Vstup a výstup – datové proudy v C

Petr Šaloun

katedra informatiky FEI VŠB-TU Ostrava

24. října 2011

Přehled, rozdělení I/O, základní pojmy

Vstupně–výstupní zařízení (I/O):

- *znaková*: vstupní – klávesnice, výstupní – monitor, tiskárna.
- *bloková*: pevný disk, flash disk, SD, microSD...

Z klávesnice čteme sekvenčně znak po znaku (*sekvenční přístup*), u binárního diskového souboru můžeme libovolně přistupovat ke zvolené části dat – *náhodný přístup*.

Pojmy:

- *Řádek textu* je posloupnost znaků ukončená symbolem (symboly) přechodu na nový řádek.
- *Soubor* je posloupnost znaků (bajtů) ukončená nějakou speciální kombinací, která do obsahu souboru nepatří – konec souboru symbolicky EOF.
- *Textový soubor* obsahuje řádky textu.
- *Binární soubor* obsahuje hodnoty v témže tvaru, v jakém jsou uloženy v paměti počítače. OS nezná typ souboru. Přípony jmen souborů jsou většinou pouze doporučeními.

Standardní vstup a výstup

Každý spuštěný program v jazyce C má otevřen

- stdin – standardní vstup,
- stdout – standardní výstup, a
- stderr – standardní chybový výstup.

Jsou napojeny na klávesnici a terminál.
Na úrovni operačního systému lze vstup a výstup přesměrovat:
app.exe $<$ in.txt $>$ out.txt
oblíbené pro tvorbu a ladění funkčního jádra programu, který zatím nemá obsažen styk s uživatelem. Vstupní data uložená z připraveného testovacího souboru, a výsledek přesměrováváme do jiného výstupního souboru.
Filtry = jednoduché programy, čtou ze standardního vstupu a zapisují do standardního výstupu; užitečné filtry: grep, more, find.

Poznámky ke standardnímu I/O

- Pro ukončení vstupu z klávesnice použijeme v MS–DOSu a MS Windows kombinaci ctrl −z, v unixu ctrl −d.
- Standardní vstup a výstup používá vyrovnávací paměť obsahující jeden textový řádek.
- Při volání funkcí standardního vstupu či standardního výstupu musíme použít hlavičkový soubor stdio.h.

Standardní vstup a výstup znaků

Základní primitiva jsou zdánlivě funkce getchar() a putchar(),
ve skutečnosti makra, volají getc(stdin) a putc(c, stdout).
int getchar(**void**);
přečte ze standardního vstupu jeden znak, který vrátí jako svou návratovou hodnotu. V případě chyby vrátí hodnotu EOF.
int putchar(**int** c);
má zcela opačnou úlohu. Znak, který je jejím argumentem, zapíše na standardní výstup. Zapsaná hodnota je současně návratovou hodnotou; nastane–li chyba, vrací EOF.
Vždy čteme/zapisujeme **int**, nikoliv **char**. Textový soubor obsahuje znaky, jeho konec, hodnota EOF do souboru nepatří, hodnota musí být odlišná od ostatních znaků, proto je typu **int**.

Standardní vstup a výstup znaků

```c
/*****************************
 * cpy.c
 * kopiruje znak ze vstupu na vystup
 *****************************/
#include <stdio.h>

int main(void)
{
  int c;

  while ((c = getchar()) != EOF)
    putchar(c);
  return 0;
}
```

Pozn. Nesmíme zapomenout přesměrovat vstup a výstup.

Standardní vstup a výstup řetězců

jednoduchá nadstavba nad čtením znaků.

```
char *gets(char *s);
int puts(const char *s);
```

gets() načte do znakového pole vstupní řetězec až do konce řádku, symbol \n není do znakového pole zapsán. Ukazatel na pole (načtený řetězec) je návratovou hodnotou. Chybu signalizuje návrat NULL.
puts() zapíše řetězec na výstup a přidá přechod na nový řádek \n. Chybu představuje návratová hodnota EOF, jinak vrací kladné celé číslo.

Pozor, gets() nemá informaci o délce oblasti vymezené pro čtený řetězec!

