Knihovny funkcí

- C run time library (CRT)
- = soubor funkcí dodávaných spolu s překladačem, optimalizované → velmi rychlé
- C = very simple, většina „funkčnosti jazyka“ → CRT
- C dle ISO/IEC9899:
  - `assert.h` makra pro ladění
  - `ctype.h` třídění znaků
  - `errno.h` globální proměnná errno
  - `float.h` vlastnosti reálných čísel
  - `limits.h` vlastnosti celých čísel
  - `locale.h` modifikace národního prostředí (nepoužívá se)
  - `math.h` matematické funkce
  - `setjmp.h`
  - `signal.h`
  - `stdarg.h` funkce s proměnným počtem parametrů
  - `stddef.h` obecná makra (jsou i jinde)
  - `stdio.h`. standardní vstup, výstup
  - `stdlib.h` dynamické proměnné, náhodná čísla, řazení, ukončení programu
  - `string.h` řetězce
  - `time.h` čas a datum

Chyby v CRT

- errno.h
- případná chyba uložena do globální proměnné `int errno`
- nutno kontrolovat ihned → možnost přepsání dalším voláním CRT funkce
- na začátku programu vhodné vynulovat
- chybové kódy dle ANSI C (`#define...`)
  - `EDOM` = vstup mimo obor
  - `ERANGE` = výstup mimo rozsah
  - `EILSEQ` = neplatná sekvence bytů
- Microsoft CRT ~40 kódů

- ukázka zpracování chyby v math.h

```
#include <stdio.h>
#include <string.h>     // pro strerror()
#include <errno.h>
#include <math.h>
double x;
errno = 0;
x = log(-5);
if (errno != 0)
      puts(strerror(errno));
```

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

- v string.h
- vrací textový popis chyby (uložené v `errno`)
    - např. pro `ERANGE` → Result too large

Matematické konstanty

- math.h

```
#define M_E        2.7182818284590452536
#define M_LOG2E    1.4426950408889634073
#define M_LOG10E   0.43429448190325182765
#define M_LN2      0.69314718055994530942
#define M_LN10     2.30258509299404568402
#define M_PI       3.14159265358979323846
#define M_PI_2     1.57079632679489661923
#define M_PI_4     0.78539816339744830962
#define M_1_PI     0.31830988618379067154
#define M_2_PI     0.63661977236758134308
#define M_2_SQRTPI 1.12837916709551257390
#define M_SQRT2    1.41421356237309504880
#define M_SQRT1_2  0.70710678118654752440
```

Matematické funkce

- většina v math.h, něco je i v stdlib.h

Absolutní hodnoty

```
int abs(int j);          // v stdlib.h
long labs(long j);       // v stdlib.h
double fabs(double j);
```

- Vrací absolutní hodnotu argumentu

goniometrické funkce

```
double sin(double x);   // vrací sinus argumentu
double cos(double x);   // kosinus
```

```
double tan(double x);   // tangens
double asin(double x);  // arkussinus
double acos(double x);  // arkuskosinus
double atan(double x);  // arkustangens
double atan2(double y, double x);   // počítá arcus tangens y/x
```

- pro `tan()` možná `ERANGE` pro hodnoty vstupu blízké `k*pi/2` (`k` je sudé)

zaokrouhlování

```
double ceil (double x); // Zaokrouhlí arg na nejmenší vyšší celé číslo
double floor (double x);// Zaokrouhlí arg na nejbližší nižší celé číslo
double round(double x); // Zaokrouhlí číslo na nejbližší celé číslo
```

- žádné chyby nevrací

logaritmy

```
double exp(double x);
```

- Vrací e na x

```
double log(double x);
```

- Vrací ln(x)

```
double log10(double x);
```

- Vrací log(x)
- chyby log(), log10
  - $x \leq 0 \rightarrow$ errno = `EDOM`
  - x blízké k 0 $\rightarrow$ errno = `ERANGE`

mocniny

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

- Vrací hodnotu x na y
- EDOM:
  - (x < 0) && (y není celé)
  - (x == 0) && (y < 0)

```
double sqrt(double x);
```

