Práce se soubory

Minule jsme začali s tématem práce se soubory, ale zaměřili jsme se hlavně na práci s textovými soubory. Teď se budeme více věnovat těm binárním.

1 Binární soubor

Textové soubory mají hlavní výhodu v tom, že jsou člověkem čitelné. Naproti tomu binární soubory, když je otevřeme v libovolném textovém editoru, tak nevidíme žádnou smysluplnou informaci, ale jen změť dat. Binární soubor může být vnitřně libovolným způsobem organizován a při zběžném pohledu nedokážeme nevíme, co máme číst. Tyto soubory nejen, že nejdou běžným editorem číst, ale ani opravovat, nebo vytvářet.

Přes tyto nevýhody se binární soubory používají v praxi často. Nejběžnější využití je při ukládání velkých dat. Pro uchování stejně velkého množství informace je v textových souborech potřeba mnohem více prostoru. Příkladem mohou být například čísla. Číslo 65535 zabere v textovém souboru prostor 5 bytů, zatímco v binárním stačí byty 2.

Další výhodou binárních souborů je rychlejší práce s nimi. Důvodem je jednak velikost souborů a také nutnost konverze čísla na text v případě textových řetězců, což je pomalá operace. Tato konverze v binárních souborech odpadá, protože se do nich zapisuje přímo obsah paměti po bytech.

Je zřejmé, že pro ukládání textů (znaků) nemá využití binárních souborů takovou výhodu, prtože znaky v paměti zabírají jeden byte, stejně jako v textovém souboru.

Malá poznámka k binárním souborům. Tyto soubory nejsou stoprocentně přenositelné mezi různými operačními systémy. Záleží na délce jednotlivých základních typů (kolik místa zabírá například int v paměti), což může být pro různé systémy různé.

2 Otevření a zavření souboru

Jazyk C nerozlišuje mezi binárními a textovými soubory. Rozlišuje **binární a textový režim otevření** souboru. Což je rozdíl. V jazyce C je možné otevřít textový soubor jako binární a binární jako textový. Dokonce to má někdy i opodstatnění. Například otevření textového souboru jako binárního může být využito při komprimování.

Jak bylo řečeno minule, chceme-li se souborem pracovat, je potřeba ho otevřít pomocí funkce `fopen`, která bere jako argumenty název souboru a jakým stylem budeme se souborem pracovat. Pro práci s binárními soubory to jsou "rb"pro čtení, "wb"pro zápis.

```
// otevreni souboru pro cteni
fr = fopen("soubor.txt", "rb");

// otevreni souboru pro zapis
fw = fopen("soubor.txt", "wb");
```

V UNIX prostředí není rozdíl mezi textovým a binárním přístupem k souboru z hlediska zpracování. Přesto, kvůli přenositelnosti, je dobré příznak b nebo t používat.

Vše ostatní (včetně např. ošetřování chyb) je stejné.

3 Vstupy a výstupy

S daty v souboru lze pracovat dvěma způsoby.

Prvním způsobem je **formátovaně**. Při tomto zpracování čtecí, případně zapisovací funkce ovlivňují data (jsou jinak reprezentovaná v souboru, než v paměti). Typickými funkcemi pro formátovou práci se soubory jsou funkce `fprintf()` a `fscanf()`. Formátované vstupy a výstupy jsou většinou využívány pro práci s textovými soubory otevřenými v textovém režimu. V binárním případě nemají smysl (i když se použít dají).

Druhým způsobem je **neformátovaně**. Zde jsou data v souboru reprezentovány stejně jako v paměti počítače. Vstupy a výstupy se dále rozlišují podle toho, kolik dat najednou zpracovávají.

Najednou můžeme číst/zapisovat buď jeden znak = byte (toho se využívá jak v textových souborech, tak binárních), jeden řádek (smysl má jen v textových souborech) nebo jeden blok dat (má smysl v binárních souborech).

3.1 Zpracování jednoho znaku

K tomu slouží příkazy `getc()`, `putc()`. Viz. minulý seminář.

Při otevření binárního souboru v textovém režimu můžeme narazit na problémy s některými hodnotami bytů. Jedním z příkladů takových bytů je byte s hodnotou 255 - tento byte může být za nevhodných okolností považován za konec souboru (pro signed char je 255 rovno -1 a tudíž rovno konstantě EOF).

3.2 Zpracování jednoho řádku

Viz. minulý seminář.

3.3 Zpracování jednoho bloku

Blok si můžeme představit jako pole o určitém počtu prvků určité velikosti. Nejjednodušším případem jsou prvky typu char. Za blokem jednoho typu může následovat blok jiné délky složený z prvků jiného typu. Tento způsob se však často nepoužívá. Nejčastěji jsou prvky v souboru jednoho typu.

Pro čtení a zápis se využívají funkce `fread()` a `fwrite()`.

```
// cteni bloku
int fread(void *kam, size_t rozmer, size_t pocet, FILE *fr);

// zapis
int fwrite(void *odkud, size_t rozmer, size_t pocet, FILE *fw);
```

Prvním argumentem `fread()` je ukazatel `kam`, který ukazuje na začátek místa v paměti, kam se budou data ze souboru načítat. `rozmer`, je velikost položky v bytech, kterou načítáme (může to být např. sizeof(typ)). Třetím argumentem je `pocet`, představující počet položek, které čteme. Celkem se tedy přečteme `pocet * rozmer` bytů. Posledním argumentem je datový proud `fr`, ze kterého data čteme.

