Vážení zákazníci,

dovolujeme si Vás upozornit, že na tuto ukázku knihy se vztahují autorská práva, tzv. copyright.

To znamená, že ukázka má sloužit výhradně pro osobní potřebu potenciálního kupujícího (aby čtenář viděl, jakým způsobem je titul zpracován a mohl se také podle tohoto, jako jednoho z parametrů, rozhodnout, zda titul koupí či ne).

Z toho vyplývá, že není dovoleno tuto ukázku jakýmkoliv způsobem dále šířit, veřejně či neveřejně např. umisťováním na datová média, na jiné internetové stránky (ani prostřednictvím odkazů) apod.

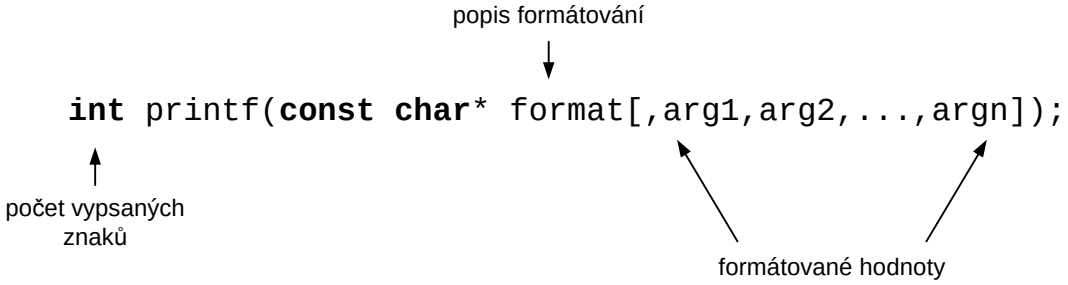
redakce nakladatelství BEN – technická literatura
redakce@ben.cz

2 Základní funkce a operátory

*V této kapitole se seznámíme s použitím funkce **printf**, probereme základní operátory a uvedeme nejdůležitější funkce.*

2.1 Funkce printf

Funkce **printf** (print formatted) slouží pro formátovaný výpis proměnných nebo výrazů základních datových typů, tato funkce je uvedena v hlavičkovém souboru **stdio.h**. Hlavička je uvedena na obr. 2.1.

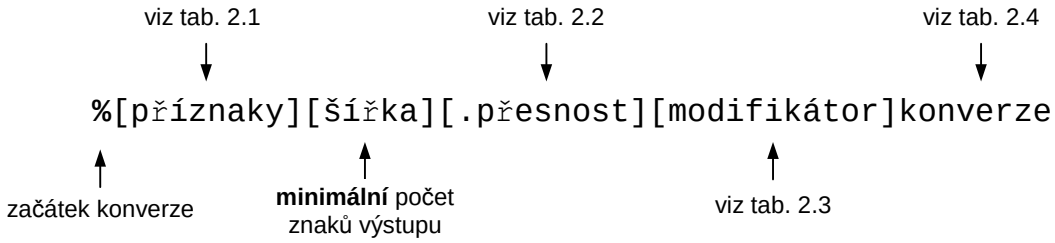


*Obr. 2.1 Hlavička funkce **printf***

Při práci s funkcí **printf** je třeba dodržet tato pravidla:

- **printf** je funkce s proměnným počtem parametrů. Povinným parametrem je pouze řetězec **format**, počet následujících parametrů je volitelný (včetně případu, že je jejich počet nulový),
- **arg1** až **argn** určují hodnoty, které mají vystoupit ve tvaru popsaném v řetězci **format**,
- popis tvaru, v jakém má hodnota arg1 až argn vystoupit, se nazývá konverzí. Každá **konverze** začíná znakem **%**, za kterým bezprostředně následuje popis konverze. Počet konverzí a formátovaných hodnot musí být stejný, jinak obdržíme „podivné“ výpisy,
- znaky nepatřící do žádné konverze vystoupí tak, jak jsou zapsány.

