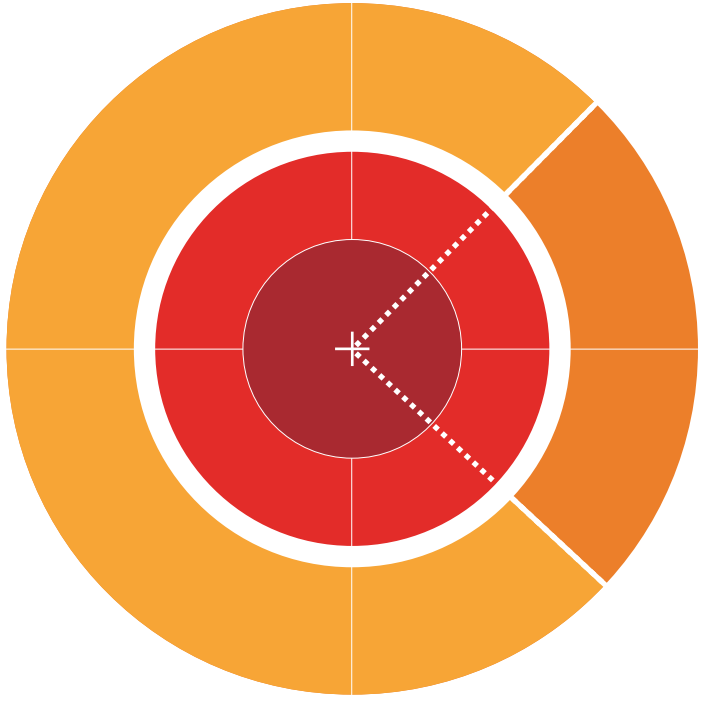
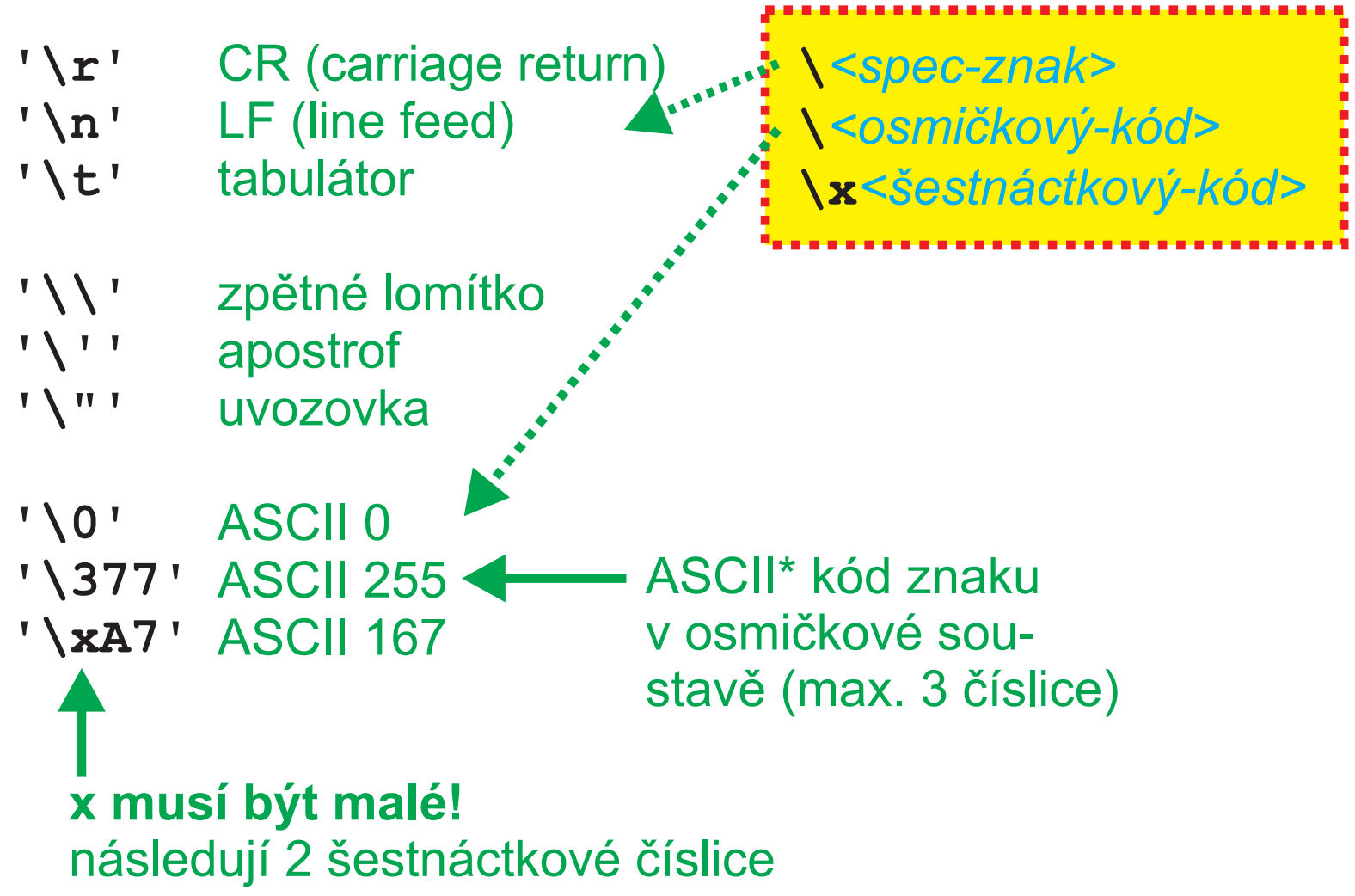
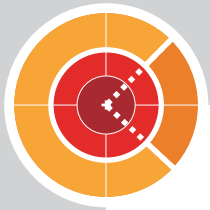