Řádek textu je posloupnost znaků ukončená symbolem přechodu na nový řádek \n.
Textový soubor představuje posloupnost řádků textu, je ukončený symbolem EOF.
Využití – kopírování souboru po řádcích – následující příklad.

Kopírování vstupu na výstup po řádcích – gets.c

```c
/**************************/
/* GETS.C                 */
/* GET String function    */
/**************************/

#include <stdio.h>
#define MAX_STR 512

int main(void)
{
  char s[MAX_STR];

  while (gets(s) != NULL)
    puts(s);
  return 0;
}
```

Formátovaný standardní výstup – funkce printf()

```
int printf (const char *format [, argument, ...]);
```

format – formátovací řetězec, tj. popis formátu pro každý argument, nebo text, který bude zapsán do výstupu.
Popis formátu začíná znakem %, výstupní znak % musíme zdvojit: %%.
Návratová hodnota – počet položek znaků zapsaných do výstupu, nebo EOF v případě chyby.

Určení formátu ve funkci printf()

```
% [flags] [width] [.prec] [h|l|L] type_char
```

položka	význam
flags	zarovnání výstupu, zobrazení znaménka a desetinných míst u čí nuly, prefix pro osmičkový a šestnáctkový výstup;
width	minimální počet znaků na výstupu, mohou být uvedeny meze nulami;
.prec	maximální počet znaků na výstupu, pro celá čísla minimum z znaků, pro racionální počet míst za desetinnou tečkou;
`l|L`	l indikuje dlouhé celé číslo, L **long double**;
type_char	povinný znak, určuje datový typ konverze.

Datový typ konverze – type_char

symbol	význam
d, i	desítkové celé číslo se znaménkem;
u	desítkové celé číslo se bez znaménka;
o	osmičkové celé číslo;
x, X	šestnáctkové celé číslo, číslice ABCDEF malé (x) nebo velké (X);
f	racionální číslo (**float**, **double**) bez exponentu, implicitně šest des
e, E	racionální číslo (**float**, **double**) v desetinném zápisu s exponent
	jedna pozice před desetinnou tečkou, šest za ní. Exponent uvozuj
	E.
g, G	racionální číslo (**float**, **double**) v desetinném zápisu s exponenten
	(podle absolutní hodnoty čísla). Nemusí obsahovat desetinnou tečk
	setinnou část). Pokud je exponent menší, než -4, nebo větší, než
	číslic, je použit.
c	znak;
s	řetězec.

Příznak flag ve funkci printf ()

příznak	význam
−	výsledek je zarovnán zleva;
+	u čísla bude vždy zobrazeno znaménko (i u kladného);
mezera	pro kladná čísla vynechá prostor pro znaménko;
#	pro formáty o, x, X výstup jako konstanty jazyka C,
	pro formáty e, E, f, g, G vždy zobrazí desetinnou tečku,
	pro g, G ponechá nevýznamné nuly,
	pro c, d, i, s, u nemá význam.

Šířka width ve funkci printf ()

šířka	význam
n	je vytištěno nejméně n znaků zarovnaných zleva či zprava, viz *příznak*, doplněno mezerami;
0n	jako předchozí, doplněno zleva nulami;
$*$	jako šířka pole bude použit následující parametr funkce printf ().

Přesnost .prec ve funkci printf ()

přesnost	význam
.0	pro e, E, f nezobrazí desetinnou tečku,
	pro d, i, o, u, x nastaví standardní hodnoty;
.n	pro d, i, o, u, x minimální počet číslic,
	pro e, E, f počet desetinných číslic,
	pro g, G počet platných míst,
	pro s maximální počet znaků;
*	jako přesnost bude použit následující parametr funkce printf ().

Formátovaný standardní vstup – funkce scanf()

```
int scanf(const char *format [, address, ...]);
```

format – formátovací řetězec, může obsahovat:

- přeskok bílých znaků (oddělovačů), tedy mezery, tabulátoru, nového řádku a nové stránky;
- srovnání znaků formátovacího řetězce se vstupními, je–li na vstupu jiný, než určený znak, je čtení ukončeno;
- specifikace formátu vstupní pro hodnoty, je vždy uvozena znakem %.

address – určuje paměťovou oblast (adrasu), do níž bude odpovídající vstupní hodnota uložena (adresa proměnné, ukazatel na pole znaků), výjimkou je použití %∗ – je načítána hodnota, nikoliv adresa, viz tabulka přesnost.
Návratová hodnota – počet bezchybně načtených položek (polí), nula je nulový počet uložených položek, EOF je pokus číst další položky po vyčerpání vstupu.

