DUM č. 15 v sadě

35. Inf-11 Objektové programování v Greenfoot

Autor: Lukáš Rýdlo

Datum: 08.06.2014

Ročník: studenti semináře

Anotace DUMu: Úvodní DUM k třetímu projektu - BMP editor v příkazové řádce. Načítání a ukládání hlavičky BMP. Zobrazení informací o souboru.

esf 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 hlaviček BMP souboru

Úvod k projektu

Třetí projekt v sadě DUMů je konzolová aplikace v jazyce C. Má ukázat multiplatformní zápis kódu a kompilaci a spuštění programu na různých systémech, také má studenty provést vynikem aplikace od textového souboru až po binárku. Proto je žádoucí nepoužívat pro tento projekt žádné IDE, ale psát jej pouze v obyčejném textovém editoru. V GNU/Linuxu to mohou být: vim, emacs, nano, mcedit nebo grafické gedit či leafpad. V MS Windows bych volil PSPad nebo Notepad+, lze i obyčejný Notepad či jiný program, který slouží k tvorbě souborů čistého textu (txt).

Projekt je poměrně dost náročný a obsahuje složitá témata (jako je dynamická alokace víceroměrných polí, předávání struktur mezi funkcemi apod.), takže některé úkoly budou studenti potřebovat nejspíš procvičit nejprve na jednodušším kódu. Pokud nemají příliš zkušeností s programováním v C, může se jim řešení zdát nad jejich síly, což svádí k prozrazování řešení. Nikdy ale neprozrazujeme řešení celé, pouze část, která je nezbytně nutná pro pochopení, a studenta necháme zbytek udělat jako procvičení (např. prozradíme postup tvorby a načítání struktury pro první – souborovou – hlavičku, ale studenta necháme zcela samostatně vytvořit druhou – obrazovou, komplikovanější).

Projekt bývá mezi studenty (obzvláště šikovnějšími) oblíbený, protože má zajímavé výsledky – dokáže upravovat známý obrázkový formát a studenti pochopí strukturu ukládání obrázků v souborech na disku.

Nemáme-li k dispozici počítač s Linuxem, můžeme použít liveCD distribuce Knoppix, které najdeme na adrese http://knopper.net/knoppix/ – v době psaní tohoto textu byl aktuální soubor s obrazem KNOPPIX_V7.0.5CD-2012-12-21-EN.iso a bylo možné jej přímo stáhnout například z mirroru TU Wien z adresy http://gd.tuwien.ac.at/linux/knoppix/KNOPPIX_V7.0.5CD-2012-12-21-EN.iso. Z něj mohou nastartovat a pracovat studenti i doma, práci si pak musí ukládat buď na flashdisk nebo v nějaké službě na síti (Dropbox, Google Disk apod.). Je rozumné se studenty v hodině projít spuštění systému z LiveCD (nastavení bootovacích priorit v BIOSu) a základy ovládání a adresářové struktury, na kterou mnozí nebudou zvyklí. Systém lze pochopitelně spustit bez vypalování přímo v nějakém virtualizačním nástroji, třeba ve volně šiřitelném VirtualBoxu (http://www.virtualbox.org).

Schopní studenti, kteří už umí alespoň základy C mohou psát kód na papír místo do počítače a vyučující jim pak pouze odpovídá na dotaz, zda je či není v kódu chyba, případně na kterém řádku. Takto se naučí kódu opravdu rozumět a ne jen „zkoušet, jestli to projde kompilátorem“.

Úkoly s řešeními

1. Vytvořte nový soubor bmp.c a v něm ve funkci main se dotažte uživatele na adresu souboru. Tento soubor otevřete v režimu binárního čtení a příslušnou funkcí pro binární čtení načtěte první dva byty ze souboru, každý zvlášť do proměnné typu char nebo do dvouprvkového pole charů (zkuste obě možnosti). Jsou-li tyto dva chary znaky 'B' a 'M', pak vypiště „Soubor je BMP“, nejsou-li, vypiště „Soubor neni BMP“. Pokud nebylo možné soubor otevřít, vypište „Soubor nelze otevrit“.