Pro lepší vysvětlení uvádíme formát zápisu jedné konverze na obr. 2.2, části zapsané v hranatých závorkách jsou nepovinné (lze je vynechat). Konverze začíná znakem %, následují nepovinné **příznaky** dle tab. 2.1, nepovinná **šířka** určuje minimální počet znaků na výstupu (na volná místa se vkládají mezery nebo nuly, pokud uvedená šířka začíná znakem 0), nepovinná **přesnost** se zapisuje za tečku (význam dle tab. 2.2), nepovinný **modifikátor** dle tab. 2.3 upravuje výpisy pro modifikované typy a poslední částí je povinná **konverze** dle tab. 2.4.



Obr. 2.2 Formát zápisu konverze

Jednotlivé části zápisu konverze jsou přehledně uvedeny v tab. 2.1 až 2.4.

Tab. 2.1 Příznaky

Příznak	Význam
–	zarovnání výpisu doleva (implicitně doprava),
+	výsledek bude opatřen znaménkem + pro kladná čísla (implicitně se používá pouze – pro záporná čísla a kladná čísla nejsou opatřena znaménkem),
mezera	pro kladná čísla bude výsledek předcházet mezera (záporná čísla budou opatřena znaménkem – a mezera se vypustí),
#	převedení výpisu do alternativního tvaru, závisí na použité konverzi (viz tab. 2.4): • **c**, **d**, **i**, **u** – nemá efekt, • **o** – pokud je číslo nenulové, tiskne se úvodní 0, • **x**, **X** – tiskne se úvodní 0x (0X), • **e**, **E**, **f** – tiskne se desetinná tečka i v případě, že za celou částí nejsou číslice (normálně se v tomto případě tečka netiskne), • **g**, **G** – stejné jako **e**, **E** s tím, že je jsou vyjmuty vodící nuly.

Tab. 2.2 Přesnost

Konverze	Význam
d, **i**, **o**, **u**, **x**, **X**	minimální počet cifer na výstupu,
e, **E**, **f**	počet cifer za desetinnou tečkou.

Tab. 2.3 Modifikátor

Modifikátor	Význam
h	• pro konverze **d**, **i**, **o**, **u**, **x**, **X** bude **int** konvertován na **short**,
l	• pro konverze **d**, **i**, **o**, **u**, **x**, **X** bude **int** konvertován na **long**, • pro konverzi **f** bude **float** konvertován na **double**,
L	• pro konverze **e**, **E**, **f** vystoupí číslo v přesnosti **long double**.

Tab. 2.4 Základní konverze

Konverze	Význam
c	vystoupí znak (**c** jako **char**),
d, i	vystoupí celé číslo se znaménkem v desítkové soustavě (**d** jako decimal, **i** jako **int**),
u	vystoupí celé číslo bez znaménka v desítkové soustavě (**u** jako **unsigned**),
o	vystoupí celé číslo v osmičkové soustavě (**o** jako octal),
e, E	vystoupí reálné číslo v exponenciálním tvaru v přesnosti **double** (**e** jako exponent), **e** pro malý exponent, **E** pro velký exponent,
f	vystoupí reálné číslo v desetinném tvaru v přesnosti **double** (**f** jako floating point),
x, X	vystoupí celé číslo v šestnáctkové soustavě (**x** jako he**x**adecimal), **x** pro malé znaky, **X** pro velké znaky.

Pro lepší vysvětlení uvedeme několik příkladů použití funkce **printf** a odpovídající výstupy (symbol □ značí mezeru). Mějme následující deklarace:

```
int a=-300, b=-5, c=0xabc;
float f=123.456;
```

```
printf("%d",a);
```
vypíše celkem 4 znaky: `–300`.

```
printf("%-5d",b);
```
vypíše celkem 5 znaků: `–5`□□□, protože bylo požadováno zarovnání doleva a celkový počet znaků byl 5.

```
printf("%05d",b);
```
vypíše celkem 5 znaků: `–0005`, protože šířka začínala znakem 0 a celkový počet znaků byl 5.

```
printf("%5x",c);
```
vypíše celkem 5 znaků: □□`abc`, protože malé x způsobí výpis malými písmeny.

```
printf("%#5X",c);
```
vypíše celkem 5 znaků: **`0XABC`**, protože velké X způsobí výpis velkými písmeny a příznak # tiskne úvodní 0X.

```
printf("%8.2f",f);
```
vypíše celkem 8 znaků: □□**`123.46`**, protože je požadována přesnost na dvě místa (zaokrouhlení) a celkový počet 8 znaků.

```
printf("%10.3e",f);
```
vypíše celkem 10 znaků: □□**`1.235e+2`**, protože je požadována přesnost na tři místa (zaokrouhlení) a celkový počet 10 znaků.