Formátová specifikace ve funkci scanf()

```
% [*] [width] [h|l|L] type_char
```

položka	význam
*	přeskoč popsaný vstup;
width	maximální počet vstupních znaků;
h|l|L	modifikace typu;
type_char	(povinný) typ konverze.

* – vstupní hodnotu

načíst, ale do paměti nezapisovat

width – maximální počet znaků, které budou použity při vstupu

(`h|l`) – pro celočíselný typ (d) – určují **short**, **long**;

(`l|L`) – pro racionální typ (f) – **float** na **double**, resp. **long double**

Určení typu dat ve funkci scanf()

symbol	význam
d	celé číslo;
u	celé číslo bez znaménka;
o	osmičkové celé číslo;
x	šestnáctkové celé číslo;
i	celé číslo, zápis odpovídá zápisu konstanty jazyka C, například 0x uvozuje celé číslo v šestnáctkové soustavě;
n	počet dosud přečtených znaků aktuálním voláním funkce scanf();
e, f, g	racionální číslo typu **float**, lze modifikovat pomocí l|L;
s	řetězec, i zde jsou úvodní oddělovače přeskočeny!, v cílovém poli je ukončen '\0';
c	vstup znaku, je–li určena šířka, je čten řetězec bez přeskočení oddělovačů!;
[search_set]	jako s, ale se specifikací vstupní množiny znaků,

Příklad – počítej x^2 zadaného x – scanwhil.c

ukončení vstupu – EOF, Win ctrl −z, Unix ctrl −d

```
#include <stdio.h>
int main(void) {
  float f;
  printf("zadej x:");
  while (scanf("%f", &f) != EOF) {
    printf("x=%10.4f x^2=%15.4f\nzadej x:", f, f*f); }
  return 0;
} /* int main(void) */
```

```
x=     2.0000 x^2=         4.0000
zadej x:3
x=     3.0000 x^2=         9.0000
zadej x:4
x=     4.0000 x^2=        16.0000
zadej x:^Z
```

Vstupní a výstupní operace v paměti

```
int sprintf (char *buffer, const char *format [, argum
int sscanf (const char *buffer, const char *format [,
```

Vstupní/výstupní řetězec je buffer, ostatní argumenty mají stejný význam jako u funkcí pro formátovaný standardní vstup a výstup.

Práce se soubory z konsoly a aplikace

Textový obsah – přesměrování vstupu a výstupu na úrovni OS z console:

```
c:>app.exe < vstup.txt > vystup.txt
```

Binární data (paměť $\leftrightarrows$ disk) bez textové konverze, pevný bajtový formát.
jméno souboru v OS vs. identifikátor pro přístup k souboru.
Binární vs. textový režim práce s obsahem souboru.

Práce se soubory – datový proud a přímý přístup

proud – ISO/ANSI normou jazyka C, tak POSIX, řada modifikátorů, definic a funkcí s definovanými vlastnostmi a rozhraním, doporučený přístup.

volání – jen POSIX, maximální rychlost (volání služeb jádra operačního systému OS), možná systémová závislost, nedoporučováno. V předmětu *Základy programování* není součástí látky.

FOPEN_MAX – počet současně otevřených souborů,
FILENAME_MAX – maximální délka jména souboru,
EOF – konec souboru.

Datové proudy

FILE * – datový typ,
stdio.h – funkční prototypy.

Běhová podpora (runtime) vždy otevře:

```
FILE *stdin;
FILE *stdout;
FILE *stderr;
```

Dva režimy proudů (ISO/ANSI norma)
textový – rozlišuje řádky textu,
binární – data jako v paměti.
Režim stanovíme při otevírání souboru.

