



# Textové soubory, standardní vstup/výstup

---

Jitka Kreslíková, Aleš Smrčka  
2023

Fakulta informačních technologií  
Vysoké učení technické v Brně

IZP – Základy programování



# Práce se soubory

---

- ☐ Standardní vstup a výstup
- ☐ Vstup a výstup znaků
- ☐ Formátovaný výstup
- ☐ Formátovaný vstup
- ☐ Vstup ze souboru, výstup do souboru
- ☐ Testování chyb



# Standardní vstup a výstup

---

## □ Tři standardní soubory

- stdin – typicky vstup z klávesnice
- stdout – typicky zápis na obrazovku
- stderr – typicky zápis chyb na obrazovku
- V hlavičkovém souboru `stdio.h` [HePa13, str. 39]

## □ Otevírání, zavírání

- Automaticky při startu a ukončení programu



# Vstup a výstup znaků

---

## ❑ **int getchar(void);**

- Vstup po znacích ze stdin
- Vrací znak v typu int (!)
- Vrací EOF, pokud dojde na konec souboru

## ❑ **int putchar(int c);**

- Výstup po znacích na stdout
- Zapisuje znak uložený v int (symetrické ke getchar)
- Vrací EOF, pokud při zápisu dojde k chybě



# Vstup a výstup znaků

## ❑ **int getchar(void);**

- vrací int, kvůli detekci konce souboru

*Příklad:* Kopie stdin do stdout. Konec: Ctrl+Z či Ctrl+D.

```
#include <stdio.h>
int main(void)
{ int c;    // zde MUSÍ být datový typ int
  // Testování konce souboru - vzor
  while((c=getchar())!=EOF) // závorky nutné
    putchar(c);

  return EXIT_SUCCESS;
}
```



# Formátovaný výstup - funkce *printf()*

---

❑ **int printf(const char \*format, ...);**

■ Formátovaný výstup do stdout

❑ Formátovací řetězec

■ Libovolný text + formátovací značky

❑ Proměnný počet argumentů

■ Musí odpovídat formátovacím značkám

○ počet

○ pořadí

○ typy



# Formátovaný výstup - funkce *printf()*

*Příklad:* Formátovaný výpis do stdout.

```
printf("2. mocnina %d. prvku pole A"  
      " je rovna cislu %f\n",  
      i,  
      A[i]*A[i]);
```

A diagram with red arrows and circles illustrates the argument passing. One arrow points from the variable 'i' in the code to the '%d' format specifier in the string. Another arrow points from the expression 'A[i]\*A[i]' to the '%f' format specifier. The variable 'i' and the expression 'A[i]\*A[i]' are each enclosed in a red circle.



# Formátovaný výstup - funkce *printf()*

*Příklad:* formátovací řetězec je zadán pomocí identifikátoru pole typu *char*

```
char a[][8]={{"Ahoj,"}, {" rad"}, {" te"},  
             {" zase"}, {"vidim!\n"}};    // ??  
char b[15]={"Ja tebe taky!\n"};  
for(int i=0; i<5; i++)  
    printf(a[i]); //!! pozor, mohl by obsahovat (%)  
printf(b);        //!! pozor nebezpečné
```

Výstup:

```
Ahoj, rad te zase vidim!    ??  
Ja tebe taky!
```





# Formátovací značky

---

## □ Formátovací značka

*%[příznak][šířka][.přesnost][modifikátor]konverze*

- Mezi znakem % a konverzí mohou být nepovinné parametry
- Konverze je povinný parametr (d, f, s, c, ..)
- Modifikátor
  - hh – short short (char), h – short, l – long, ll – long long, L – long double
- Příznak
  - + zarovnání doprava, – zarovnání doleva



# Formátovací značky

*Příklad:* Ukázky různých konverzí

Hodnota:	Konverze:	Výstup:	Poznámka:
3.141592	%7.3f	□□3.142	mezery zleva
3.141592	%-7.3	3.142□□	mezery zprava
369.24	%+11.3E	+3.692E+002	
369.24	%+9.3E	+3.692E+002	
369.24	%+13.3E	□□+3.692E+002	
3.6	%4.0f	□□□4	
Ahoj !	%2s	Ahoj !	
Ahoj !	%.2s	Ah	



