

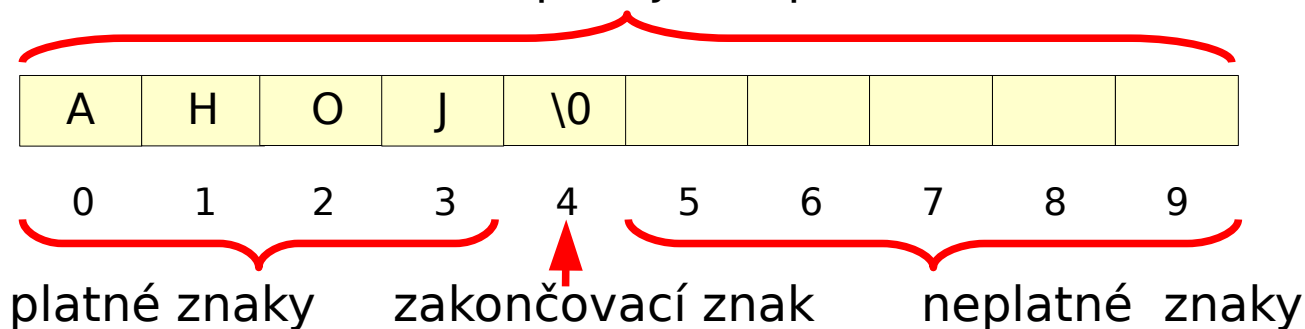
Programování v jazyce C pro chemiky (C2160)

4. Textové řetězce, zápis dat do souboru

Textové řetězce

- V jazyce C neexistuje typ proměnné, který by byl určen výhradně pro ukládání textu
- Pro práci s texty používáme pole znaků, tj. pole typu **char**, v praxi mluvíme o tzv. **řetězcích** (jsou to řetězce znaků)
- Posledním znakem řetězce je vždy zakončovací znak **'\0'**, který indikuje konec řetězce
- Při definici pole musíme jeho velikost zvolit tak, aby zbylo místo na zakončovací znak **'\0'**
- Příklad: `char str[5] = {'A', 'H', 'O', 'J', '\0'};`
- Počet platných znaků řetězce může být menší než je velikost pole; za platné znaky jsou považovány pouze ty, které předchází zakončovacímu znaku **'\0'**
- Příklad: `char str[10] = {'A', 'H', 'O', 'J', '\0'};`

velikost pole je 10 prvků



Inicializace řetězce

- Při definici můžeme řetězec inicializovat jednoduše uvedením textu v uvozovkách
- Velikost pole v tomto případě nemusíme uvádět, překladač automaticky nastaví velikost pole a naplní pole specifikovaným textem včetně zakončovacího '\0'
- Pokud nechceme řetězec inicializovat žádným textem, inicializujeme ho jako vždy alespoň jako prázdný řetězec (tj. uvedeme prázdné uvozovky)

```
char str[] = "AHOJ"; // Při inicializaci tohoto retezce
                    // prekladac automaticky alokuje pole
                    // potrebne velikosti a na jeho
                    // konec vlozi znak '\0'
char text[10] = ""; // Inicializace prazdnym retezcem
```

- Tento postup lze použít **pouze při definici** řetězce, **nelze ho použít pro kopírování znaků** do jiného řetězce. Pro zkopírování obsahu jednoho řetězce do jiného musíme kopírovat postupně jednotlivé znaky

```
char str[] = "AHOJ";
// Nasledujici prirazeni nebude fungovat !!!
str = "NAZDAR";
```

Unicode, diakritika, speciální znaky

- Práce se znaky mimo znakovou sadu US-ASCII je komplikovanější
- Dnes běžné kódování znaků UTF-8: každý **ne-ASCII znak** zabere **2-4 bajty**. Jeden **char** je vždy právě **jeden bajt**
- Ne-ASCII znak tedy nelze celý uložit do skalární proměnné typu **char**
- Iterace přes řetězce funguje vždy normálně, jen se zvlášť zpracuje každý bajt ne-ASCII znaku
- Pozor při ručním počítání znaků k určení délky řetězce! Raději to při inicializaci nechte na kompilátoru

```
char c = 'č';           // Nefunguje, znak potřebuje 2 B

char str[6] = "kočka"; // Chyba, 5 písmen zabere všech 6 B,
                       // nevejde se ukončovací znak '\0'
char str2[] = "kočka"; // Odpovídá str2[7]

// Nyní je v str2[2] první polovina č a ve str2[3] ta druhá.
```