Zápis znaků pomocí výjimkových kódů





Řetězcové konstanty

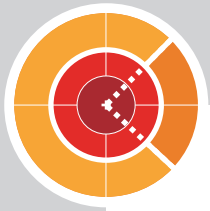
- posloupnost znaků uzavřená v uvozovkách, lze používat **výjimkové kódy** stejně jako u znakových konstant

```
""
"\"Nazdar!\""
"Stiskněte libovolnou klávesu.\n"
"Velmi dlouhá řetězcová konstanta je roz\
dělená na více řádek (i v klasickém C)"
```

text musí začínat na začátku řádky - případné mezery, tabulátory, apod. budou součástí řetězce

```
"Tato dlouhá řetězcová konstanta pak je roz"
  "dělená pouze v ANSI C"
```

v ANSI C je možné "sestavit" řetězcovou konstantu z více částí, obsah řetězce je pouze uvnitř uvozovek, tj. mezery, tabulátory, apod. na začátku řádky nevadí



Deklarace řetězcových konstant

```
char s1[] = "abcdef";
char *s2 = "ghijkl";
```

znaky v řetězci lze vždy přepisovat

znaky v řetězci **nelze** přepisovat

- je-li nutné **explicitně zabránit přepisování** řetězce, lze uvést `const char msg[] = "Message ...";`

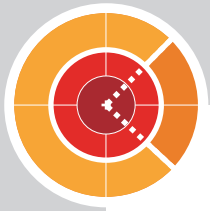
POZOR!

```
sizeof("abcdef") == 7
```

překladač za řetězcovou konstantu automaticky připojí `'\0'`

```
char *str1, *str2;
str1 = "abcd";
str2 = "abcd";
if (str1 == str2) ...
```

překladač může (ale nemusí) stejné řetězcové konstanty sdílet



Znakové a řetězcové konstanty

- **znaková konstanta:** jeden nebo více znaků uzavřených v apostrofech

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

hodnota `c` je 65

hodnota `c` je `0x41424344` nebo (dle arch) `0x44434241`

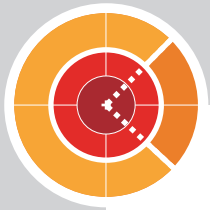
- implicitně je znaková konstanta typu **`int`**, je ale možné překladač přimět, aby jí deklaroval jako **`long int`** a to uvedení prefixu L: **`L'A'`**
- některé překladače (např. Watcom) znakové konstanty s prefixem L překládají jinak, než definuje norma ANSI C
- pokud se zapsaná znaková konstanta nevejde do implicitního datového typu (**`int`**), hlásí překladač chybu