Otevření datového proudu

```
FILE *fopen(const char *filename, const char *mode);
```

vrací FILE *, nebo NULL při neúspěchu.
filename – jméno souboru
mode – režim práce se souborem a typ, například: `"a+b"`, `"wb"`, ...
textový režim je implicitní, uvést `t` se doporučuje.
Otevření – určuje režim přístupu k datům v proudu. Proud lze znovuotevřít či spojit s novým souborem.

Režimy práce s datovým proudem

řetězec	význam (otevření pro:)
r	čtení;
w	zápis;
a	připojení;
r+	aktualizace (update) – jako rw;
w+	jako r+ a existující proud ořízne na nulovou délku; jinak nový soul
a+	pro aktualizaci, pokud neexistuje, vytvoří;
t	textový režim;
b	binární režim.

Zavření/znovuotevření datového proudu

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

uzavře proud. úspěch vrátí nulu, jinak EOF.
Uvolní paměť vyhrazenou pro strukturu FILE * a vyprázdní vyrovnávací paměť, aktualizuje adresářový záznam.

```
FILE *freopen(const char *filename, const char *mode,
```

uzavře soubor proudu stream (jako při volání fclose()),
otevře soubor jménem filename (jako při fopen(filename, mode))
návratová hodnota – otevřený proud, jinak NULL.

Proudy a vstup/výstup znaků

Konverze znaku na **int** po načtení umožní rozlišit případné EOF. Při zápisu do proudu je znak konvertován opačným postupem.

```
int getc(FILE *stream);
```

úspěch – převod (bez znaménka) na typ **int**.
chyba nebo konec proudu vrací EOF.

```
int putc(int c, FILE *stream);
```

zapíše c do stream a vrátí stejnou hodnotu, jako zapsal.
V případě chyby, nebo dosažení konce proudu vrací EOF.

```
int ungetc(int c, FILE *stream);
```

je–li c různé od EOF, pak zápis do stream a případně zruší příznak konce souboru.
je–li c rovno EOF, nebo nemůže–li zápis proběhnout, vrací EOF.
Jinak vrací c, přesněji **unsigned char** c.

Datové proudy a vstup/výstup řetězců

```
char *fgets(char *s, int n, FILE *stream);
```

načte řetězec (řádek až po jeho konec včetně znaku konce řádku) z proudu stream do vyrovnávací paměti s, nejdéle n−1 znaků.
Vrátí ukazatel na řetězec (vyrovnávací paměť); při chybě NULL.

```
int fputs(const char *s, FILE *stream);
```

zapíše do proudu řetězec ukončený zarážkou. Ani zarážku, ani případný konec řádku (obsažený na konci řetězce) do proudu nezapíše.
V případě úspěchu vrátí počet zapsaných znaků (délku řetězce), jinak EOF.

gets() nahradí znak konce řádku řetězcovou zarážkou.
puts() při zápisu do proudu nahradí zarážku řetězce znakem konce řádku.

Formátovaný vstup/výstup z/do proudu

```
int fprintf (FILE *stream, const char *format [, argument,
int printf  (              const char *format [, argument,
int fscanf  (FILE *stream, const char *format [, address,
...]);
int scanf   (              const char *format [, address,
...]);
```

Datové proudy a blokový přenos dat

Blokový přenos dat je nezbytný při práci s binárním proudem.
size_t – typ, určuje velikosti paměťových objektů, případně jejich počet

```
size_t fread(void *ptr, size_t size, size_t n, FILE *stream
```

přečte z stream položky o velikosti size v počtu n jednotek do paměti ptr
úspěch – vrátí počet načtených položek, jinak hodnota menší (pravděpodobně nula). Načtené položky jsou platné.

```
size_t fwrite(const void *ptr, size_t size, size_t n, FILE*
```

zápis položek do proudu.

Datove proudy – užitečné funkce

```
int feof(FILE *stream);
```

true (tj. různá od nuly), nacházíme se na konci proudu
false (nula) – jinak.

```
int fflush(FILE *stream);
```

vyprázdní bafr souboru (zápis).
vrací: nula = úspěch, EOF signalizuje chybu.

```
int fseek(FILE *stream, long offset, int whence);
```

přenese aktuální pozici CP v proudu stream na stanovené místo;
offset – posun;
whence je vztažný bod: SEEK_SET, SEEK_CUR, SEEK_END.