Výpis textového řetězce pomocí printf()

- Pro výpis řetězce ve funkci `printf()` používáme formátovací prvek **%s**
- Funkce `printf()` vypisuje všechny znaky, dokud nenarazí na zakončovací znak `'\0'`

```
char str[] = "AHOJ";  
  
printf("Text retezce je: %s\n", str);
```

Zkrácení řetězce

- Obsah řetězce můžeme kdykoliv změnit, tj. zapsat do něj jiný řetězec, počet nově zapsaných znaků však nesmí překročit velikost pole (včetně znaku '\0')
- Počet platných znaků řetězce může být menší než je velikost pole; funkce které s řetězcem pracují (např. printf()) poznají počet platných znaků podle umístění zakončovacího znaku '\0'
- Změnou polohy znaku '\0' lze řetězec zkrátit

A	H	O	J	\0					
0	1	2	3	4	5	6	7	8	9

```
char s[20] = "AHOJ";
```

```
printf("Text retezce je: %s\n", s); // Vypise se AH0J
```

```
// Retezec zkratime tak ze znak '\0' umistime o pozici drive
```

```
s[3] = '\0';
```

```
printf("Text retezce je: %s\n", s); // Vypise se AHO
```

```
s[2] = '\0';
```

```
printf("Text retezce je: %s\n", s); // Vypise se AH
```

Výpis řetězce od jiného než prvního znaku

- Chceme-li ve funkcích (např. `printf()`) pracovat s řetězcem tak aby se začínalo jiným než prvním znakem, přičteme ke jménu předávané řetězcové proměnné číslo odpovídající počtu znaků o které se má začátek posunout

```
char s[20] = "AHOJ";

printf("Text retezce je: %s\n", s);      // Vypise se AHOJ
printf("Text retezce je: %s\n", s+1);   // Vypise se HOJ
printf("Text retezce je: %s\n", s+2);   // Vypise se OJ
printf("Text retezce je: %s\n", s+3);   // Vypise se J
printf("Text retezce je: %s\n", s+4);   // Nevypise se nic
```

Načtení textového řetězce ze vstupu

- Pro načtení řetězce ze vstupu (zpravidla z klávesnice) používáme funkci **scanf()**, formátovací prvek je **%s**
- Jméno řetězcové proměnné předávané funkci **scanf()** uvádíme bez hranatých závorek a **bez &**
- Abychom zajistili, že při načítání nebude překročena velikost pole, uvedeme maximální počet načítaných znaků mezi **%** a **s**. Musí zůstat jedno volné místo na zakončovací znak **'\0'**, proto uvedeme velikost o jedničku menší než je velikost pole
- Funkce **scanf()** načítá text ze vstupu **dokud nenarazí na mezeru**, pak načítání ukončí => načítají se slova oddělená mezerou
- Funkce **scanf()** automaticky přidává znak **'\0'** na konec řetězce

```
char string[20] = ""; // Inicializujeme prazdnym retezcem

scanf("%19s", string); // Nacte max. 19 znaku do string + '\0'
printf("Text retezce s je: %s\n", string);
```

```
char s1[20] = "", s2[20] = "";

scanf("%19s %19s", s1, s2); // Nacte dve slova (oddeleno mezerou)
printf("Prvni slovo: %s, druhe slovo: %s\n", s1, s2);
```


Zápis dat do souboru

Proměnná typu FILE slouží k identifikaci souboru.

Soubor otevřeme voláním funkce fopen()

```
FILE *f = NULL;
```

Název a cesta k souboru

```
f = fopen("/home/martinp/testdata/data1.txt", "w");
```

```
if (f == NULL)
```

```
{
```

```
    printf("Cannot open file!\n");
```

```
    return 1;
```

```
}
```

```
fprintf(f, "Tento text se zapise do souboru\n");
```

```
fclose(f);
```

```
f = NULL;
```

Režim otevření souboru "w" vytvoří nový soubor a otevře ho pro zápis.

Pokud bylo otevření souboru úspěšné, bude v *f* hodnota různá od NULL.

