DUM č. 16 v sadě

35. Inf-11 Objektové programování v Greenfoot

Autor: Lukáš Rýdlo

Datum: 08.06.2014

Ročník: studenti semináře

Anotace DUMu: Načítání dat BMP souboru do dynamicky alokovaného dvourozměrného pole pixelů. Ukládání obrazových dat. Struktura dat v BMP souboru, ošetření zarovnání dat.

evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Projekt: konzolový BMP editor v C
Čtení a zápis dat BMP souboru

Úvod

V předchozí lekci se měli studenti především seznámit s psaním kódu v linuxu a kompilaci čistě na příkazové řádce. Přesto se v příkladech vyskytla i poměrně náročná práce s korektním používáním structů, načítání binárních souborů a rozdělení kódu na zdrojový c-soubor a hlavičkový h-soubor.

V této lekci se zaměříme na dynamickou alokaci paměti a procvičíme ji také se strukturami, protože budeme potřebovat načítat data BMP souboru, která tvoří dvourozměrné pole pixelů (údajů o červené, zelené a modré barvě). Jelikož alokovat takovou strukturu není jednoduché, procvičíme si to nejprve na dvou jednodušších příkladech mimo projekt.

Při řešení této lekce je dobré znovu pročíst článek http://www.root.cz/clanky/graficky-format-bmp-pouzivany-a-pritom-neoblibeny/ a především si uvědomit, že data v BMP souboru jsou doplňovaná nulovými byty tak, aby byl počet bytů (nikoliv pixelů!) každého obrazového řádku dělitelný čtyřmi. Dále znovu připomínám, že některé grafické editory nedodržují standardní velikost hlaviček a přidávají si data navíc, proto je potřeba před čtením obrazových dat přeskočit funkcí fseek na správné místo v souboru.

Naše implementace BMP formátu bude podporovat pouze 24bitovou barevnou hloubku, jednu vrstvu dat a žádnou kompresi. Nepodporujeme ani paletu. Pro účely výuky to bohatě postačuje, protože klíčové koncepty si demonstrujeme, ale nemusíme řešit výjimky a kód zůstane přiměřeně krátký. Navíc většina BMP souborů se používá právě v tomto nastavení. Zájemci (opravdu nadaní programátoři) mohou v případě velkého časového odstupu od zbytku skupiny implementovat podporu RLE komprese. Během načítání data dekomprimují, ukládají do pole pixelů stejně jako nekomprimovaná. Při opětovném zápisu se data znovu komprimují, pokud je to nastaveno v hlavičce. To také umožní uložit nekomprimovaný soubor komprimovaně – stačí před uložením v hlavičce změnit hodnotu komprimace.

Úkoly s řešeními

1. Napište program (pozpatku.c), který se uživatele zeptá, kolik chce zadat čísel. Pak čísla od uživatele načte a vypíše v opačném pořadí. Použijte dynamicky alokované pole čísel. Zkompilujte v konzoli a vyzkoušejte.

   Triviální prográmek u kterého se ověří pochopení dynamické alokace. Důkladně zkontrolujeme, zda je v řešení i dealokace a test úspěšného alokování… Při kontrole je možné zkusit jak moc velké musí být pole, aby alokace selhala.

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
   int count;
   int i;
   int □ array;

   printf("Zadejte počet hodnot: ");
   scanf("%i",□count);

   array = (□) malloc(count*□(int));
   if(array == □) {
      printf("Nelze alokovat pamet.\n");
      return 1;
   }

   for(i=0;i<count;i++) {
      printf("Zadej %i. cislo: ",i);
      scanf("%i",□);
   }

   for(i=count;i>0;i--) {
      printf("%i. cislo: je %i\n", i, array[□]);
   }

   free(□);
   return 0;
}
```

2. Napište program asciiart.c, který se nejprve uživatele zeptá na výšku a šířku „obrázku“, následně alokuje dvourozměrné pole znaků a naplní jej mezerami. V cyklu se ptejte uživatele na souřadnice pole, na které se vloží hvězdička. Pokud bude souřadnice mimo rozsah, program končí, jinak se vypíše pole na obrazovku a pokračuje se dalším dotazem. Nezapomeňte kontrolovat, zde se alokace zdařila, v případě neúspěchu dealokovat, což alokované bylo, stejně tak na konci programu. Zkuste takto nakreslit smajlíka z hvězdiček.

   Řešení je komplikovanější a většině studentů už poměrně zamotá hlavu, především v tom, kolik kde je vězdiček (co je ukazatel na co). Je vcelku důležité nakreslit si na tabuli obrázek, jak a čím

se při alokaci 2D pole zaplňuje paměť, kde je adresa, kde ukazatel, kde hodnota, co ukazuje na co. Nejvíce chyb bude v dealokaci a kontrole úspěšné alokace. Pokud student nepochopí tento úkol, nemá smysl, aby pokračoval dále. Vzor opět vynechává podstatné znaky nebo celé výrazy, můžeme jej tentokrát použít jako „doplňovačku“, na které se pozná porozumění.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[]) {
    int width=0, height=0;
    char □ array;
    int i=0, x=0, y=0, posx, posy;

    printf("Zadejte sirku a vysku pole: ");
    scanf("%i %i",&width, &height);

    array = (□) malloc(sizeof(char□)*width);

    if (array==□) {
       printf("Alokace se nezdarila. Koncim.\n");
       return 1;
    }

    for(i=0;i<width;i++) {
       array[i] = (□) malloc(sizeof(char)*□);
       if (array[i]==□) {
          printf("Alokace se nezdarila. Koncim.\n");
          for(i--;□;i--) {
             free(array[i]);
          }
          free(array);
          return 1;
       }
    }

    for(x=0; x<□; x++) {
       for(y=0; y<□; y++) {
          array[x][y]=' ';
       }
    }

    while(1) {
       for(□=0; □<height; □++) {
          for(□=0; □<width; □++) {
            printf(" %c",array[x][y]);
          }
          printf("\n");
       }
```

```
        printf("Zadejte x a y pozici pro hvezdicku: ");
        scanf("%i %i",&posx, &posy);

        if(posx<0 || posy<0 || posx>=□ || posy>=□) break;
        array[□][□]='*';
    }

    for(i=0;i<□;i++) {
        free(array[i]);
    }
    free(□);

    return 0;
}
```

3. Doplňte do souboru bmp.h deklaraci struktury (použijte typedef) sPixel, která ponese informace o jednom pixelu, tedy červenou, zelenou a modrou barevnou složku. Zvolte vhodný datový typ s ohledem na šetření paměti, jestliže počítáme s ukládáním 24bitových barev. Deklarujte v mainu proměnnou image, která bude dvourozměrným polem pixelů, jež se bude později alokovat dynamicky.

   Do souboru doplníme k ostatním strukturám následující kód. Datový typ nebudu prozrazovat vyjma informace, že obrazová data jsou bez znaménka (unsigned) a že 3 barevné složky dohromady mají dát 24 bitů, což jsou 3 B a proto každá složka má 1 B.

```
/* struct to store pixel data */
□ struct {
        unsigned □ red;
        unsigned □ green;
        unsigned □ blue;
        } sPixel;
```

   Deklarace proměnné image je vcelku jednoduchá. Jedná se o pole pixelů, čili ukazatel, který je ale dvojitý, vzhledem k dvojrozměrnosti pole (`sPixel ** image;`).

4. Do souboru bmp.h doplňte deklarace následujících funkcí, které také implementujte v souboru bmp.c.

```
void freeImage(sPixel □ image, picHeader ph);
```

   Funkce, která slouží k dealokaci pole pixelů image. Přebírá také strukturu obrazové hlavičky, ze které bude potřeba převzít informace o šířce a výšce alokovaného pole. Hlavička se tentokrát pro zjednodušení předává hodnotou (kopie), nikoli odkazem. V deklaraci sami zvažte, kolik hvězdiček je třeba uvést u image. Podotýkám, že pole je dvourozměrné a ukazatel samotný se měnit nebude, stačí jej (tento ukazatel) předat hodnotou. Uvnitř funkce předpokládejte, že je možné, že se pole nealokovalo celé, v takovém případě bude image[i] rovno NULL a nesmí se uvolňovat pomocí free!

```
int readBMPData(FILE * fp, fileHeader * fh, picHeader * ph, sPixel □
imageData);
```

