DÚ 4 – Skupiny 06 a 23

Termín odevzdání DÚ (prvního pokusu) je **NEDĚLE 22. 12. 23:59**.

V posledním domácím úkolu půjde hlavně o procvičení práci se soubory. Vytvoříme program, který bude fungovat jako jednoduchá evidence pohybů na bankovním účtu. Výsledný program bude velice podobný přikladu z přednášky, který jsem vám nahrál do studijních materiálů k tomuto zadání, doporučuji vám z něj vycházet. Podívejte se také na program `serializace.c`, který máte v materiálech ze cvičení.

K samotnému programu:

Program bude evidovat pohyby na bankovním účtu uživatele. Typ platby (příchozí/odchozí) může být určen pouze znamínkem u částky (-1000 je odchozí, 254.3 je příchozí). Ke každé platbě bude evidováno datum uskutečnění platby (např. můžete zvlášť ukládat den, měsíc a rok do jednotlivých celočíselných proměnných, ale nebráním se i jiným řešením) a její **víceslovný** popis. Můžete použít následující strukturu:

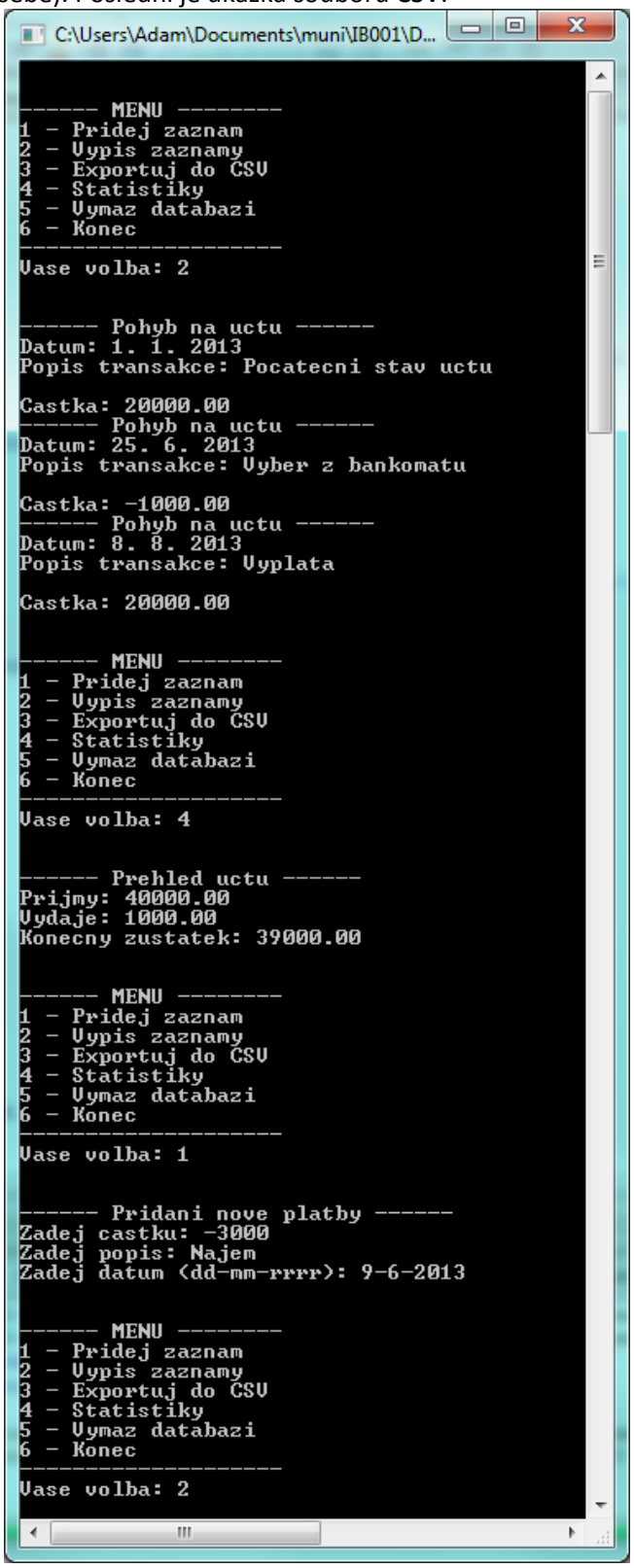
```
typedef struct
{
    float castka;
    unsigned short den;
    unsigned short mesic;
    unsigned short rok;
    char popis[100];
} pohyb;
```

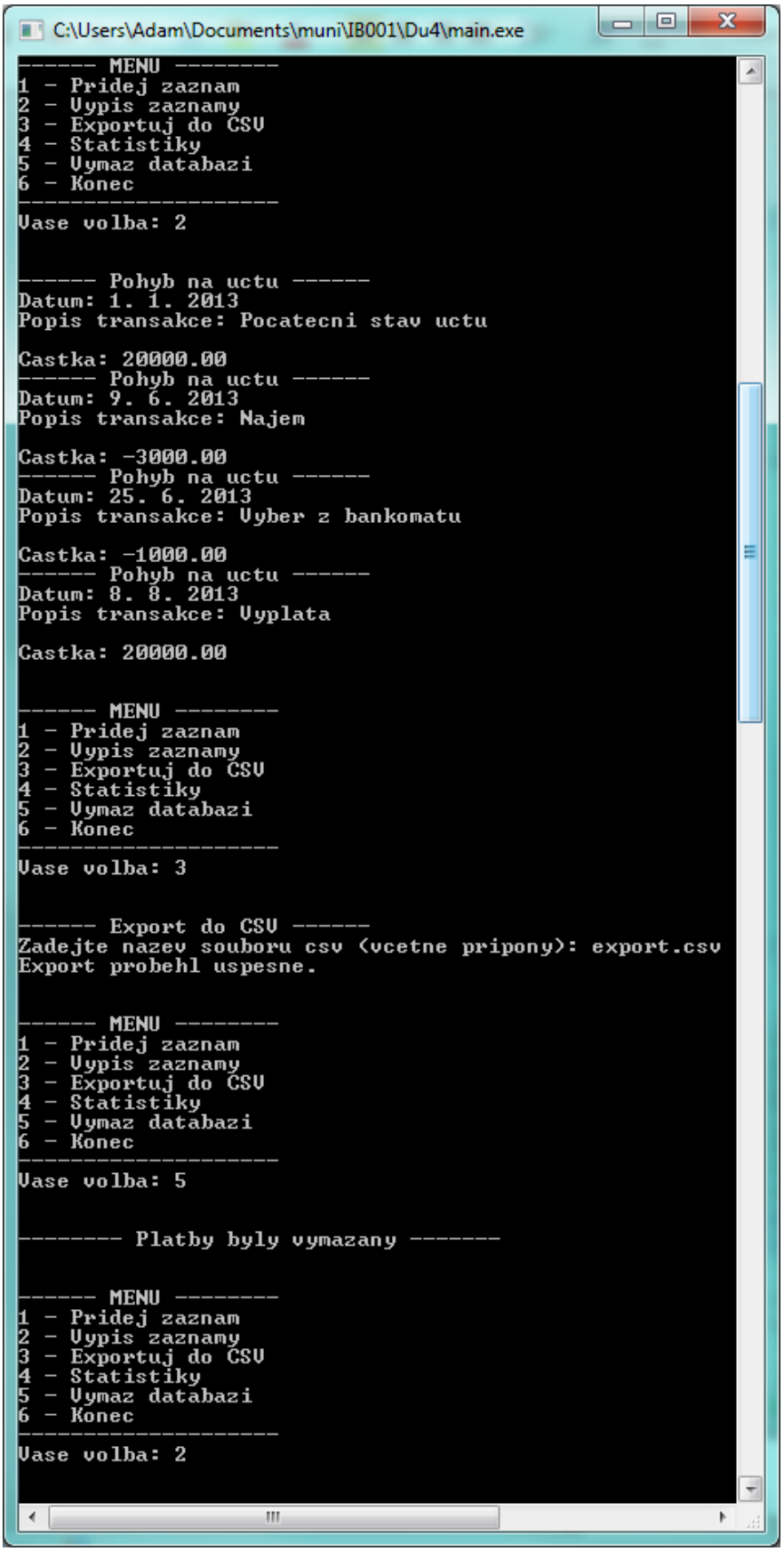
Program bude ukládat databázi plateb do **binárního souboru** tak, aby i po ukončení a novém spuštění programu byl schopný načíst platby, zadané při minulém spuštění. Program bude podporovat následující funkce:

- Vypsání existujících plateb - Obdoba funkce `vypis` v příkladu z přednášky.
- Vymazání databáze plateb - Obdoba funkce `nova` v příkladu z přednášky.
- Souhrnné informace o účtu – Celkem přijatých plateb, Celkem odeslaných plateb, Konečný zůstatek (počítáme s tím, že před první evidovanou platbou byl zůstatek 0)
- Export plateb do CSV souboru – Obdoba funkce `exportuj` v příkladu z přednášky. Export pro zpracování v dalších nástrojích (např Excel). Výsledkem bude **textový soubor**, kde bude na každém řádku jedna platba, jednotlivé položky budou odděleny čárkou nebo středníkem. Např.:
  ```
  1. 1. 2013;1000.00;Platba 1
  8. 8. 2013;-1000.00;Platba 2
  9. 9. 2013;1400.00;Platba 3
  12. 12. 2013;-200.00;Platba 4
  ```
- Přidání nového záznamu – Přidání záznamu do souboru. Uznám i přidání záznamu na konec, volitelně můžete naprogramovat vkládání záznamu na správnou pozici v souboru (tzn. za všechny platby, které byly pořízeny ve dnech před provedením nové platby, a také za ty, které byly provedeny ve stejném dni, ale v souboru už jsou). Zkuste to hlavně ve vlastním zájmu, ale pokud budete mít rozšířenou verzi, je pravděpodobné, že v případě nějaké jiné drobné chyby v programu přimhouřím oko a nebudu vás nutit do opravy.

- Program na začátku zobrazí uživateli menu, ze kterého bude moci zvolit akci, po jejím dokončení se menu zobrazí znova. Takhle bude program fungovat tak dlouho, dokud uživatel nerozhodne o jeho ukončení (viz příklad z přednášky).
- Program bude korektně pracovat se souborem, to znamená hlavně to, že při svém ukončení soubor zavře.

Na dalších stránkách najdete výpisy programu (nevešlo se to do jednoho okna, ale navazuje to na sebe). Poslední je ukázka souboru CSV:



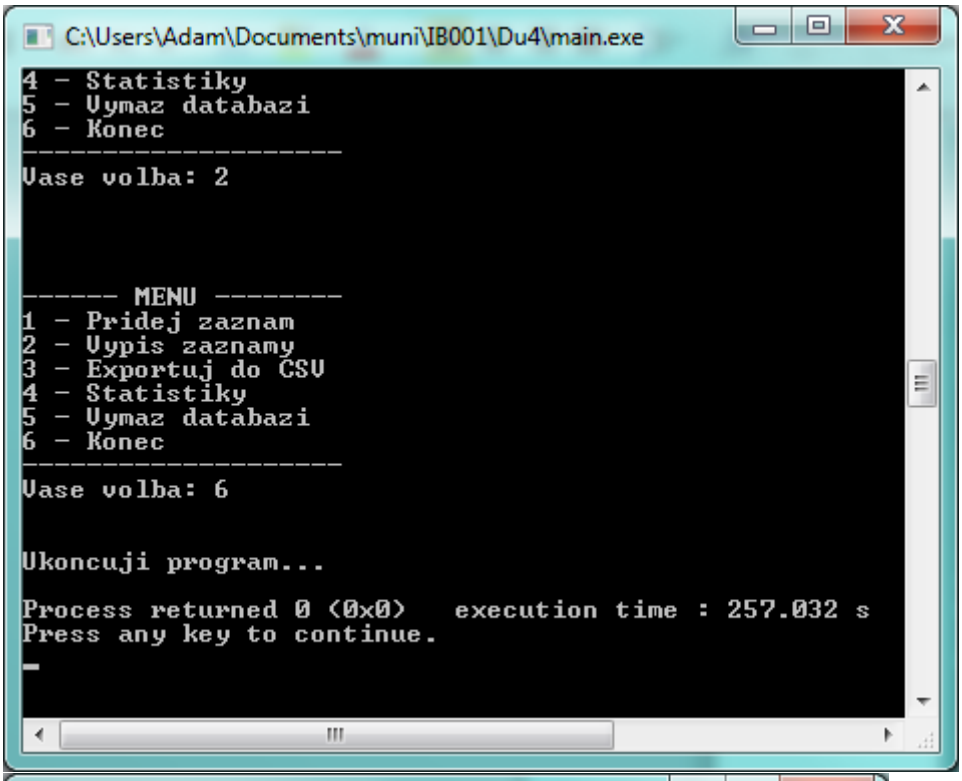

C:\Users\Adam\Documents\muni\IB001\Du4\main.exe
------ MENU --------
1 - Pridej zaznam
2 - Vypis zaznamy
3 - Exportuj do CSV
4 - Statistiky
5 - Vymaz databazi
6 - Konec

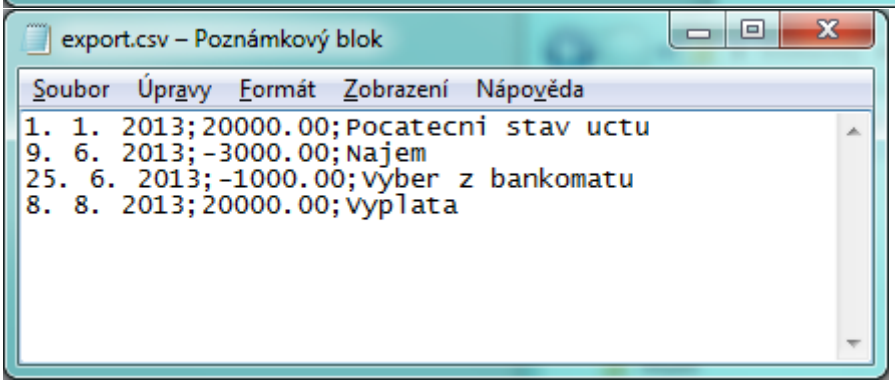
Vase volba: 2
------ Pohyb na uctu ------
Datum: 1. 1. 2013
Popis transakce: Pocatecni stav uctu
Castka: 20000.00
------ Pohyb na uctu ------
Datum: 9. 6. 2013
Popis transakce: Najem
Castka: -3000.00
------ Pohyb na uctu ------
Datum: 25. 6. 2013
Popis transakce: Vyber z bankomatu
Castka: -1000.00
------ Pohyb na uctu ------
Datum: 8. 8. 2013
Popis transakce: Vyplata
Castka: 20000.00
------ MENU --------
1 - Pridej zaznam
2 - Vypis zaznamy
3 - Exportuj do CSV
4 - Statistiky
5 - Vymaz databazi
6 - Konec

Vase volba: 3
------ Export do CSV ------
Zadejte nazev souboru csv (vcetne pripony): export.csv
Export probehl uspesne.
------ MENU --------
1 - Pridej zaznam
2 - Vypis zaznamy
3 - Exportuj do CSV
4 - Statistiky
5 - Vymaz databazi
6 - Konec

Vase volba: 5
-------- Platby byly vymazany --------
------ MENU --------
1 - Pridej zaznam
2 - Vypis zaznamy
3 - Exportuj do CSV
4 - Statistiky
5 - Vymaz databazi
6 - Konec

Vase volba: 2





Ukázka exportu:

Tohle bylo celé zadání, dál už jsou různé rady a podobně.

Následuje popis funkcí, proměnných a postupů pro práci se soubory, které budete pro vypracování úkolu určitě potřebovat.

Otevření souboru: pomocí funkce `fopen`. Tato funkce vrací ukzatel na strukturu `FILE`, kterou potom používáme při práci se souborem. První parametr je název souboru. Pokud neuvedete plnou cestu, soubor se vytvoří nebo se bude hledat ve složce, odkud program spouštíte (typicky `adresarProjektu\bin\Debug` nebo `Release`). Druhý parametr je typ přístupu. Základní tři:

- `w` – otevření souboru pro zápis. Pokud soubor už existuje, přepíše se **vždy** novým (prázdným) souborem
- `r` – otevření souboru pro čtení. Soubor musí existovat.
- `a` – otevření souboru pro zápis tak, že je možné zapisovat do něj (tzn. **nepřepíše** se novým). Pokud soubor neexistuje, vytvoří se.

Za každý specifikátor je možné přidat +, což značí, že je možné provádět i „to druhé“. Takže takže `r+` i `w+` obojí dovoluje čtení i zápis, ale `r+` nefunguje, pokud soubor neexistuje (nevytvoří nový). Nakonec můžeme připsat ještě `b`, což značí práci v binárním režimu (tzn. chceme používat funkce `fwrite` a `fread`). Příklady:

```
/* otevření souboru data.dat v binárním režimu, soubor umožňuje zápis i
čtení, nepřepisuje původní soubor) */
FILE* f = fopen("data.dat", "a+b");
/* vytvoření nového souboru data.tmp (případně přepsání existujícího) a
otevření pouze pro zápis v binárním režimu*/
FILE* tmp = fopen("data.tmp", "wb");
```

Zavření souboru: pomocí funkce fclose. Na tento příkaz nesmíme zapomínat. Zavření souboru provádíme samozřejmě až potom, co už z něj nechceme nic číst ani do něj nic zapisovat. Příklad:

```
fclose(f);
fclose(tmp);
```

Zápis a čtení v binárním režimu: Používáme funkce `fwrite` a `fread`, obě s následujícími parametry:

- Ukazatel na místo v paměti, kam (odkud) chci data ukládat (brát)
- Velikost datového typu
- Počet prvků, které chci zapisovat nebo číst
- Ukazatel na strukturu `FILE`

Obě funkce vrací počet prvků, které se povedlo zapsat/načíst. Příklad:

```
int pole[10] = {0,1,2,3,4,5,6,7,8,9};
/* zápis deseti prvků typu int do souboru tmp (viz výše). Jako zdroj slouží
pole (pamatujme, že název pole je ukazatel na jeho první prvek */
fwrite(pole, sizeof(int), 10, tmp);
/* načítání celočíselných hodnot ze souboru f „dokud je co číst“ – fread
vrací počet přečtených prvků, 0 znamená, že jsme na konci. Načítáme do
proměnné pom, funkci musíme předat její adresu */
int suma = 0;
int pom;
while(fread(&pom, sizeof(int), 1, f) == 1)
      suma+= pom;
```

Zápis a čtení v textovém režimu: Soubor musí být otevřen **bez** specifikátoru b. Používáme funkce fscanf a fprintf, fgets, fputs. Jedná se o obdoby již známých funkcí, jedním z parametrů je vždy ukazatel na soubor. Příklady:

```
FILE* souborTxt = fopen("cislo.txt", "w+");
// parametry: kam, jak, co
fprintf(souborTxt, "%.2f %.2f ", 254.2, 847.5);
// posun na začátek souboru
Rewind(souborTxt);
while(!feof(souborTxt))
{
  fscanf(souborTxt, "%f", &c);
  printf("%f\n", c);
}
```

Další pomocné funkce: `rewind(f)` přesouvá ukazatel na začátek souboru, `feof(f)` zjišťuje, jestli už je ukazatel na konci souboru, `fflush(f)` vyprazdňuje vyrovnávací paměť – hodí se, když do souboru zapisujeme a potom ho chceme číst (něco by mohlo teoreticky zůstat jenom ve vyrovnávací paměti a ještě se nepřenést do souboru). Prakticky ho voláme na začátku každé funkce. Zbytek funkcí najdete popsané v přednášce, podívejte se také na studijní materiály.

Teď už další poznámky, které se netýkají souborů.

Pro načítání a výpis typu `unsigned short` (pokud se rozhodnete použít navrženou strukturu) je třeba použít specifikátor `%hu`.

Při načítání popisu použijte funkci `fgets`. Pokud byste použili `scanf`, nemohl by uživatel zadávat víceslovné komentáře k platbám. Problém je v tom, že pokud je před `fgets` příkaz `scanf`, `fgets` jako první načte konec řádku, co tam zbyl po předchozím čtení, a nenechá uživatele popis zadat. Použijeme tedy funkci `getchar`:

```
printf("Zadej castku: ");
scanf("%f", &(z.castka));
getchar();
printf("Zadej popis: ");
fgets(z.popis, 100, stdin);
```

Nejzrádnější částí úkolu je přidávání nového záznamu na správné místo (je to volitelné, v pořádku bude i přidání záznamu na konec). Musíme dát pozor na situace, kdy v databázi nic není, když ukládáme na konec nebo na začátek nebo když už v databázi platby ze stejného dne jsou. V každém případě bude třeba vytvořit nový soubor, do něj zkopírovat existující záznamy a přidat nový. Podobně v příkladu z přednášky funguje mazání osoby.