Pro zápis do souboru používáme funkci **fprintf()**. Můžeme použít formátovací prvky podobně jako u funkce printf().

Prvním parametrem funkce fprintf() je proměnná typu FILE identifikující daný soubor.

Po ukončení používání souboru jej uzavřeme funkcí **fclose()**.

Zápis dat do souboru

- Proměnná identifikující soubor je typu **FILE ***
- Funkce **fopen(cesta a jméno souboru, režim přístupu)** otevře soubor a vrátí identifikátor, který je v případě úspěšného otevření různý od NULL

Možné režimy přístupu:

"r" otevře existující soubor pro čtení

"w" vytvoří nový soubor pro zápis (popř. přepíše existující)

"r+", "w+", "a", "a+" - viz "man fopen"

- Funkce **fprintf(identifikátor souboru, formátovací řetězec, proměnné...)** se používá podobně jako printf(), prvním argumentem je však identifikátor souboru (typu FILE *)
- Funkce **fclose(identifikátor souboru)** slouží k uzavření souboru

```
FILE *f = NULL;  
int i = 10;  
float a = 2.3;  
char c = 'R';
```

```
// Funkci fprintf() lze používat stejně jako printf()  
// avšak jako první argument musíme uvést identifikátor souboru
```

```
fprintf(f, "Promenne: %i, %f, %c\n", i, a, c);
```

Formátovaný výstup

- Funkce `printf()` a `fprintf()` používají pro výpis hodnot proměnných formátovací prvky (`%i`, `%f`, `%c`, `%s` atd.)
- Formátovací prvky specifikují typ vypisované proměnné (**int**, **float**, **char**), kromě toho umožňují specifikovat formát výpisu např. počet desetinných míst, zarovnávání doleva nebo doprava a pod.
- Obecný zápis formátovacích prvků:
`%[příznaky][šířka][.přesnost][modifikátor]konverze`
- Význam volitelných parametrů se může lišit pro různé typy vypisovaných hodnot

- Příklad: `%+ #12.5Lf`

Annotation: `+` and `#` are **příznaky**. `12` is **šířka**. `.5` is **přesnost**. `L` is **modifikátor**. `f` is **konverze (typ hodnoty)**.

Výstup celých čísel - konverze %i

- Pro výstup celých čísel (typ **int**) slouží konverze **%i (%d)**
- **Šířka** nastavuje **minimální počet vypisovaných znaků**; je-li vypisované číslo kratší než šířka, doplní ze **zleva mezery**
- **Přesnost** nastavuje **minimální počet vypisovaných znaků** (funguje tedy podobně jako šířka), je-li je číslo kratší jsou **zleva doplněny nuly**
- **Příznaky:**
 - výsledek se **zarovnává doleva** (namísto implicitního doprava)
 - + číslo bude vždy vytištěno **se znaménkem + nebo -** (standardně se znaménko vypisuje jen u záporných čísel)**mezera** kladná čísla jsou vypisována **s mezerou na začátku** (záporná čísla budou mít místo mezery znaménko-)
- Příklad výpisu čísla 47 (tečka `·` představuje mezeru):

<code>%i</code>	<code>47</code>	standardní výpis
<code>%5i</code>	<code>···47</code>	min. 5 znaků, zleva mezery
<code>%.5i</code>	<code>00047</code>	min. 5 znaků, zleva nuly
<code>%-5i</code>	<code>47···</code>	min. 5 znaků, zarovnání doleva (mezery zprava)
<code>%+5i</code>	<code>··+47</code>	min. 5 znaků, bude vytištěno znaménko + nebo -
<code>% i</code>	<code>·47</code>	kladná čísla budou mít na začátku mezeru
<code>%- 5i</code>	<code>·47··</code>	min. 5 znaků, zarovnání doleva, mezera před kladným číslem

Výstup desetinných čísel - konverze %f

- Pro výstup desetinných čísel (typ **float**) slouží konverze **%f**
- **Šířka** nastavuje **minimální počet vypisovaných znaků** (vč. desetinné tečky a znaménka); bude-li vypisované číslo kratší než šířka, doplní ze **zleva mezery**
- **Přesnost** specifikuje počet cifer za desetinnou tečkou (v případě potřeby je číslo zaokrouhleno podle standardních pravidel)
- **Příznaky** jsou stejné jako u konverze %i
- Příklad výpisu čísla 45.375 (tečka `.` představuje mezeru):

