Práce s řetězci

- string.h

```
size_t strlen(const char *str);
```

- vrací: délku řetězce `str` (bez `'\0'`)

```
int strcmp(const char *str1, const char *str2);
```

- porovná řetězce
- vrací:
  - <0: `str1` je lexikograficky menší než `str2`
  - 0: řetězce jsou lexikograficky identické
  - >0: `str1` je lexikograficky větší než `str2`

```
char *strcpy(char *strDest, const char *strSource);
```

- zkopíruje `strSource` do `strDest`
- vrací:
  - úspěch: pointer na `strDest`
  - neúspěch: nic
- pozor na délku `strDestination`

```
char *strcat(char *strDestination, const char *strSource);
```

```
char *strncat(char *strDest, const char *source, size_t count);
```

- připojí `strSource` k `strDestination` (`'\0'` cílového řetězce přepsána 1. znakem zdroje)
- vrací:
  - úspěch: pointer na `strDestination`
  - neúspěch: nic
- `strncat()` – pouze `count` znaků

```
char *strchr(const char *str, int c);
char *strrchr(const char *str, int c);
```

- hledá výskyt znaku `c` v řetězci `str` (`strrchr()` → od konce)
- vrací:
  - znak nalezen: pointer na první výskyt znaku
  - znak nenalezen: `NULL`

```
char *strstr(const char *str, const char *strSearch);
```

- hledá výskyt řetězce `strSearch` v řetězci `str`
- vrací:
  - řetězce nalezen: pointer na první výskyt znaku
  - řetězce nenalezen: `NULL`
  - prázdný `str`: pointer na `str`

Formátovaný vstup / výstup z řetězce

```
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format,...);
```

- = `printf()` do řetězce `str`

```
int sscanf( const char *str, const char *format, ...);
```

  - = `scanf()` z řetězce `str`.

Závěrečné poznámky

- velmi mnoho dalších funkcí – viz MSDN
- bezpečné varianty (jen Microsoft CRT library ve VS2005 a novější)
  - jména ve tvaru xxx_s
  - není dle ANSI C!!!

Převody řetězec ↔ číslo

- stdlib.h

```
int atoi(const char *str);
long atol(const char *str);
double atof(const char *str);
```

- převedou `str` na číslo
- vrací: úspěch: číslo, konverzi nelze provést: nedefinováno (obvykle 0, 0.0)
- jen kvůli kompatibilitě s UNIX, v ANSI C lepší viz dále

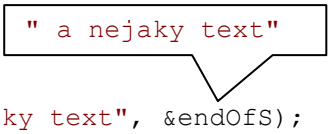
```
double strtod(const char *str, char **endptr);
```

- převede `str` na číslo, nastaví `endptr` na první pozici za převedené číslo (znak, který nelze převést na číslo v dané soustavě)
- číslo ve formátu `[whitespace] [{+ | -}] [0 [{ x | X }]] [digits]`
- případné mezery na začátku přeskočí

```
long strtol(const char *str, char **endptr, int base);
unsigned long strtoul(const char *str, char **endptr, int base);
```

- base je základ soustavy ⟨2; 36⟩
- příklady

```
char *endOfS;
double x;
int i;
x = strtod("   1.2345 a nejaky text", &endOfS);
i = strtol("0x12", &endOfS, 0);          // autodetekce zakladu
i = strtol("10", &endOfS, 16);           // zaklad rucne
```



Práce s pamětí I

- memory h nebo string.h

```
int memcmp(const void *s1, const void *s2, size_t n);
```

- Porovná prvních `n` bajtů polí (blok mem) `s1` a `s2`.
- Vrací
  - 0 = bloky stejné
  - > 0, když s1 > s2
  - < 0, když s1 < s2.

```
void *memcpy(void *dest, const void *src, size_t n);
```

- Zkopíruje `n` bajtů z bloku paměti `src` do `dest`.
- Bloky se nesmějí překrývat.

```
void *memmove(void *dest, const void *src, size_t n);
```

- Zkopíruje `n` bajtů z bloku paměti `src` do `dest`.
- Bloky se smějí překrývat

```
void *memset(void *s, int c, size_t n);
```

- Vyplní blok paměti `s` o délce `n` bajtů hodnotou `c`.
- Vrací ukazatel `s`.

