F), ale má význam příkazu pro konzoli (např. pípnout, apod.).

```
int isprint(int c);
```

- Opak ↑, testuje, zda je předaný znak **tisknutelný**, tj. zobrazuje se v konzoli; má význam dat, nikoliv příkazu pro konzoli.

```
int isgraph(int c);
```

- Testuje, zda má předaný znak **grafickou podobu**, tj. zda lze vypsat do konzole (což testuje funkce **`isprint(·)`**) a bude tam následně vidět (tzn. např. mezera '␣' (0x20) takovým znakem **není**).

Funkce pro klasifikaci znaků
Interpunkce a velká/malá písmena

```
int ispunct(int c);
```

- Testuje, zda je předaný znak **interpunkčním znaménkem**.
- Aktivní locale ovlivňuje, co je a není interpunkcí, ale každopádně je to znak, pro který nabývá nenulové hodnoty výraz **`isgraph(·) && !isalnum(·)`**.

```
int islower(int c);
```

- Testuje, zda je předaný znak **minuskulí**, tj. malým písmenem.

```
int isupper(int c);
```

- Testuje, zda je předaný znak **verzálou**, tj. velkým písmenem.

Funkce pro převod znaků
Převod velkých na malá písmena a naopak

```
int tolower(int c);
```

- Převádí **velké písmeno na malé** (to vrací jako návratovou hodnotu). Pokud je předaný znak malé písmeno nebo nelze převést, vrací nezměněný argument.

```
int toupper(int c);
```

- Převádí **malé písmeno na velké** (to vrací jako návratovou hodnotu). Pokud je předaný znak velké písmeno nebo nelze převést, vrací nezměněný argument.

- **Obě funkce jsou výrazně ovlivněny aktivním locale** (zejm. např. problémy se znaky národních abeced v různ. kódování:
  'š' (CP1250 0x9A) → 'Š' (CP1250 0x8A)
  'č' (CP1250 0xE8) → 'Č' (CP1250 0xC8)
  } „poloha" znaků v kódu ad hoc

Matematické funkce
Obecné skutečnosti

- Většina matematických funkcí je definovaná v knihovně **math** → připojit hlavičkový soubor příkazem preprocesoru: `#include <math.h>`

- Několik málo matematických funkcí (např. celočíselné absolutní hodnoty) je v knihovně **stdlib**.

- Všechny funkce z knihovny **math** přijímají jako (primární) argument datový typ `double` a vrací také `double`.

- Definice pro typ `double` postačuje, `float` na něj lze automaticky rozšířit bez ovlivnění přesnosti! (navíc CPU/FPU obvykle pracuje s 80-bitovou vnitřní reprezentací reálného čísla, typem `extended` podle standardu IEEE 754-1985)

Matematické funkce
Obecné skutečnosti – diagnostika chyb

- Návratová hodnota představuje vypočítanou hodnotu matematické funkce, tj. z návratové hodnoty nelze zjistit, zda výpočet proběhl bez chyb.

- O případných chybách při výpočtu informuje globální proměnná **`int errno`** z knihovny **errno**. Nastat může buď chyba oboru **`EDOM`** (např. při **`log(-1.0);`**) nebo rozsahu **`ERANGE`** (např. při **`log(0.0);`**).

```
errno = 0;
x = log(-1.0);

if (errno == EDOM) printf(…);
if (errno == ERANGE) printf(…);
if (errno) printf("%s\n", strerror(errno));
```

Matematické funkce
Absolutní hodnota a celočíselné dělení

```
int abs(int x);
long labs(long x);
double fabs(double x);
```

} definovány ve **stdlib**

- Vrací absolutní hodnotu svých argumentů.

```
div_t div(int n, int d);
ldiv_t ldiv(long n, long d);
```

} definovány ve **stdlib**

- Vypočítávají zároveň podíl a zbytek po celočíselném dělení čísla **int/long** `n` číslem **int/long** `d`.

- Návratový typ **div_t** je definován jako:
  struct `div_t { int quot; int rem; };`
  analogicky pro argumenty typu **long** je pak definován návratový typ **ldiv_t**.