<code>%f</code>	<code>45.375000</code>	standardní výpis (implicitní přesnost je 6)
<code>%11f</code>	<code>..45.375000</code>	min. 11 znaků, zleva mezery
<code>%.2f</code>	<code>45.38</code>	přesnost je 2 znaky, číslo je zaokrouhleno
<code>%.5f</code>	<code>45.37500</code>	přesnost je 5 znaků (tj. 5 deset. míst)
<code>%10.5f</code>	<code>..45.37500</code>	min. 10 znaků, přesnost je 5
<code>%-10.5f</code>	<code>45.37500..</code>	jako předchozí, ale zarovnání doleva
<code>%+10.5f</code>	<code>..+45.37500</code>	bude vytištěno znaménko + nebo -
<code>%-+10.5f</code>	<code>+45.37500.</code>	jako předchozí, ale zarovnání doleva
<code>% f</code>	<code>..45.375000</code>	kladná čísla budou mít na začátku mezeru

Výstup znaku - konverze %c

- Pro výstup znaku (typu **char**) slouží konverze **%c**
- **Šířka** nastavuje **minimální počet vypisovaných znaků**; podle potřeby se doplní **zleva mezery**
- **Přesnost** nemá žádný efekt u této konverze
- **Příznaky:**
 - výsledek se **zarovnává doleva** (namísto implicitního doprava)
- Chceme-li vypsat znak % použijeme formátování %% nebo \%
- Příklad výpisu znaku 'A' (tečka · představuje mezeru):

%c	A	standardní výpis znaku
%6c	· · · · · A	min. 6 znaků, zleva mezery
%-6c	A · · · · ·	jako předchozí ale zarovnání doleva

Výstup řetězce - konverze %s

- Konverze **%s** slouží pro výstup řetězce (tj. řetězcové proměnné typu **char []** obsahující zakončovací znak **\0**)
- **Šířka** nastavuje **minimální počet vypisovaných znaků**; bude-li řetězec kratší než šířka, doplní ze **zleva mezery**
- **Přesnost** nastavuje **maximální počet vypisovaných znaků řetězce** (což nemusí odpovídat celkovému počtu znaků, protože ten může být nastaven hodnotou šířky a v takovém případě se podle potřeby doplňují mezery - viz. příklad níže)
- **Příznaky**:
 - výpis řetězce **zarovnává doleva** (namísto implicitního doprava)
- Příklad výpisu řetězce "AH0J" (tečka **·** představuje mezeru):

%s	AH0J	standardní výpis řetězce
%6s	· · AH0J	min. 6 znaků, zarovnání doprava(zleva mezery)
%-6s	AH0J · ·	jako předchozí, ale zarovnání doleva
%.2s	AH	vypíše max. 2 znaky řetězce
%6.2s	· · · · AH	vypíše max. 2 znaky řetězce, ale dohromady minimálně 6 znaků (zleva mezery)
%-6.2s	AH · · · ·	jako předchozí, ale zarovnání doleva

Vykonání systémového příkazu

- Funkce `system(příkaz)` slouží k vyvolání příkazu operačního systému, který je interpretován shellem systému
- Funkce slouží převážně k jednoduchému spouštění externích programů
- Příkaz se zapisuje stejně, jako bychom ho uvedli na příkazovém řádku terminálu
- Běh programu je pozastaven do té doby, než je příkaz vykonán
- Při použití funkce `system()` je třeba na začátek programu vložit `#include <stdlib.h>`

```
// Nasledující program vypíše obsah aktuálního adresáře po sloupcích  
  
int main()  
{  
    system("ls -C /bin");  
  
    return 0;  
}
```


Dodržujte následující pravidla

- Všechny řetězce inicializujte vhodným textem nebo prázdným řetězcem.
- Pro jednoduchost ve všech řetězcích uvažujte jen US-ASCII znaky.
- Dávejte pozor, aby všechny zpracovávané řetězce byly vždy řádně ukončené nulovým znakem.
- Na začátku každého programu uveďte stručný komentář vysvětlující účel programu.
- Dbejte na správné odsazování textu.