Zpracování znaků

- všechny funkce pro zpracování znaků (dále popsané) jsou definované ve standardní knihovně **`ctype`**, tj. připojit pomocí `#include <ctype.h>`

- funkce jsou dvojího druhu: **klasifikační** a **převodní**
- jména **klasifikačních funkcí** začínají prefixem **`is`**... a vrací nulovou nebo nenulovou hodnotu (**`int`**) podle toho, zda daný znak nepatří nebo patří do určité třídy znaků
- jména **převodních funkcí** začínají prefixem **`to`**... a vrací hodnotu typu **`int`** reprezentující převedený znak

- **POZOR!** - v některých implementacích působí problémy nestandardní znaky (znaky nár. abeced, "široké" znaky)



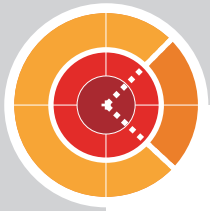
isalnum(), isalpha() a iscntrl()

```
int isalnum(int c);
int isalpha(int c);
int iscntrl(int c);
```

- **`isalnum()`** vrací nenulovou hodnotu, je-li **`c`** v množině znaků {'0'..'9', 'A'..'Z', 'a'..'z'}
  (shodné s **`isalpha(c) || isdigit(c)`**)

- **`isalpha()`** vrací nenulovou hodnotu, je-li **`c`** velké nebo malé písmeno
  (shodné s **`islower(c) || isupper(c)`**)

- **`iscntrl()`** vrací nenulovou hodnotu, je-li **`c`** v množině řídicích znaků {'\0'..'\037', '\177'} (pro ASCII)



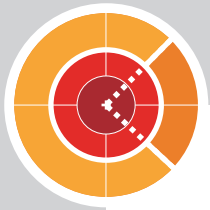
isdigit(), isxdigit() a isspace()

```
int isdigit(int c);
int isxdigit(int c);
int isspace(int c);
```

- **`isdigit()`** vrací nenulovou hodnotu, je-li **`c`** v množině znaků {'0'..'9'}

- **`isxdigit()`** vrací nenulovou hodnotu, je-li **`c`** v množině znaků {'0'..'9', 'A'..'F', 'a'..'f'}

- **`isspace()`** vrací nenulovou hodnotu, je-li **`c`** v množině tzv. bílých znaků {'**`\t`**', '**`\r`**', '**`\n`**', '**`\f`**', '**`\v`**', ' '}



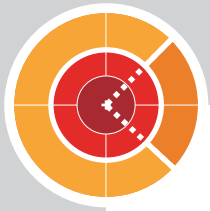
isgraph(), isprint() a ispunct()

```
int isgraph(int c);
int isprint(int c);
int ispunct(int c);
```

- **`isgraph()`** vrací nenulovou hodnotu, je-li **`c`** v množině grafických znaků, tj. všech kromě bílých znaků

- **`isprint()`** vrací nenulovou hodnotu, je-li **`c`** v množině tisknutelných znaků - pro ASCII {'\040'..'\176'}
  (ve většině implementací shodné s **`!iscntrl(c)`**)

- **`ispunct()`** vrací nenulovou hodnotu, je-li **`c`** v množině tzv. interpunkčních znaků {'!', '"', '#', '$', '%', '^', '(', ')', ...}



islower() a isupper() a tolower() a toupper()

