6 Textové soubory, standardní vstup/výstup

Kreslíková, 9. 10. 2023

Tato publikace je určena výhradně jako podpůrný text pro potřeby výuky. Bude užita výhradně v přednáškách výlučně k účelům vyučovacím či jiným vzdělávacím účelům. Nesmí být používána komerčně. Bez předchozího písemného svolení autora nesmí být kterákoliv část této publikace kopírována nebo rozmnožována jakoukoliv formou (tisk, fotokopie, mikrofilm, snímání skenerem či jiný postup), vložena do informačního nebo jiného počítačového systému nebo přenášena v jiné formě nebo jinými prostředky.

Veškerá práva vyhrazena © Jitka Kreslíková a kol., Brno 2023

6.1 Úvod

Tato kapitola se zabývá standardním vstupem a výstupem programu a soubory. V jazyce C nejsou soubory součástí jazyka, ale standardní knihovny, proto se podíváme na funkce, pomocí nichž se soubory pracujeme.

6.2 Práce se soubory

Jazyk C nedefinuje žádné klíčové slovo pro práci se soubory. Vstupy a výstupy jsou řešeny pomocí standardní knihovny jazyka C *stdio.h* [HePa13, str. 39], která obsahuje rozsáhlou sadu vstup/výstup (I/O) funkcí.

6.2.1 Standardní vstup a výstup

Jakmile je spuštěn program v jazyce *C*, automaticky se otevírají tři soubory: standardní vstup, standardní výstup a standardní výstup pro chybová hlášení. Jde o virtuální soubory, protože existují pouze v operační paměti a nikoliv fyzicky na disku. Pro tyto soubory definuje standardní knihovna jazyka C identifikátory *stdin, stdout, stderr*. Obvykle je jako standardní vstup použita klávesnice a jako standardní výstup (i pro chybová hlášení) obrazovka. V tomto odstavci se budeme zabývat funkcemi, které umožňují komunikaci programu se standardním vstupem a výstupem.

Standardní vstup a výstup se používá zejména u programů, které zpracovávají data proudovým způsobem – to znamená, že čtou znaky ze standardního vstupu a ihned po zpracování zapisují výsledek na standardní výstup. Tomuto typu programů se někdy říká filtry, protože „filtrují“ vstupní proud dat a výsledek poskytují dále. Funkčnost těchto programů lze spojovat pomocí tzv. „rour“ (angl. pipe), což jsou speciální programy operačního systému. Funguje to tak, že pokud jsou dva programy spojeny rourou, pak to, co první program považuje za standardní výstup, to se druhému programu jeví jako standardní vstup.

Příklad: Ukázka propojení dvou programů pomocí roury (ve Windows i v Linuxu se to provádí pomocí znaku |):

```
$ program1 | program2
```

Programy, které pracují pouze se standardním vstupem a výstupem mohou zpracovávat i skutečné textové soubory, pokud použijeme přesměrování souborů. Přesměrovávání souborů je opět operace, kterou provádí operační systém (nesouvisí tedy přímo s jazykem C, ten pouze umožňuje tuto vlastnost využívat).

Příklad: Ukázka příkazového řádku s přesměrování souborů do a z programu (přesměrování se provádí ve Windows i v Linuxu pomocí znaků < a >):

```
$ program < vstupni.txt > vystupni.txt
```

6.2.2 Vstup a výstup znaků

Funkce *getchar()* je určena pro vstup znaků ze standardního vstupu, funkce *putchar()* pro výstup znaků na standardní výstup. Funkce mají následující funkční prototypy:

```
int getchar(void);
int putchar(int c);
```

Funkce *getchar()* nemá žádný parametr, návratovou hodnotou této funkce je kód znaku, přečtený ze standardního vstupu *stdin*. (Jestliže bylo dosaženo konce souboru, vrací funkce hodnotu *EOF*.)

Funkce *putcha()r* má jako argument hodnotu typu **int**, zadávající kód znaku, který má být zapsán na standardní výstup *stdout*. Návratová hodnota funkce je kód zapsaného znaku, nebo *EOF* pokud při výstupu došlo k chybě.

Funkce *getchar()* pracuje s datovým typem **int** namísto **char**, aby bylo možné detekovat konec souboru (konstanta *EOF*). Pokud by pracovala s datovým typem **char**, nebylo by možné odlišit hodnotu této konstanty od některého ze znaků (*EOF* není znak). Funkce *putchar()* pracuje s tímto datovým typem z důvodu symetričnosti.

Příklad: Program kopíruje znaky ze standardního vstupu na standardní výstup. Ukončovací „znak“ souboru *EOF* lze zadat kombinací 2 kláves: Ctrl+Z (ve Windows) nebo Ctrl+D (v Linuxu).

```
#include <stdio.h>
int main()
{ int c;  // zde MUSI byt datovy typ int
  // Testovani konce souboru
  while((c=getchar())!=EOF) // c=getchar() musi byt v zavorce
    putchar(c);
  return EXIT_SUCCESS;
}
```

6.2.3 Formátovaný výstup – funkce *printf()*

S funkcemi *printf()* a *scanf()* jsme se již setkali. Vyložíme nyní systematicky základní možnosti použití těchto funkcí.

Funkce *printf()* se používá pro formátovaný výstup dat na standardní výstup *stdout*. Prototyp *printf()* má následující tvar:

```
int printf(const char *retezec, ...);
```

Funkce má proměnný počet parametrů, volání funkce ale musí obsahovat alespoň první argument *retezec,* tj. formátový řetězec, který obsahuje jednak libovolné znaky, jež se beze změny kopírují na výstup, jednak formátové specifikace (konverze) určující, v jakém tvaru se následující argumenty zobrazí. Vzájemné přiřazení argumentů a konverzí je provedeno podle pořadí zleva doprava. V případě, že ve formátovém řetězci specifikujeme více konverzí, než kolik jsme zadali argumentů, vznikne chyba a nelze předpovědět co se zobrazí.

Návratová hodnota funkce *printf()* udává počet znaků, které byly funkcí zapsány do *stdout*. Pokud nastane chyba, vrací záporné číslo.

Příklad:

```
printf("\n2. mocnina %d. prvku pole A je rovna cislu %f",
i,A[i]*A[i]);
```

Znak *'\n'* je zde interpretován jako přechod na novou řádku. Kombinace *%d, %f* jsou konverze určené pro výstup hodnot po řadě typu **int** a **double**, přitom hodnota *i* se tiskne podle konverze *%d* a hodnota *A[i]*A[i])* podle konverze *%f*.

Kdyby byl uvedený příkaz *printf()* umístěn v cyklu:

```
for(int i=0;i<3;i++){...}
```

vypadal by výstup pro $A_i = i$, $i = 0,1,2$ takto:

```
2. mocnina 0. prvku pole A je rovna cislu 0.000000
2. mocnina 1. prvku pole A je rovna cislu 1.000000
2. mocnina 2. prvku pole A je rovna cislu 4.000000
```

První argument funkce *printf()* je, jak vyplývá z jejího funkčního prototypu, ukazatel na konstantní řetězec. V následujícím příkladu je tento ukazatel zadán pomocí identifikátoru pole typu **char**.

Příklad: Formátovací řetězec je zadán pomocí identifikátoru pole typu **char**.

```
#include <stdio.h>
int main(void)
{
  char a[][8]={{"\nAhoj,"},{" rad"},{" te"},{" zase"},{"
vidim!"}};
  char b[15]={"\nJa tebe taky!"};

  for(int i=0;i<5;i++)
    printf(a[i]); // !! pozor nebezpečné mohl by obsahovat (%)
  printf(b);      // !! pozor nebezpečné
  return;
}
```

Výstup tohoto programu by vypadal takto:

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

Výpis tímto způsobem může být ovšem nebezpečný, protože pokud by se v textovém řetězci vyskytl znak *'%'*, funkce jej bude interpretovat jako formátovací značku, a protože nenásleduje žádný další parametr, bude výsledkem nedefinovaný řetězec a pravděpodobně i pokus o nelegální přístup do paměti. Textové řetězce se vždy vypisují pomocí formátovací značky *%s*.

```
printf("%s", a[i]);
```

Formátovací značky

Formátová specifikace má obecně následující tvar:

%[příznaky][šířka][.přesnost][modifikátor]konverze

Závorky *[]* zde označují nepovinné parametry.

Jednotlivé položky specifikace vysvětlíme v pořadí jejich důležitosti.