Matematické funkce
Manipulace s reálným číslem

```
double frexp(double x, int *exponent);
double ldexp(double x, int exponent);
double modf(double x, double *integer);
```

- **`frexp(·)`** rozděluje reálné číslo **`x`** na normalizovanou **mantisu**, tj. z intervalu [0.5, 1) ∪ 0, kterou vrací jako návratovou hodnotu, a **exponent**, který uloží na adresu **`*exponent`**.
- **`ldexp(·)`** pracuje opačně, tj. „složí" číslo z mantisy **`x`** a exponentu **`int exponent`**.
- **`modf(·)`** dělí reálné číslo **`x`** na **desetinnou část**, kterou vrací v návratové hodnotě, a na **celou část**, kterou uloží na adresu **`*integer`**.

```
x = frexp(-3.625, &ex);
```
▪▪▪▶ x = -0.906250, ex = 2

```
x = modf(-3.625, &int);
```
▪▪▪▶ x = -0.625000, int = -3.0

Matematické funkce

Reálnému číslu nejbližší celá čísla

```
double ceil(double x);
double floor(double x);
```

- **`ceil(·)`** vrací nejmenší reálné číslo takové, že jeho hodnota není menší než **`x`** a je celočíselná, tj. **nejbližší větší celé číslo**.
- **`floor(·)`** vrací největší reálné číslo takové, že jeho hodnota není větší než **`x`** a je celočíselná, tj. **nejbližší menší celé číslo**.

Zbytek po celočíselném dělení

```
double fmod(double x, double y);
```

- Vrací zbytek po celočíselném dělení čísla **`x`** číslem **`y`**.

Nevhodně zvolené názvy funkcí **`fmod(·)`** a **`modf(·)`** mohou vést k záměně (prototyp je až na název zcela identický).

Matematické funkce
Mocniny a odmocniny

```
double pow(double x, double y);
double sqrt(double x);
```

- **`pow(·)`** vypočítává **obecnou mocninu**, tj. číslo x^y. Může nastat chyba oboru **`EDOM`**, je-li **`x`** záporné a **`y`** není celé nebo když je **`x`** = 0 a **`y`** není kladné.
- **`sqrt(·)`** vypočítává **druhou odmocninu** z čísla **`x`**. Chyba nastane, je-li **`x`** záporné.

Obecná odmocnina

- Knihovna **neposkytuje funkci pro obecnou odmocninu**, je třeba využít matematického poznatku, že

$$x^{\frac{1}{n}} = \sqrt[n]{x}$$

Matematické funkce
Logaritmus a exponenciální funkce

```
double log(double x);
double log10(double x);
double exp(double x);
```

- **`log(·)`** vypočítává **přirozený logaritmus** čísla **`x`**.
- **`log10(·)`** vypočítává **dekadický logaritmus** čísla **`x`**. V obou případech může nastat chyba oboru **`EDOM`**, je-li **`x`** záporné, nebo chyba rozsahu **`ERANGE`** pro $x \to 0$.
- **`exp(·)`** vypočítává **exponenciální funkci** čísla **`x`**, tj. e^x, kde e je Eulerovo číslo, základ přirozených logaritmů. Chyba rozsahu **`ERANGE`** nastane pro velké hodnoty **`x`**.

Matematické funkce
Logaritmus o obecném základu

- Knihovna jazyka ANSI C **neposkytuje** funkce pro výpočet logaritmů o jiném základu než 10 (`log10(·)`) a *e* (`log(·)`).
- K výpočtu logaritmu o jiném základu je třeba využít matematického poznatku, že

$$\log_n x = \frac{\log_{10} x}{\log_{10} n}$$

- V informatice velmi užitečný **logaritmus dualis** (tj. při základu 2), který určuje, kolik bitů je potřeba k uložení `x` různých stavů (hodnot), lze nadefinovat jako

```
double log2(double x) {
  return log(x) / log(2);
}
```

Matematické funkce
Goniometrické funkce

```
double sin(double x);
double cos(double x);
double tan(double x);
```

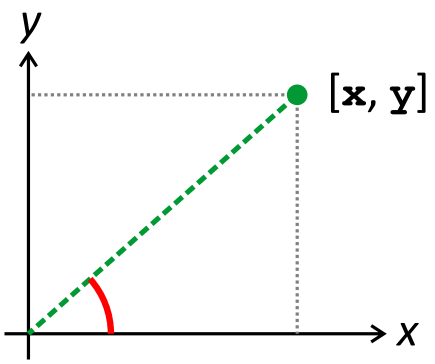
- Vypočítávají hodnoty goniometrických funkcí dle názvu.
- Argument se předává **v radiánech!**
- Chyba oboru **`EDOM`** nenastává, ale pro velké hodnoty argumentu nemusí být výsledek správný.
- Chyba rozsahu **`ERANGE`** může nastat u makra **`tan(·)`** pro hodnoty argumentu blízké sudému násobku π / 2.