Práce s pamětí II

- stdlib.h

```
void *calloc(size_t nmemb, size_t size);
void *malloc(size_t size);
void free(void *ptr);
void *realloc(void *ptr, size_t size);
```

Spolupráce s OS

- stdlib.h

funkce

```
void abort(void);
```

- abnormální ukončení programu.

```
void exit(int status);
```

- ukončí program kdekoliv v programu – program vrací hodnotu status.

```
int atexit(void (* function)(void));
```

- Přiřadí funkci do seznamu funkcí, které jsou volány při ukončení programu funkcí exit nebo skončením main().
- `function` = ukazatel na ukončující funkci. Registrované jsou volány v opačném pořadí než jsou registrovány.
- vrací 0 = úspěch, -1 = chyba.

```
int system (const char *string);
```

- Spustí příkazový interpret a předá mu řetězec `string`.
- např. `system("PAUSE");` `system("format c:");` `system("mujprogram.exe")`

Makra

`EXIT_FAILURE`

- hodnota argumentu funkce exit pro neúspěšné ukončení programu. (nenulová hodnota)

`EXIT_SUCCESS`

- hodnota argumentu funkce exit pro úspěšné ukončení programu. (nulová hodnota)

`NULL`

- nulový ukazatel.

Generování náhodných čísel

- stdlib.h

`RAND_MAX`

- makro – největší hodnota generovaná `rand()`

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

- Inicializuje generátor pseudonáhodných čísel.
- obvyklé použití

```
#include <time.h>
srand( (unsigned)time( NULL ) )
```

```
int rand(void);
```

- Vrací pseudonáhodná čísla s rovnoměrným rozložením mezi 0 a RAND_MAX (min. 32767)
- bez inicializace vrací stále stejnou posloupnost

Řazení a vyhledávání

```
void qsort(void *base, size_t nmemb, size_t size,
      int (*compar)(const void *, const void *));
```

- řazení algoritmem quicksort pole libovolného typu, na které ukazuje `base` o `nmemb` položkách velikosti `size`.
- `compar` = pointer na porovnávací funkci, musí vracet
  - < 0 pokud (první arg < druhý arg)
  - < 0 pokud (první arg == druhý arg)
  - > 0 pokud (první arg > druhý arg)
- binární vyhledávání

```
void *bsearch(const void *key, const void *base, size_t num, size_t width,
int (*compare) (const void *, const void *));
```

- ve vzestupně setříděném poli `base` (počet prvků `num` a velikost jednoho `width`) hledá výskyt hodnoty `key`
- `compare` → viz. `qsort()`
- vrací: nalezeno → pointer na nalezený prvek
  nenalezeno → `NULL`
- nesetříděné pole → nepředvídatelné chování

- Příklad – setřiďte vzestupně pole čísel typu `int`

```
#include <stdio.h>
#include <stdlib.h>
#define N 6
int porovnej(const int *prvni, const int *druhy)
{
      if ((*druhy - *prvni) < 0) return 1;
      else if ((*druhy - *prvni) > 0) return -1;
      else return 0;
}
int main(void)
{
      int pole[N] = {10,-5,20,31,8,88};
      qsort((void *)pole, N, sizeof(pole[0]), porovnej);
      return 0;
}
```

  - lze také `(int (*)(const void*, const void*)) porovnej`

Práce se znaky

- ctype.h
- makra pro určení typu znaku
  - parametr `char c` – např.
  - vrací `int` ve smyslu true /false

`isalnum()`	je číslice nebo písmeno
`isalpha()`	je písmeno
`isascii()`	je ASCII (0-127)
`iscntrl()`	je ctrl znak (1-26)
`isdigit()`	je číslice
`islower()`	je malé písmeno
`isprint()`	je tisknutelný znak (32 – 126)
`ispunct()`	je interpunkční znak
`isspace()`	je bílý znak
`isupper()`	je velké písmeno
`isxdigit()`	je znak do hexa číslice (0-9. A-F, a-f)
`isgraph()`	je viditelný znak (33 – 126)

- makra pro konverze znaků – vrací konvertovaný znak

`tolower()`	na malé písmeno (jen velká, ostatních si nevšímá)
`toupper()`	na velké písmeno (jen malá, ostatních si nevšímá)
`toascii()`	

- princip použití:

```
char c;
printf("Zadej znak: ");
scanf("%c", &c);
if (islower(c)) {
      printf("znak je maly, konverze na velky\n");
      c = (char)toupper(c);
}
```