Budeme potřebovat mechanismus, jak porovnávat data. Je dobré vytvořit si nějakou funkci, třeba `int porovnejData(pohyb z1, pohyb z2)`, která bude vracet -1, pokud je první záznam menší, 0 pokud jsou stejné a 1 pokud je druhý menší. Logika porovnávání je podobná jako ve druhé písemce: Pokud se roky nerovnají, je to jasné. Pokud se rovnají, musíme se dívat na měsíce. Pokud se měsíce nerovnají, je to jasné. Pokud se měsíce rovnají, musíme do toho zapojit i dny.

Ohledně nalezení správného místa pro nový záznam: Obecně jsou dva způsoby, jak to udělat.

Buď načteme celý soubor do pole, přidáme záznam (buď na správné místo a vše za ním posuneme o jeden prvek dozadu, nebo prvek přidáme prostě na konec a poté pole setřídíme) a potom zase zapíšeme do souboru. V tomto případě (práce s polem v paměti) musíme správně vypočítat, kolik prvků bude pole mít. Můžeme použít velikost souboru a velikost datového typu pohyb:

```
// kouzelná formule pro zjištění velikosti souboru
fseek(soubor, 0L, SEEK_END);
long velikostSouboru = ftell(soubor);
long velikostZaznamu = sizeof(pohyb);
// návarat ukazatele na začátek souboru
fseek(fp, 0L, SEEK_SET);
long velikostPole = 0 // to už je na vás…
```

Řešení přes práci v poli je však paměťově neefektivní, protože musíme mít všechny záznamy naráz v operační paměti. Vhodnější je tedy druhý způsob, kdy kopíruji data z původního souboru do dnového tak dlouho, dokud nenarazím na místo, kam vložit nový záznam, potom vložit nový záznam a nakonec vložit zbytek z původního souboru. Tady se ale taky dočkáme jedné neintuitivní věci. Podívejme se na následující kód:

```
// vytvoření dočasného souboru
FILE* tmp = fopen("bilance.tmp", "wb");

// vyprázdnění vyrovnávací paměti
fflush(f);
```

```
// přesun ukazatele na začátek souboru
rewind(f);

pohyb novy = zadej(); // pridavany zaznam
pohyb z; // pomocna promenna pro uchovavani starych nacitanych zaznamu

// načtení plateb starších než přidávaná položka
while(fread(&z, sizeof(pohyb), 1, f) == 1 && porovnejData(z, novy) <= 0)
{
  fwrite(&z, sizeof(pohyb), 1, tmp);
}
// zápis nové položky
fwrite(&novy, sizeof(pohyb), 1, tmp);
// zápis zbytku původního souboru
while(fread(&z, sizeof(pohyb), 1, f) == 1)
        fwrite(&z, sizeof(pohyb), 1, tmp);
```

Mělo by to dělat to, že načítáme soubory z původního souboru a rovnou je kopírujeme do toho nového, dokud jsou data operací na účtu menší nebo rovna nové platbě. Poté zkopírujeme nový záznam na správné místo a pokračujeme v kopírování ostatních. Když to vyzkoušíte, zjistíte, že nový záznam nahradí jeden z již existujících (konkrétně první, který měl následovat po tom novém). Proč to? Funkce `fread` načte do proměnné `z` hodnotu ze souboru a poté její datum porovná s novým záznamem. Pokud zjistí, že datum v novém záznamu je menší, než datum posledního záznamu, zapíše nový záznam a poté pokračuje dál se zbytkem souboru. Jenže jeden záznam přeskočí, a to právě ten, u kterého přestala platit druhá polovina podmínky v prvním `while` cyklu. Máme ho načtený v proměnné `z`, příkaz pro zápis v těle prvního `while` cyklu se však neprovede (podmínka už neplatí) a ukazatel pozice v souboru už je za tímto záznamem. Můžeme zkusit za příkaz pro zápis nové hodnoty vložit: `fwrite(&z, sizeof(pohyb), 1, tmp);`
V tomto případě však program nebude fungovat, pokud vkládáme platbu do nového souboru. V takovém případě totiž `fread` nic do proměnné `z` nenačte, vrátí nulu, první `while` cyklus skončí a my se přesto pokusíme do souboru za nový záznam uložit obsah proměnné `z`, což budou nějaké podivné iniciální hodnoty.

Jedním ze způsobů, jak tyhle problémy řešit, je použít proměnnou pro ukládání výsledku příkazu `fread`, pak už se to dá vyřešit.