

Práce se soubory

Jiří Zacpal



DEPARTMENT OF COMPUTER SCIENCE
PALACKÝ UNIVERSITY, OLOMOUC

KMI/ZP2 Základy programování 2

2. úkol



Napište funkce pro práci s množinami. Pro reprezentaci množin použijte vhodný odvozený datový typ **mnozina** definovaný pomocí typedef (pole, struktura, ...).

Pro takto definované množiny napište funkce:

int vloz_prvek(mnozina M, int i) – Tato funkce vloží prvek i do množiny M.

int odeber_prvek(mnozina M, int i) – Tato funkce odeber prvek i z množiny M.

mnozina prunik(mnozina A, mnozina B) – Tato funkce vrátí průnik množin A a B.

mnozina sjednoceni(mnozina A, mnozina B) – Tato funkce vrátí sjednocení množin A a B.

mnozina rozdil(mnozina A, mnozina B) – Tato funkce vrátí rozdíl množin A a B.

Dále vytvořte funkci:

mnozina spocitej(char* r, ...);

která vrátí množinu, jež vznikne aplikací operací, které jsou zadány v řetězci r na množiny, které jsou zadány jako proměnný počet parametrů této funkce. V řetězci r se mohou vyskytovat tyto znaky:

- p – pro průnik dvou množin
- s – pro sjednocení dvou množin
- r – pro rozdíl dvou množin.

2. úkol



Příklad spuštěného programu:

```
C:\WINDOWS\system32\cmd.exe
Mnozina A:
5, 12, 8, 7, 14, 20,
Mnozina A po odebrani prvku 20:
5, 12, 8, 7, 14,
Pokus o odebrani prvku 4:
Prvek neni v mnozine.
Mnozina B:
5, 14, 9, 7, 19,
Sjednoceni mnozin A a B:
5, 12, 8, 7, 14, 9, 19,
Prunik mnozin A a B:
5, 7, 14,
Rozdil mnozin A a B:
12, 8,
A prunik B sjednoceno B rozdil A:
9, 19,
```

- soubor je tvořen posloupností bytů uložených v určitých blocích na médiu (např. disku), obecněji hovoříme o datových proudech (anglicky stream)
- o konkrétní strukturu souborů a přístup k nim (čtení, zápis) se stará operační systém
- přístup k periférním zařízením je mnohem pomalejší než k paměti, proto se při přístupu k souborům používá tzv. **buffering**
 - **buffering vstupu** – do paměti (buffer) se načte blok dat určité velikosti, jednotlivé položky se čtou již z paměti; pokud je třeba načte se další blok dat
 - **buffering výstupu** – data se zapisují do paměti (buffer), pokud je buffer plný, zapíše se jeho obsah na disk

Začátek práce se souborem

- funkce, konstanty a struktury používané při práci se soubory jsou definovány v `stdio.h`
- deklarace „ukazatele na datový proud“:
`FILE *ident;`
- otevření souboru:
`FILE *fopen(char* filename, char* mode);`
- pokud při otevírání souboru dojde k chybě, vrací funkce hodnotu NULL
- textový řetězec mode může nabývat následujících hodnot: "r", "w", "a", "r+", "w+", "a+", "rt", "wt", "at", "r+t", "w+t", "a+t", "rb", "wb", "ab", "r+b", "w+b", "a+b".

Módy otevření souboru



- soubory mohou být otvírány jako binární (znak b) nebo textové (znak t nebo žádný ze znaků b, t)
- "r" – otevření textového souboru pro čtení
- "w" – otevření textového souboru pro zápis
- "a" – otevření textového souboru pro zápis na konec
- "r+", "w+", "a+" módy umožňují čtení i zápis

| požadavek \ mód otevření | "r" | "w" | "a" | "r+" | "w+" | "a+" |
|-----------------------------------|-----|-----|-----|------|------|------|
| soubor musí existovat | * | | | * | | |
| existující soubor bude vymazán | | * | | | * | |
| existující soubor bude rozšiřován | | | * | | | * |
| neexistující soubor bude založen | | * | * | | * | * |
| data lze odkudkoli číst | * | | * | * | * | * |
| data lze kamkoli zapisovat | | * | | * | * | |
| data lze zapisovat jen na konec | | | * | | | * |

Čtení z textového souboru / proudu



- čtení jednoho znaku

```
int getc(FILE *);  
(obdoba funkce int getchar();)
```

- čtení řádky

```
char *fgets(char* str, int max, FILE* f);  
(obdoba funkce char *gets(char* str);)
```

Funkce načte do textového řetězce str z proudu f nejvýše jeden řádek (včetně znaku '**\n**') a nejvíce

max-1 znaků (max je tedy délka řetězce včetně '**\0**').