- Původní ANSI C verze knihovny makro **`tan(·)`** neobsahovala. Bylo zavedeno až standardem C99 → je-li požadována zpětná kompatibilita, je vhodnější **`tan(·)`** nepoužívat a tangens vypočítat jako podíl **`sin(x) / cos(x)`**.
  (z ↑ je zřejmé, proč je **`tan(·)`** definován jako makro)

Matematické funkce
Cyklometrické funkce

```
double asin(double x);
double acos(double x);
double atan(double x);
double atan2(double x, double y);
```

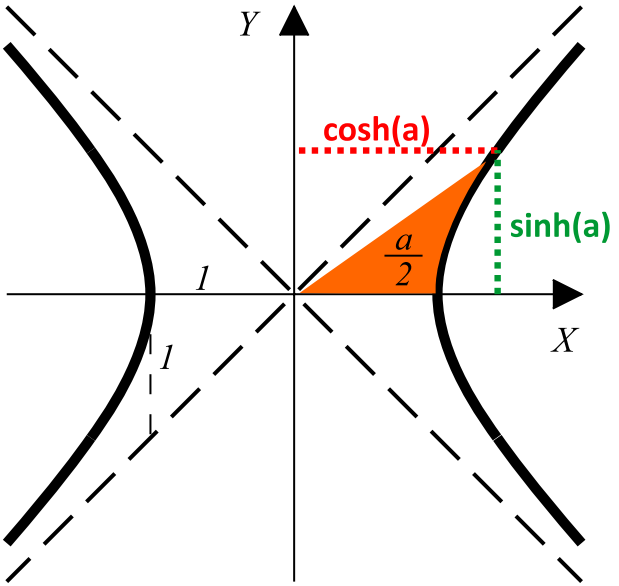
- Vypočítávají hodnoty cyklometrických funkcí dle názvu.
- Návratová hodnota (tj. úhel) je **v radiánech!**
- Chyba oboru **`EDOM`** nastává v případě **`asin(·)`** a **`acos(·)`** tehdy, leží-li argument mimo interval (-1; 1).
- **`atan2(·)`** vrací úhel mezi osou *x* kartézské soustavy souřadnic a přímkou procházející počátkem soustavy a bodem [**`x`**, **`y`**]; chyba nastane, jsou-li oba argumenty nulové.



Matematické funkce
Hyperbolické funkce

```
double sinh(double x);
double cosh(double x);
double tanh(double x);
```

- Vypočítávají hodnoty hyperbolických funkcí dle názvu.
- Chyba rozsahu **ERANGE** nastane tehdy, je-li absolutní hodnota parametru funkcí **sinh(·)** a **cosh(·)** velmi velká.
- Použití: řešení některých diferenciálních rovnic, …



$$x^2 - y^2 = 1$$

Generování náhodných čísel
Funkce generátoru náhodných čísel

```
int rand();
void srand(unsigned seed);
```

} defin. ve **stdlib**

- **`rand(·)`** vrací při každém volání jinou celočíselnou hodnotu, **pseudonáhodné číslo** z intervalu ⟨0, **`RAND_MAX`**⟩.
- Hodnota konstanty **`RAND_MAX`** je dle normy ANSI C alespoň 32767.