Funkce bude sloužit ke čtení dat. Zatím ji neimplementujte! Pouze ji deklarujte a v bmp.c ji nechejte prázdnou. Zvažte, kolik hvězdiček musí být u sPixel image. Uvědomte si, že tentokrát je sice pole stále dvourozměrné, ale ukazatel na něj se bude měnit, protože se uvnitř funkce bude nově alokovat paměť. Proto se ukazatel na pole pixelů musí předávat ukazatelem ne hodnotou, jako předchozí funkci, což znamená…

```
int writeBMPData(FILE * fp, fileHeader * fh, picHeader * ph, sPixel □
imageData);
```

Funkce bude sloužit k ukládání dat. Zatím ji také neimplementujte! Pouze ji deklarujte a v bmp.c ji nechejte prázdnou. Také zvažte, kolik hvězdiček musí být u sPixel image. Ukazatel se tentokrát nebude měnit, stačilo by ho tedy předat hodnotou, nikoli odkazem. Přesto v zájmu sjednocení syntaxe pro zápis a čtení použijte stejně jako u čtení předání ukazatelem na ukazatel.

Počty hvězdiček prozrazovat nebudu, pouze podotýkám, že dvourozměrné pole je ukazatel na ukazatel. Pokud ho předáváme hodnotou, předáváme ukazatel na ukazatel. Je-li nutné předat jej ukazatelem, aby se dala změnit hodnota tohoto ukazatele na ukazatel (tj. adresa v paměti, kde se nachází pole pixelů), pak předáváme ukazatel na ukazatel na ukazatel. Implementace dealokace obrázku je následující:

```
void freeImage(sPixel □ image, picHeader ph) {
     unsigned int i=0;
     if (image==□) return;
     for (i=0; i<ph.□; i++) {
       if (image[i]!=□) free(image[i]);
     }
     free(image);
}
```

5. Do souboru bmp.h doplňte deklarace následujících funkcí, které také implementujte v souboru bmp.c.

```
int readBMPFile(char * fileName, □ fh, □ ph, sPixel □ imageData);
```

Funkce bude načítat celý soubor (hlavičku i data) a využije k tomu už existující funkce readBMPHeader a readBMPData (kterou musíme v dalším příkladě doprogramovat). Soubor bude otevírat podle jména v režimu binárního čtení. Zvažte sami, zda budete souborovou fh a obrázkovou hlavičku předávat jako příslušnou strukturu (hodnotou) nebo jako ukazatel na tuto strukturu (ukazatelem). K tomu připomínám, že obě hlavičky (jejich umístění) se sice nebude měnit, ale minimálně obrazová je poměrně velká a je možná zbytečné vytvářet pro běh funkce její kopii. Také obě funkce readBMPHeader a readBMPData, které se budou volat, vyžadují předání ukazatelem. Data (sPixel imageData) se zcela jistě měnit budou, proto toto dvojrozměrné pole musíme předat ukazatelem. Kolik bude hvězdiček? Pokud se nepodaří soubor otevřít, vraťte chybu ERR_INFILE, pokud nastanou nějaké chyby v readBMPHeader nebo readBMPData, vracejte jejich chybové kódy. Nezapomeňte v případě neúspěchu během nahrávání souboru dealokovat obrazové pole. Před koncem funkce vždy zavřete soubor (pokud byl otevřený). Při úspěchu vraťte ERR_OK.

```
int writeBMPFile(char * fileName, □ fh, □ ph, sPixel □ imageData);
```

