Zápočtové příklady na procvičení

Zde uvedené příklady jsou určené k procvičení před absolvováním zápočtového testu nebo jeho opravy. Zápočtový příklad ale může zkoušet vše, co jste se během semestru měli naučit a může se odlišovat od uvedených příkladů.

Tabulkovač

Váš program dostane jeden textový argument na příkazové řádce – název vstupního souboru. Tento soubor obsahuje předem neznámý počet řádků (UNIXový textový soubor, každý řádek končí znakem LF), a tyto řádky mají předem neznámou délku. Řádky jsou rozděleny určitým počtem čárek – tento počet je pro všechny řádky stejný. Mezi dvěma čárkami ale nemusí být žádný text. Řádky jsou řádky v tabulce, čárky oddělují jednotlivé sloupce. Vaším úkolem bude načíst položky tabulky do vhodné struktury, a následně ji na standardní výstup naformátovat tak, že každý sloupec má velikost nejdelší položky ve sloupci (s vhodným ohraničením buněk).

Příklad, vstup:

```
SKUPINA,,,
ovoce,jablko,hruška,slivovice
zelenina,mrkev,zelí,paprika
operační systém,DOS,Linux,BSD
```

http://www.fi.muni.cz/~xzarevuc/pb071/tabulka.txt [http://www.fi.muni.cz/~xzarevuc/pb071/tabulka.txt]

Výstup:

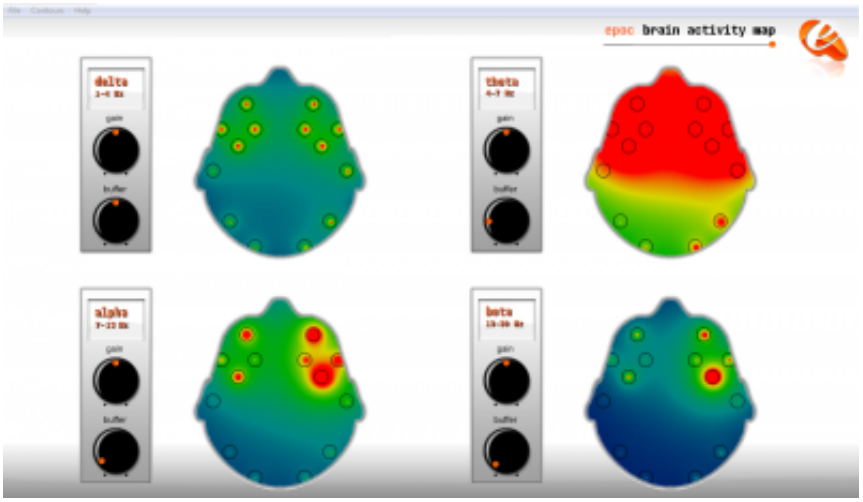
```
o-----------------o--------o--------o-----------o
| SKUPINA         |        |        |           |
o-----------------o--------o--------o-----------o
| ovoce           | jablko | hruška | slivovice |
o-----------------o--------o--------o-----------o
| zelenina        | mrkev  | zelí   | paprika   |
o-----------------o--------o--------o-----------o
| operační systém | DOS    | Linux  | BSD       |
o-----------------o--------o--------o-----------o
```

Poznámky

- Program nemusí brát v úvahu ne-ASCII znaky. Můžete předpokládat, že každý bajt řetězce zabere stejné místo na výstupu. (Můžete zkusit správně zpracovat UTF-8.)
- Můžete předpokládat maximální délku řádku 1024 bajtů. (Měli byste zvládnout i pro libovolnou délku.)
- Můžete předpokládat maximum 64 sloupců. (Měli byste zvládnout i pro libovolný počet.)
- Pište program už od začátku rozdělený na vhodné funkce. První by měl být napsaný několikařádkový main().
- Jednotlivé řádky tabulky je vhodné ukládat do spojovaného seznamu. Vzhledem k formě výpisu je nutné načíst celý soubor před výpisem.

Odstranění offsetu z EEG křivky (13.5.2013)

Program dostane na příkazové řádce jméno vstupního souboru *X* se záznamem 14 kanálů z EEG (Electroencephalogram). Program se soubor pokusí otevřít a načíst jeho obsah do vhodně pojmenované struktury. Každý kanál (sloupeček ve vstupním souboru) bude mít svou vlastní položku - dynamicky alokované pole hodnot. Vytvoříte funkci, která projde hodnoty v každém z kanálů a odstraní posun ve směru osy Y (offset) tak, aby nejmenší hodnota v daném kanálu po opravě byla 0. Výsledné upravené kanály se uloží do nového souboru *X.noff* ve stejném formátu jako vstupní soubor. Do výstupního souboru budou vypsány pouze ty položky, které byly zachyceny v čase v intervalu [A,B], kde hodnoty A a B obdržíte jako vstup na příkazovém řádku.



- program obdrží na příkazovém řádku tři argumenty: jméno_souboru čas_od čas_po
- pokud nedostane žádný argument, tak vypíše nápovědu a skončí
- program se tedy spouští např. jako './program eeg.dat 120 570' a vygeneruje výstupní soubor *eeg.dat.noff*
- vstupní soubor bude dodržovat tento formát (každý řádek obsahuje separátní záznam - jednu hodnotu pro každý kanál + dodatečné informace):
  - čas_měření stav_baterie gyro_x gyro_y kanal1 kanal2 … kanal14
  - všechny hodnoty jsou celá čísla se znaménkem oddělená bílým místem
  - ukázkový soubor pro testování: eeg.dat [http://www.fi.muni.cz/~xsvenda/eeg.dat]
- offset odstraňujete pouze u kanálů, nikoli u dodatečných informací
- do vystupního souboru vypisujete pouze záznamy pořízené v intervalu [A,B] (první sloupeček v souboru)
- program po uložení upraveného souboru vypíše na standardní výstup i statistiky ze zpracování
  - pro každý kanál hodnotu odebíraného offsetu
- zdrojový kód bude dodržovat dobré programovací návyky, nebude obsahovat memory leaks ani neuzavřené soubory
- zdrojový program nebude celý umístěn ve funkci main, ale bude vhodně rozdělen do smysluplných funkcí
- program nebude vytvářet přehnaně veliká pole struktur, která nebudou využita (tj. alokujete jen tolik, kolik budete využívat)
- program by neměl spadnout na Segmentation fault v případě že by byl puštěn bez parametrů nebo pokud by se nepodařilo najít vstupní soubor
- při vypracování příkladu můžete používat své vlastní kódy a dokumentaci nalezenou na internetu. Není povoleno spolupracovat s někým dalším

Příklad vstupních dat:

```
248 0 0 0  8637 8952 7692 8101 8811 9106 8113 9313 8034 8718 9600 8903 9115 8911
256 0 3 1  8630 8913 7693 8068 8854 9077 8128 9282 8014 8693 9618 8875 9097 8929
264 0 -1 2  8643 8906 7690 8070 8880 9050 8127 9281 8003 8696 9619 8878 9086 8944
271 0 0 0  8662 8906 7685 8080 8907 9034 8128 9306 8008 8700 9626 8903 9088 8958
279 0 0 0  8667 8886 7684 8077 8919 9014 8120 9322 8006 8695 9631 8914 9098 8971
287 0 0 0  8681 8883 7685 8082 8925 9001 8107 9323 8001 8703 9627 8911 9102 8972
```

První hodnota pro kanál1 bude po odstranění offsetu rovna hodnotě 7 (nejmenší hodnota v tomto kanálu je před zpracováním byla 8630)

Zjištění velikosti souborů

Program dostane na příkazové řádce jména souborů, ty se pokusí otevřít a pokud existují, tak zjistí jejich velikost. Do výstupního souboru se jménem *files.txt* zapíše jméno souboru a jeho velikost.

- program obdrží na příkazovém řádku volitelný počet argumentů (tj. 0…N)
- pokud nedostane žádný argument, tak vypíše nápovědu a skončí
- pokud dostane alespoň jeden argument, tak argumenty interpretuje jako jména souboru a postupně se je pokusí otevřít pro čtení

- program se tedy spouští např. jako './program soubor1 soubor2 soubor3'
- pro každý soubor, který se podařilo otevřít, zjistí program jeho velikost (např. pomocí funkcí fseek a ftell)
- pokud se soubor nepodaří otevřít, bude jeho velikost nastavena na -1
- program použije pro uchování dvojice jméno souboru a velikost souboru vhodně deklarovanou strukturu struct
- program nejprve zkusí otevřít všechny soubory, zjistí jejich velikosti a výsledné struktury uloží ve vhodné dynamicky alokované struktuře (pole resp. spojovaný seznam)
- po obsloužení všech souborů projde dynamicky alokovanou strukturu a vypíše její obsah do souboru *files.txt* ve formátu *jmeno_souboru* = *velikost_souboru*, každý záznam bude na samostatném řádku
- zdrojový kód bude dodržovat dobré programovací návyky, nebude obsahovat memory leaks ani neuzavřené soubory
- program nebude vytvářet přehnaně veliká pole struktur, která nebudou využita (tj. alokujete jen tolik, kolik budete využívat)
- program by neměl spadnout na Segmentation fault v případě že by byl puštěn bez parametrů nebo pokud by se nepodařilo najít vstupní soubor
- při vypracování příkladu můžete používat své vlastní kódy a dokumentaci nalezenou na internetu. Není povoleno spolupracovat s někým dalším

Předpokládejme, že existují soubory soubor1 (100B), soubor3 (existuje, ale 0B) a soubor4 (50B). Soubor 2 se nepodaří otevřít.

Výstup programu (v souboru files.txt) spuštěného jako './program soubor1 soubor2 soubor3 soubor4' může vypadat takto (výstup nemusí být nijak řazen):

```
soubor1 = 100
soubor2 = -1
soubor3 = 0
soubor4 = 50
```

Zjištění typu souboru

Program dostane jako argument na příkazové řádce jména souborů. Ty se pokusí otevřít a na základě jejich „hlavičky“ (první tři bajty) určí jejich typ. Na konci vypíše celkový počet souborů konkrétního typu (velikost znaků z hlavičky nehraje roli, tj. 'bmp' a 'BMP' jsou soubory stejného typu)

- program obdrží na příkazovém řádku volitelný počet argumentů (tj. 0…N)
- pokud nedostane žádný argument, tak vypíše nápovědu a skončí
- pokud dostane alespoň jeden argument, tak argumenty interpretuje jako jména souboru a postupně je otevře v *binárním* režimu pro čtení
- program se tedy spouští např. jako './program soubor1 soubor2 soubor3'
- z každého souboru načte první tří bajty (funkce fread()) a interpretuje je jako řetězec (např. 'bmp', 'Avi', 'JPG'…) označující typ souboru
- typ souboru je nezávislý na velikosti písmen ('bmp' == 'BMP' == 'BmP')
- program spočítá počet výskytů souboru konkrétního typu a na závěr vypíše na standardní výstup statistiku ve formátu 'typ_souboru : pocet_vyskytu'
- Nápověda: typ souboru a počet výskytů lze udržovat ve vhodné dynamicky alokované struktuře a další přidávat jen v případě, že daný typ souboru se ještě nevyskytuje
- zdrojový kód bude dodržovat dobré programovací návyky, nebude obsahovat memory leaks ani neuzavřené soubory
- program nebude vytvářet přehnaně veliká pole struktur, která nebudou využita (tj. alokujete jen tolik, kolik budete využívat)
- program by neměl spadnout na Segmentation fault v případě že by byl puštěn bez parametrů nebo pokud by se nepodařilo najít vstupní soubor
- při vypracování příkladu můžete používat své vlastní kódy a dokumentaci nalezenou na internetu. Není povoleno spolupracovat s někým dalším

Předpokládejme, že existují soubory soubor1 (hlavička 'bmp'), soubor2 (hlavička 'jpg'), soubor3 (hlavička 'BMP'), soubor4 (hlavička 'AVI')

Výstup programu spuštěného jako './program soubor1 soubor2 soubor3 soubor4' může vypadat takto (výstup nemusí být nijak řazen):

```
jpg : 1
bmp : 2
avi : 1
```