Datove proudy – další užitečné funkce

```
long ftell(FILE *stream);
```

vrátí aktuální pozici v proudu – pro binární proud = počet bajtů vzhledem k začátku, `-1L` = chyba + nastaví globální proměnnou errno.

ferror () – informuje o případné chybě při práci s proudem.
clearerr () – ruší nastavení příznaku chyby a konce proudu.
perror() – pošle řetězec chybového hlášení do stderr.
tmpfile() – otevře přechodný soubor v binárním režimu pro aktualizaci.
(souvisí s: tmpnam() a tempnam()).
fgetpos() a fsetpos()
umožňují uchovat (získat) pozici v proudu a pak ji (opětně) nastavit.
setbuf() a setvbuf() umožňují nastavit a případně modifikovat velikost vyrovnávací paměti pro určený proud.

Tvorba textového souboru, datový proud io−pv04.C II

```
int main(void) {
 FILE *soubor; int i; char *s, *jmeno = "soubor.txt";
 if ((soubor = fopen(jmeno, "wt")) == NULL)
          chyba(1);
 s = "Toto je textovy soubor vytvoreny v C.\n"
     "pristup pomoci proudu - FILE *\n";
 fputs(s, soubor);
 for (i = 1; i < 10; i++)
   fprintf(soubor, "%5d", i);
 fputs("\n\n", soubor);
 s="Uz to umim!\n";  fputs(s, soubor);
 s="Jeste pridat posledni radek na konec.";fputs(s, so
 if (fclose(soubor) == EOF)
     chyba(1);
 return 0;
} /* int main(void) */
```

Tvorba textového souboru, datový proud io−pv04.C I

```
/* soubor IO-DP01.C                                    */
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <process.h>
#include <string.h>
#include <sys\stat.h>
void chyba(int er) {
 extern int errno;
 if (er != 0)    {
    perror(strerror(errno));
    exit(errno);
   }
 return er;
} /* void chyba(int er) */
```

Tvorba textového souboru, datový proud io – pv04.C – vystup

```
Cteni textoveho souboru <soubor.txt> funkcemi pro dato
Toto je textovy soubor vytvoreny v C.
pristup pomoci proudu - FILE *
    1    2    3    4    5    6    7    8    9

Uz to umim!
Jeste pridat posledni radek na konec.

...KONEC...
```

Čtení textového souboru, datový proud io−dp02.c I

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <process.h>
#include <sys\stat.h>
#include <string.h>

const int MAX_DELKA_RADKU = 81;

void chyba(int er) {
 extern int errno;
 if (er != 0)    {
    perror(strerror(errno));
    exit(errno);
  }
} /* void chyba(int er) */
```

Čtení textového souboru, datový proud io−dp02.c II

```
int cti_radek(FILE *f, char *radek, int max) {
 int i = 0, ch = 0;
 char *s = radek;
 while ((i < max)
           && ((ch = getc(f)) != EOF)
           && (ch != '\n'))
   {
    s[i] = ch;
    i++;
   }
 s[i] = '\x0';
 return (ch == EOF) ? ch : ((i == 0) ? 1 : i);
} /* int cti_radek(int hdl, char *radek, int max) */
```

Čtení textového souboru, datový proud io−dp02.c III

```
int main(void) {
 int nacteno;  FILE *soubor;
 char s[MAX_DELKA_RADKU], *jmeno = "soubor.txt";
 if((soubor=fopen(jmeno,"rt"))==NULL)  chyba(1);
 printf("\nCteni textoveho souboru <%s> funkcemi pro datovy
 while((nacteno=cti_radek(soubor,s,sizeof(s)-1))!= EOF){
    printf("%s\n", s);
    *s = 0x0;  }
 if ((nacteno == EOF) && (strlen(s) != 0))
    printf("%s\n\n...KONEC...\n", s);
 else
    chyba(nacteno);
 if (fclose(soubor) == -1)
    chyba(1);
 return 0;
} /* int main(void) */
```