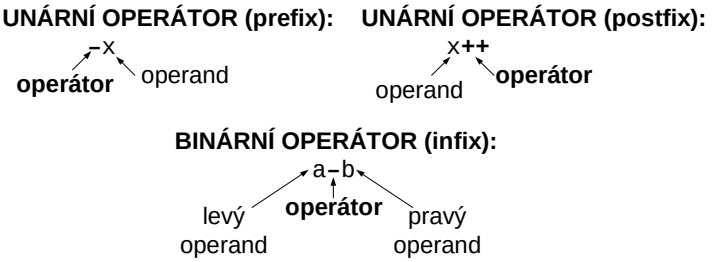
```
printf("%0.f",f);
```
vypíše celkem: **`123`**, desetinná část nebude nyní zobrazena (zaokrouhlí se).

2.2 Základní operátory

Operátory jsou symboly označující matematické nebo logické operace. V této chvíli se seznámíme pouze se základními operátory.

Rozdělení operátorů podle počtu operandů

- **Unární operátory** – mají jeden operand, zápis je obvykle v prefixové podobě (existují dvě výjimky, které používají postfixový zápis),
- **Binární operátory** – mají dva operandy, zápis je vždy v infixové podobě,
- **Ternární operátor** – má tři operandy (viz kapitolu 13.1.3).



Obr. 2.3 Rozdělení operátorů

Prefixový a postfixový zápis se používá pouze pro unární operátory. Prefix znamená, že se symbol operátoru píše před operand. U postfixové formy se symbol operátoru píše za operand.

Infixový zápis se používá pro binární operátory a symbol operátoru se uvádí mezi levý a pravý operand.

Aritmetické operátory

Všechny aritmetické operátory jsou použitelné jak pro celá čísla (včetně typu **char**), tak i pro reálná čísla. Výjimkou je operátor **%**, který lze používat pouze pro celá čísla. Níže jsou uvedeny jednotlivé aritmetické operátory:

- **Unární +**
  - unární + slouží pro označení kladného znaménka čísla,
  - zapisuje se v prefixové podobě, tedy **+x**,
- **Unární –**
  - unární – slouží pro označení záporného znaménka čísla (tedy násobení –1),
  - zapisuje se v prefixové podobě, tedy **–x**.

Pro níže uvedené binární operátory +, –, *, / platí:
- jsou-li oba operandy celá čísla, je výsledkem celé číslo,
- je-li alespoň jeden z operandů reálný, je výsledek také reálný.

- **Binární +**
  - binární + odpovídá operaci součtu dvou čísel,
  - zapisuje se v infixové podobě, tedy **x+y**,
- **Binární –**
  - binární – odpovídá operaci rozdílu dvou čísel,
  - zapisuje se v infixové podobě, tedy **x–y**,
- **Násobení ***
  - binární operátor * odpovídá operaci součinu dvou čísel,
  - zapisuje se v infixové podobě, tedy **x*y**,
- **Dělení /**
  - binární operátor / odpovídá operaci podílu dvou čísel,
  - zapisuje se v infixové podobě, tedy **x/y**,
  - jsou-li oba operandy celá čísla, jedná se o celočíselné dělení,
  - je-li alespoň jeden z operandů reálné číslo, jedná se o reálné dělení,
- **Zbytek po dělení % (modulo)**
  - binární operátor % získá zbytek po dělení dvou čísel,
  - zapisuje se v infixové podobě, tedy **x%y**,
  - je použitelný pouze pro celá čísla.

Operátory **+**, **–**, *****, **/** lze použít v libovolné kombinaci celočíselných i reálných datových typů.

Je-li alespoň jedním z operandů reálné číslo, je výsledek opět reálné číslo.

Zvláštní postavení má operátor **/**, který podle souvislosti představuje celočíselné dělení (pokud jsou oba operandy celá čísla) nebo reálné dělení (pokud je alespoň jeden operand reálné číslo).

Níže je uveden příklad použití operátoru /, jsou kombinovány operandy různých typů a vypsána výsledná hodnota podílu.