Funkce bude zapisovat na disk celý soubor (hlavičku i data) a využije k tomu už existující funkce writeBMPHeader a writeBMPData (kterou musíme v dalším příkladě doprogramovat). Soubor bude otevírat podle jména v režimu binárního zápisu. K hlavičkám a datům platí stejný komentář jako u readBMPFile (zvažte výhodnost předávání ukazatelem). Data (sPixel imageData) se sice při zápisu měnit (nově alokovat) nebudou, ale v zájmu sjednocení syntaxe s čtecí funkcí použijte také předání ukazatele ukazatelem. Pokud se nepodaří soubor otevřít, vraťte chybu ERR_OUTFILE, pokud nastanou nějaké chyby ve writeBMPHeader nebo writeBMPData, vracejte jejich chybové kódy. Během ukládání nedealokujte obrazové pole! Může se používat dál i po uložení pro další úpravy. Před koncem funkce vždy zavřete soubor (pokud byl otevřený). Při úspěchu vraťte ERR_OK.

Implementace funkcí je překvapivě jednoduchá. Dokonce ani není potřeba řešit speciální podmínkou situaci, kdy se soubor nepodaří otevřít nebo vytvořit, protože požadovanou chybu vrací funkce readBMPHeader a writeBMPHeader, pokud se jim pokusíme předat ukazatel na soubor, který je neplatný (NULL). V místech □ opět chybí alespoň jeden znak. Podmínka u čtení a zápisu dat má zajistit, že se nepokusíme číst nebo zapisovat data, pokud již došlo k chybě během čtení či zápisu hlaviček.

```
int writeBMPFile(char * fileName, □ fh, □ ph, sPixel □ imageData) {
  FILE * file;
  int result = 0;

  file = fopen(fileName,□);
 result = writeBMPHeader(file, □, □);
  if (□) result = writeBMPData(file, □, □, imageData);
  if (file□NULL) fclose(file);

  return result;
}

int readBMPFile(char * fileName, □ fh, □ ph, sPixel □ imageData) {

  FILE * file;
  int result = 0;

  file = fopen(fileName,□);

  result = readBMPHeader(file, □, □);
  if (□) result = readBMPData(file, □, □, imageData);

  if (result == ERR_INFILE □
      result == ERR_NOTBMP □
      result == ERR_ALLOC  □
      result == ERR_DATA) freeImage(□imageData,□ph);
  if (file□NULL) fclose(file);

  return result;
}
```

6. Implementujte funkce pro zápis a čtení dat. Funkce readBMPData má nejprve testovat, zda předané ukazatele nejsou NULL. Pak musí alokovat dvourozměrné pole pixelů. Pokud se nepodařilo pole alokovat, dealokujeme alokované pomocí volání freeImage. Voláním fseek se přesune na začátek oblasti dat (viz offset obrazových dat v hlavičce). Následně načítá přímo obrazová data po jednotlivých bytech barev (pozor, v BMP souborech jsou složky v pořadí BGR) a také se začíná spodním levým pixelem (což nutně nemusíme řešit).

   Funkce writeBMPData také testuje předané ukazatele. Pak ale pouze zapisuje data v pořadí BGR. Nezapomeňte vrátit chybu ERR_ALLOC, pokud se nepodařilo při čtení alokovat paměť a ERR_DATA, pokud se nepodařilo načíst nebo uložit všechna data (součet zapsaných/přečtených záznamů neodpovídá trojnásobku počtu pixelů).

   Otestujte, zda aplikace funguje zatím na obrázcích s počtem bytů v řádku dělitelným 4. Např. 512 × 512 px. Dosud jsme totiž nevyřešili, že formát BMP přidává nulové byty k obrazovým datům tak, aby každý řádek měl počet bytů dělitelný 4. Stačí když pro zkoušku bude program překopírovávat soubor na soubor. Můžete ale také mezi čtením a zápisem pozměnit pár pixelů v poli obrazových dat (např. vynulovat).

