

## **Popis funkcí a maker standardních knihoven**

`<assert.h>` Makro používané při ladění.

## <ctype.h> Práce se znaky.

isalnum()	čísllice nebo malá či velká písmena
isalpha()	malá či velká písmena
iscntrl()	řídící znaky
isdigit()	čísllice
isgraph()	znaky s grafickou podobou
islower()	malá písmena
isprint()	tisknutelné znaky
ispunct()	interpunkční znaky
isspace()	bílé znaky
isupper()	velká písmena
isxdigit()	hexadecimální číslice
tolower()	konverze velkých pímen na malá
toupper()	konverze malých písmen na velká

<math.h>

sin(x)	$\sin(x)$
cos(x)	$\cos(x)$
tan(x)	$\tan(x)$
asin(x)	$\arcsin(x)$
acos(x)	$\arccos(x)$
atan(x)	$\arctan(x)$
atan2(x, y)	<b>arkustangents</b> $x, y$
sinh(x)	$\sinh(x)$
cosh(x)	$\cosh(x)$
tanh(x)	$\tanh(x)$
exp(x)	$e^x$
log(x)	$\ln(x)$
log10(x)	$\log(x)$
pow(x, y)	$x^y$
sqrt(x)	$\sqrt{x}$
ceil(x)	$y \in \mathbb{Z}, y \geq x$ , nejmenší
floor(x)	$y \in \mathbb{Z}, y \leq x$ , největší
fabs(x)	$ x $
ldexp(x, n)	$x \times 2^n$
frexp(x, exp)	$x = m \times 2^e xp$
modf(x, n)	$x$ na celou $n$ a desetinou část
fmod(x, n)	reálný zbytek po dělení $x/y$

1. Parametry funkcí a návratové hodnoty jsou typu double.

2. Při chybě je v errno

EDOM - vstup mimo definiční obor funkce.

ERANGE - Výstup  $\pm$ HUGE\_VAL nebo 0.0.

```
int i
double g, f;

printf("%f \n", ceil(3.14));    4.000000
printf("%f \n", floor(3.14));   3.000000

f = frexp(3.14, &g);
printf("%f x 2^%d \n", f, i);  0.785000 x 2^2

f = modf(-3.14, &g)
printf("%f + %f\n", g, f);    -3.000000

printf("%f \n", fmod(3.14, 2.0));  1.140000
```

<locale.h> Přizpůsobení C národnímu prostředí.

## Problémy

- znaková sada, např. `isupper(Š)`,
- desetinná tečka versus naše desetinná čárka,
- zápisu času a data, např. 10.6.1995 nebo 6-10-1995.

`setlocale()` nastavuje  
`localeconv()` vrací

<setjmp.h> Umožnění "dlouhých" skoků.

`setjmp()` připraví podmínky a návratový bod  
`longjmp()` provede nelokální skok.

Funkce `setjmp()` uloží a funkce `longjmp()` obnoví obsah potřebných registrů procesoru, typ je `jmp_buf`.

Princip jednotné reakce na chybu s následným restartem programu.

```
#include <studio.h>
#include <setjmp.h>

void chyba(void); /* funkčni prototyp */
jmp_buf hlavni_smycka; /* globalni promenna*/

int main(void)
{
    int    i, j;

    printf(" Zacatek programu  \n") ;
    do {
        printf("Zadej dve cisla :");
        scanf("%d %d", &i , &j );
        if(j !=0)
```

```
    printf(" %d /%d=%d\n",i,j, i/j);
else
    chyba();
} while(i !=0);
}
```

```
void chyba(void);
{
    printf("Chyba - nulovy delitel \n\n");
    longjmp(hlavni_smycka,1);
}
```

```
void chyba(jmp_buf kam_se_vratit, char *chybova_zprava)
{
    printf("Chyba: %s\n", chybova_zprava );
    longjmp(kam_se_vratit ,1);
}
```

<signal.h> Zpracování signálu.