- Vrací odmocninu z čísla x
- `EDOM` pro `sqrt()` záporného čísla

Práce s reálným číslem

```
double frexp(double x, int *nptr);
```

- číslo dle IEEE754 rozdělí na mantisu $\langle 0{,}5;\ 1)$ v návratové hodnotě a exponent `nptr`

```
double ldexp(double x, int n);
```

- z mantisy `x` a exponentu `n` vytvoří reálné číslo dle IEEE754

```
double modf(double x, double *iptr);
```

- Vrátí desetinnou část `x` a celou část uloží do `*iptr`

Celočíselné dělení

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

- vrací celočíselný zbytek po celočíselném dělení x/y

```
div_t div(int n, int d);
```

```
ldiv_t ldiv(long n, long d);
```

- vrací

```
quot = n/d
rem = n%d
```

- ve struktuře

```
typedef struct
{
	int quot;
	int rem;
}
div_t
```

Vstupní a výstupní operace

Úvod

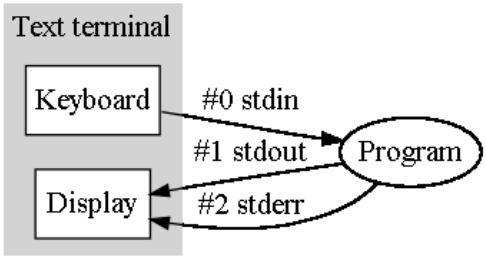
- v stdio.h (Standard Input Output)
- založeny na bufferovaných proudech (bufferred streams)
- stdio.h definuje strukturu `FILE` – informace o proudu
  - každá CRT překladače jiná – příklad Microsoft CRT→ nikdy ne přímý přístup k položkám

```
typedef struct
{
	char *_ptr;
	int   _cnt;
	char *_base;
	int   _flag;
	int   _file;
	int   _charbuf;
	int   _bufsiz;
	char *_tmpfname;
} FILE;
```

- proud pro jakékoli I/O zařízení – soubory, konzole, paměť, ...
- proud – určuje mechanizmy práce s daty (R/W)

standardní proudy

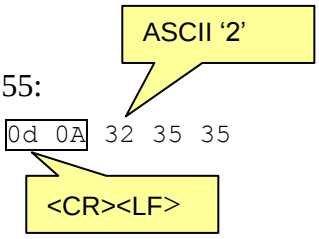
- po spuštění programu k dispozici 3 proudy (obr. z wikipedia.org)




```
extern FILE *stdin
extern FILE *stdout
extern FILE *stderr
```

- chybový výstup

Soubory v C



- textový vs. binární soubor – uložení dvou čísel typu `char`: 1 a 255:
  - textový soubor (každé číslo na samostatném řádku): `31 0d 0A 32 35 35`
  - binární soubor: `1 FF`
-
- textové soubory – organizovány po řádcích, končí domluveným znakem:
  - Windows → `<CR><LF>` (`"\n\r"`)
  - Linux → `<LF>` (`"\n"`)
  - Macintosh → `<CR>` (`"\r"`)
- Práce se souborem
  1) otevření souboru
  2) práce s obsahem
     - čtení, zápis
     - pohyb po souboru – seek
  3) uzavření souboru

Otevření souboru

- = „namapování“ konkrétního souboru na proměnnou typu `FILE`

```
FILE * fopen(jmeno_souboru,"mod");
```

- `Jmeno_souboru` = plné platné jméno souboru v OS (možné včetně cesty)
  - např. `"C:\\Dokumenty\\soubor.txt"`
- `mod` = přístupové modifikátory pro práci se souborem
  - Textové soubory:
    - `r` čtení (read)
    - `w` zápis, nebo přepsání (write)

`r+`	čtení a zápis
`w+`	čtení, zápis nebo přepsání
`a`	zápis na konec (append)
`a+`	čtení a zápis na konec

- Binární soubory:
  = textové, jen s `b` (`rb`, `ra+`…)
- vrací:
  - OK → handle (platnou proměnnou typu `FILE *`)
  - chyba → `NULL`
  - upřesnění chyby – `errno`

- přehled činnosti a reakcí

mód	soubor		počáteční seek
	existuje	neexistuje	
r	R	`NULL`	0
w	Del, W	W	0
a	W	W	End
r+	R/W	`NULL`	0
w+	Del, R/W	R/W	0
a+	R/W	R/W	End

- pozn.: módy +
  - po zápisu lze číst až po volání `fsetpos()`, `fseek()`, `rewind()` nebo `flush()`
- ošetření chyb

```
FILE *fr;
errno = 0;
fr = fopen("source.txt", "r");
if (fr == NULL)
      printf("Chyba pri otevirani souboru: \"%s\"\n", strerror(errno));
```

Uzavření souboru

- po skončení práce
- bufferované streamy → neuzavření souboru před skončením programu = možná ztráta dat

```
int fclose(FILE *stream);
```

- uzavře soubor a vyprázdní `stream`

Přesměrování proudů

- do/ze souboru
- přes možnosti OS (DOS, Konzole Win32, UNIX)
  - c:\program.exe > stdout.txt
  - c:\program.exe < stdin.txt

```
FILE *freopen(const char *path, const char *mode, FILE *stream);
```

- přesměruje výstup z proudu `stream` do souboru `path` otevřený s příznaky `mode`

```
int a;
freopen("vystup.txt", "w", stdout);
freopen("vstup.txt", "r", stdin);
scanf("%d", &a);  // bude se cist ze souboru
printf("%d", a);  // vypis do souboru
```

Vstup / výstup znaku

```
int fgetc(FILE* stream);
int getc(FILE* stream);
int getchar();
```

makro (efektivita)

```
#define getchar() getc(stdin)
```

- vrací jeden znak ze proudu `stream`
- návratová hodnota `int` → vrací -1 jako chybu své práce
- při uložení do `char` → přetypování:

```
char znak = (char)getchar();
```

- problém bufferovaného `stdin`

```
int znak1, znak2;
znak1 = getchar();
znak2 = getchar();
```

vyčká na uživatele;
uživatel na klávesnici `'a'`;
buffer klávesnice `'a''\n'`;
fce vybere z bufferu `'a'`

z bufferu přečte `'\n'`
na uživatelský vstup nečeká

  - nutno přečtení bufferu **po každém!!!** volání `getchar()` až do `'\n'`:

```
Znak1 = getchar();
while (getchar() != '\n');    // vyprazdneni bufferu
Znak2 = getchar();
while (getchar() != '\n');    // vyprazdneni bufferu
```

```
fputc(int c, FILE* stream);
putc(int c, FILE* stream);
putchar(int znak);
```

makro (efektivita)

```
#define putchar(c) putchar(c, stdout)
```

- výstup znaku `c` do proudu `stream`
- použití:

```
char znak;
puchar(znak);      // vypise znak
puchar('\n');      // odratkuje
```

```
int ungetc(int c, FILE* stream);
```

- vrací znak `c` do proudu `stream`
- případě úspěchu vrací `c`, jinak `EOF`.
- volání max 1x za sebou, jinak nutno čtení nebo změna pozice ve streamu

Vstup / výstup řetězce

```
char *fgets(char *str, int n, FILE *stream);
```

- do `str` uloží max. n znaků z proudu `*stream`, za načtené znaky přidá `'\0'`
- čtení končí:
  - konec souboru
  - konec řádku
  - načteno n–1 znaků
- Poznámky:
  - pracuje správně i případě, že na posledním řádku není znak konce řádku
  - do `str` je uložen i `'\n'`

```
char *gets(char *str);
```

- přečte znaky z stdin až do `'\n'`, který nahradí `'\0'`, a uloží je do `str`

```
char retezec[50];
gets(retezec);
```

- nebezpečná funkce → hrozí přetečení bufferu = velikost `retezec` < znaků na klávesnici

```
char *fputs(char *str, FILE *stream);
```

- zapíše řetězec `str` do proudu `stream`

```
char *puts(char *str);
```

- zapíše řetězec `str` do proudu stdout až (`'\0'` nahradí `'\n'`)

```
char retezec[50];
puts(retezec);
```

Formátovaný vstup / výstup řetězce

funkce s PPP

```
int printf(const char *format [,argument]...);
int fprintf(FILE *stream, const char *format [,argument]...);
```

- zapíše formátovaný řetězec `format` do proudu `stream` (stdout)
- formátovací řetězec viz Příloha I
- Příklad

```
SouborOUT = fopen("output.txt","w");
fprintf(SouborOUT, "Realne cislo = %lf\n", RealneCislo);
fprintf(SouborOUT, "Retezec = %s", Retezec);
fclose(SouborOUT);
```

- obsah souboru

```
Realne cislo = 1.234000
Retezec = Ahoj mami
```

```
int fscanf(FILE *stream, const char *format [,argument ]...);
```

```
int scanf(const char *format [,argument ]...);
```

- přečte z proudu `stream` (stdin) řetězec `format` a dle formátovacích řetězců v něm obsaženém uloží obsah do proměnných z seznamu argument
- viz příloha II
- příklad

```
int i;
double fp;
char c, s[81];
scanf("%d %lf %c %80s", &i, &fp, &c, s);
printf("%d %lf %c %s", i, fp, c, s);
```

Detekce konce řádku

- neexistuje konstanta EOL (End Of Line) – různé OS definovány jinak

```
int znak;
while ((znak = getc(soubor)) != '\n')
{
      if (znak >= 32)
      {
            // zpracovani znaku
      }
}
```

ve Win přeskočení `'\r'`

Detekce konce souboru

- konstanta `EOF` (End Of File) – obvykle `-1`

```
int znak;
while ((znak = getc(soubor)) != EOF)
{
      // zpracovani znaku
}
```

Příklad I

- Náhrada `.` za `,` v souboru s naměřenými daty

```
int Znak;
FILE *SouborIN, *SouborOUT;
SouborIN = fopen("input.txt","r");
SouborOUT = fopen("output.txt","w");
while ((Znak = getc(SouborIN)) != EOF)
{
      if (Znak == '.')
            Znak = ',';
      putc(Znak, SouborOUT);
}
fclose(SouborIN);
fclose(SouborOUT);
```

- opět chybí ošetření chyb

Příklad II

- Program přečte soubor `input.txt` z disku, zobrazí ho na obrazovce a uloží kopii do souboru `output.txt`. Na začátek souboru `output.txt` připojí poznámku „Toto je kopie souboru input.txt“.

```
#define MAX_ZNAKU_NA_RADEK 80       // uzitecnych, konzole 80x25 znaku
char Radek[MAX_ZNAKU_NA_RADEK+1];   // +1 pro '\0'
FILE *SouborIN, *SouborOUT;
// otevreni souboru
SouborIN = fopen("input.txt","r");
SouborOUT = fopen("output.txt","w");
fputs("Toto je kopie souboru input.txt\n\n", SouborOUT);
while (fgets(Radek, MAX_ZNAKU_NA_RADEK+1, SouborIN) != NULL)
{
     // fgets cte max (n-1) znaku, +1 na posl. znak, +2 na '\n'
      printf("%s", Radek);          // na obrazovku
      fputs(Radek, SouborOUT);      // do jineho souboru
}
fclose(SouborIN);
fclose(SouborOUT);
```

- výpis na obrazovku → `printf()`, `puts()` = `'\n'` navíc (je v řetězci od `fgets()`)
    - opět chybí ošetření chyb

Příklad III

- čtení čísla ze souboru (délka řetězce před číslem neznámá)

```
Realne cislo = 1.234000
Retezec = Ahoj mami
```

```
while (((Znak = getc(SouborIN)) < '0') || (Znak > '9'));
ungetc(Znak,SouborIN);  // vraceni '1' zpet du bufferu
fscanf(SouborIN, "%lf", &JineRealneCislo);
```

Další souborové funkce

- stdio.h

```
int fflush(FILE *stream);
```

- vyprázdní datového proudu `stream`. Vrací 0 = úspěch, jinak `EOF`.

```
int feof(FILE *stream);
```

- vrací kladnou nenulovou hodnotu v případě dosažení konce souboru `stream`, jinak 0.

```
int ferror(FILE *stream);
```

- v případě chyby souboru `stream` vrací kladnou nenulovou hodnotu, jinak 0.

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

- nuluje indikátor konce souboru a chyby pro souborový proud `stream`.

```
int fileno(FILE *stream);
```

- vrací (odkazem) deskriptor souboru datového proudu `stream`.
- pro přístup dle POSIX

Práce se souborem na úrovni OS

- stdio.h

```
int remove(const char *pathname);
```

- smaže soubor `pathname`

```
int rename(const char *oldpath, const char *newpath);
```

- přejmenuje soubor `oldpath` na soubor `newpath`.