- **`srand(·)`** lze použít k nastavení počáteční hodnoty generátoru pseudonáhodných čísel.
- Inicializuje-li se generátor **stejnou hodnotou**, sekvence následných volání **`rand(·)`** vrací **stejné posloupnosti** čísel!
- Není-li generátor zinicializován voláním **`srand(·)`**, vrací **`rand(·)`** takovou posloupnost, jako by inicializace byla provedena číslem 1.
- **Generátor produkuje čísla s rovnoměrným rozdělením!**

Generování náhodných čísel
Použití generátoru náhodných čísel

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
  int i, n = 5;

  srand((unsigned) time(NULL));

  for (i = 0; i < n; i++) {
    printf("%d\n", rand() % 50);
  }

  ⋮
```

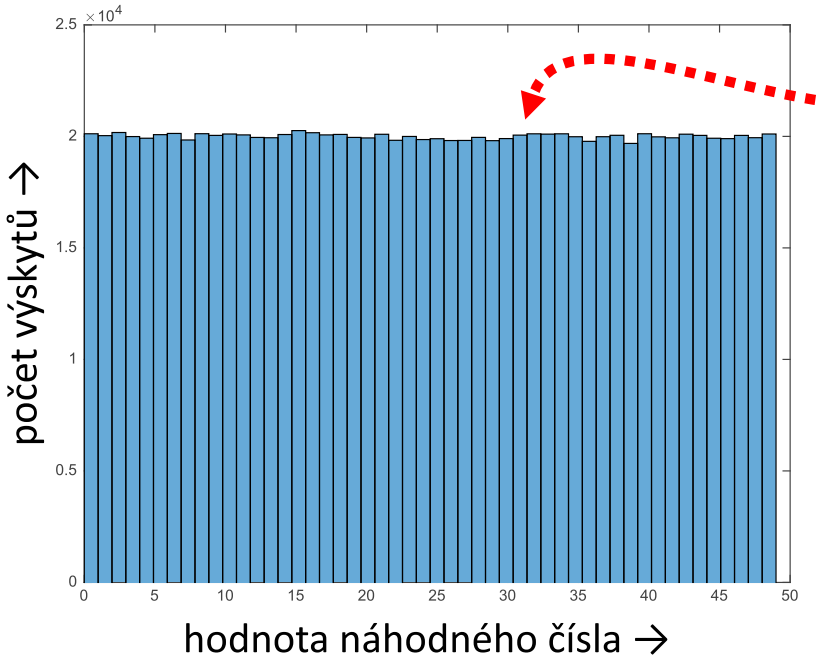
Trik k dosažení „náhodnosti” generované posloupnosti:
Inicializovat generátor aktuální časovou značkou...

```
05
23
31
39
06
```

modulo dělení
→ **zajišťuje požadovaný rozsah generovaných pseudonáhodných čísel, tj. zde v intervalu** ⟨0, 50).

Generování náhodných čísel
Použití generátoru náhodných čísel

- Takto vypadá **histogram** čísel, vygenerovaných kódem z předchozí ukázky pro `n` = 1 000 000, tj. pravděpodobnost vygenerování určitého čísla má **rovnoměrné rozdělení p-sti**.



Generátor s takovým rozdělením je ale **naprosto nevhodný** pro př. simulační experimenty, krypto, ap.

Generování náhodných čísel
Generátor s normálním rozdělením

- Existuje řada různých postupů, jak generovat náhodná čísla s nějakým rozdělením pravděpodobnosti, např. velmi jednoduchá **metoda přijetí-zamítnutí** (***Exclusion Method***):

```
#define nrand() (1.0 * rand() / RAND_MAX)

double gauss_rand(double mean, double dev) {
  double x, y, z, w = 10.0 * dev;

  do {
    x = nrand() * w + (mean - w / 2.0);
    z = nrand();
    y = gauss_pdf(x, mean, dev);
  } while (z >= y);

  return x;
}
```

Generování náhodných čísel
Generátor s normálním rozdělením

- Funkce **`gauss_pdf(·)`** vypočítává hodnoty gaussovské funkce:

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

hustota pravděpodobnosti
(*Probability Density Function*)

```
double gauss_pdf(double x, double mu,
                 double sigma) {
  double ds = 2 * pow(sigma, 2.0);
  double xm = pow(x - mu, 2.0);
  return 1.0 / (SQRT_2PI * sigma) *
    exp(-xm / ds);
}
```

2.506628275

- MPZ může pracovat s **libovolnou** funkcí hustoty pravděpodobnosti → lze implementovat generátory s různým rozdělením (Bernoulliho, binomickým, Poissonovým, atp.).

Generování náhodných čísel
Generátor s normálním rozdělením

- Takto vypadá **histogram** 100 000 čísel, vygenerovaných kódem MPZ (z ukázky) s gaussovskou funkcí hustoty pravděpodobnosti a parametry $\mu = 3.0$ a $\sigma = 1.5$: