IJC: DU2

```
Jazyk C                         DU2                        27.3.2017
--------------------------------------------------------------------

                          Domácí úkol č.2


Termín odevzdání: 25.4.2017                          (Max. 15 bodů)

1) (max 5b)
   a) V jazyku C napište program "tail.c", který  ze zadaného
   vstupního souboru vytiskne posledních  10 řádků. Není-li
   zadán vstupní soubor, čte ze stdin. Je-li programu zadán
   parametr -n číslo, bude se tisknout tolik posledních řádků,
   kolik je zadáno parametrem 'číslo' >= 0.
   Případná chybová hlášení tiskněte do stderr. Příklady:

     tail  soubor
     tail  -n 20 <soubor

   [Poznámka: výsledky by měly být +-stejné jako u POSIX příkazu tail]

   Je povolen implementační limit na délku řádku (např. 1024 znaků),
   v případě prvního překročení mezí hlaste chybu na stderr (řádně otestujte)
   a pokračujte se zkrácenými řádky (zbytek řádku přeskočit/ignorovat).

   b) Napište stejný program jako v a) v C++11 s použitím standardní
   knihovny C++. Jméno programu: "tail2.cc". Tento program
   musí zvládnout řádky libovolné délky a jejich libovolný počet,
   jediným možným omezením je volná paměť.
   Použijte funkci
      std::getline(istream, string)
   a vhodný STL kontejner (např. std::queue<string>).
   Poznámka: Pro zrychlení použijte std::ios::sync_with_stdio(false);
             protože _nebudete_ používat <cstdio>


2) (max 10b)
   Přepište následující C++ program do jazyka ISO C

    // wordcount-.cc
    // Použijte GCC>=4.9: g++ -std=c++11
    // Příklad použití STL kontejneru map<> nebo unordered_map<>
    // Program počítá četnost slov ve vstupním textu,
    // slovo je cokoli oddělené "bílým znakem" === isspace

    #include <string>
    #include <iostream>
    #include <unordered_map>

    int main() {
        using namespace std;
        unordered_map<string,int> m;  // asociativní pole
        string word;

        while (cin >> word) // čtení slova
            m[word]++;      // počítání výskytů slova

        for (auto &mi: m)   // pro všechny prvky kontejneru m
            cout << mi.first << "\t" << mi.second << "\n";
        // tisk     slovo (klíč)        počet (data)
    }
```

Výstupy programů musí být pro stejný vstup stejné (kromě pořadí a příliš dlouhých slov).
Výsledný program se musí jmenovat "wordcount.c".

Veškeré operace s tabulkou budou v samostatné knihovně (vytvořte statickou
i dynamickou/sdílenou verzi). V knihovně musí být každá funkce ve
zvláštním modulu - to umožní případnou výměnu hash_function() ve vašem
staticky sestaveném programu (vyzkoušejte si to: definujte svoji
hash_function v programu).

Knihovna s tabulkou se musí jmenovat
"libhtab.a" (na Windows je možné i "htab.lib") pro statickou variantu,
"libhtab.so" (na Windows je možné i "htab.dll") pro sdílenou variantu
a rozhraní "htab.h".

Podmínky:
- Implementace musí být dynamická (malloc/free) a musíte zvládnout
  správu paměti v C (použijte valgrind, nebo jiný podobný nástroj).

- Asociativní pole implementujte nejdříve prototypově jednoduchým seznamem
  a potom tabulkou (hash table). Odevzdává se řešení s tabulkou.

- Vhodná rozptylovací funkce pro řetězce je podle literatury
  (http://www.cse.yorku.ca/~oz/hash.html varianta sdbm):

```
  unsigned int hash_function(const char *str) {
    unsigned int h=0;
    const unsigned char *p;
    for(p=(const unsigned char*)str; *p!='\0'; p++)
        h = 65599*h + *p;
    return h;
  }
```

  její výsledek modulo arr_size určuje index do tabulky:
```
   index = (hash_function("mystring") % arr_size);
```
  Zkuste použít i jiné podobné funkce a porovnejte efektivitu.

- Tabulka je (pro knihovnu privátní) struktura obsahující pole seznamů,
  jeho velikost a počet položek tabulky v následujícím pořadí:

```
   +----------+
   | arr_size | // velikost pole ukazatelů
   +----------+
   | n        | // aktuální počet záznamů [key,data,next]
   +----------+
   +---+
   |ptr|-->[key,data,next]-->[key,data,next]-->[key,data,next]--|
   +---+
   |ptr|-->[key,data,next]-->[key,data,next]--|
   +---+
   |ptr|--|
   +---+
```

  Položka arr_size je velikost následujícího pole ukazatelů (použijte
  C99: "flexible array member"). Paměť pro strukturu se dynamicky alokuje
  tak velká, aby se do ní vešly i všechny položky pole.
  V programu zvolte vhodnou velikost pole a v komentáři zdůvodněte vaše
  rozhodnutí.
  (V obrázku platí velikost arr_size==3 a počet položek n==5.)
  Nápověda: rozhraní knihovny obsahuje jen neůplnou deklaraci struktury