Konverze

Konverze je povinný parametr, je označena jedním znakem, mezi znakem % a označením konverze mohou být umístěné další (nepovinné) parametry. Následující tabulka ukazuje, jaké konverze se používají pro výstup hodnot jednotlivých datových typů, eventuálně v jakém tvaru se zobrazí.

```
konverze:   typ položky seznamu odpovídající konverzi:
%c          znak; (je-li hodnota typu int, je převedena na typ
            unsigned char)
%d %i       číslo typu signed int, desítkový (dekadický) zápis
%u          číslo typu unsigned int, desítkový zápis
%o          číslo typu unsigned int, osmičkový (oktalový) zápis
%x %X       číslo typu unsigned int, šestnáctkový (hexadecimální)
            zápis, číslice označené a,b,c,d,e,f nebo A,B,C,D,E,F
%f %F       číslo typu float, double, desetinný tvar
%e %E       číslo typu float, double, semilogaritmický tvar,
            exponent označen podle konverze e nebo E
%g %G       číslo typu float, double, tvar zvolen podle výhodnosti
            zápisu jako desetinný nebo semilogaritmický, exponent
            je podle konverze (v semilogaritmickém tvaru) e nebo E
%s          řetězec (bez ukončovacího znaku '\0')
%p          ukazatel; tiskne se jeho obsah nejčastěji v šestnáctkovém
            zápisu
%n          ukazatel na typ int. Na adresu na kterou ukazuje se za-
            píše počet znaků který byl až dosud tímto voláním zapsán
            na výstup, netiskne se nic,
```

Modifikátor

```
h           modifikuje konverze d,i na typ signed short int
                        konverze u,o,x,X na typ unsigned short int
l           modifikuje konverze d,i na typ signed long int
                        konverze u,o,x,X na typ unsigned long int
ll          modifikuje konverze d, i na typ signed long long int
                        konverze u,o,x,X na typ unsigned long long int
L           modifikuje konverze f,e,E,g,G na typ long double
```

Šířka

```
n           tiskne se alespoň n znaků, mezery se doplňují zprava
            nebo zleva, viz příznaky
0n          tiskne se alespoň n znaků, namísto mezer se doplňují nuly
*           šířka je zadána nepřímo: argument, který "je na řadě"
            obsahuje šířku, (musí být typu int), následuje argument,
            který bude vystupovat
```

Přesnost

Přesnost je dekadické číslo, které pro konverze *d, i, o, u, x, X* znamená minimální počet cifer na výstupu, pro konverze *f, e, E,* znamená počet cifer za desetinnou tečkou, pro konverze *g, G* znamená počet významových cifer a pro konverzi *s* maximální počet znaků. Kromě toho má .* a . následující význam:

```
.*          přesnost je zadána nepřímo: argument, který "je na řadě",
            obsahuje přesnost,(musí být typu int), následuje argument,
            který bude vystupovat
.           znamená totéž co .0
```

Příznak

```
-           výsledek se zarovná doleva, zprava se doplní mezery
            není-li uveden, výsledek se zarovná doprava a zleva
            se doplní mezery nebo nuly
+           číslo typu signed se vytiskne vždy se znaménkem
            není-li uveden vynechá se znaménko '+' u kladných
            hodnot
mezera      kladné číslo se vytiskne bez znaménka, "+" bude nahra-
            zeno mezerou
```

Příklady:

```
 Hodnota:     Konverze:      Výstup:          Poznámka:
 3.141592     %7.3             3.142          mezery se doplní
zleva
 3.141592     %-7.3          3.142            mezery se doplní
zprava
 369.24       %+11.3E        +3.692E+002
 Ahoj!        %2s            Ahoj!
 Ahoj!        %.2s           Ah
```

Příklad: Nepřímo zadaná přesnost.

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

Výpis speciálních hodnot u %f

Racionální datové typy mohou obsahovat hodnoty nekonečno *INF* a *NaN* – neplatné číslo (Not a Number). Pokud dojde k požadavku na tisk takové hodnoty, vypíše se textový řetězec *"inf"* nebo *"nan"* (nebo *"INF"* či *"NAN"* pro *%F*). Funkce *scanf()* tyto řetězce na vstupu také akceptuje a interpretuje je jako číselnou hodnotu.

6.2.4 Formátovaný vstup - funkce *scanf()*

Funkce *scanf()* se používá pro formátovaný vstup dat ze standardního vstupu *stdin*. Funkční prototyp má následující tvar:

```
int scanf(const char *retezec, ...);
```

Parametry za parametrem *retezec* jsou adresy (ukazatele!) proměnných, jejichž hodnoty se mají přečíst ze standardního vstupu.