   Obě funkce jsou poměrně krátké, ale složité na práci s ukazateli. Předáváním dat ukazatelem je často potřeba z ukazatele dereferencovat hodnotu (hvězdička – nikoliv ampersand – před jménem proměnné) Funkce pro čtení je navíc komplikovaná alokací dvourozměrného pole pixelů. Vzorové řešení opět tají některé podstatné části symbolem □.

```
int writeBMPData(FILE * fp, fileHeader * fh, picHeader * ph, sPixel □
imageData) {
  unsigned int x=0;
  unsigned int y=0;
  unsigned int ic=0;

  if(□==NULL □ □==NULL □ □==NULL □ □==NULL) return ERR_OUTFILE;

  for(y=0;y<ph->□;y++) {
          for(x=0;x<ph->□;x++) {
                  ic+=fwrite( □((□imageData)[x][y].blue), 1, 1, fp);
                  ic+=fwrite( □((□imageData)[x][y].green), 1, 1, fp);
                  ic+=fwrite( □((□imageData)[x][y].red), 1, 1, fp);
          }
  }

  if(ic!=3*ph->□*ph->□) return ERR_OUTFILE;
  return ERR_OK;
}

int readBMPData(FILE * fp, fileHeader * fh, picHeader * ph, sPixel □
imageData) {
  unsigned int x=0;
  unsigned int y=0;
  unsigned int ic=0;
```

```
if(□==NULL □ □==NULL □ □==NULL □ □==NULL) return ERR_INFILE;

*imageData = (□) malloc(ph->width*sizeof(□));
if(□==NULL) return ERR_ALLOC;

for(i=0;i<ph->width;i++) {
        (□imageData)[i] = (□) malloc(ph->height*sizeof(□));
        if ((□)[i]==NULL) {
           freeImage(□imageData, □ph);
           return ERR_ALLOC;
        }
}

fseek(fp,□,SEEK_□);

for(y=0;y<ph->□;y++) {
    for(x=0;x<ph->□;x++) {
                ic+=fread( □((□imageData)[x][y].blue), 1, 1, fp);
                ic+=fread( □((□imageData)[x][y].green), 1, 1, fp);
                ic+=fread( □((□imageData)[x][y].red), 1, 1, fp);
    }
}

if(ic!=3*ph->width*ph->height) {
   freeImage(□imageData, □ph);
   return ERR_DATA;
}
```

7. Do obou funkcí pro čtení i zápis dat doplňte kód, který ošetří dělitelnost počtu bytů v řádku obrazových dat čtyřmi (tj. bude načítat/zapisovat nuly navíc). Zkuste vymyslet vzorec, který z šířky řádku určí počet bytů v něm a pak správně vypočte číslo: Je-li počet bytů dělitelný čtyřmi (tj. zbytek po dělení počtu bytů čtyřmi je 0), pak se přidává 0 bytů. Je-li zbytek 1, přidávají se 3 byty, pro zbytek 2 se přidají 2, pro zbytek 3 se přidá 1.

   Výpočet je pěkný příklad komplikovanějšího použití operátoru % – zbytku po dělení. Uvědomíme si, že šířka řádku je v pixelech, nikoli v bytech. My ale máme jeden pixel velký 24 bitů, což je na byty… Tím dostáváme počet bytů v řádce. Určíme-li zbytek po dělení, dostáváme čísla: 0, 1, 2, 3, 4. My ale potřebujeme dostávat 0, 3, 2, 1. Pokud bychom zbytky odečítali od čtyř, máme řadu 4, 3, 2, 1. Zbývá odstranit problém s prvním číslem, které nemá být 4, ale 0, ostatní zůstávají. Jaká funkce změní 4 na 0, ale čísla 0 až 3 ponechá stejná? Následující cyklus patří do čtecí funkce do místa, kde bylo dokončeno čtení celého řádku pixelů. Hodnotu pixel nikde jinde nepoužijeme. Při zápisu ve funkci writeBMPData má být v proměnné pixel nula. Nezapomeňte deklarovat proměnné i a pixel. V řešení je symbol □ náhradou za jeden nějaký operátor a symbol ○ za nějaké jedno číslo.

```
for(i=0;i<(○□((ph->width□○)□○))□○;i++) {
     fread( &pixel, 1, 1, fp);
}
```

Úkoly

1. Napište program (pozpatku.c), který se uživatele zeptá, kolik chce zadat čísel. Pak čísla od uživatele načte a vypíše v opačném pořadí. Použijte dynamicky alokované pole čísel. Zkompilujte v konzoli a vyzkoušejte.

