IJC: DU1

```
Jazyk C                    DU1                          24.2.2017
--------------------------------------------------------------------


                      Domácí úkol č.1


Termín odevzdání:  27.3.2017
 čtěte pokyny na konci tohoto textu
 Hodnocení celkem max. 15 bodů

Příklady: (budou opravovány v prostředí Linux/GCC,
           LC_ALL=cs_CZ.utf8
           překlad: gcc -O2 -std=c99 -Wall -pedantic)

a) V rozhraní "bit_array.h" definujte pro datovou strukturu typu pole bitů:
   Typ:
     typedef bit_array_t
       Typ bitového pole (pro předávání parametru do funkce odkazem)

   Makra:
     ba_create(jmeno_pole,velikost)
       definuje a _nuluje_ proměnnou jmeno_pole
       (POZOR: opravdu musí _INICIALIZOVAT_ pole bez ohledu na
       to, zda je pole statické nebo automatické/lokální!  Vyzkoušejte si obě
       varianty, v programu použijte lokální pole.)
       Př: static ba_create(p,100); // p = pole 100 bitů, nulováno
           ba_create(q,100000L); // q = pole 100000 bitů, nulováno

     ba_size(jmeno_pole)
       vrátí deklarovanou velikost pole v bitech

     ba_set_bit(jmeno_pole,index,výraz)
       nastaví zadaný bit v poli na hodnotu zadanou výrazem
       (nulový výraz == bit 0, nenulový výraz == bit 1)
       Př: ba_set_bit(p,20,1);

     ba_get_bit(jmeno_pole,index)
       získá hodnotu zadaného bitu, vrací hodnotu 0 nebo 1
       Př: if(ba_get_bit(p,i)==1) printf("1");
           if(!ba_get_bit(p,i))   printf("0");

   Kontrolujte meze polí. V případě chyby volejte funkci

     error_msg("Index %lu mimo rozsah 0..%lu",
               (unsigned long)index, (unsigned long)mez).

   (Použijte například modul error.c/error.h z příkladu b)

   Program musí fungovat na 32 (gcc -m32) i 64bitové platformě.

   Podmíněným překladem zajistěte, aby se při definovaném symbolu
   USE_INLINE místo těchto maker definovaly inline funkce stejného jména
   všude kde je to možné (bez změn v následujícím testovacím příkladu!).
   USE_INLINE nesmí být definováno ve zdrojovém textu --
   překládá se s argumentem -D (gcc -DUSE_INLINE ...).

   Napište pomocná makra BA_GET_BIT_(p,i), BA_SET_BIT_(p,i,b)
   pro indexování bitů v poli T *p nebo T p[NN] bez kontroly mezí,
   kde T je libovolný celočíselný typ (char, unsigned short, ...).
   (Tato makra pak použijete v dalších makrech a inline funkcích.)
```

```
   Pro vaši implementaci použijte pole typu: unsigned long [].
   Implementace musí efektivně využívat paměť (využít každý
   bit pole až na posledních max. CHAR_BIT*sizeof(unsigned long)-1).

   Jako testovací příklad implementujte funkci, která použije algoritmus známý
   jako Eratostenovo síto (void Eratosthenes(bit_array_t pole);) a použijte ji
   pro výpočet posledních 10 prvočísel ze všech prvočísel od 2 do
   N=303000000 (303 milionů). (Doporučuji program nejdříve odladit pro N=100.)
   Funkci Eratosthenes napište do samostatného modulu "eratosthenes.c".

   Budete pravděpodobně potřebovat zvětšit limit velikosti zásobníku.
   Na Unix-like systémech použijte příkaz ulimit -a pro zjištění velikosti
   limitu a potom "ulimit -s zadana_velikost_v_KiB".

   Každé prvočíslo tiskněte na zvláštní řádek v pořadí
   vzestupném.  Netiskněte  nic  jiného  než  prvočísla (bude se
   automaticky  kontrolovat!).  Pro kontrolu správnosti prvočísel
   můžete použít program  factor (./primes|factor).

   Zdrojový text programu se musí jmenovat "primes.c" !
   Napište Makefile tak, aby příkaz "make" vytvořil všechny varianty:
    primes      používá makra
    primes-i            inline funkce
   a aby příkaz "make run" všechny varianty spustil stylem:
     time ./primes

   (Při nesplnění podmínek: až 0 bodů.)

(7b)

Poznámky:  Eratosthenovo síto (přibližná specifikace):
   1) Nulujeme bitové pole  p  o rozměru N,
      p[0]=1; p[1]=1; // 0 a 1 nejsou prvočísla
      index i nastavit na 2
   2) Vybereme nejmenší index i, takový, že p[i]==0.
      Potom je i prvočíslo
   3) Pro všechny násobky i nastavíme bit p[n*i] na 1
      ('vyškrtneme' násobky - nejsou to prvočísla)
   4) i++; dokud nejsme za sqrt(N), opakujeme bod 2 až 4
      (POZOR: sestavit s matematickou knihovnou parametrem -lm)
   5) Výsledek: v poli p jsou na prvočíselných indexech hodnoty 0

   https://en.wikipedia.org/wiki/Prime_number

   Efektivita výpočtu: cca 2.2s na Intel i5-4690 @ 3.50GHz (gcc -O2)
   Porovnejte efektivitu obou variant (makra vs. inline funkce).