Funkce má proměnný počet parametrů, volání funkce ale musí obsahovat alespoň první argument *retezec*, tj. formátový řetězec. Tento řetězec obsahuje formátové specifikace (konverze) určující, jak se budou jednotlivé čtené posloupnosti slabik interpretovat. Dále může formátový řetězec obsahovat bílé znaky a ostatní znaky (ASCII znaky různé od % a bílých znaků). Pokud funkce *scanf()* najde ve formátovém řetězci bílý znak, přečte všechny následující bílé znaky ze vstupu až po první jiný znak. Tyto bílé znaky nejsou přitom transformovány v žádnou vstupní hodnotu. Pokud najde funkce *scanf()* ostatní (nebílý) znak ve formátovém řetězci očekává, že následující znak z stdin bude s tímto znakem totožný. Tento znak bude přečten a ignorován.

Formátové specifikace mají následující tvar:

%[šířka][modifikátor]konverze

Význam parametrů *modifikátor, konverze* je stejný jako ve formátových specifikacích funkce *printf()*. Parametr *šířka* určuje počet znaků tzv. vstupního pole. Bude přečteno maximálně tolik znaků, kolik zadává parametr *šířka*. Pokud narazí funkce *scanf()* na bílý znak nebo na znak, který nepatří do zápisu čtené hodnoty, ukončí se čtení dříve. Na rozdíl od funkce *printf()*, kde lze konverze *f, e, E, g, G* použít pro výstup hodnot typu **float** i typu **double**, lze tyto konverze ve funkci *scanf()* obvykle použít pouze pro vstup hodnot typu **float**, zatímco pro hodnoty typu double je nutné použít tyto konverze s modifikátorem *l*, tedy *lf, le, lE, lg, lG*. Kromě toho mají konverze *f, e, E, g, G* na vstupu stejný význam, všechny je lze použít k přečtení čísla zapsaného v desetinném i semilogaritmickém tvaru.
Návratová hodnota funkce *scanf()* je počet přečtených vstupních polí.