Práce s binárními soubory

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

- posune kurzor v souboru `stream` na pozici vzdálenou `offset` od místa `whence`.

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

- Vrátí aktuální pozici kurzoru souboru `stream`.

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

- Posune kurzor na začátek souboru `stream`.

```
int fgetpos(FILE *stream, fpos_t *pos);
```

- uloží aktuální pozici kurzoru do proměné `pos`.

```
int fsetpos(FILE *stream, fpos_t *pos);
```

- obnoví pozici kurzoru z proměné `pos`.

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

- načte `count` dat o velikosti 1 položky `size` bytů z proudu `stream` do paměti, kam ukazuje `ptr`. Vrací počet správně načtených položek.

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

- zapíše `count` dat o velikosti 1 položky `size` bytů do proudu `stream` z paměti kam ukazuje `ptr`. Vrací počet správně zapsaných položek.

Příloha I– formátovací řetězce printf()

`%[flags][width][.precision][length]specifier`

- `specifier` určuje typ a interpretaci příslušného argumentu

	výstup	příklad
`c`	znak	`a`
`d` nebo `i`	znaménkové celé číslo	`392`
`e`	reálné číslo v exp. tvaru	`3.9265e+2`
`E`	reálné číslo v exp. tvaru	`3.9265E+2`
`f`	reálné číslo dekadicky	`392.65`
`g`	kratší z `%e` nebo `%f`	`392.65`
`G`	kratší z `%E` nebo `%f`	`392.65`
`o`	znaménkové celé číslo v osmičkové soustavě	`610`
`s`	řetězec (pointer na `char`)	`sample`
`u`	bezznaménkové celé číslo	`7235`
`x`	bezznaménkové celé číslo v hexa soustavě	`7fa`
`X`	bezznaménkové celé číslo v hexa soustavě	`7FA`
`p`	adresa (hodnota pointer)	`B800:0000`
`n`	nic netiskne. Argument musí být `int*`, kde je počet znaků	
`%`	(= zdvojené `%`) vypíše `%`	`%`

- `length`

	význam
`h`	argument interpretován jako `short` nebo `unsigned short` (platí pouze pro `i`, `d`, `o`, `u`, `x`, `X`)
`l`	argument interpretován jako `long` nebo `unsigned long` (platí pouze pro `i`, `d`, `o`, `u`, `x`, `X`)
`L`	argument interpretován jako `long double` (platí pouze pro `e`, `E`, `f`, `g`, `G`)

- `precision`

	význam
`číslo`	pro `i`, `d`, `o`, `u`, `x`, `X` – číslo doplněno nulami zleva na délku `číslo` pro `e`, `E`, `f` – počet míst za řádovou čárkou pro `g`, `G` – maximální počet tisknutých znaků pro `s` – počet významných číslic
`*`	přesnost udána jako další argument předcházející aktuálnímu argumentu (nikoli tedy ve formátovacím řetězci jako `číslo`)

- `width`

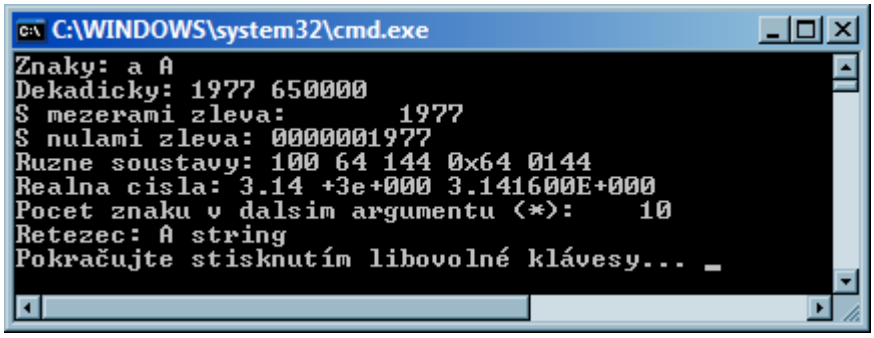
	význam
`číslo`	minimální počet tisknutých znaků. Je-li aktuální hodnota menší, argument doplněn nulami mezerami zleva
`*`	počet znaků udán jako další argument předcházející aktuálnímu argumentu (nikoli tedy ve formátovacím řetězci jako `číslo`)