   Zamyslete se, jak by se ověřila efektivita pro (neinline) funkce.
```

```
b) Napište modul "error.c" s rozhraním v "error.h", který definuje
   funkci void warning_msg(const char *fmt, ...) a
   funkci void error_msg(const char *fmt, ...). Tyto funkce mají
   stejné parametry jako printf(); tisknou text "CHYBA: " a potom
   chybové hlášení podle formátu fmt. Vše se tiskne do stderr
   (funkcí vfprintf) a potom pouze error_msg ukončí program voláním
   funkce exit(1).  Použijte definice ze stdarg.h.

 * Napište modul "ppm.c" s rozhraním "ppm.h",
   ve kterém definujete typ:
     struct ppm {
        unsigned xsize;
        unsigned ysize;
        char data[];    // RGB bajty, celkem 3*xsize*ysize
     };
   a funkce:
     struct ppm * ppm_read(const char * filename);
```

```
        načte obsah PPM souboru do touto funkcí dynamicky
        alokované struktury. Při chybě formátu použije funkci warning_msg
        a vrátí NULL.  Pozor na "memory leaks".

      int ppm_write(struct ppm *p, const char * filename);
         zapíše obsah struktury p do souboru ve formátu PPM.
         Při chybě použije funkci warning_msg a vrátí záporné číslo.

    Můžete doplnit další funkce, ale pro DU1 to není nutné.
    [Zamyslete se nad (ne)vhodností použití warning_msg() a promyslete alternativní
    způsoby hlášení chyb.]

    Můžete omezit max. velikost obrazových dat vhodným implementačním
    limitem (např 1000*1000*3), aby bylo možné použít statickou inicializaci
    bit_array_t pro následující testovací program.

    Popis formátu PPM najdete na Internetu, implementujte pouze
    binární variantu P6 s barvami 0..255 a bez komentářů:
      "P6" <ws>+
      <xsizetxt> <ws>+ <ysizetxt> <ws>+
      "255" <ws>
      <binarni data, 3*xsize*ysize bajtu RGB>
      <EOF>

  * Napište testovací program "steg-decode.c", kde ve funkci main načtete ze
    souboru zadaného jako jediný argument programu obrázek ve formátu PPM
    a v něm najdete uloženou "tajnou" zprávu. Zprávu vytisknete na stdout.

    Zpráva je řetězec znaků (char, včetně '\0') uložený po jednotlivých bitech
    (počínaje LSb) na nejnižších bitech (LSb) vybraných bajtů barevných složek
    v datech obrázku. Dekódování ukončete po dosažení '\0'.
    Pro DU1 budou vybrané bajty určeny prvočísly -- použijte Eratostenovo síto
    podobně jako v příkladu "primes.c".

    Program použije error_msg v případě chyby čtení souboru (chybný formát),
    a v případě, že zpráva není korektně ukončena '\0'.

    Použijte program "make" pro překlad/sestavení programu.
    Testovací příkaz:  ./steg-decode du1-obrazek.ppm

    Zájemci si mohou vytvořit i program "steg-encode.c" (nehodnotí se).
    Zamyslete se nad (ne)vhodností implementačních limitů vynucených
    konstantní velikostí pole bitů.

(8b)

   Zařiďte, aby příkaz "make" bez parametrů vytvořil všechny spustitelné
   soubory pro DU1.  Při změně kteréhokoli souboru musí přeložit jen změněný
   soubor a závislosti. Pokud bude Makefile vypadat jako skript odečtou se 3b.
```

Testovací obrázek: du1-obrazek.ppm

```
Předmět: Jazyk C                                    rev 20.2.2017

Obecné pokyny pro vypracování domácích úkolů

*  Pro úkoly v jazyce C používejte ISO C99 (soubory *.c)
   Použití nepřenositelných konstrukcí není dovoleno.
   C11 nebudete potřebovat.

*  Úkoly zkontrolujte překladačem například takto:
      gcc -g -std=c99 -pedantic -Wall -Wextra priklad1.c
   místo gcc můžete použít i jiný překladač
!  (nebude-li úkol podle normy ISO C99, bude za 0 bodů!)
   v  souvislosti s tím napište do poznámky na začátku
```

```
    souboru jméno překladače, kterým byl program přeložen
    (implicitní je verze GNU C instalovaná na serveru 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.

 Příklad:
   // enum.c
   // Řešení IJC-DU1, příklad a), 20.3.2111
   // Autor: Jároslav Cimrman, FIT
   // Přeloženo: gcc 4.9
   // ...popis příkladu - poznámky, atd

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

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

 * Řešení se odevzdává elektronicky v IS FIT (velikost souboru je omezena)

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

 * Úkoly neodevzdané v termínu 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: 23. February 2017
Pokud naleznete na této stránce chybu, oznamte to dopisem na adresu peringer AT fit.vutbr.cz