Příklad:
```
char c; int i; float r_1; double r_2; char text[9];
                  ...
scanf("%c%d%f%lf%8s",&c,&i,&r_1,&r_2,text);


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

V uvedeném příkladu je čtení řetězce omezeno na jeho prvních 8 znaků, (devátý znak je vyhrazen pro ukončovací znak *'\0'*). Návratová hodnota funkce by v našem příkladu byla rovna *5*.

6.2.5 Vstup po řádcích

Jak jsme uvedli v předchozí části, pokud narazí funkce *scanf()* na bílý znak nebo na znak, který nepatří do zápisu čtené hodnoty, ukončí se čtení dříve. To poněkud komplikuje čtení řetězců obsahujících v textu mezery. Naproti tomu funkce *gets(), puts()* pracují s textovými řádkami jako s celky. Funkční prototyp má následující tvar:

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

Funkce *gets()* přečte řetězec znaků až do znaku *'\n'* tj. textovou řádku ze standardního vstupního zařízení a uloží ji do řetězce *str*. Znak *'\n'* se neukládá a řetězec je automaticky ukončen znakem *'\0'*. Návratová hodnota funkce je ukazatel na řetězec *str*. Pokud je řetězec prázdný, vrací funkce hodnotu symbolické konstanty *NULL*. Řetězec, kam tato funkce bude ukládat musí být správně alokován.

POZOR! Tato funkce je nebezpečná, protože nelze omezit počet znaků načítaných ze vstupu. Pokud je na vstupu delší řádek než řetězec, který jsme alokovali, dojde k nelegálnímu zápisu do paměti (překročení rozsahu pole). Tuto funkci **NIKDY NEPOUŽÍVEJTE!** Namísto této funkce lze použít funkci *fgets()* – viz dále.

6.2.6 Výstup po řádcích

Prototyp funkce pro výstup řádku má následující tvar:

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

Funkce *puts()* vytiskne řetězec str a odřádkuje, tj. vypíše znak '\n'. Funkce vrací nezáporné číslo, v případě že operace nemůže z nějakého důvodu proběhnout je návratová hodnota rovna *EOF*. Na rozdíl od *printf(),* funkce *puts()* text neinterpretuje.

6.3 Datový typ soubor

Kromě standardních, automaticky otevíraných souborů *stdin, stdout, stderr,* lze otevřít a používat další soubory. V hlavičkovém souboru *stdio.h* je pro práci s nimi zaveden datový typ *FILE*. Identifikátor souboru je typu *FILE* * tj. ukazatel na typ *FILE*. Nemá smysl pracovat s datovým typem *FILE* bez ukazatele. Norma *ANSI* jazyka *C* rozeznává dva druhy souborů - **textový** a **binární**.

6.3.1 Textové soubory

V textovém souboru lze knihovními funkcemi vytvářet a rozeznávat textové řádky nezávisle na operačním systému. To je umožněno tím, že systém při zápisu automaticky některé znaky (konec řádku '\n') do souboru doplňuje v souladu s konvencí[1], která pro textové soubory v daném operačním systému platí. Při čtení textového souboru se tyto znaky naopak automaticky vyřazují. Tento režim práce s textovými soubory zajišťuje, že jejich obsah si lze prohlédnout, vytvořit nebo opravit běžným editorem. Výhodou je také poměrně bezproblémová přenositelnost.

6.3.2 Binární soubory

Binární soubor není funkcemi čtení a zápisu tímto způsobem nijak ovlivňován. To znamená, že co do binárního souboru zapíšeme, to v něm také přesně bude, a co je v binárním souboru zapsáno, to se také přesně přečte. Výhoda binárních souborů spočívá v tom, že často pro uchování stejného množství informace potřebují mnohem méně prostoru než textové soubory. Nevýhodou je problematická přenositelnost mezi platformami (problémy s little-endian, big-endian, zarovnáváním datových typů). Pro zápis některých dat (obrázky, video, atd.) lze použít pouze tento typ souborů.

6.3.3 Otevření a uzavření souboru

Pro **otevření** a uzavření souboru se používá dvojice knihovních funkcí *fopen(), fclose().* Funkce *fopen()* slouží k otevření existujícího souboru nebo k vytvoření nového souboru.

---

[1] Tyto konvence jsou dány historickými podmínkami z dob vzniku sálových počítačů, kdy se textový výstup nerealizoval na obrazovku, ale tisknul se na papír. Odtud pochází i zkratky znaků, sloužících pro ukončení řádku CR – carriage return, LF – line feed (návrat vozíku, posun řádku). Paradoxně systémy firmy Microsoft, které nikdy výstup na papír nepotřebovaly, stale ukončují řádky textových souborů dvojicí znaků CR LF, kdežto systémy vycházející z původního Unixu šetří místem a ukončují řádky pouze znakem LF. A abychom to neměli příliš snadné, systémy firmy Apple ukončují řádky v textových souborech znakem CR.

Prototyp funkce pro otevření souboru má následující tvar:

```
FILE *fopen(const char *jmeno, const char *modus);
```

První parametr *jmeno* je označení souboru, který se má otevřít nebo vytvořit. Za tento parametr lze dosadit řetězec nebo pole typu *char* obsahující jméno souboru (eventuálně včetně označení disku a cesty). Druhý parametr *modus* určuje mód, ve kterém se má s otevíraným souborem pracovat (pozor, jde o řetězec, takže musí být ve dvojitých uvozovkách, i když obsahuje pouze jediný znak). V následující tabulce jsou uvedeny možnosti, jak lze druhý parametr zvolit:

"r"	textový soubor pro čtení
"w"	textový soubor pro zápis nebo pro přepsání
"a"	textový soubor pro připojení 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

Pokud lze soubor *jmeno* otevřít nebo vytvořit v daném módu, vrátí funkce ukazatel na typ *FILE*. Typ *FILE* je definován v <*stdio.h*>. Je to struktura, která obsahuje různé typy informací o souboru (velikost, aktuální pozici, přístupové režimy, apod.).
Pokud se soubor nepodaří otevřít nebo vytvořit, vrátí funkce *fopen()* hodnotu symbolické konstanty *NULL*.

Program musí **vždy** otestovat, zda se povedlo soubor otevřít, protože není samozřejmé, že se ho podaří otevřít. Soubor může mít nastavena práva tak, že jej uživatel nesmí modifikovat nebo vůbec nemusí existovat.

Pro **uzavření** souboru je určena funkce *fclose()*. Prototyp funkce pro uzavření souboru má následující tvar:

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

Pokud se soubor identifikovaný ukazatelem *file* nepodaří uzavřít, (např. nebyl-li otevřen,) vrací funkce hodnotu symbolické konstanty *EOF*. Při úspěchu vrací funkce *fclose()* nulu. Pokud program neuzavře otevřený soubor, může dojít k jeho poškození, protože není zaručeno, že se do něj zapíší všechna data, která do něj program zapsal. Příčinou je to, že standardní knihovna přistupuje k souborům přes vyrovnávací paměť a ukončením programu (například funkcí *abort()*) bez uzavření souboru by mohlo dojít ke ztrátě dat (na některých systémech).

Příklad: Použití funkcí *fopen()*, *fclose()*:

```
FILE *soubor;
      ...
soubor=fopen("DATA.TXT","r");
      ...
fclose(soubor);
      ...
```

V uvedeném příkladu je funkcí *fopen()* otevřen soubor DATA.TXT jako textový soubor pro čtení. Vyvoláním funkce *fclose()* je v našem příkladu soubor DATA.TXT uzavřen. (Patří k dobrým programátorským zvykům uzavřít soubor ihned po ukončení práce s ním.) Operace otevření a uzavření souboru nemusí ovšem proběhnout správně, program by proto měl testovat úspěšnost provedení těchto operací.

Příklad: Testování úspěšnosti otevření a zavření souborů.

```
FILE *soubor;
char *jmeno = "DATA.TXT"
     ...
soubor=fopen(jmeno,"r");
if(soubor==NULL)
{
  fprintf(stderr, "\nChyba pri otevreni %s\n", jmeno);
  return;
}
     ...
if(fclose(soubor)==EOF)
{
  fprintf(stderr, "\nChyba pri uzavreni %s\n", jmeno);
  return;
}
     ...
```

6.3.4 Souborový vstup a výstup po znacích

Funkce *getc(), putc()* určené pro vstup znaků ze souboru a výstup znaků do souboru jsou analogické funkcím *getchar()* a *putchar()* pro standardní vstup a výstup znaků, které jsme probrali v předchozí části.

Funkce *getc()* je tedy určena pro vstup znaků ze souboru, funkce *putc()* pro výstup znaků do souboru. Uveďme pro srovnání funkční prototypy všech čtyř funkcí:

```
int getchar(void);
int getc(FILE *file);

int putchar(int c);
int putc(int c, FILE *file);
```

Funkce *getc()* má jediný parametr typu ukazatel na *FILE* identifikující soubor, ze kterého má být znak přečten, návratovou hodnotou této funkce je kód znaku, přečtený z určeného souboru. Jestliže bylo dosaženo konce souboru, vrací funkce hodnotu symbolické konstanty *EOF*.

Poznámka: Jak už víme *EOF* není znak, ale pouze symbolická konstanta určená k detekci konce souboru. Ačkoli si mnoho začínajících programátorů myslí, že *EOF* je znak, který se nachází na konci každého souboru, není to pravda. O tom, jakým způsobem operační systém rozpozná konec souboru rozhoduje použitý systém souborů (např. FAT32, EXT3, …).
V praxi není vůbec zaručeno, že soubor bude označen nějakou speciální značkou.

Funkce *putc()* má jako první argument hodnotu typu **int**, zadávající kód znaku, který má být zapsán do souboru, který je identifikován druhým parametrem. Návratová hodnota funkce je kód zapsaného znaku, nebo *EOF* pokud při výstupu došlo k chybě.
Tyto funkce opět pracují s datovým typem **int**, aby bylo možné detekovat konec souboru (*EOF*).

Následující program využívá knihovní funkce *fopen(), fclose(), getc()* pro určení počtu řádků textového souboru.

Příklad: Program určí celkový počet řádků a počet neprázdných řádků textového souboru.

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

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

int c;
  while((c=getc(soubor))!=EOF)
  { // test - konec souboru
    if(c=='\n')
    { // test - konec řádků
      rd_0++;
      if(zn>0)
        rd++;
      zn=0;
    }
    if(isspace(c)==0) // podmínka neplatí pro "bílý" znak
      zn++;
  }

rd_0++;
  if(zn>0)
    rd++;
  printf("\nPocet neprazdnych radku=%d\nPocet vsech
radku=%d\n"
          ,rd,rd_0);

  if(fclose(soubor)==EOF)
  {
    fprintf(stderr, "\nChyba pri uzavreni TEXT\n");
    return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
}
```

Uvedený program využívá knihovní funkci (přesněji, jde o předdefinované makro) *isspace()* k identifikaci bílých znaků. Proto je na začátku programu vložen hlavičkový soubor *ctype.h* příkazem *#include.*