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

- **islower()** vrací nenulovou hodnotu, je-li **c** v množině malých písmen {'a'..'z'}

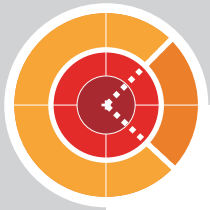
- **isupper()** vrací nenulovou hodnotu, je-li **c** v množině velkých písmen {'A'..'Z'}

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

- převádí c na malé/velké písmeno nebo vrací nezměněný
- **POZOR!** - pro některé znaky národních abeced neumí převod provést => vrací nezměněný parametr

Matematické funkce

- většina matematických funkcí je definovaná v knihovně **`math`**, tj. připojit pomocí **`#include <math.h>`**
- některé (celočíselné, absolutní hodnoty) jsou definované v knihovně **`stdlib`**
- o vzniklých chybách informuje globální proměnná **`int errno`** z knihovny **`errno`**
- může nastat chyba oboru **`EDOM`** nebo rozsahu **`ERANGE`**
- tradiční **návratovou hodnotou** v případě chyby je 0, ale některé implementace mají možnost vracet speciální hodnoty **NaN** *(Not a Number)*, **+Inf** *(Positive Infinity)* a **-Inf** *(Negative Infinity)* - tuto možnost ostatně nabízí i norma IEEE pro zobrazení čísel s pohyblivou řádovou čárkou



abs(), labs(), fabs(), div() a ldiv()

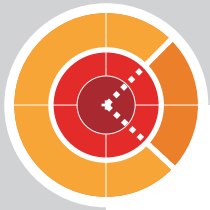
```
int abs(int x);
long labs(long x);
double fabs(double x);
```

} definovány ve **<stdlib.h>**

- vrací absolutní hodnotu svých argumentů

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

- vypočítávají zároveň podíl a zbytek po celočíselném dělení hodnoty **n** hodnotou **d**
- typ **div_t** je definován jako:
  struct div_t { int quot; int rem; }
- **ldiv** analogicky pro typ **long**

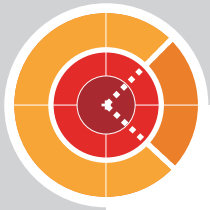


ceil(), floor(), fmod()

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

- **ceil()** vrací nejmenší reálné číslo, které není menší než **x** a jehož hodnota je celočíselná
- **floor()** vrací největší reálné číslo, které není větší než **x** a jehož hodnota je celočíselná
- **fmod()** vrací zbytek po celočíselném dělení hodnoty **x** hodnotou **y**

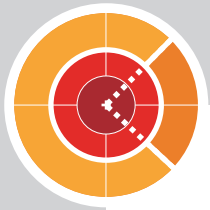
- definice pro typ **double** stačí, proměnné typu **float** se na něj **automaticky rozšíří** (na PC se obvykle vnitřně pracuje s 80-bitovým desetinným číslem, typ **extended** - není součástí ANSI C)



exp(), log(), log10()

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

- **`exp()`** vypočítává **exponenciální funkci** proměnné **`x`**, tj. e^x, kde *e* je základ přirozených logaritmů
- **`log()`** vypočítává **přirozený logaritmus** hodnoty **`x`** - je-li **`x`** záporné, nastává chyba oboru (EDOM); je-li **`x`** blízké nule, může nastat chyba rozsahu (ERANGE) směrem k *-Inf*
- **`log10()`** vypočítává **logaritmus při základu 10** - podmínky chyby jsou stejné jako v případě **`log()`**



Logaritmus o jiném základě

- jazyk C **neposkytuje** funkci pro výpočet logaritmu o jiném základě než 10 (`log10()`) a e (`log()`)
- je třeba využít matematického poznatku, že

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

- v informatice užitečný *logaritmus dualis* (dvojkový logaritmus), který vypočítává počet bitů nutných k uložení **x** různých stavů (hodnot), lze nadefinovat takto