# Formátovací značky

## □ Přesnost

- u celých čísel – minimální počet cifer
- u reálných čísel – zaokrouhlení
- `.*` – přesnost lze zadat argumentem

```
double a = 369.2468;  
printf("zobrazeni cisla: %.*f\n", 5, a);  
// zobrazeni cisla: 369.24680
```



# Formátovací značky

---

- Výpis speciálních hodnot u %f, %F
  - Nekonečno u racionálních typů – INF
    - Výsledek při dělení nulou
  - Neplatné číslo – NaN (Not a Number)
    - Výsledek např. 0.0/0.0
  - Tisknou se řetězce
    - `inf` či `nan` pro %f
    - `INF` či `NAN` pro %F
  - Tyto hodnoty akceptuje i funkce *scanf()*



# Formátovaný vstup - funkce *scanf()*

---

❑ **int scanf(const char \*format, ...);**

- Formátovaný vstup ze stdin
- Během čtení přeskakuje bílé znaky

❑ Formátovací řetězec

- Vzor, očekávaný na vstupu

❑ Proměnný počet argumentů

- Nutno předávat odkazem!

❑ **POZOR!**

- **scanf("%s", buffer)** – hrozí čtení za hranicí alokované paměti!



# Formátovaný vstup - funkce *scanf()*

---

## ❑ Formátovací značka

*%[šířka][modifikátor]konverze*

- Význam podobný jako u *printf()*
- *%\*konverze* – přečti a zahod'

## ❑ Návratová hodnota

- Počet úspěšně přečtených parametrů
- Pokud se vstup neshoduje se vzorem, funkce ukončí čtení těsně před prvním nerozpoznaným znakem → funkce vrátí menší výsledek než se čekalo
- Pokus o čtení za koncem souboru → vrací EOF



# Formátovaný vstup - funkce *scanf()*

*Příklad:* čtení hodnot různých typů

```
char c; int i; float r_1;
double r_2; char text[9];
...
int err = scanf("%c%d%f%lf%8s", &c, &i, &r_1, &r_2,
               text);
if (err != 5 && err != EOF) chyba();

int a,b;
scanf("%d A %d", &a, &b); // vstup 5    A7
printf("zobrazeni cisel:%d  %d\n", a,b);
// zobrazeni cisel:5 7
```



# Vstup po řádcích

---

## ❑ **char \*gets(char \*str);**

- Přečte celý řádek ukončený '\n' a uloží jej do bufferu str
- Čte i bílé znaky, znak '\n' se neukládá
- Automaticky vkládá '\0' na konec

**POZOR! raději NEPOUŽÍVAT! NEBEZPEČNÉ!**

- Neumožňuje omezit počet čtených znaků
- Hrozí zápis mimo alokovaný buffer
- Raději používat *fgets()*, která je bezpečná





# Výstup po řádcích

---

## □ **int puts(char \*str);**

- Vytiskne řetězec str a odřádkuje
- Na rozdíl od printf neinterpretuje řetězec
- Ekvivalentní volání
  - `printf("%s\n", str);`
- Při neúspěchu zápisu vrací EOF



# Vstup ze souboru, výstup do souboru

---

[HePa13, str. 69]

## □ textové soubory

- Knihovní funkce interpretují konce řádků podle zvyklostí OS (CR LF x LF x CR)

## □ binární soubory

- Knihovní funkce neinterpretují konce řádků
- Knihovní funkce pracují přesně s daty, která jsou v souboru uložena
- Jiným principiálním způsobem se textové a binární soubory **neliší**



# Vstup ze souboru, výstup do souboru

---

## ☐ Jazyk C

- Nezná datový typ soubor
- Nemá syntaktické prostředky pro práci se soubory
- Práce se soubory pomocí knihovních funkcí

## ☐ **FILE \***

- Typ ze stdio.h
- FILE – struktura popisující soubor, nelze s ní pracovat přímo
- Práce výhradně přes ukazatel a knihovní funkce



# Vstup ze souboru, výstup do souboru

❑ **FILE \*fopen(const char \*jmeno,  
const char \*mod);**

- Otevře soubor zadaného jména [HePa13, str. 79]  
Pokud se nepovede otevřít – vrací **NULL**
- mod – způsob otevření souboru, řetězec (!)