6 Funkce – základní rysy

*Pro návrh složitějšího programu je vhodné použít **hierarchickou strukturu**, která souvisí s postupným návrhem programu metodou **shora dolů**.*

Tato metoda spočívá v opakovaném rozkladu složitějších problémů na dílčí podproblémy. Přecházíme tak od původně abstraktních příkazů (řešící složitější problémy) na příkazy méně abstraktní (řešící dílčí podproblémy).

6.1 Základy používání funkcí

Nejdříve uvedeme základní vlastnosti funkcí.

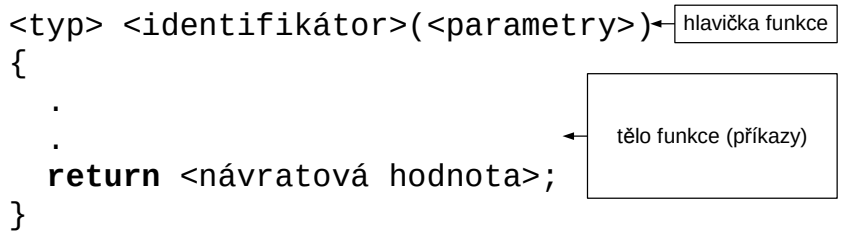
Výhody používání funkcí:

- umožňují rozdělit program do kratších celků,
- opakující části není třeba několikrát opisovat (případnou chybu opravíme na jediném místě),
- správnou činnost funkce lze ověřit nezávisle na zbytku programu,
- vyšší srozumitelnost, návrh programu metodou shora dolů (rozdělení celku na dílčí problémy, které se pak řeší odděleně).

Obecný zápis funkce

Zápis funkce začíná tzv. **hlavičkou**, která definuje tři významné vlastnosti funkce (viz obr. 6.1):

- typ návratové hodnoty (typ výsledku funkce),
- identifikátor funkce (jméno funkce),
- parametry (vstupně/výstupní operandy), každý parametr se musí definovat samostatně!



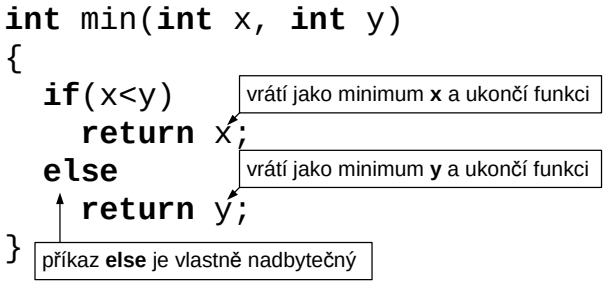
Obr. 6.1 Obecný formát zápisu funkce

Jednotlivé operace, které má funkce vykonat, se zapisují do **těla funkce**.

Příkaz **return** ukončí okamžitě prováděnou funkci a předá její výsledek.

Jako praktický příklad si můžeme uvést funkci, která ze dvou vstupních celých čísel určí minimum a vrátí ho jako svůj výsledek, viz obr. 6.2. Příkaz **else** je možné dokonce vynechat, protože příkaz **return** okamžitě ukončí funkci.

Takže pokud je podmínka splněna, vrátí se hodnota **x** a funkce **min** se ukončí. V opačném případě se vrátí hodnota **y** a funkce **min** se ukončí.



*Obr. 6.2 Zápis funkce **min***

Typ návratové hodnoty

Funkce mohou vracet hodnoty prakticky libovolného typu. Omezením je pouze skutečnost, že typem návratové hodnoty nesmí být pole. Situaci lze však řešit tak, že funkce vrátí ukazatel na pole.

Typ void

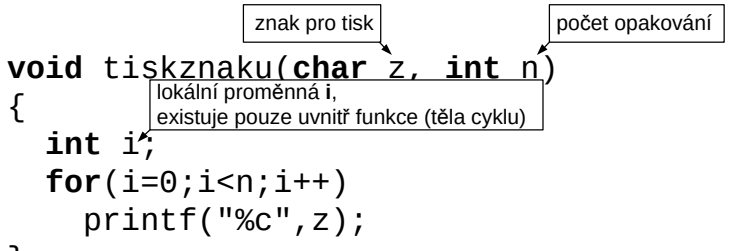
Existují případy funkcí, které nemají žádnou návratovou hodnotu. Pak je typ návratové hodnoty **void**.