signal() zavede novou obsluhu signálu  
raise() vyšle signál programu

```
#include <studio.h>
#include <signal.h>
#include <setjmp.h>
#define POCET_POKUSU 2
#define vynech 5000
/* znak se vypisuje jednou za vynech*/

jmp_buf navrat;
char vypisovany_znak = '#';

void obsluha_signalu(int nevyuzity_parametr)
{static int pocet_vstupu = 0;
  pocet_vstupu++;
  if (pocet_vstupu <= pocet_pokusu) {
    printf("\n%d: A zrovna ne \n", pocet_vstupu);
    vypisovany_znak++; }
}
```



```

    longjmp(navrat, pocet_vstupu);
}

int main(void)
{long int i = 0;
  int pid;
  void (*stara_adr)(int);
  /* zavedeni nove adresy obsluhy SIGQUIT */
  stara_adr = signal(SIGQUIT, obsluha_signalu);
  /* ziskani identifikacniho cisla procesu */
  pid=getpid();
  printf("\nPID =%d \n\n" , pid);
  printf("Zadej trikrat prikaz: kill -%d %d \n",
        SIGQUIT, pid);
  if (setjmp(navrat) < 0 pocet_pokusu) {
    while(1) {
      if (++i % vynech == 0) {
        putchar(vypisovany_znak);
        fflush(stdout); } } }
  printf("\n Tak tedy ano ... \n"];
  /* obnoveni puvodni adresy obsluhy SIGQUIT */
  signal(SIGQUIT, stara_adr);
}

```

<stdarg.h> Práce s proměnným počtem parametrů.

va_start()	nastaví na začátek
va_arg()	dá další položky seznamu
va_end()	ukončí práci se seznamem

<stdio.h> Funkce pro vstup a výstup.

<code>fopen()</code>	otevívá soubor
<code>freopen()</code>	přesměruje otevřený soubor do jiného souboru
<code>fflush()</code>	zapiše obsah bufferu do souboru
<code>fclose()</code>	uzavře soubor
<code>remove()</code>	vymaže soubor
<code>rename()</code>	přejmenuje soubor
<code>tmpfile()</code>	vytvoří dočasný soubor
<code>tmpnam()</code>	vytvoří dočasné jméno souboru
<code>setbuf()</code>	změní buffer
<code>setvbuf()</code>	změní buffer a způsob práce s ním
<code>fprintf()</code>	zapisuje formátovaně do souboru
<code>printf()</code>	zapisuje formátovaně do stdout
<code>sprintf()</code>	zapisuje formátovaně do řetězce
<code>vfprintf()</code>	zapisuje do souboru proměnný počet parametrů
<code>vprintf()</code>	zapisuje do stdout proměnný počet parametrů
<code>vsprintf()</code>	zapisuje do řetězce proměnný počet parametrů
<code>fscanf()</code>	čte formátovaně ze souboru
<code>scanf()</code>	čte formátovaně ze stdin
<code>sscanf()</code>	čte formátovaně z řetězce
<code>vfscanf()</code>	čte ze souboru proměnný počet parametrů
<code>vsscanf()</code>	čte ze stdin proměnný počet parametrů
<code>fgetc()</code>	čte znak ze souboru - vždy funkce
<code>getc()</code>	čte znak ze souboru - může být makro
<code>fputs()</code>	zapisuje řetězec do souboru
<code>puts()</code>	zapisuje řetězec do stdout
<code>getchar()</code>	čte znak ze stdin

<code>fread()</code>	čte neformátovaně ze souboru
<code>fwrite()</code>	zapisuje neformátovaně do souboru
<code>fseek()</code>	přesune se v souboru
<code>ftell()</code>	vrátí pozici aktuálního místa v souboru
<code>rewind()</code>	přesune se na začátek souboru
<code>fgetpos()</code>	uloží pozici aktuálního místa v souboru
<code>fsetpos()</code>	vrátí se na uloženou pozici v souboru
<code>clearerr()</code>	nuluje indikátor chyby a konce souboru
<code>feof()</code>	testuje příznak konce souboru
<code>ferror()</code>	testuje příznak chyby práce se souborem
<code>perror()</code>	tiskne chybu podle obsahu <code>errno</code>

<stdlib.h>

Obecně využitelné funkce.