*"r" textový soubor pro čtení*

*"w" textový soubor pro zápis nebo pro přepsání*

*"a" textový soubor pro připisování na konec*

*"r+" textový soubor pro čtení a zápis*

*"w+" textový soubor pro čtení, zápis nebo přepsání*

*"a+" textový soubor pro čtení a zápis na konec*



# Vstup ze souboru, výstup do souboru

---

## ❑ **int fclose(FILE \*file);**

- Uzavře soubor.

[HePa13, str. 73]

- Pokud se nepovede uzavřít – vrací **EOF**.

## ❑ (Ne)uzavírání souborů

- Std. knihovna musí zapsat všechny vnitřní buffery.

- Předčasné ukončení (např. funkcí abort()) →  
→ poškození souborů.

- Slušný program uzavírá své otevřené soubory a nespolehá na operační systém!



# Vstup ze souboru, výstup do souboru

*Příklad:* použití funkcí `fopen()`, `fclose()`.

```
FILE *soubor;  
...  
soubor=fopen("DATA.TXT", "r");  
if (soubor == NULL)  
    return ERR_FOPEN;  
...  
!fclose(soubor);  
...
```

Je nutné testovat případné chyby těchto operací!



# Vstup ze souboru, výstup do souboru

*Příklad:* testování úspěšnosti otevření a zavření souborů.

```
FILE *soubor;  
char *jmeno = zjistijmeno(); // např. DATA.TXT  
...  
soubor = fopen(jmeno, "r");  
if(soubor == NULL)  
{  
    //! fprintf(stderr, "Chyba pri otevreni %s\n",  
    //!          jmeno);  
    return ERR_FOPEN;  
}  
...  
if(fclose(soubor) == EOF)  
    return ERR_FCLOSE;
```



# Znakový vstup/výstup

---

## ☐ Standardní vstup/výstup

[HePa13, str. 39]

- **int getchar(void);**

- **int putchar(int c);**

## ☐ Souborový vstup/výstup

[HePa13, str. 73]

- **int getc(FILE \*file);**

- **int putc(int c, FILE \*file);**

- Pracují s otevřenými soubory.

- Práce s neotevřeným souborem → chyba.





# Vstup znaků ze souboru

*Příklad:* program určí počet řádků a neprázdných řádků textového souboru.

```
FILE *soubor;
int zn=0;      // počet znaků na řádku (bílé
               // znaky nejsou započítány)
int sp=0;      // počet bílých znaků na řádku
int rd=0;      // počet řádků (řádky pouze s
               // bílými znaky nejsou započítány)
int rd_0=0;    // počet řádků (včetně "prázdných")
if((soubor=fopen("TEXT", "r"))==NULL)
{ // test - otevření
    fprintf(stderr, "Chyba při otevření TEXT\n");
    return EXIT_FAILURE;
}
```



# Vstup znaků ze souboru

*Příklad: pokračování*

```
int c;
while( (c=getc(soubor)) !=EOF)
{ // test - konec souboru
  if(c=='\n')
  { // test - konec řádku
    rd_0++;
    if(zn>0) rd++;
    zn=sp=0;
  }
  if(!isspace(c)) zn++;
  else sp++;
}
```



# Vstup znaků ze souboru

*Příklad: pokračování*

```
if (zn>0)
{ rd_0++; rd++; }
else if (sp>0)
{ rd_0++; }

printf("Počet neprázdných řádků=%d\n"
      "Počet všech řádků=%d\n", rd, rd_0);

if (fclose(soubor)==EOF)
{
    fprintf(stderr, "Chyba při uzavření TEXT\n");
    return EXIT_FAILURE;
}
return EXIT_SUCCESS;
```



# Formátovaný vstup a výstup

---

## □ Standardní vstup/výstup

[HePa13, str. 39]

- **int scanf(const char \*format, ...);**

- **int printf(const char \*format, ...);**

## □ Souborový vstup/výstup

[HePa13, str. 73]

- **int fscanf(FILE \*file, const char \*format, ...);**

- **int fprintf(FILE \*file, const char \*format, ...);**




# Vstup a výstup řádků

---

❑ **char \* fgets(char \*str, int max, FILE \*file);**

- Čte nejvýše *max* znaků do konce řádku.
- Znak '\n' se **ukládá** (!).
- Pokud nenačte celý řádek, nepřejde na nový, ale příště bude pokračovat, kde skončila.
- Při úspěchu vrátí ukazatel na řetězec, jinak **NULL**.