- formátované čtení

```
int fscanf(FILE *f, const char* format, ...);  
(obdoba int scanf(const char* format, ...);)
```

- pokud již nelze z proudu číst (např. přečetli jsme data až do konce souboru), vrací funkce hodnotu EOF nebo NULL

Zápis do textového souboru / proudu

- zápis jednoho znaku

```
int putc(int , FILE* );  
(obdoba funkce int putchar(int);)
```

- zápis textového řetězce

```
int fputs(const char *, FILE *);  
(obdoba funkce int puts(const char *);)
```

- formátovaný zápis

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

- pokud dojde při zápisu k chybě, vrátí funkce hodnotu EOF, jinak se vrátí nezáporné číslo (funkce fprintf vrátí počet zapsaných znaků)

Uzavření souboru

- počet současně otevřených souborů je omezen
- pokud v programu dojde k chybě, data zapsaná pouze do bufferu budou pravděpodobně ztracena
- funkce pro vyprázdnění bufferu
`int fflush(FILE *);`
- funkce pro vyprázdnění bufferu a zavření proudu
`int fclose(FILE *);`
- v případě neúspěchu operace vracejí obě funkce konstantu EOF

Příklad 1



Napište v jazyku C funkci

```
int soucty(const char *vstup, const char *vystup),
```

která čte ze vstupního souboru vstup desetinná čísla, počítá součty na jednotlivých řádcích a zapisuje je do výstupního souboru vystup. Na konec výstupního souboru pak navíc vloží sumu všech čísel ve vstupním souboru.

Příklad vstupního souboru:

```
7.134 0.5198 2.436 0.9626 1.27 1.324 0.9639 1.538 0.4995 1.503 4.95 0.3466 0.454  
2.367 0.6877 9.057 0.1807 1.112 4.287 8.675 1.511 0.4296 0.2331
```

Příklad výstupního souboru:

```
11.0524 5.5954 6.7996 13.8584 15.1357 Suma: 52.4415
```

Příklad 1



```
soucty(const char *vstup, const char *vystup)
{
    FILE *in, *out;
    char znak;
    float suma = 0, suma_r = 0, cislo = 0;
    if((in = fopen(vstup, "rt")) == NULL)
        return -1;
    out = fopen(vystup, "wt");
    znak = ' ';
    while(znak != EOF)
    {
        while(((znak = getc(in)) != '\n') && (znak != EOF))
        {
            ungetc(znak, in);
            fscanf(in, "%f", &cislo);
            suma_r += cislo;
            while((znak = getc(in)) == ' ');
            ungetc(znak, in);
        }
        fprintf(out, "%f\n", suma_r);
        suma += suma_r;
        suma_r = 0;
    }
    fprintf(out, "Suma: %f", suma);
    fclose(in);
    fclose(out);
}
```

Standardní proudy



- načítání vstupů z klávesnice a výpisy na obrazovku jsou ve skutečnosti také realizovány pomocí datových proudů
- v `stdio.h` jsou definovány ukazatele

```
FILE *stdin;  
FILE *stdout;  
FILE *stderr;
```
- tyto proudy jsou otevřeny již při spuštění programu
- proud `stdin` je otevřen pro čtení (implicitně z klávesnice), proudy `stdout` (standardní výstup) a `stderr` (chybový výstup) pro zápis (oba na obrazovku)
- s těmito proudy lze pracovat stejně jako s proudy odpovídajícími souborům
- implicitní nastavení proudů lze změnit přesměrováním

```
program.exe > vystup.txt
```

Možnosti formátovaného I/O 1/4

- řídicí řetězec formátovaného I/O může obsahovat části pro zápis (resp. čtení) hodnot proměnných
`%[příznaky][šířka][.přesnost][modifikátor]konverze`
- části uvedené v hranatých závorkách jsou nepovinné
- možnosti části *konverze*
 - d nebo i = desítkové číslo typu `signed int`
 - u = desítkové číslo typu `unsigned int`
 - o = osmičové číslo typu `unsigned int`
 - x nebo X = šestnáctkové číslo typu `unsigned int`
- f = desetinné číslo typu `float` (při zápisu i `double`)
 - e nebo E = semilogaritmické číslo typu `float` (`double`)
- g nebo G = des. nebo sem. číslo typu `float` (`double`)