atof()	konverze řetězce na double
atoi()	konverze řetězce na int
atol()	konverze řetězce na long int
strtod()	konverze řetězce na double
strtol()	konverze řetězce na long int
strtoul()	konverze řetězce na unsigned long int
rand()	generátor náhodného čísla
srand()	inicializace generátoru náhodného čísla
calloc()	alokace dynamické paměti pro pole objektů
malloc()	alokace bloku dynamické paměti
realloc()	změna velikosti bloku dynamické paměti
free()	uvolnění dynamické paměti
abort()	nestandardní ukončení programu
exit()	standardní ukončení programu
atexit()	registrace ukončující funkce
system()	volání systémového příkazu
getnev()	poskytnutí informace o systémové proměnné
bsearch()	binární vyhledávání
qsort()	třídění logaritmem quick–sort
abs()	absolutní hodnota typu int
labs()	absolutní hodnota typu long int
div(x, y)	$x/y$ a $x\%y$ současně pro typy int
ldiv(x, y)	$x/y$ a $x\%y$ současně pro typy long int

### *Konverze řetězců na čísla*

Poznámka: ANSI C nedefinuje opačné funkce, tj. pro převod čísel na řetězce, např. itoa(). V těchto případech je nutno použít funkci sprintf().

```
char *s, *chyba;
long int l;

s = " 1234567";
l = atol(s);
printf("s=%s  l = %ld\n",s,l);
    /* s=1234567  l=1234567 */
s="1234567";
l= strtol(s, &chyba , 10);
printf("s=%s  l = %ld\n",s,l);
    /* s=1234567  l=1234567 */

s="ffff";
l=strtol(s, &chyba ,16);
printf("s=%s  l = %ld \n",s,l);
    /* s=ffff      l=65535 */
```

```
s= " 0xFGFFF"
l=strtol(s, &chyba , 16);
printf("s=%s  l=%ld \nchybny znak: %s \n",
       s, l ,chyba);
    /* s = 0xFGFFF  l = 15 */
s= 1111111111111111; l = 15;
    /* chybný znak:  GFFF */
l= strtol(s, &chyba, 2);
printf("s=%s  \n l = %ld \n" ,s,l);
    /* s = 1111111111111111 l = 65535 */
```



## *Generátor pseudonáhodných čísel*

Používají se dvě funkce:

1. `int rand(void)`

Rozsah  $< 0, RAND\_MAX$ ), při každém spuštění programu je posloupnost stejná. Použití: `rand() % rozsah`, např. `rand() % 20`

```
#define real_rand()\n((double)rand()/(RAND_MAX+1.0))
```

2. `void srand(unsigned int start)`

Inicializuje generátor `rand()` počáteční hodnotou `start`.

```
#include <studio.h>
#include <studio.h>
#include <time.h>

int main(void)
{
int i;

srand(( unsigned)  time(NULL));
printf("\nDeset nahodnych cisel od 0 do 99\n");

for(i=0; i<10; i++)
    printf("%2d  ", rand() % 100);
}
```

*Funkce pracující s dynamickou pamětí* Funkce typu `malloc()`.

*Funkce pro spolupráci s operačním systémem* Funkce `abort()` nestandardně ukončí program, tzn. že nezapíše buffery, nemaže dočasné soubory, nespouští funkce registrované pomocí `atexit()`, atd. V UNIXU se snaží zavřít soubory a generuje soubor `core`.

*Funkce* `atexit()` zaregistruje funkci `stop_funkce()`. Zaregistrovaná funkce se spustí po ukončení funkce `main()`. Lze jich zaregistrovat až 32; jsou volány v obráceném pořadí, než byly registrovány.

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

void exit_fn1(void)
{ printf("1.exit_funkce\n"); }
void exit_fn2(void)
{ printf("2.exit_funkce \n"); }
int main(void)
{printf("Zacatek & registrace exit_fn \n");
  atexit(exit_fn1);
  atexit(exit_fn2);
  printf("Konec programu \n");
}
```

Program vypíše:

```
Zacatek & registrace exit_fn
Konec programu
2. exit_funkce
1. exit_funkce
```

*Funkce* `exit()` Standardně ukončí provádění programu, funguje jako příkaz `return` v `main()`, tzn. zapíše se buffery, uzavřou se soubory, volají se všechny funkce registrované pomocí `atexit()`, mažou se dočasné soubory, atd.