- Napište funkce

```
  t=htab_init(size)         konstruktor: vytvoření a inicializace tabulky

  t=htab_move(newsize,t2)   move konstruktor: vytvoření a inicializace
                            tabulky daty z tabulky t2,
                            t2 nakonec zůstane prázdná a alokovaná

  size_t s=htab_size(t)           vrátí počet prvků tabulky (n)
```

```
    size_t c=htab_bucket_count(t)   vrátí počet prvků pole (arr_size)

    ptr=htab_lookup_add(t,key)    vyhledávání - viz dále

    ptr=htab_find(t,key)          vyhledávání - viz dále

    htab_foreach(t,func)      volání funkce func pro každý prvek

    b=htab_remove(t,key)      vyhledání a zrušení zadané položky
                              vrací b==false pokud neexistuje

    htab_clear(t)             zrušení všech položek, tabulka zůstane prázdná

    htab_free(t)              destruktor: zrušení tabulky (volá htab_clear())

  kde t,t2    je ukazatel na tabulku (typu htab_t *),
      b       je typu bool,
      ptr     je ukazatel na záznam (položku tabulky),
      func    je funkce s parametry: func(constkey,valueptr)

- Vhodně zvolte typy parametrů funkcí.

- Záznam [key,data,next] je typu
      struct htab_listitem
  a obsahuje položky:
      key .... ukazatel na dynamicky alokovaný řetězec,
      data ... počet výskytů a
      next ... ukazatel na další záznam

- Funkce  htab_foreach(t,function)  volá zadanou funkci pro každý prvek
  tabulky, obsah tabulky nemění. (Vhodné např. pro tisk obsahu.)

- Funkce
    struct htab_listitem * htab_lookup_add(htab_t *t, const char *key);
  V tabulce  t  vyhledá záznam odpovídající řetězci  key  a
    - pokud jej nalezne, vrátí ukazatel na záznam
    - pokud nenalezne, automaticky přidá záznam a vrátí ukazatel
  Poznámka: Dobře promyslete chování této funkce k parametru key.

    struct htab_listitem * htab_find(htab_t *t, const char *key);
  Totéž jako htab_lookup_add, ale bez alokace:
    - pokud nenalezne, vrací NULL

- Když htab_init, htab_move, htab_lookup_add nemohou alokovat paměť,
  vrací NULL


Napište funkci

    int get_word(char *s, int max, FILE *f);

  která čte jedno slovo ze souboru f do zadaného pole znaků
  a vrátí délku slova (z delších slov načte prvních max-1 znaků,
  a zbytek přeskočí). Funkce vrací EOF, pokud je konec souboru.
  Umístěte ji do zvláštního modulu "io.c" (nepatří do knihovny).
  Poznámka: Slovo je souvislá posloupnost znaků oddělená isspace znaky.

Omezení: řešení v C může tisknout jinak seřazený výstup
  a je povoleno použít implementační limit na maximální
  délku slova (např. 127 znaků), delší slova se ZKRÁTÍ a program
  při prvním delším slovu vytiskne varování na stderr (max 1 varování).

Poznámka: Vhodný soubor pro testování je například seznam slov
          v souboru /usr/share/dict/words
          nebo texty z http://www.gutenberg.org/
          případně výsledek příkazu:  seq 1000000 2000000|shuf
```

(10b)

Použijte implicitní lokalizaci (= nevolat setlocale()).

Napište soubor Makefile tak, aby příkaz make vytvořil programy
"tail", "tail2", "wordcount", "wordcount-dynamic" a knihovny "libhtab.a",
"libhtab.so" (nebo "htab.dll").
Program "wordcount" musí být staticky sestaven s knihovnou "libhtab.a".
Program "wordcount-dynamic" musí být sestaven s knihovnou "libhtab.so".
Tento program otestujte se stejnými vstupy jako u staticky sestavené verze.

Porovnejte efektivitu obou (C i C++) implementací (viz např. příkaz time)
a zamyslete se nad výsledky (pozor na vliv vyrovnávacích paměťí atd.)
Také si zkuste překlad s optimalizací i bez ní (-O2, -O0) a porovnejte
efektivitu pro vhodný vstup.

Poznámky:
- 1b) pokud možno maximálně využívejte standardní knihovny C++
- 2) pro testy wordcount-dynamic na linuxu budete potřebovat nastavit
  LD_LIBRARY_PATH="." (viz "man ld.so" a odpovídající přednáška)
- Čtěte pokyny pro vypracování domácích úkolů (viz dále)

Obecné pokyny pro vypracování domácích úkolů

* Pro úkoly v jazyce C používejte ISO C99 (soubory *.c)
  Pro úkoly v jazyce C++ používejte ISO C++11 (soubory *.cc)
  Použití nepřenositelných konstrukcí není dovoleno.

* Úkoly zkontrolujte překladačem například takto:
  gcc -std=c99 -pedantic -Wall -Wextra priklad1.c
  g++ -std=c++11 -pedantic -Wall priklad.cc
  Místo gcc můžete použít i jiný překladač - podle vašeho prostředí.
  V souvislosti s tím napište do poznámky na začátku
  souboru jméno a verzi překladače, kterým byl program přeložen
  (implicitní je GCC `g++ --version` na počítači merlin).

* Programy pište, pokud je to možné, do jednoho zdrojového
  souboru. Dodržujte předepsaná jména souborů.

* Na začátek každého souboru napište poznámku, která bude
  obsahovat jméno, fakultu, označení příkladu a datum.

* Úkoly je nutné zabalit programem zip takto:
  zip xnovak99.zip *.c *.cc *.h Makefile

  Jméno xnovak99 nahradíte vlastním. Formát souboru bude ZIP.
  Archiv neobsahuje adresáře. Každý si zkontroluje obsah ZIP archivu jeho
  rozbalením v prázdném adresáři a napsáním "make".

* Posílejte pouze nezbytně nutné soubory -- ne *.EXE !

* Řešení se odevzdává elektronicky v IS FIT

* Úkoly neodevzdané v termínu (podle WIS) budou za 0 bodů.

* Opsané úkoly budou hodnoceny 0 bodů pro všechny zůčastněné
  a to bez výjimky (+ bonus v podobě návštěvy u disciplinární komise).

Poslední modifikace: 27. March 2017
Pokud naleznete na této stránce chybu, oznamte to dopisem na adresu peringer AT fit.vutbr.cz