Možnosti formátovaného I/O 2/4

- možnosti části *konverze* (pokračování)
 - c = znak (typ **char**)
 - s = textový řetězec (typ **char ***)
 - p = adresa (libovolný ukazatel)
 - % = výpis znaku procento
- část *modifikátor* (mění velikost typu *konverze*)
 - h modifikuje d a i na typ **signed short int**,
u, o, x a X na typ **unsigned short int**
 - l modifikuje d a i na typ **signed long int**,
u, o, x a X na typ **unsigned long int**
a f na typ **double** (pouze při načítání)
 - L modifikuje f, e, E, g a G na typ **long double**

- možnosti části *šířka*
 - n – tiskne vždy alespoň n znaků; má-li výstupní hodnota méně než n znaků, doplňují se zleva mezery
 - $*$ – jako n , ale počet tištěných znaků je dán předchozím parametrem
 - $0n$ – jako n , ale místo mezer se doplňují nuly
 - 0^* – jako $*$, ale místo mezer se doplňují nuly
- možnosti části *přesnost*
 - pro konverze i, d, u, o, x, X, s se chová jako *šířka*
 - pro f, e, E nastavuje maximální počet cifer za desetinnou tečkou
 - pro g, G nastavuje maximální počet významových cifer

Možnosti formátovaného I/O 4/4



- možnosti části *přesnost* (pokračování)
 - *.n* – tiskne *n* znaků, desetinných míst, významových cifer
 - *.* – stejně jako *.0*
 - *.** – jako *.n*, ale počet tiště znaků, desetinných míst nebo významových cifer je dán předchozím parametrem
- možnosti části *příznak*
 - *minus* – zarovná se doleva, zprava se doplňují mezery
 - *plus* – číslo bude vždy vytištěno se znaménkem
 - *nic* – nezáporné číslo bude vytištěno bez znaménka
 - *mezera* – nezáporné číslo bude vytištěno bez znaménka, ale na místo znaménka se dodá mezera navíc
 - *#* – *o*, *x*, *X* – číslo zapisuje jako konstantu jazyka C
f, *e*, *E*, *g*, *G* – vždy zobrazuje desetinnou tečku

Testování při práci s proudy

- test konce souboru
 - pomocí konstanty EOF (def. v `stdio.h`, většinou -1)
 - pomocí makra `feof(stream)`, které se přepíše na nenulový výraz, pokud poslední čtení proudu *stream* bylo ukončeno na konci souboru
- test konce řádku
 - přímo v textových souborech jsou řádky označeny v závislosti na použitém kódování (znak 0xD, znak 0xA, dvojice znaků 0xD a 0xA)
 - při otevření v textovém režimu jsou ovšem všechny možné konce řádek nahrazeny jediným znakem `'\n'`
- test při otevření či uzavření souboru
 - `fopen` vrací při chybě NULL, `fclose` konstantu EOF

Vrácení znaku do bufferu

- v reálných aplikacích se často dozvídáme o tom, že máme přestat číst až ve chvíli, kdy jsme přečetli jeden znak navíc
- tento znak samozřejmě nemůžeme zahodit (či si ho někde složitě pamatovat)
- funkce pro vrácení znaku do bufferu
`int ungetc(int, FILE *);`
- funkce vrací při úspěšném vrácení znaků tento znak, v případě neúspěchu hodnotu EOF
- doporučuje se vracet pouze jeden znak
- lze „vrátit“ i jiný znak než ten, který se četl

Bodovaný úkol



Napište v jazyku C funkci

```
zlomek soucet(const char *vstup),
```

která čte ze vstupního textového souboru vstup zlomky (resp. dvojice celých čísel), vypočítává a vrací součet všech zlomků zapsaných ve vstupním souboru. Pro snadnější práci se zlomky si definujte strukturovaný typ zlomek. Výsledný zlomek by měl být upraven do základního tvaru. Vykrácení zlomku do základního tvaru docílíte vydělením čitatele i jmenovatele jejich největším společným dělitelem, který můžete určit například použitím **Euklidova algoritmu**.

Bodovaný úkol



```
D:\dokumenty\SkyDrive\dokumenty_aktualni\vyuka_aktualni\LS\zp2_zaklady_programovani\bodovane_u...  
1. zlomek: 2/3  
2. zlomek: -1/6  
3. zlomek: 3/4  
4. zlomek: -3/8  
  
Soucet zlomku je 7/8  
Press any key to continue . . .
```