```
if(( fr=fopen("nesmysl.dat","r")) == NULL) {  
printf("Soubor neexistuje \n ");  
exit(1);  
}
```

*Funkce* `system()` vyvolá z programu příkaz operačního systému.

```
#include<studio.h>
#include<stdlib.h>

int main(void)
{
    printf("Vyvolani prikazu dir z programu \n");
    system("dir *.txt");
}
```

## *Funkce pro hledání a třídění*

Používá se funkce:

```
void qsort( void *pole_trid, size_t n_prv,
           size_t vel_prv,
           int(*porov_fce)(const void*,
                          const void*));
```

a `porov_fce()` vrací:

$< 0$  je-li první parametr menší než druhý,

$0$  jsou-ji parametry shodné,

$> 0$  je-li první parametr větší než druhý.

## *Binární vyhledávání v setříděném poli*

```
void *bsearch(const void *klic,  
             const void *pole_trid,  
             size_t n_prv,  
             size_t vel_prv,  
             int(*porov_fce)  
             (const void*,const void*));
```



## <string.h>

strcpy()	kopírování řetězce
strncpy()	kopírování řetězce s omezením délky
strcat()	spojení řetězců
strncat()	spojení řetězců
strcmp()	porovnání řetězců
strncmp()	porovnání řetězců
strchr()	hledání znaku v řetězci
strrchr()	hledání znaku v řetězci od jeho konce
strstr()	hledání podřetězce v řetězci
strlen()	aktuální délka řetězce
strspn()	hledání prvního rozdílného znaku
strcspn()	hledání prvního shodného znaku
strpbrk()	počet rozdílných znaků
strerror()	poskytnutí chybového hlášení
strtok()	dělení řetězce na části
memcpy()	kopírování bloků bajtů
memmove()	jako memcpy(), bloky se mohou překrývat
memcmp()	porovnání bloků bajtů
memchr()	hledání znaku v bloku bajtů
memset()	vyplnění bloku bajtů konstantou

```
#include <stdio.h>
#include <string.h>

int main(void)
{
char *s1 = "nazdar ahoj, dobry_den";
char *s2 = "qwx radzan";
char *s3= " qwxr";
char *s4= "od";
int pozice;
char *p_c;
pozice=strspn(s1,s2);
printf("Prvnich %d znaku s1 je v s2\n", pozice);
pozice=strcspn(s1,s3);
printf("Prvnich %d znaku s1 neni v s3\n", pozice);
p_c=strpbrk(s1,s4);
if(p_c != NULL)
    printf(" Prvni shodny znak s1 a s4: %c\n",*p_c);
else
    printf("Zadny shodny znak \n");
p_c =strtok(s1," , -"); /*mezera ,carka pomlcka */
```

```
if(p_c !=NULL)
    printf("%s \n",p_c);

while((p_c = strtok(NULL, " ,-")) !=NULL)
    printf("%s\n",p_c);
}
```

Program vypíše:

```
Prvních 8 znaků s1 je obsazeno v s2
prvních 5 znaků s1 není obsazeno v s3
První shodný znak : d
nazdar
ahoj
dobry
den
```

```
#include <stdio.h>
#include <string.h>
#define delka 1000
```

```
int main(void)
{
int a[delka],b[delka];
int i, *p_i;

memset((void *)a,0,(size_t)(delka* sizeof(int)));
printf("memset: a[%d] = %d \n", delka/2, a[delka/2]);
a[delka/2]=1;
p_i =(int *) memchr((void *) a, 1,(size_t)(delka *
printf("memchr: i=%u \n",p_i - a);
memcpy((void*)b,(void*a,(size_t)(delka*sizeof(int)))
printf("memcpy: b[%d]=%d \n", delka/2 ,b[delka /2])

for(i=1; i < 9; i++)
    a[i]=i-1;
for(i=0 ; i < 9; i++)
printf("a [%d]=%d ",i,a[i]);