Obdobně funkce `fwrite()` zapíše `pocet` prvků velikosti `rozmer` do souboru `fw` z paměti kam ukazuje pointer `odkud`.

Obě funkce vrací počet skutečně přečtených, nebo zapsaných položek. Ten se liší od parametru `pocet`, pokud jsme při čtení došli na konec souboru nebo pokud došlo k chybě. To lze zjistit pomocí funkcí `feof()` a `ferror()`.

Funkce `ferror()` zda nedošlo k chybě (obdobně jako `feof()` testuje konec souboru). Pokud nedošlo k chybě, vrací 0, jinak kladnou hodnotu.

Příklad použití `fread()`:

```
#define VELIKOST_BLOKU 10

int data[VELIKOST_BLOKU];
FILE *fr = fopen("soubor.dat", "rb");
fread(data, sizeof(int), VELIKOST_BLOKU, fr);
```

Příklad použití `fwrite()`:

```
#define VELIKOST_BLOKU 10

int data[VELIKOST_BLOKU];
FILE *fw = fopen("soubor.dat", "wb");

fwrite(data, sizeof(int), VELIKOST_BLOKU, fw);
```

4 Práce se souborem

Při otevření souboru pro čtení, ukazuje souborový indikátor pozice (kurzor) na začátek souboru. Pokud použijete libovolnou funkci pro čtení jednoho bytu, posune se ukazatel na další byte. Tímto způsobem se čte soubor od začátku do konce byte za bytem. Pokud bychom chtěli číst soubor zase od začátku, tak vás určitě napadne, že ho můžeme otevřít znovu. To je sice možnost, ale je to pomalé a nepraktické. Nebo, co když budeme chtít přečíst jen pár bytů na konci? Kvůli tomu bychom museli projít celý soubor.

Při otevření souboru pro zápis zase ukazuje kurzor na konec souboru a problém je, jak řešit situaci, kdybychom chtěli zapisovat doprostřed.

C má nástroje na nastavování kurzoru na požadovanou pozici v souboru. Jednou z funkcí je funkce `fseek()`.

```
int fseek(FILE *f, long posun, int odkud);
```

Tato funkce pracuje se souborovým streamem `f` a posouvá kurzor o `posun` od místa `odkud`. `posun` může být i záporná hodnota. Funkce vrací v případě úspěchu nulu.

Speciální konstanty, které můžeme použít jako parametr `posun` jsou:

- `SEEK_SET` - značí relativní vzdálenost od začátku souboru

- `SEEK_CUR` - značí relativní vzdálenost od aktuální pozice
- `SEEK_END` - značí relativní vzdálenost od konce souboru

Další používané funkce jsou `ftell()` a `rewind()`.

```
long ftell(FILE *f);
void rewind(FILE *f);
```

Funkce `ftell()` vrací aktuální pozici kurzoru v souboru `f` (posunu od začátku souboru v bytech). Funkce `rewind()` posune kurzor na začátek souboru `f`. Je to totéž, jako `fseek(f, 0, SEEK_SET)`.

4.1 Příklad 1

Praktickým příkladem, kdy použít funkci `ftell()` je pro zjištění velikosti souboru.

To je možné udělat tak, že kurzor posuneme na konec souboru a zjistíme jeho vzdálenost od začátku.

```
#include <stdio.h>

int main(void)
{
    FILE *soubor;
    size_t i;
    char pole[100];
    soubor = fopen("priklad1.txt", "fb");

    // posun na konec souboru
    fseek(soubor, 0L, SEEK_END);

    // %lu  znaci, ze vystupem bude unsigned long cislo
    printf("Velikost souboru je %lu bytu.\n", ftell(soubor));

    fclose(soubor);
    return 0;
}
```

5 Režimy otevření souboru

Zatím jsme zmínili jen 4 režimy otevření souboru a to "wt", "rt" (nebo jen "w", "r") pro textový režim pro zápis / čtení a "wb", "rb" pro binární pro zápis / čtení.

Režimů je však více:

- "a" otevření souboru pro přepisování (pokud neexistuje, je vytvořen).
- "r+" otevření existujícího souboru pro čtení i zápis. Pokud soubor neexistuje chyba.
- "w+" vytvoření nového souboru pro čtení i zápis. Pokud soubor existuje, je jeho obsah vymazán.
- "a+" otevření souboru pro čtení a zápis na konec (nedá se zapisovat jinam než na konec).

Na konec můžeme přidat `t` nebo `b` pokud chceme pracovat s textovými nebo binárními soubory (pokud nedodáme nic, soubor se otevře v textovém režimu).

6 Cvičení

1. V Příkladu 1 nejsou ošetřené žádné chyby (správnost otevření/uzavření souboru a pod.). Dopište ošetření chyb.
2. Napište program, který uloží čísla od 0 do 9 do binárního i textového souboru. Zjistěte velikosti těchto souborů obdobně, jako v předchozím příkladu. Otevře si tyto soubory v textovém editoru.
3. Napište program, který uloží číslo od 1234567 do binárního i textového souboru. Zjistěte velikosti těchto souborů obdobně, jako v předchozím příkladu. Proč je v minulém případě velikost binárního souboru větší a v tomto ne?
4. Napište program, který do binárního souboru uloží pole desetinných čísel (záleží na vás, jak velké).
5. Napište program, který z desetinných čísel uložených v binárním souboru (například vytvořeného v předchozí úloze) spočítá průměr. Měl by být napsán obecně, aby fungoval i když nebudeme vědět, kolik je v souboru čísel.