2. Napište program asciiart.c, který se nejprve uživatele zeptá na výšku a šířku „obrázku“, následně alokuje dvourozměrné pole znaků a naplní jej mezerami. V cyklu se ptejte uživatele na souřadnice pole, na které se vloží hvězdička. Pokud bude souřadnice mimo rozsah, program končí, jinak se vypíše pole na obrazovku a pokračuje se dalším dotazem. Nezapomeňte kontrolovat, zde se alokace zdařila, v případě neúspěchu dealokovat, což alokované bylo, stejně tak na konci programu. Zkuste takto nakreslit smajlíka z hvězdiček.

3. Doplňte do souboru bmp.h deklaraci struktury (použijte typedef) sPixel, která ponese informace o jednom pixelu, tedy červenou, zelenou a modrou barevnou složku. Zvolte vhodný datový typ s ohledem na šetření paměti, jestliže počítáme s ukládáním 24bitových barev. Deklarujte v mainu proměnnou image, která bude dvourozměrným polem pixelů, jež se bude později alokovat dynamicky.

4. Do souboru bmp.h doplňte deklarace následujících funkcí, které také implementujte v souboru bmp.c.

```
void freeImage(sPixel □ image, picHeader ph);
```

Funkce, která slouží k dealokaci pole pixelů image. Přebírá také strukturu obrazové hlavičky, ze které bude potřeba převzít informace o šířce a výšce alokovaného pole. Hlavička se tentokrát pro zjednodušení předává hodnotou (kopie), nikoli odkazem. V deklaraci sami zvažte, kolik hvězdiček je třeba uvést u image. Podotýkám, že pole je dvourozměrné a ukazatel samotný se měnit nebude, stačí jej (tento ukazatel) předat hodnotou. Uvnitř funkce předpokládejte, že je možné, že se pole nealokovalo celé, v takovém případě bude image[i] rovno NULL a nesmí se uvolňovat pomocí free!

```
int readBMPData(FILE * fp, fileHeader * fh, picHeader * ph, sPixel □
imageData);
```

Funkce bude sloužit ke čtení dat. Zatím ji neimplementujte! Pouze ji deklarujte a v bmp.c ji nechejte prázdnou. Zvažte, kolik hvězdiček musí být u sPixel image. Uvědomte si, že tentokrát je sice pole stále dvourozměrné, ale ukazatel na něj se bude měnit, protože se uvnitř funkce bude nově alokovat paměť. Proto se ukazatel na pole pixelů musí předávat ukazatelem ne hodnotou, jako předchozí funkci, což znamená…

```
int writeBMPData(FILE * fp, fileHeader * fh, picHeader * ph, sPixel □
imageData);
```

Funkce bude sloužit k ukládání dat. Zatím ji také neimplementujte! Pouze ji deklarujte a v bmp.c ji nechejte prázdnou. Také zvažte, kolik hvězdiček musí být u sPixel image. Ukazatel se tentokrát nebude měnit, stačilo by ho tedy předat hodnotou, nikoli odkazem. Přesto v zájmu sjednocení syntaxe pro zápis a čtení použijte stejně jako u čtení předání ukazatelem na ukazatel.