Typ **void** (prázdný) nemůžeme použít pro definici proměnné, nebude mu totiž přidělena žádná paměť. Používá se vlastně pouze v případě funkcí.

Příkaz **return** nyní nepředává výsledek, ale pouze ukončuje funkci. Pokud není třeba provádění funkce předčasně ukončit, nemusíme tento příkaz vůbec používat!

Jako příklad funkce bez návratové hodnoty uvedeme funkci, která má na obrazovce opakovaně zobrazit zvolený znak dle jednoho parametru tolikrát za sebou, kolik udává druhý parametr.

Funkci můžeme nazvat například **tiskznaku**. První parametr bude tedy znak a druhý celé číslo, viz obr. 6.3.




```
void tiskznaku(char z, int n)
{
  int i;
  for(i=0;i<n;i++)
    printf("%c",z);
}
```

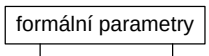
*Obr. 6.3 Zápis funkce **tiskznaku***

6.2 Volání hodnotou

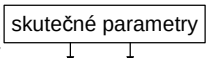
Až dosud jsme používali takzvané volání hodnotou. Platí pro něj tato pravidla:

- hodnota skutečného parametru se do formálního parametru zkopíruje,
- formální parametr je vlastně lokální proměnná,
- změna formálního parametru tedy nemá vliv na skutečný parametr,
- jako skutečný parametr muže být použita proměnná nebo literál.

Na obr. 6.4 je připomenuta funkce **tiskznaku**. Tato funkce má dva formální parametry (znak označený **z** a celé číslo označené **n**).




```
void tiskznaku(char z, int n)
{
  int i;
  for(i=0;i<n;i++)
    printf("%c",z);
}
.
.
int i=8;
tiskznaku('x',i);
```



Obr. 6.4 Vysvětlení pojmů formální a lokální parametr

Na obr. 6.4 je uvedena možnost volání této funkce. Skutečnými parametry mohou být buď proměnné nebo literály. Jako první parametr je dosazen literál **'x'**, jako druhý parametr je dosazena proměnná **i**.

Jsou-li parametry volané hodnotou, nevzniká mezi skutečným a formálním parametrem vazba. Hodnota skutečného parametru se prostě překopíruje do formálního parametru. Takže tělo funkce pracuje s kopií původní hodnoty. Jakékoli změny této kopie se tedy nemohou projevit jako změny skutečného parametru.

Volání odkazem se tedy hodí pouze pro tzv. vstupní parametry.

6.3 Volání přes ukazatel

Programovací jazyk C nedisponuje voláním parametrů odkazem jako jiné programovací jazyky. Realizace výstupních parametrů je založena na volání přes ukazatel.

Volání parametrů přes ukazatel znamená, že místo hodnot jednotlivých parametrů předáme jejich adresy. Přes ukazatel bude mít funkce přístup k proměnné jak pro čtení tak i pro zápis.

Vazba na skutečný parametr je zprostředkována pomocí adresy skutečného parametru.

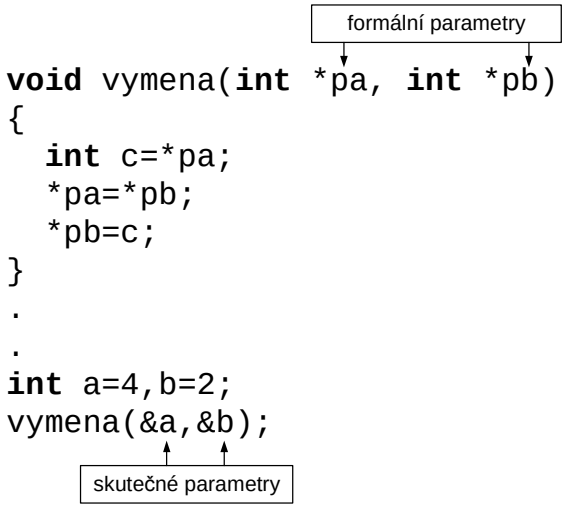
Jako praktický příklad volání parametrů přes ukazatel je uvedena funkce **vymena** dle obr. 6.5. Účelem této funkce je vzájemně vyměnit hodnoty dvou celočíselných proměnných. Taková operace je nezbytná například pro některé algoritmy řazení.

