IJC: DU2

```
Jazyk C                       DU2                      20.3.2019
----------------------------------------------------------------

                        Domácí úkol č.2


Termín odevzdání: 24.4.2019                       (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 +3 soubor
     tail -n 20 <soubor

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

   Je povolen implementační limit na délku řádku (např. 1023 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> (pozor na valgrind)


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

    // wordcount-.cc
    // Použijte: g++ -std=c++11 -O2
    // Příklad použití STL kontejneru nebo unordered_map<>
    // Program počítá četnost slov ve vstupním textu,
    // slovo je cokoli oddělené "bílým znakem"

    #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";
            //      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".

   Implementujte tabulku s rozptýlenými položkami (hash table) - viz dále.
   Veškeré operace s tabulkou budou v samostatné knihovně (vytvořte statickou
   i dynamickou/sdílenou verzi).  V knihovně musí být prakticky každá funkce ve
   zvláštním modulu - to umožní případnou výměnu htab_hash_function() ve vašem
   staticky sestaveném programu. (V dynamicky sestaveném programu je to možné vždy.)
   Vyzkoušejte si to: definujte svoji verzi htab_hash_function() v programu
   s podmíněným překladem -- použijte #ifdef HASHTEST.

   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).

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

        unsigned int htab_hash_function(const char *str) {
          uint32_t h=0;     // musí mít 32 bitů
          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 = (htab_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:

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

      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 .size==5.)
      Rozhraní knihovny obsahuje jen neúplnou deklaraci struktury.

    - Napište funkce podle následujícího hlavičkového souboru:

==================================================================
// htab.h -- rozhraní knihovny htab (řešení IJC-DU2)
// Licence: žádná (Public domain)

// následující řádky zabrání násobnému vložení:
#ifndef __HTABLE_H__
#define __HTABLE_H__

#include <string.h>     // size_t
#include <stdbool.h>    // bool

// tabulka:
struct htab;    // neúplná deklarace struktury - uživatel nevidí obsah
typedef struct htab htab_t;     // typedef podle zadání
// iterátor do tabulky:
struct htab_item;               // neúplná deklarace struktury

// iterátor:
typedef struct htab_iterator {
    struct htab_item *ptr;      // ukazatel na položku
    const htab_t *t;            // ve které tabulce
    int idx;                    // ve kterém seznamu tabulky (TODO: optimalizovat?)
} htab_iterator_t;              // typedef podle zadání

// rozptylovací (hash) funkce
// pokud si v programu definujete stejnou funkci, použije se ta vaše
unsigned int htab_hash_function(const char *str);

// funkce pro práci s tabulkou:
htab_t *htab_init(size_t n);
htab_t *htab_move(size_t n, htab_t *from);
size_t htab_size(const htab_t * t);             // počet záznamů v tabulce
size_t htab_bucket_count(const htab_t * t);     // velikost pole

htab_iterator_t htab_lookup_add(htab_t * t, const char *key);

htab_iterator_t htab_begin(const htab_t * t);
htab_iterator_t htab_end(const htab_t * t);
htab_iterator_t htab_iterator_next(htab_iterator_t it);
inline bool htab_iterator_valid(htab_iterator_t it) { return it.ptr!=NULL; }
inline bool htab_iterator_equal(htab_iterator_t it1, htab_iterator_t it2) {
  return it1.ptr==it2.ptr && it1.t == it2.t;
}
const char * htab_iterator_get_key(htab_iterator_t it);
int htab_iterator_get_value(htab_iterator_t it);
int htab_iterator_set_value(htab_iterator_t it, int val);

void htab_clear(htab_t * t);
void htab_free(htab_t * t);

#endif // __HTABLE_H__
==================================================================

      Hlavičkový soubor můžete celý převzít (je v "Public domain").

    - Stručný popis základních funkcí:

        t=htab_init(numb)         konstruktor: vytvoření a inicializace tabulky
                                  numb = počet prvků pole (.arr_size)

        t=htab_move(newnumb,t2)   move konstruktor: vytvoření a inicializace
                                  nové tabulky přesunem dat z tabulky t2,
                                  t2 nakonec zůstane prázdná a alokovaná
                                  (tuto funkci cvičně použijte v programu
                                   podmíněným překladem #ifdef TEST)

        size_t s=htab_size(t)           vrátí počet prvků tabulky (.size)

        size_t n=htab_bucket_count(t)   vrátí počet prvků pole (.arr_size)

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

        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),

    - Vhodně zvolte typy parametrů funkcí (včetně použití const).

    - Záznam [key,data,next] je typu
          struct htab_item
      a obsahuje položky:
          key .... ukazatel na dynamicky alokovaný řetězec,
          data ... počet výskytů a
          next ... ukazatel na další záznam
      Tento záznam je definován v privátním hlavičkovém souboru pro všechny
      moduly tabulky a není dostupný při použití knihovny ("Opaque data type").
      Každý iterátor obsahuje ukazatel na tuto strukturu.

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

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

    - Funkce pro iteraci přes prvky tabulky:
        iterator = htab_begin(t)
          vrací iterátor označující první záznam
        iterator = htab_end(t)
          vrací iterátor označující (neexistující) první záznam za koncem
        iterator = htab_next(iterator)
          posun iterátoru na další záznam v tabulce (nebo na htab_end(t))

    - Základní funkce pro testování iterátorů:
        b = htab_iterator_valid(iterator)
          test, zda iterátor označuje platný prvek v tabulce
        b = htab_iterator_equal(iterator1, iterator2)
          porovnání dvou iterátorů na rovnost

    - Funkce pro čtení/zápis přes iterátor:
        cstr = htab_iterator_get_key(iterator);
          vrací klíč, cíl musí existovat
        i = htab_iterator_get_value(iterator);
          vrací hodnotu, cíl musí existovat
        htab_iterator_set_value(iterator, nova_hodnota);
          přepisuje hodnotu, cíl musí existovat


    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" atd.).

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 pamětí 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).
```