5. Do souboru bmp.h doplňte deklarace následujících funkcí, které také implementujte v souboru bmp.c.

   ```
   int readBMPFile(char * fileName, □ fh, □ ph, sPixel □ imageData);
   ```

   Funkce bude načítat celý soubor (hlavičku i data) a využije k tomu už existující funkce readBMPHeader a readBMPData (kterou musíme v dalším příkladě doprogramovat). Soubor bude otevírat podle jména v režimu binárního čtení. Zvažte sami, zda budete souborovou fh a obrázkovou hlavičku předávat jako příslušnou strukturu (hodnotou) nebo jako ukazatel na tuto strukturu (ukazatelem). K tomu připomínám, že obě hlavičky (jejich umístění) se sice nebude měnit, ale minimálně obrazová je poměrně velká a je možná zbytečné vytvářet pro běh funkce její kopii. Také obě funkce readBMPHeader a readBMPData, které se budou volat, vyžadují předání ukazatelem. Data (sPixel imageData) se zcela jistě měnit budou, proto toto dvojrozměrné pole musíme předat ukazatelem. Kolik bude hvězdiček? Pokud se nepodaří soubor otevřít, vraťte chybu ERR_INFILE, pokud nastanou nějaké chyby v readBMPHeader nebo readBMPData, vracejte jejich chybové kódy. Nezapomeňte v případě neúspěchu během nahrávání souboru dealokovat obrazové pole. Před koncem funkce vždy zavřete soubor (pokud byl otevřený). Při úspěchu vraťte ERR_OK.

   ```
   int writeBMPFile(char * fileName, □ fh, □ ph, sPixel □ imageData);
   ```

   Funkce bude zapisovat na disk celý soubor (hlavičku i data) a využije k tomu už existující funkce writeBMPHeader a writeBMPData (kterou musíme v dalším příkladě doprogramovat). Soubor bude otevírat podle jména v režimu binárního zápisu. K hlavičkám a datům platí stejný komentář jako u readBMPFile (zvažte výhodnost předávání ukazatelem). Data (sPixel imageData) se sice při zápisu měnit (nově alokovat) nebudou, ale v zájmu sjednocení syntaxe s čtecí funkcí použijte také předání ukazatele ukazatelem. Pokud se nepodaří soubor otevřít, vraťte chybu ERR_OUTFILE, pokud nastanou nějaké chyby ve writeBMPHeader nebo writeBMPData, vracejte jejich chybové kódy. Během ukládání nedealokujte obrazové pole! Může se používat dál i po uložení pro další úpravy. Před koncem funkce vždy zavřete soubor (pokud byl otevřený). Při úspěchu vraťte ERR_OK.

6. Implementujte funkce pro zápis a čtení dat. Funkce readBMPData má nejprve testovat, zda předané ukazatele nejsou NULL. Pak musí alokovat dvourozměrné pole pixelů. Pokud se nepodařilo pole alokovat, dealokujeme alokované pomocí volání freeImage. Voláním fseek se přesune na začátek oblasti dat (viz offset obrazových dat v hlavičce). Následně načítá přímo obrazová data po jednotlivých bytech barev (pozor, v BMP souborech jsou složky v pořadí BGR) a také se začíná spodním levým pixelem (což nutně nemusíme řešit).

   Funkce writeBMPData také testuje předané ukazatele. Pak ale pouze zapisuje data v pořadí BGR. Nezapomeňte vrátit chybu ERR_ALLOC, pokud se nepodařilo při čtení alokovat paměť a ERR_DATA, pokud se nepodařilo načíst nebo uložit všechna data (součet zapsaných/přečtených záznamů neodpovídá trojnásobku počtu pixelů).

   Otestujte, zda aplikace funguje zatím na obrázcích s počtem bytů v řádku dělitelným 4. Např. 512 × 512 px. Dosud jsme totiž nevyřešili, že formát BMP přidává nulové byty k obrazovým datům tak, aby každý řádek měl počet bytů dělitelný 4. Stačí když pro zkoušku bude program překopírovávat soubor na soubor. Můžete ale také mezi čtením a zápisem pozměnit pár pixelů v poli obrazových dat (např. vynulovat).

7. Do obou funkcí pro čtení i zápis dat doplňte kód, který ošetří dělitelnost počtu bytů v řádku obrazových dat čtyřmi (tj. bude načítat/zapisovat nuly navíc). Zkuste vymyslet vzorec, který z šířky řádku určí počet bytů v něm a pak správně vypočte číslo: Je-li počet bytů dělitelný čtyřmi (tj. zbytek po dělení počtu bytů čtyřmi je 0), pak se přidává 0 bytů. Je-li zbytek 1, přidávají se 3 byty, pro zbytek 2 se přidají 2, pro zbytek 3 se přidá 1.