- `flags`

	význam
`-`	výstup je zarovnán doleva v rámci počtu znaků daného `width` (standardně doprava)
`+`	vynutí tisk znaménka (standardně jen –)
`#`	pro `specifier o` vytiskne na začátku `0` pro `x` vytiskne na začátku `0x` pro `X` vytiskne na začátku `0X` pro `e E f` vynutí tisk řádové čárky pro `g G` vynutí tisk řádové čárky a doplní nevýznamné nuly zprava za řádovou čárkou
`0`	tiskne nevýznamné nuly zleva

Příklady

```
printf ("Znaky: %c %c \n", 'a', 65);
printf ("Dekadicky: %d %ld\n", 1977, 650000);
printf ("S mezerami zleva: %10d \n", 1977);
printf ("S nulami zleva: %010d \n", 1977);
printf ("Ruzne soustavy: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
printf ("Realna cisla: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
printf ("Pocet znaku v dalsim argumentu (*): %*d \n", 5, 10);
printf ("Retezec: %s \n", "A string");
```



Příloha II – scanf()

```
int scanf(const char *format [,argument ]...);
```

- `format` může obsahovat
  - bílé znaky – na příslušné pozici v `stdin` bude `scanf()` ignorovat jakýkoli počet bílých znaků (mezera, tab, enter). První nebílý znak za bílými bude přečten
  - Nebílé znaky (kromě `%`) – `scanf()` přečte další znak a porovná jej s příslušným znakem ve formátovacím řetězci. Bude-li:
    - shodný – `scanf()` jej vynechá
    - různý – `scanf()` ukončí čtení (zbytek znaků zůstane v `stdin`)
  - formátovací řetězce – uvozené `%`

Formátovací řetězec

```
%[*][width][length]specifier
```

- `specifier` určuje typ a interpretaci příslušného argumentu

	vstup	argument
`c`	znak	`char *`
`d`	dekadické číslo (se znaménkem + –)	`int *`
`e E f g G`	reálné číslo (se znaménkem + –) (v exp. tvaru `3.9e+2`)	`float *`
`o`	celé číslo v osmičkové soustavě	`int *`
`s`	řetězec (čte dokud nenarazí na bílý znak)	`char *`
`u`	bezznaménkové celé číslo	`unsigned int *`
`x X`	celé číslo v hexa soustavě	`int *`

- `length`

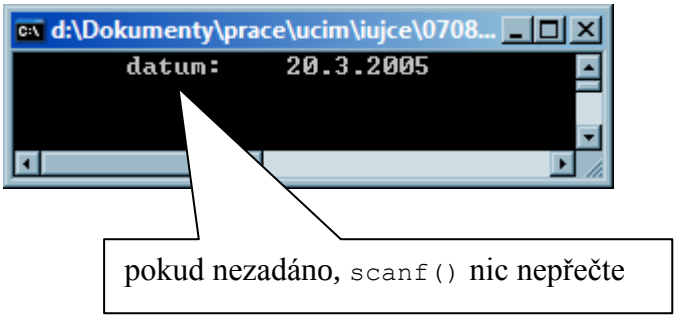
	význam
`h`	argument interpretován jako `short` nebo `unsigned short` (platí pouze pro `i`, `d`, `o`, `u`, `x`, `X`)
`l`	argument interpretován jako `long` nebo `unsigned long` (platí pouze pro `i`, `d`, `o`, `u`, `x`, `X`) argument interpretován jako `double` (platí pouze pro `e`, `E`, `f`, `g`, `G`)
`L`	argument interpretován jako `long double` (platí pouze pro `e`, `E`, `f`, `g`, `G`)

- `length`
  maximální počet znaků čtených pro daný formátovací řetězec

- `*`
  pro daný formátovací řetězec bude stdin čten, ale nebude ukládáno do argument

Příklady

```
int den, mesic, rok;
scanf(" datum: %d.%d.%d", &den, &mesic, &rok);
```



pokud nezadáno, `scanf()` nic nepřečte