Úlohy - část 1

1. Vytvořte program, který načte od uživatele text (jedno slovo bez mezer). Program potom vypíše na obrazovku počet znaků v načteném řetězci. Potom vypíše část zadaného textu počínaje 6. znakem. Řetězec potom zkrátíte na 5 znaků (v původním umístění, bez kopírování do další proměnné) a výsledek vypíšete na obrazovku. (Příklad: "kockopes", vypíše se počet znaků 8, potom "pes" a nakonec "kocko"). **1 bod**
2. Vytvořte program, který načte od uživatele jedno slovo, to pozpátku zkopíruje do další proměnné výsledek vypíše na obrazovku. (Příklad: uživatel zadá "kockopes" a vypíše se "sepokcok"). **1 bod**
3. Vytvořte program, který od uživatele načte dvě slova, každé do jiné řetězcové proměnné. Dále v programu definujte třetí řetězcovou proměnnou do které zkopírujete první a pak druhé načtené slovo oddělené mezerou (tj. bude obsahovat spojené první dva řetězce). Řetězec vypíšete na obrazovku. Dále obraťte pořadí znaků tohoto složeného řetězce stejně jako v úloze 2 a výsledek vypíšete na obrazovku. (Příklad: uživatel zadá "dobry" "den" a vypíše se "dobry den" a potom "ned yrbd"). **nepovinná, 1 bod**

Úlohy - část 2

4. Vytvořte program který vytvoří soubor a zapíše do něj seznam čísel -5 až 10 a odpovídající hodnoty (viz. níže). Na každém řádku bude číslo a za ním postupně příslušné hodnoty. Formátování bude odpovídat obrázku níže, na prvních 3 řádcích souboru bude níže uvedený komentář. Program doplňte o zavolání editoru *kate* ve kterém se automaticky otevře vytvořený soubor. **1 bod**
5. Vytvořte program který do souboru zapíše seznam čísel 1 až 20 a jejich druhé mocniny tak, že na každém řádku bude číslo a za ním jeho druhá mocnina (oddělené mezerou). Program dále vytvoří soubor s příkazy pro *gnuplot*, tak aby obsahoval příkazy pro zobrazení souboru s mocninami. Zavolejte z programu program *gnuplot* tak, aby došlo k zobrazení grafu mocnin. (Pozn.: pro vypsání uvozovek ve funkci `fprintf()` použijte `\`). **1 bod**

Ukázka souboru s příkazy pro *gnuplot* (data budou v souboru *mocniny.dat*):

```
plot "mocniny.dat" using 1:2  
pause -1 "Hit return"
```

Bude-li se soubor s příkazy jmenovat např. *gnuplot.cmd* spustíme *gnuplot* příkazem:
gnuplot gnuplot.cmd

Úloha 4 - ukázka

01234567890123456789012345678901234567890123456789

Seznam cisel a souvisejicich matematickych hodnot

i	i*i	i*i*i	exp(i)	sin(i)	cos(i)
-5	25	-125	0.007	0.958924	+0.2837
-4	16	-64	0.018	0.756802	-0.6536
-3	9	-27	0.050	-0.141120	-0.9900
-2	4	-8	0.135	-0.909297	-0.4161
-1	1	-1	0.368	-0.841471	+0.5403
0	0	+0	1.000	0.000000	+1.0000
1	1	+1	2.718	0.841471	+0.5403
2	4	+8	7.389	0.909297	-0.4161
3	9	+27	20.086	0.141120	-0.9900
4	16	+64	54.598	-0.756802	-0.6536
5	25	+125	148.413	-0.958924	+0.2837
6	36	+216	403.429	-0.279415	+0.9602
7	49	+343	1096.633	0.656987	+0.7539
8	64	+512	2980.958	0.989358	-0.1455
9	81	+729	8103.084	0.412118	-0.9111
10	100	+1000	22026.466	-0.544021	-0.8391

Úlohy - část 3

6. Program z úlohy č. 5 upravte tak, že do souboru zapíše seznam prvních 20 prvočísel tak, že na každém řádku bude pořadí prvočísla a za ním prvočíselo. Nakonec zobrazí příslušný graf v *gnuplot*, podobně jako v úloze 5. **nepovinná, 1 bod**