   Jednoduchý příklad na procvičení binárního čtení. Zdrojový kód má vynechaný poslední parametr funkce fread, který každý student jistě správně doplní úvahou nebo zjištěním v dokumentaci: http://www.tutorialspoint.com/c_standard_library/c_function_fread.htm, http://stackoverflow.com/questions/8589425/how-does-fread-really-work, http://www.sallyx.org/sally/c/c33.php. Zakomentované řádky obsahují variantu s polem, povšimneme si, že v fread pak nepoužíváme znak &, protože název pole je sám o sobě ukazatel a není potřeba ukazatel dereferencovat.

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

int main(int argc, char * argv[]) {
   FILE * fp;
   char filename[255];
   char buffer1, buffer2; //char buffer[2];

   printf("Zadej nazev souboru: ");
   scanf("%s",filename);

   if((fp=fopen(□,"rb"))==NULL) {
       printf("Soubor nelze otevrit.\n");
       return 1;
   }

   fread(&buffer1, 1, 1, □);
   fread(&buffer2, 1, 1, □);
// fread(buffer,1,2,□);

   if (buffer1=='B' && buffer2=='M') {
// if (buffer[0]=='B' && buffer[1]=='M') {
      printf("Soubor je BMP.\n");
   } else {
      printf("Soubor neni BMP.\n");
   }

   fclose(fp);
   return 0;
}
```

2. V předchozím příkladu doplňte počítání počtu načtených záznamů (pokud jste tak již neučinili) a v podmínce přidejte, že musely být načtené opravdu dva záznamy, aby bylo možné soubor považovat za BMP (počet načtených záznamů vrací funkce fread, viz dokumentace). Vytvořte několik testovacích souborů: zcela prázdný, textový, obsahující jen znaky „BM“, bitmapový obrázek, jiný soubor (např. PNG obrázek). Program zkompilujte na příkazové řádce pomocí kompilátoru GCC a spusťte. Otestujte soubory.

   Požadovaná úprava počítání načtených záznamů slouží k ověření, že se záznamy opravdu načetly a nedošlo dříve na konec souboru (nebo jeho poškození). Stačí vytvořit novou proměnnou int bytecount = 0; a připočítávat bytecount += fread(…). Doplnění podmínky snad netřeba vysvětlovat.

   Samotná kompilace na příkazové řádce je jednoduchá, stačí v konzoli spustit kompilátor příkazem `gcc -o program bmp.c`*. Je dobré studenty nechat nastudovat parametry programu pomocí manuálových stránek příkazem* `man gcc`*. Ve Windows je třeba mít instalaci gcc (např. jako součást instalace Dev-C) a spouštět* `gcc.exe -o program bmp.c` *s případným doplněním cest, kde se soubor či gcc.exe nachází. V Linuxu ještě zkompilovanému programu přidáme právo spuštění příkazem* `chmod a+x program` *a spustíme* `./program`*. Je možné, že program uložený na flashdisku nepůjde spustit kvůli bezpečnostnímu nastavení systému, pak jej musíme překopírovat na disk.*

3. Přečtěte si na serveru root.cz článek http://www.root.cz/clanky/graficky-format-bmp-pouzivany-a-pritom-neoblibeny/ a vytvořte struct pro souborovou a struct pro obrazovou hlavičku. Obě hlavičky načtěte a pak doplňte předchozí program tak, aby za BMP soubor považoval jen takový, který úspěšně načte celé obě dvě hlavičky (správný počet záznamů) a vypište: rozměry obrázku, počet bitů na pixel (obvykle 24) a zda je soubor komprimován. Program překompilujte a ověřte správnou činnost na několika obrázcích. Obrázky a jejich strukturu si také můžete „ručně“ prohlédnout a zkusit pozměnit v jejich binární podobě některým z programů ghex2 nebo hexdump (Linux) či PSPad (Windows).

   Je nezbytně nutné pokusit se porozumět zmíněnému článku, s čímž může pomoci právě ruční procházení souboru v některém hex-editoru a také třeba změna údajů (přepsat v hlavičce šířku a výšku z 512 × 512 px na 256 × 1024 px a 1024 × 256 px a dívat se, jak se obrázek změnil. Upravit hodnoty dat (změnit barvu pixelu).

   Vzorově ukazuji řešení pouze pro souborovou hlavičku, vytvoření a načtení hlavičky obrazové musí zvládnout student sám, pokud pochopil princip structu a binárního čtení. Poslední parametr fread student již dohledal v předchozím řešení a vynechaná čísla velikosti záznamu a počtu záznamů při načítání velikosti a offsetu bystrý student doplní bez dokumentace…

```
//definice typu v záhlaví programu
typedef struct {
        unsigned char type[2];
        unsigned int size;
        unsigned short reserved[2];
        unsigned int offs;
        } fileHeader;
```

```
//načítání hlavičky po otevření souboru v mainu
fileHeader fh;
bytecount += fread( fh.type, 1, 2, □);
bytecount += fread( &(fh.size), □, 1, □);
bytecount += fread( fh.reserved, 2, 2, □);
bytecount += fread( &(fh.offs), 4, □, □);
if (bytecount == 6 && fh.□[0] == 'B' && fh.□[0] == 'M' ) {
      printf("Soubor je BMP.\n");
}
```

Analogicky se vytvoří struktura a načte i obrazová hlavička, pak stačí data z hlavičky vypsat. Test, zda se jedná o BMP soubor lze udělat až po kompletním načtení hlaviček.

4. Přepište kód do funkcí. Vytvořte hlavičkový bmp.h soubor. V něm nechť je deklarace struktur, definice konstanty _BMP_H a vypodmínkování kódu (direktivou preprocesoru #ifdef nebo #ifndef), pokud je již tato direktiva zavedena (předejde to opětovnému vložení hlavičkového souboru). Deklarujte v hlavičkovém souboru direktivy pro chybové stavy a zvažte, proč zvolená čísla chyb nejsou popořadě o jedna větší:

```
ERR_OK      0   //vše ok
ERR_ARG     1   //je potřeba více parametrů (bude potřeba později)
ERR_INFILE  2   //nelze otevřít vstupní soubor
ERR_NOTBMP  4   //otvíraný soubor není BMP
ERR_ALLOC   8   //nelze alokovat paměť (bude potřeba později)
ERR_DATA    16  //příliš málo obrazových dat (bude potřeba později)
ERR_OUTFILE 32  //nelze otevřít nebo zapsat výstupní soubor
```

Deklarujte v hlavičkovém souboru a v souboru bmp.c implementujte funkce (použijte je pak v main, aby si zachoval svou funkci) – nezapomeňte includovat bmp.h:

```
void printHeaderInfo(fileHeader * fh, picHeader * ph);
```

Vypíše textově všechny informace z hlaviček, které předáte již načtené v daných strukturách pomocí ukazatelů na tyto struktury, aby se nemusely v paměti překopírovávat (šetří to RAM). Nezapomeňte, že položky struktury čteme operátorem tečka, když jde o strukturu, a operátorem „šipka“, když jde o ukazatel (rozdíl `fh.size` a `fh->size`). Nezapomeňte otestovat, že fh a ph nebyly předány jako NULL. V takovém případě nevypisujte nic, ale program nesmí skončit chybou.

```
int writeBMPHeader(FILE * fp, fileHeader * fh, picHeader * ph);
```

Zapíše do (již otevřeného) souboru předaného ukazatelem obě hlavičky. Opět pozor na operátor pro získání položek struktury, když předáváme ukazatel. Uvědomte si, že délka zapsané picHeader (obrazové) hlavičky je pevně daná tím, co všechno zapisujete a ne údajem o velikosti hlavičky, který jsme načetli. GIMP a některé jiné programy uvádějí do hlavičky další dodatečná data, která nejsou ve standardu. Pokud zapíšete délku, kterou jste načetli (delší), ale pak už nedoplníte další data na konci hlavičky (což pro naše potřeby není nutné), nebude odpovídat uložená délka množství reálně zapsaných dat. Nezapomeňte testovat, že fp, fh ani ph nejsou NULL, pak vracejte chybu ERR_OUTFILE. Při úspěšném dokončení vraťte ERR_OK.

```
int readBMPHeader(FILE * fp, fileHeader * fh, picHeader * ph);
```

Načítejte z již otevřeného souboru předaného ukazatelem obě hlavičky. Pokud některý z argumentů je NULL, vracejte ERR_INFILE, pokud načtený typ souboru není BMP (kvůli prvním dvěma znakům nebo proto, že se nepodařilo načíst hlavičku celou), vracejte ERR_NOTBMP. Při úspěšném dokončení vraťte ERR_OK.

Řešení tohoto úkolu je rozsáhlá „mravenčí“ práce, kterou označíme jako refactoring stávajícího kódu a zabere mnoho času. Vzorové řešení uvádím, ale vynechávám některé údaje (slova, proměnné, hodnoty), které by měl student sám domyslet a nechci je prozrazovat. Soubor bmp.h vypadá takto:

```
#□ _BMP_H
#define _BMP_H

#□ ERR_OK      0   //return ok
#□ ERR_ARG     1   //need more arguments
#□ ERR_INFILE  2   //cannot open input file
#□ ERR_NOTBMP  4   //not a BMP file (bad format)
#□ ERR_ALLOC   8   //cannot allocate memory
#□ ERR_DATA    16  //file too small (premature end of data)
#□ ERR_OUTFILE 32  //cannot open output file

/* struct to store bmp file header */
typedef struct {
        unsigned □ type[2];
        unsigned □ size;
        unsigned □ reserved[2];
        unsigned □ offs;
        } fileHeader;

/* struct to store bmp image header */
typedef struct {
        □ □ size;
        □ □ width;
        □ □ height;
        □ □ planes;
        □ □ bitCount;
        □ □ compression;
        □ □ sizeImage;
        □ □ XperMeter;
        □ □ YperMeter;
        □ □ used;
        □ □ important;
        } □;

void printHeaderInfo(fileHeader * fh, picHeader * ph);
int writeBMPHeader(FILE * fp, fileHeader * fh, picHeader * ph);
int readBMPHeader(FILE * fp, fileHeader * fh, picHeader * ph);

#endif
```

Příslušný bmp.c soubor by mohl (opět s vynechávkami znakem □) vypadat takto:

```
#include <stdio.h>
#include <stdlib.h>
#□ "bmp.h"

void printHeaderInfo(fileHeader * fh, picHeader * ph) {

   if(□ == □ □ ph == NULL) return;

   printf("--- Printing headers ---\n** file header\n");
   printf("Type:     %c%c [%x %x]\n",fh□type[0],fh□type[1],□,□);
   printf("Size:     %u\n",fh□size);
   printf("Reserved: %u %u\n",fh□reserved[0],fh□reserved[1]);
   printf("Offs:     %u\n\n** image header\n",fh□offs);

   printf("Size:        %□\n",ph□size);
   printf("Width:       %□\n",ph□width);
   printf("Height:      %□\n",ph□height);
   printf("Planes:      %□\n",ph□planes);
   printf("BitCount:    %□\n",ph□bitCount);
   printf("Compression: %□\n",ph□compression);
   printf("SizeImage:   %□\n",ph□sizeImage);
   printf("XperMeter:   %□\n",ph□XperMeter);
   printf("YperMeter:   %□\n",ph□YperMeter);
   printf("Used:        %□\n",ph□used);
   printf("Important:   %□\n",ph□important);
   printf("--- headers end---\n\n");
}

int writeBMPHeader(FILE * fp, fileHeader * fh, picHeader * ph) {
  int ic=0;
  unsigned int newSize; //we write allways 54 B long header