printf("\nmemmove: \n");
memmove((void*) &a[0],(void*) &a[1],
```

```
(size_t)(9*sizeof(int)));
```

```
for(i=0; i < 9;i++)  
printf("a[%d]=%d", i, a[i]);  
}
```

```
memset: a[500] =0
```

```
memchr: i=500
```

```
memcpy: b[500] =1
```

```
a[0]=0 a[1]=0 a[2]=1 a[3]=2 a[4]=3
```

```
a[5]=4 a[6]=5 a[7]=6 a[8]=7
```

```
remove:
```

```
a[0]=0 a[1]=1 a[2]=3 a[3]=3 a[4]=4
```

```
a[5]=5 a[6]=6 a[7]=7 a[8]=0
```

`<time.h>` Práce s datem a časem.

<code>clock()</code>	tiků procesoru
<code>time()</code>	sekund od 1.ledna 1970
<code>difftime()</code>	rozdíl dvou časů
<code>mktime()</code>	práce s položkami času
<code>asctime()</code>	složky času do řetězce
<code>ctime()</code>	počet sekund na řetězec
<code>gmtime()</code>	počet sekund na čas
<code>localtime()</code>	složky času z počtu sekund
<code>strftime()</code>	form. složky času do řetězce

`CLK_TCK` počet procesorových tiků za sekundu,

`clock_t` obvykle `long`, pro funkci `clock()`,

`time_t` obvykle `long`, pro ostatní funkce než `clock()`,

`tm` pro uložení složek času.

## Struktura

tt tm obsahuje tyto položky:

int tm_sec	sekundy (0, 59)
int tm_min	minuty (0, 59)
int tm_hour	hodiny (0, 23)
int tm_mday	den v měsíci (1, 31)
int tm_mon	měsíc od ledna (0, 11)
int tm_year	roky od 1970 (0, )
int tm_wday	dny do neděle (0, 6)
int tm_yday	počet dnů od 1. ledna (0, 365)
int tm_isdst	příznak letního času

```
clock_t  zac, konec;
```

```
zac =clock();
```

```
konec = clock();
```

```
printf("program trval :  %f [sec] \n",  
      (konec-zac)/(double) clk_tck);
```

---

```
time_t  pocet_sec, n_sec;
```

```
pocet_sec = time(&n_sec);  
pocet_sec = time(&pocet_sec);  
pocet_sec = time(NULL);
```

---

```
struct tm tm_promenna, *p_tm;  
time_t cas;  
time(&cas);  
tm_promenna = *localtime(&cas); /*prirazeni strukturu*/  
if(tm_promenna.tm_min == 2)  
    p_tm = localtime(&cas); /*prirazeni ukazatelu*/  
if(p_tm->tm_min ==2)  
    ...
```

---

Mon Jun 19 16:00:14 1995



```
time_t t ;
```

```
time(&t);
```

```
printf("%s\n", asctime( localtime(&t)));
```

```
printf("%s\n" , ctime(&t));
```

Program vypíše:

```
Mon Jun 19 16:00:14 1995
```

```
Mon Jun 19 16:00:14 1995
```

```
#include <studio.h>
#include <time.h>

#define posun 20000
int main(void)
{
    struct tm *p_pak;
    time_t ted;

    char*dny []={"Ne", "Po", "Ut", "St", "Ct", "Pa", "So"};
    time(&ted);
    p_pak=localtime( &ted);
    p_pak->tm_min += posun;

    mktime(p_pak);
    printf("Za %d bude %s. \n",posun,
           dny [p_pak->tm_wday] );
```

Příklad: Následující program bude pokračovat ve své činnosti dnes ve 20 hodin plus tolik minut, kolik je nyní, plus ještě 15 minut.

```
struct tm *p_pak;
time_t ted, spusteni;
time(&ted);
p_pak = localtime(&ted);
p_pak ->tm_min +=15 ;    /*prirustkove*/
p_pak ->tm_hour =20 ;    /*absolutni*/
spusteni = mktime(p_pak);
while(difftime(time(NULL), spusteni) < 0)
    /* prazdny cyklus cekani*/
```