6.3.5 Formátovaný vstup a výstup

Pro formátovaný vstup ze souboru a formátovaný výstup do souboru se používají funkce *fscanf(), fprintf()* analogické funkcím *scanf(), printf()* určeným pro formátovaný standardní vstup a výstup. Funkční prototypy všech čtyř funkcí:

```
int scanf(const char *retezec, ...);
int fscanf(FILE *file, const char *retezec, ...);

int printf(const char *retezec, ...);
int fprintf(FILE *file, const char *retezec, ...);
```

Funkční prototypy funkcí *fscanf(), fprintf()* obsahují tedy oproti funkcím *scanf(), printf()* navíc parametr typu ukazatel na *FILE* identifikující soubor, se kterým se příslušná vstupní nebo výstupní operace provádí. Ostatní parametry mají stejný význam jako u funkcí *scanf(), printf()*, stejně je také definována návratová hodnota obou funkcí.

Funkce *fprintf()* se často používá pro výstup chybových hlášení na standardní chybový výstup *stderr* .

6.3.6 Souborový vstup a výstup po řádcích

Pro **vstup** řádky ze souboru slouží funkce *fgets()*. Prototyp funkce pro vstup řádky ze souboru má následující tvar:

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

Funkce *fgets()* přečte řetězec znaků až do znaku *'\n'* nejvýše však *max* znaků ze souboru identifikovaného ukazatelem *file* a uloží jej do řetězce *str*. Znak *'\n'* se ukládá a řetězec je automaticky ukončen znakem *'\0'*. Návratová hodnota funkce je ukazatel na řetězec *str*. Pokud je soubor prázdný, vrací funkce hodnotu symbolické konstanty *NULL*. Pokud při čtení nenarazí na konec řádku, tak **nepřejde** na další řádek, ale při dalším čtení bude pokračovat uprostřed rozečteného řádku.

Funkce *fputs()* je určena pro **zápis** řetězce do souboru, funguje analogicky funkci *puts()*. Po zapsání řetězce ale neodřádkuje.

Prototyp funkce pro zápis řetězce do souboru má následující tvar:

```
int fputs(char *str, FILE *file);
```

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

```
#include <stdio.h>
int main(void)
{
  char nazev[20];
  printf("Ktery soubor se ma kopirovat?\n"
         "Zadejte nazev existujiciho textoveho souboru!\n");
    scanf("%19s", nazev); // cteni nazvu souboru,
                          //  ktery budeme kopirovat
  FILE *file_r;
```

```
  if((file_r=fopen(nazev,"r"))==NULL)
  {
    fprintf(stderr, "Chyba pri otevreni souboru pro
cteni!\n");
    return EXIT_FAILURE;
  }
printf("Do ktereho souboru kopirovat?\n"
         "Zadejte novy odlisny nazev textoveho souboru!\n");
 scanf("%19s", nazev); // cteni nazvu souboru,
                       // do ktereho ulozime kopii
 FILE *file_w;
 if((file_w=fopen(nazev,"w"))==NULL)
 {
  fprintf(stderr, "\nChyba pri otevreni souboru pro
zapis!\n");
  return EXIT_FAILURE;
 }
// Kopirovani souboru:
  int c;
  while((c=getc(file_r))!=EOF) // c=getc() musi byt v zavorce
  putc(c,file_w);
  int rclose = fclose(file_r);
  int wclose = fclose(file_w);
  if(rclose==EOF || wclose==EOF)
  {
    fprintf(stderr, "Nektery ze souboru nelze uzavrit!\n");
    return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
}
```

Pro kopírování souborů je zde použito funkcí *getc()*, *putc()*, tj. soubor je kopírován znak po znaku. Namísto toho je možné také použít funkce *fgets()*, *fputs()*, tj. kopírování po řádkách. V uvedeném programu by bylo třeba změnit jen část programu označenou komentářem */* Kopirovani souboru: */* a doplnit definici pole typu **char**, která nahradí roli proměnné *c*. Program bude pracovat za předpokladu, že textové řádky neobsahují více znaků než 100*:*

Příklad: Úprava předchozího programu s použitím *fgets()*.

```
  ...
char radka[101];   // k popisum doplnime definici pole 'radka'
  ...
/* Kopirovani souboru: */
  while(fgets(radka,100,file_r)!=NULL) // kopirovani po
radkach
    fputs(radka,file_w);
```

6.3.7 Testování chyb

Funkce pro práci se soubory vždy vracejí buďto *NULL* nebo *EOF* pro detekci chybového stavu. Chybový stav lze také detekovat pomocí funkce *ferror()*.
Prototyp funkce pro testování chyb má následující tvar:

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

Funkce *ferror()* vrací nenulovou hodnotu, pokud došlo při práci se souborem spojeným s *fp* k chybě, jinak vrací nulu.