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

  ic+=fwrite( fh□type, 1, 2, □);
  ic+=fwrite( &(fh->size), 4, 1, □);
  ic+=fwrite( fh->reserved, 2, 2, □);
  newSize = 54;
  ic+=fwrite( &newSize, 4, 1, □);

  newSize = 40;
  ic+=fwrite( □newSize, 4, 1, □);
  ic+=fwrite( □(ph□width), 4, 1, □);
  ic+=fwrite( □(ph□height), 4, 1, □);
  ic+=fwrite( □(ph□planes), 2, 1, □);
  ic+=fwrite( □(ph□bitCount), 2, 1, □);
  ic+=fwrite( □(ph□compression), 4, 1, □);
```

```
  ic+=fwrite( □(ph□sizeImage), 4, 1, □);
  ic+=fwrite( □(ph□XperMeter), 4, 1, □);
  ic+=fwrite( □(ph□YperMeter), 4, 1, □);
  ic+=fwrite( □(ph□used), 4, 1, □);
  ic+=fwrite( □(ph□important), 4, 1, □);

  if(ic!=□) return ERR_OUTFILE;
  return ERR_OK;
}

int readBMPHeader(FILE * fp, fileHeader * fh, picHeader * ph) {
  int ic = 0;

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

  ic += fread( fh□type, 1, 2, □);
  ic += fread( &(fh□size), 4, 1, □);
  ic += fread( fh□reserved, 2, 2, □);
  ic += fread( &(fh□offs), 4, 1, □);

  if (□[0] != 'B' || □[1] != 'M') return ERR_NOTBMP;

  ic += fread( □(ph□size), 4, 1, □);
  ic += fread( □(ph□width), 4, 1, □);
  ic += fread( □(ph□height), 4, 1, □);
  ic += fread( □(ph□planes), 2, 1, □);
  ic += fread( □(ph□bitCount), 2, 1, □);
  ic += fread( □(ph□compression), 4, 1, □);
  ic += fread( □(ph□sizeImage), 4, 1, □);
  ic += fread( □(ph□XperMeter), 4, 1, □);
  ic += fread( □(ph□YperMeter), 4, 1, □);
  ic += fread( □(ph□used), 4, 1, □);
  ic += fread( □(ph□important), 4, 1, □);

  if(ic!=□) return ERR_NOTBMP;

  return ERR_OK;
}

int main(int argc, char *argv[])
{
  char filename[255];
  fileHeader fh;
  picHeader  ph;

  printf("Zadej soubor: ");
  scanf("%s",□);

  if((fp=fopen(□,"rb"))==NULL) {
```

```
        printf("Soubor nelze otevrit.\n");
        return ERR_INFILE;
    }

    readBMPHeader(□,□fh,□ph);
    printHeaderInfo(□fh,□ph);

    getchar();

    return ERR_OK;
}
```

Úkoly s řešeními

1. Vytvořte nový soubor bmp.c a v něm ve funkci main se dotažte uživatele na adresu souboru. Tento soubor otevřete v režimu binárního čtení a příslušnou funkcí pro binární čtení načtěte první dva byty ze souboru, každý zvlášť do proměnné typu char nebo do dvouprvkového pole charů (zkuste obě možnosti). Jsou-li tyto dva chary znaky 'B' a 'M', pak vypiště „Soubor je BMP“, nejsou-li, vypiště „Soubor neni BMP“. Pokud nebylo možné soubor otevřít, vypište „Soubor nelze otevrit“.

2. V předchozím příkladu doplňte počítání počtu načtených záznamů (pokud jste tak již neučinili) a v podmínce přidejte, že musely být načtené opravdu dva záznamy, aby bylo možné soubor považovat za BMP (počet načtených záznamů vrací funkce fread, viz dokumentace). Vytvořte několik testovacích souborů: zcela prázdný, textový, obsahující jen znaky „BM“, bitmapový obrázek, jiný soubor (např. PNG obrázek). Program zkompilujte na příkazové řádce pomocí kompilátoru GCC a spusťte. Otestujte soubory.

3. Přečtěte si na serveru root.cz článek http://www.root.cz/clanky/graficky-format-bmp-pouzivany-a-pritom-neoblibeny/ a vytvořte struct pro souborovou a struct pro obrazovou hlavičku. Obě hlavičky načtěte a pak doplňte předchozí program tak, aby za BMP soubor považoval jen takový, který úspěšně načte celé obě dvě hlavičky (správný počet záznamů) a vypište: rozměry obrázku, počet bitů na pixel (obvykle 24) a zda je soubor komprimován. Program překompilujte a ověřte správnou činnost na několika obrázcích. Obrázky a jejich strukturu si také můžete „ručně“ prohlédnout a zkusit pozměnit v jejich binární podobě některým z programů ghex2 nebo hexdump (Linux) či PSPad (Windows).

4. Přepište kód do funkcí. Vytvořte hlavičkový bmp.h soubor. V něm nechť je deklarace struktur, definice konstanty _BMP_H a vypodmínkování kódu (direktivou preprocesoru #ifdef nebo #ifndef), pokud je již tato direktiva zavedena (předejde to opětovnému vložení hlavičkového souboru). Deklarujte v hlavičkovém souboru direktivy pro chybové stavy a zvažte, proč zvolená čísla chyb nejsou popořadě o jedna větší:

```
ERR_OK      0   //vše ok
ERR_ARG     1   //je potřeba více parametrů (bude potřeba později)
ERR_INFILE  2   //nelze otevřít vstupní soubor
ERR_NOTBMP  4   //otvíraný soubor není BMP
ERR_ALLOC   8   //nelze alokovat paměť (bude potřeba později)
ERR_DATA    16  //příliš málo obrazových dat (bude potřeba později)
ERR_OUTFILE 32  //nelze otevřít nebo zapsat výstupní soubor
```

   Deklarujte v hlavičkovém souboru a v souboru bmp.c implementujte funkce (použijte je pak v main, aby si zachoval svou funkci) – nezapomeňte includovat bmp.h:

```
void printHeaderInfo(fileHeader * fh, picHeader * ph);
```

Vypíše textově všechny informace z hlaviček, které předáte již načtené v daných strukturách pomocí ukazatelů na tyto struktury, aby se nemusely v paměti překopírovávat (šetří to RAM). Nezapomeňte, že položky struktury čteme operátorem tečka, když jde o strukturu, a operátorem „šipka“, když jde o ukazatel (rozdíl `fh.size` a `fh->size`). Nezapomeňte otestovat, že fh a ph nebyly předány jako NULL. V takovém případě nevypisujte nic, ale program nesmí skončit chybou.

```
int writeBMPHeader(FILE * fp, fileHeader * fh, picHeader * ph);
```

Zapíše do (již otevřeného) souboru předaného ukazatelem obě hlavičky. Opět pozor na operátor pro získání položek struktury, když předáváme ukazatel. Uvědomte si, že délka zapsané picHeader (obrazové) hlavičky je pevně daná tím, co všechno zapisujete a ne údajem o velikosti hlavičky, který jsme načetli. GIMP a některé jiné programy uvádějí do hlavičky další dodatečná data, která nejsou ve standardu. Pokud zapíšete délku, kterou jste načetli (delší), ale pak už nedoplníte další data na konci hlavičky (což pro naše potřeby není nutné), nebude odpovídat uložená délka množství reálně zapsaných dat. Nezapomeňte testovat, že fp, fh ani ph nejsou NULL, pak vracejte chybu ERR_OUTFILE. Při úspěšném dokončení vraťte ERR_OK.

```
int readBMPHeader(FILE * fp, fileHeader * fh, picHeader * ph);
```

Načítejte z již otevřeného souboru předaného ukazatelem obě hlavičky. Pokud některý z argumentů je NULL, vracejte ERR_INFILE, pokud načtený typ souboru není BMP (kvůli prvním dvěma znakům nebo proto, že se nepodařilo načíst hlavičku celou), vracejte ERR_NOTBMP. Při úspěšném dokončení vraťte ERR_OK.