Katalog osob

- program načte po řádcích data z textového souboru - např. mydata.txt
- cesta k načítanému souboru je uvedena v prvním argumentu příkazového řádku
- program načte po řádcích data z textového souboru - např mydata.txt
- cesta k načítanému souboru je uvedena v prvním argumentu příkazového řádku
- počet řádků v souboru je uveden jako druhý argument příkazového řádku
- program se tedy spouští jako './program mydata.txt 4'
- každý řádek souboru obsahuje data ve formátu věk#jméno(formát nemusíte kontrolovat)
- řádek rozdělte dle symbolu # a naplňte jím dynamicky alokovanou položku typu 'struct Person', kterou vhodně zadeklarujte
- všechny řádky budou uloženy v poli ukazatelů na struct Person
- vytvořte samostatnou funkci, která vezme jako parametry ukazatel na začátek pole, délku pole a jméno hledané osoby. Funkce projde všechny prvky a vrátí věk pro hledanou osobu resp. -1 v případě, že osoba neexistuje
- ve funkci main vhodně demonstrujte hledaní jedné existující a jedné neexistující osoby
- na standardní výstup vypište informaci o věku a jméno třetího člověka v poli
- zdrojový kód bude dodržovat dobré programovací návyky, nebude obsahovat memory leaks
- program by neměl spadnout na Segmentation fault v případě že by byl puštěn bez parametrů nebo pokud by se nepodařilo najít vstupní soubor
- při vypracování příkladu můžete používat své vlastní kódy a dokumentaci nalezenou na internetu. Není povoleno spolupracovat s někým dalším

Výstup programu může vypadat například takto:

```
Hledana osoba: 'Pepa' vek: '51'
Hledana osoba: 'Neznamy' vek: (neexistuje)
Treti osoba: 'Jon Doe' vek: '103'
```

Rozvinutí zkomprimovaného zápisu

Program provádí rozvinutí komprimovaného zápisu z jednoho souboru do několika jiných souborů dle definovaných pravidel.

- Program dostane na příkazové řádce dva argumenty: vstupní soubor a maximální počet načítaných řádků
- Program otevře vstupní soubor, načte po řádcích data z textového souboru do pole struktur (zadeklarujte si vhodně sami).
- Program načítá řádky, dokud nenarazí na konec souboru nebo nedosáhne maximálního počtu načítaných řádků
    - Uzavře vstupní soubor
    - Provede změny v každé ze struktur popsané níže
    - Vytvoří nový výstupní soubor pro každý řádek vstupu (*výstupní_soubor*, viz. níže) a zapíše do něj výstup ve formátu popsaném níže
    - Uzavře výstupní soubor
    - Uvolní všechny ostatní zdroje, které za běhu využil (uvolnění dynamické paměti, apod)
    - Na stdout vypíše, kolik řádků bylo reálně zpracováno
- Formát vstupního řádku (můžete spoléhat na jeho korektnost)
    - nepřekročí délku 255 znaků

    - *výstupní_soubor*:*počet_opakování*:*znak*
    - např. 'data1.txt: 5 :x' (bez ')
- Požadovaná úprava v každé ze struktur
    - pokud je *počet_opakování* lichý, dojde ke snížení o jedna.
    - pokud je *počet_opakování* sudý, zůstane nezměněn
    - pokud je *počet_opakování* roven 0, výstupní soubor není pro tuto strukturu vůbec vytvářen
- Formát výstupního řádku
    - *výstupní_soubor*: *znak* zopakován bez mezer k-krát, kde k je rovno *počet_opakování*
    - např. vstupní řádek 'data1.txt: 5 :x' způsobí vytvoření souboru data1.txt s obsahem 'data1.txt:xxxx' (bez ')
    - např. vstupní řádek 'data2.txt: 2 :y' způsobí vytvoření souboru data2.txt s obsahem 'data2.txt:yy' (bez ')
    - např. vstupní řádek 'data3.txt: 0 :y' nevytvoří žádný soubor

Kopie souboru s duplikací

Program vytváří kopie zadaných souborů s tím, že výskyty cifer jsou duplikovány o zadaný počet