Parametry **pa** a **pb** jsou ukazatele na celá čísla. Takže při volání funkce musíme dosadit adresy dvou celočíselných proměnných (není tedy možné jako parametr dosadit literál).

Ve funkci je deklarována lokální proměnná **c**. Do této proměnné nejdříve zkopírujeme hodnotu první proměnné, na kterou ukazuje ukazatel **pa**. Operace ***pa** poskytne přístup k proměnné přes ukazatel **pa**.

Následně bude hodnota proměnné odkázaná ukazatelem **pa** přepsána hodnotou proměnné z adresy **pb** zápisem: ***pa=*pb**.

Na závěr se původní hodnota první proměnné (dočasně uložená v proměnné **c**) zapíše do druhé proměnné přes ukazatel **pb**.



Obr. 6.5 Vysvětlení volání parametrů přes ukazatel

Na obr. 6.5 je rovněž uvedena ukázka volání funkce **vymena**. Jsou deklarovány proměnné **a**, **b** s hodnotami 4 a 2. Do parametrů funkce **vymena** musíme dosadit adresy těchto proměnných získanými operacemi **&a**, **&b**. Funkce vymění hodnoty obou proměnných, takže po jejím volání bude platit: a=2, b=4.

Parametr typu pole

Jak již víme z kapitoly 5.3, představuje název pole vlastně i jeho adresu (resp. adresu prvního prvku pole). Parametr typu pole se tedy předává pomocí adresy.

Volání pole přes ukazatel přináší další výhodu.
Předává se pouze adresa, nedochází ke kopírování prvků.
Tento způsob volání parametru je tedy velmi efektivní.

Jako praktický příklad můžeme uvážit funkci **vypis**, která vypíše jednotlivé prvky pole (připomeňme, že funkce **printf** může prvky vypsat pouze tak, že v cyklu vypisuje pole po jednom prvku).

Když předáme adresu pole do funkce, nedá se již zjistit, jaká je velikost pole (pole se předává přes ukazatel, který má stále stejnou šířku), proto je nutný druhý parametr, který určuje počet prvků.

Na obr. 6.6. vlevo je ukázán klasický zápis. První parametr je ukazatel na **int** (představuje ukazatel na pole celých čísel) a druhý parametr slouží k předání počtu prvků pole.

Na obr. 6.6 vpravo je alternativní zápis, který může být poněkud přehlednější.

```
void vypis(int *pole, int n)
{
.
.
}
```

parametr typu ukazatel

```
void vypis(int pole[], int n)
{
.
.
}
```

parametr typu ukazatel

Obr. 6.6 Parametr typu pole předávaný přes ukazatel

Diskutujme ještě chvíli zmíněnou funkci **vypis**. Jelikož tato funkce bude prvky pole pouze číst a nebude provádět jejich změnu, je účelné tuto skutečnost zdůraznit modifikátorem **const**. Vznikne tak ukazatel na konstantní objekt, v případě pokusu změny prvků přes konstantní ukazatel podá překladač varovné hlášení.

```
void vypis(const int *pole, int n)
{
.
.
}
```

parametr typu ukazatel

```
void vypis(const int pole[], int n)
{
.
.
}
```

parametr typu ukazatel

*Obr. 6.7 Parametr typu pole předávaný přes **konstantní** ukazatel (viz PROG_06_03)*

6.4 Příklady

Po předcházejícím úvodu přistoupíme k praktickým příkladům.

Příklad PROG_06-01

Zadání: Napište funkci **sgn**, která má jako parametr celé číslo a jako výsledek vrací celé číslo. Chování funkce je popsáno tímto výrazem: $\operatorname{sgn}(x) = \begin{cases} -1; & x < 0 \\ 0; & x = 0 \\ +1; & x > 0 \end{cases}$.

Takto jednoduchý příklad jistě není třeba dokládat vývojovým diagramem.

Vlastní funkce **sgn** je velmi jednoduchá a zápis bude podobný jako v případě funkce **min** dle obr. 6.2. Nyní je příkaz **else** již zcela vynechán.

Hlavní program obsahuje cyklus **do..while**, který přečte celé číslo z klávesnice a následně vypíše jeho znaménko.