❑ **int fputs(char \*str, FILE \*file);**

- Zapiše řetězec do souboru a neodřádkuje. 
- Při neúspěchu vrátí **EOF**.



# Vstup znaků ze souboru

*Příklad:* program překopíruje zadaný textový soubor do souboru, jehož název (eventuálně i s cestou) určí uživatel.

```
char nazev[20];
printf("Ktery soubor se ma kopirovat?\n");
scanf("%19s", nazev); // ctení názvu souboru,

FILE *file_r;
if((file_r=fopen(nazev, "r"))==NULL)
{
    fprintf(stderr,
        "Chyba pri otevreni souboru pro cteni!\n");
    return EXIT_FAILURE;
}
```



# Vstup znaků ze souboru

*Příklad: pokračování*

```
printf("Do ktereho souboru kopirovat?\n");  
!scanf("%19s", nazev); // ctení názvu souboru  
  
FILE *file_w;  
if((file_w=fopen(nazev, "w"))==NULL)  
{  
    fprintf(stderr,  
        "Chyba pri otevreni souboru pro zapis!\n");  
    return EXIT_FAILURE;  
}
```



# Vstup znaků ze souboru

*Příklad: pokračování*

```
// Kopírování souboru:
int c;
while( (c=getc(file_r)) != EOF)
{ putc(c, file_w); }
int rclose = fclose(file_r);
int wclose = fclose(file_w);
if(rclose==EOF || wclose==EOF)
{
    fprintf(stderr,
        "Některý ze souboru nelze uzavřít!\n");
    return EXIT_FAILURE;
}
```

**Přepište program a použijte funkce *fgets()*, *fputs()*, tj. kopírování po řádcích.**





# Testování chyb

---

- ❑ I/O funkce vracejí NULL nebo EOF pro detekci chyby.
- ❑ **int ferror(FILE \*file);**
  - Vrací nenulovou hodnotu, pokud poslední operace se souborem způsobila chybu.

```
int ch = fgetc(file);  
if(ferror(file))  
{  
    printf("Chyba souboru\n");  
    break;  
}
```



# Testování chyb

---

## ❑ **int errno;**

- Globální proměnná z `<errno.h>`
- Obsahuje kód chyby poslední I/O operace.
- Tuto proměnnou nastavují všechny I/O operace.


## ❑ **char \* strerror (int errnum);**

- Funkce z `<string.h>`
- Vypíše anglicky chybové hlášení.
- Parametrem je kód chyby (errno).



# Testování chyb

*Příklad:* otestuje správnost otevření souboru.

```
// Pokusí se otevřít soubor zadaného jména.  
// Pokud dojde k chybě, vypíše hlášení  
FILE *testOpen(const char *name, const char *mod)  
{  
    FILE *f = fopen(name, mod);  
    int error = errno;  
    if (f == NULL)  
    {  
        fprintf(stderr, "%s", strerror(error));  
         exit(EXIT_FAILURE); // TAKTO NE!  
    }  
    return f;  
}
```



# Textové soubory, standardní vstup/výstup

---





# Kontrolní otázky

---

1. Co dělají funkce `fprintf()` a `fscanf()`?
2. Proč vrací funkce `getc()` hodnotu typu `int`, i když umí číst pouze osmibitové znaky?
3. Proč bychom se měli vyvarovat používání funkce `gets()`?
4. Jaký je hlavní rozdíl mezi textovým a binárním souborem?



# Úkoly k procvičení

---

1. Napište program, který ve standardním vstupu najde všechny číslice ('0'-'9') a vypočte jejich četnost. Formátovanou tabulku četnosti těchto číslic vypíše do textového souboru.
2. Napište program, který čte text ze standardního vstupního proudu (stdin) a na každém řádku nahradí každou posloupnost bílých znaků jedinou mezerou. Takto upravený text zapisujte do standardního výstupního proudu (stdout).
3. Upravte předchozí program tak, aby četl text z textového souboru a upravený text zapisoval do jiného textového souboru.
4. Modifikujte předchozí program tak, aby se každý výskyt tabulátoru nahrazoval osmi mezerami.