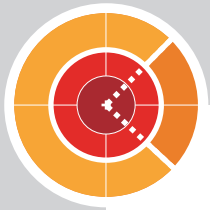
```
double log2(double x) {
   return log(x) / log(2);
}
```

frexp(), ldexp(), modf()

```
double frexp(double x, int *nptr);
double ldexp(double x, int n);
double modf(double x, double *nptr);
```

- **frexp()** rozdělí reálné číslo **x** na mantisu z intervalu **0** U **[0.5, 1)** a exponent - mantisa se předává jako návratová hodnota, exponent se uloží na adresu **nptr**
- **ldexp()** je funkce inverzní
- **modf()** rozdělí reálné číslo **x** na desetinnou část, která se vrací jako návratová hodnota a na celou část, která se uloží na adresu **nptr**

modf() je nevhodné jméno, protože navrácená hodnota je desetinnou částí předaného argumentu



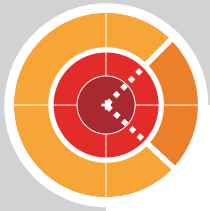
Mocniny - pow() a sqrt()

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

- **pow()** vypočítá hodnotu x^y - chyba oboru (EDOM) nastane, když je **x** záporné a **y** není celé, nebo je-li **x** nula a **y** není kladné
- **sqrt()** vypočítává nezápornou druhou odmocninu z **x** - chyba nastane, je-li **x** záporné

Obecná odmocnina

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

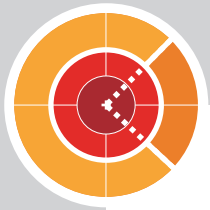


Generátor náhodných čísel - srand() a rand()

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

} definovány ve **<stdlib.h>**

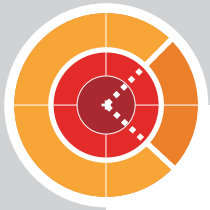
- **rand()** vrací při každém volání jinou celočíselnou hodnotu z intervalu <0, **RAND_MAX**> - **RAND_MAX** je definováno ve **stdlib** a musí mít (dle ANSI normy) hodnotu alespoň 32767
- **srand()** lze použít pro nastavení počáteční hodnoty generátoru pseudonáhodných čísel
- inicializuje-li se generátor **stejnou hodnotou**, sekvence následných volání **rand()** vrací stejné posloupnosti čísel
- není-li generátor zinicializován voláním **srand()**, vrací **rand()** takovou posloupnost hodnot, jako by inicializace byla provedena číslem 1
- generátor produkuje čísla **s rovnoměrným rozdělením**



Goniometrické funkce - sin(), cos(), tan()

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

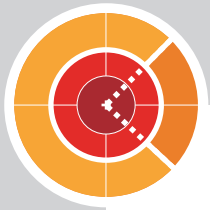
- argument se očekává **v radiánech**
- chyba oboru (EDOM) ani rozsahu (ERANGE) nenastane, ale výsledek nemusí mít význam pro velké hodnoty **`x`**
- chyba rozsahu může nastat u funkce **`tan()`** pro hodnoty **`x`** blízké sudému násobku



Cyklometrické funkce - asin(), acos(), atan(), atan2()

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

- návratová hodnota je **v radiánech**
- chyba oboru (EDOM) nastane v případě funkcí **`asin()`** a **`acos()`** tehdy, leží-li argument mimo interval **(-1, 1)**
- chyba oboru (EDOM) ani rozsahu (ERANGE) nenastane v případě **`atan()`**
- **`atan2()`** vrací (v termínech kartézského souřadného systému) **úhel** mezi osou **x** a přímkou procházející bodem o souřadnicích **[x, y]**; chyba nastane, jsou-li oba parametry rovny nule



Hyperbolometrické funkce - sinh(), cosh(), tanh()

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

- chyba rozsahu (ERANGE) nastane tehdy, je-li absolutní hodnota parametru funkcí **`sinh()`** nebo **`cosh()`** veliká