`long ftell(FILE *f)`
vrátí aktuální pozici ukazovátka souboru (při chybě vrací **-1L**).

`int fflush(FILE *f)`
vyprázdní vyrovnávací paměť (buffer) souboru (tím zajistí zápis dat na disk).

9.4 Příklady práce se soubory

Níže jsou uvedeny příklady práce s textovými a binárními soubory.

Příklad PROG_09-01

Napište program, který nageneruje řadu deseti náhodných čísel v rozsahu 1 až 1000 a uloží ji do textového souboru s názvem **CISLA.TXT**.

Na začátku programu jsou deklarovány proměnné: dvě celá čísla (jedno pro řízení cyklu a druhé pro generované číslo) a hlavně proměnná typu ukazatel na **FILE**.

Následně je vyvolána funkce **fopen**. První parametr je řetězec názvu souboru a druhý odpovídá otevření souboru pro zápis v textové formě.

Následuje test úspěšného otevření a zápis nagenerovaných čísel pomocí funkce **fprintf**. Nakonec je soubor zavřen pomocí funkce **fclose**.

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

int main(int argc, char *argv[])
{
  int i,c;
  FILE *f;

  f=fopen("CISLA.TXT","w");   <- otevření souboru

  if(f!=NULL)   <- test úspěšného otevření
  {
    for(i=0;i<10;i++)
    {
      c=rand()%1000+1;
      fprintf(f,"%d\n",c);   <- zápis čísel
    }
    fclose(f);   <- zavření souboru
    printf("Cisla uspesne zapsana\n");
  }
  else
    printf("Soubor se nepodarilo otevrit\n");

  system("PAUSE");
  return 0;
}
```

Obsah souboru CISLA.TXT (\r\n odpovídají znaků s pořadovými čísly 13 a 10):

```
42\r\n
468\r\n
335\r\n
501\r\n
170\r\n
725\r\n
479\r\n
359\r\n
963\r\n
465\r\n
```

Příklad PROG_09-02

Napište program, který čte čísla z textového **CISLA.TXT**, stanoví jejich součet a průměr.

Proměnné **n**, **s** slouží pro počítání čísel a jejich součtu. Proto jsou nejdříve

vynulovány. Proměnná **c** slouží pro načtení čísla a proměnná **f** zajišťuje přístup k souboru.

Soubor je nejdříve otevřen voláním funkce **fopen** do vstupního textového režimu.

Čísla ze souboru budeme číst tak dlouho, dokud nebude dosaženo konce souboru. Připomeňme, že funkce **fscanf** vrátí 1 pro úspěšné načtení (jedné konverze). Při nalezení konce souboru vrací hodnotu **EOF**. Je tedy třeba zapsat volání funkce **fscanf** přímo jako podmínku příkazu **while**.

V těle cyklu pak za každé úspěšně načtené číslo zvýšíme obsah proměnné **n** o 1 a načtené číslo přičteme k obsahu proměnné **s**.

Po ukončení cyklu je zobrazen počet načtených čísel a pokud je počet nenulový, vypočítá a zobrazí se i aritmetický průměr.

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

int main(int argc, char *argv[])
{
  int n=0,s=0,c;
  FILE *f;

  f=fopen("CISLA.TXT","r");   <- otevření souboru

  if(f!=NULL)   <- test úspěšného otevření
  {
    while(fscanf(f,"%d",&c)==1)   <- čtení čísla spojené s testem konce
    {
      n++;
      s=s+c;
    }
    fclose(f);   <- zavření souboru

    printf("Pocet cisel: %d\n",n);
    printf("Soucet cisel: %d\n",s);
    if(n>0)
      printf("Aritmeticky prumer: %.1f\n",(float)s/n);
  }
  else
    printf("Soubor se nepodarilo otevrit\n");

  system("PAUSE");
  return 0;
}
```

Příklad PROG_09-03

Napište program, který uloží údaje osob uložené v poli prvků typu **TOsoba** (viz kapitolu 8.4) do binárního souboru **OSOBY.BIN**.

Nejdříve deklarujeme pole prvků typu **TOsoba** s pěti položkami nazvané **Udaje**. Pro jednoduchost je provedena inicializace přímo při deklaraci.

Následně se voláním funkce **fopen** otevře soubor **OSOBY.BIN** pro zápis v binárním režimu.