- Program dostane na příkazové řádce dva argumenty: vstupní soubor a maximální počet načítaných řádků
- Program otevře vstupní soubor, načte po řádcích data z textového souboru do pole struktur (zadeklarujte si vhodně sami, jednotlivé struktury jsou dynamicky alokovány)
- Program načítá řádky, dokud nenarazí na konec souboru nebo nedosáhne maximálního počtu načítaných řádků
- Provede změny v každé ze struktur popsané níže (po načtení všech řádků do struktury, ne během načítání)
- Vytvoří nový výstupní soubor pro každý řádek vstupu (*výstupní_soubor*, viz. níže) a zapíše do něj výstup ve formátu popsaném níže
- Uvolní všechny ostatní zdroje, které za běhu využil (uvolnění dynamické paměti, apod)
- Na stdout vypíše, kolik řádků bylo reálně zpracováno
- Formát vstupního řádku (můžete spoléhat na jeho korektnost)
    - nepřekročí délku 255 znaků
    - *vstupní_soubor@výstupní_soubor*:*počet_opakování_cifry*
    - např. 'dataIn.txt@dataout.txt : 5' (bez ')
- Požadovaná úprava v každé ze struktur
    - pokud je jméno vstupního i výstupního souboru shodné, je k výstupnímu souboru přidána koncovka .res
    - pokud je jméno různé, jméno výstupního souboru zůstane nezměněno
- Formát výstupního souboru
    - pokud vstupní soubor neobsahuje žádné cifry, tak je obsah výstupního souboru identický
    - pokud obsahuje cifry, tak je výskyt každé cifry duplikován k-krát, kde k je rovno *počet_opakování_cifry*
    - např. sekvence ze vstupního souboru fdkljsefa45jksdf2nnn se změní na fdkljsefa444555jksdf222nnn, pokud je *počet_opakování_cifry* rovno 3

Sociální síť

Program provádí analýzu struktury sociální sítě.

- Program dostane na příkazové řádce dva argumenty: vstupní soubor a maximální počet načítaných řádků
    - Pokud nejsou zadány korektně data na příkazovém řádku, program vypíše nápovědu a skončí
- Vstupní soubor obsahuje seznam přátel na sociální síti ve formátu osoba:přítel (obě položky jsou jména z tisknutelných znaků)
    - Každý řádek obsahuje právě jeden záznam
    - Můžete předpokládat korektní formát vstupního souboru
    - Jedna osoba může mít více přátel, jejich celkový počet **není** omezen
- Zadefinujte si vhodně strukturu pro každou osobu.

    - Tato struktura bude v sobě obsahovat jméno osoby, počet přátel a dynamicky alokované pole nebo seznam se jmény všech přátel.
    - Pole nebo seznam bude právě tak velký, kolik má osoba přátel (tj. žádná nadbytečná alokace)
- Všechny načtené osoby (struktury) uložte do vhodného pole nebo seznamu
- Po načtení všech osob nalezněte a vypište osobu s největším počtem přátel ve formátu *jméno_osoby* = *počet_přátel (jména přátel)*

- Nápověda: Vstupní soubor můžete projít a zpracovat vícekrát
- Program nebude obsahovat memory leaky, neuzavřené soubory, zápis mimo alokovanou pamět a bude dodržovat konvence programování běžně používané v předmětu

Pro vstupní data:

```
Tomas Jedno: Michal Blazek
Roman Havran:Misa Rychtarova
Tomas Jedno:Martin Smrk
```

Vypíše:

```
Tomas Jedno = 2 (Michal Blazek, Martin Smrk)
```

Váhy

- program načte po řádcích data z textového souboru - např data-02.txt
- cesta k načítanému souboru je uvedena v prvním argumentu příkazového řádku
- počet řádků v souboru je uveden jako druhý argument příkazového řádku
- program se tedy spouští jako './program data-02.txt 4'
- každý řádek souboru obsahuje data ve formátu jmeno:vaha(formát nemusíte kontrolovat)
- řádek rozdělte dle symbolu : a naplňte jím dynamicky alokovanou položku typu 'struct Person', kterou vhodně zadeklarujte
- všechny řádky budou uloženy v poli ukazatelù na struct Person
- vytvořte samostatnou funkci, která vezme jako parametry ukazatel na začátek pole a délku pole, projde všechny prvky a vrátí součet hmotností všech lidí
- na standartní výstup vypište informaci o celkové hmotnosti a jméno prvního člověka v poli (data budou vždy obsahovat alespoň jednu osobu)
- zdrojový kód bude dodržovat dobré programovací návyky, nebude obsahovat memory leaks
- program by neměl spadnout na Segmentation fault v případì že by byl spuštěn bez parametrů nebo pokud by se nepodařilo najít vstupní soubor
- při vypracování příkladu můžete používat své vlastní kódy a dokumentaci nalezenou na internetu. Není povoleno spolupracovat s nìkým dalším

```
výstup programu na vzorových datech může vypadat například takto:
Total mass: 105
First person: Jana Novakova
```