Příklad: Testování chyby po vstupní operaci.

```
FILE *fp;
.
.
  int ch = fgetc(fp);
  if(ferror(fp)) {
    printf("Chyba souboru\n");
    break;
  }
```

Pokud chceme otestovat konkrétní kód chyby, použijeme k tomu pomocnou globální proměnnou *errno*, deklarovanou v <*errno.h*>.

```
int errno;
```

Tuto proměnnou nastavují všechny vstupně/výstupní operace. Kód chyby tedy musíme otestovat bezprostředně po vykonání testované operace (pozor, v/v operací je i *printf()*).

Kódy jednotlivých chyb lze najít v dokumentaci standardní knihovny jazyka C. Pokud se spokojíme s anglicky psanými chybovými zprávami, můžeme použít funkci *strerror()* z rozhraní <*string.h*>

Příklad: otestuje správnost otevření souboru.

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

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

6.4 Shrnutí

Po přečtení této kapitoly je čtenář obeznámen s rozhraním standardní knihovny jazyka C pro práci se standardním vstupem a výstupem a se soubory. Dokáže při spouštění programu přesměrovávat soubory na standardní vstup programu a standardní výstup programu do souboru. Dokáže také spojovat výstupy a vstupy programů pomocí roury.

Čtenář dokáže realizovat vstup a výstup po znacích i formátovaný vstup a výstup po slovech a to jak s využitím standardního vstupu a výstupu, tak s využitím souborů. Umí používat vzory cyklů pro efektivní zpracování dat od začátku do konce souboru bez nadbytečného volání systémových funkcí a bez zbytečného vícenásobného průchodu celým souborem.

Při zpracování souborů je dokáže správně otevírat pro čtení i zápis a ošetřovat chyby, které při tom mohou nastat. Je si vědom skutečnosti, že otevřené soubory je potřeba v programu po skončení práce zavřít, aby nedošlo ke ztrátě dat.

Čtenář si je vědom rozdílů mezi textovými a binárními soubory. Po přečtení předchozích kapitol je nyní čtenář již schopen vytvářet užitečné a obecné programy v jazyce C, které používají vstupy a výstupy.

6.5 Úlohy k procvičení

1. Napište program, který ve standardním vstupu najde všechny číslice ('0'-'9') a vypočte jejich četnost. Formátovanou tabulku četnosti těchto číslic vypište 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 každý výskyt tabulátoru nahrazoval osmi mezerami.
5. Napište program, který bude na standardním vstupu očekávat posloupnost čísel typu **float** oddělených libovolnými bílými znaky (každé číslo bude zapsáno textově, podle zvyklostí jazyka C). Na standardní výstup vypište průměrnou hodnotu načtených čísel. Pokud na vstupu bude číslo ve špatném formátu, program ukončete s odpovídající chybovou zprávou.
6. Napište program, který bude zpracovávat posloupnost racionálních čísel (použijte typ double) ze standardního vstupu (ne ze souboru). Tyto hodnoty budou zadávány textově a odděleny bílými znaky. Délka posloupnosti nebude předem známa a její zpracování skončí až tehdy, když detekujete konec vstupu (EOF). Po zpracování této posloupnosti vypište na výstup tyto hodnoty: průměr ze všech vstupních hodnot, průměr druhých mocnin všech vstupních hodnot, minimální a maximální zadanou vstupní hodnotu. Ošetřete všechny chyby, které mohou při zpracování nastat. Při řešení smíte použít jediný cyklus.
7. Modifikujte předchozí úlohu tak, že program bude syntaktické chyby na vstupu ignorovat (přeskakovat).
8. Modifikujte předchozí dvě úlohy tak, aby pracovaly s textovými soubory.
9. Vytiskněte na obrazovku formátovanou tabulku s hodnotami funkce sinus od 0° do 180° po deseti stupních. Využijte funkci *sin()* z rozhraní <*math.h*>.

6.6 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?
5. Čím se liší standardní vstup a výstup od jiných textových souborů?
6. Jakým způsobem lze při čtení souboru detekovat jeho konec?
7. Kde v souboru lze nalézt znak *EOF*?