Képfeldolgozó – programozói dokumentáció

Szatmáry Zoltán

Modulokra bontás

- ***main:*** *A main modul alkotja a főprogramrészt, melynek feladata, hogy a parancssori argumentumokat figyelembe véve meghívja a többi modul megfelelő képbetöltő és képmanipulációs függvényeit, valamint hiba esetén a hibakódot továbbítsa a status modul felé.*
- ***image:*** *Az image modul implementálja a képek létrehozását, felszabadítását, valamint a képeken végzett műveleteket megvalósító függvényeket.*
- ***bmp:*** *A bmp modul intézi a BMP formátum betöltésével és fájlba mentésével kapcsolatos teendőket, így a formátumok helyességének ellenőrzését is.*
- ***cmd:*** *A cmd modul valósítja meg a parancssori argumentumok által megadható kapcsolók/jelzések értelmezését.*
- ***status:*** *A status modul az összes többi modul által generált hibakódok kiírását végzi.*

Képek kezelése absztrakcióval

Minden beolvasott képet absztrakcióval kezel a program, ami azt jelenti, hogy formátumtól független módon tárolja a képet, amelyen így aztán különböző általánosan megfogalmazható transzformációk valósíthatók meg. Ennek a megoldásnak nagy előnye, hogy amennyiben a BMP-n kívül más képformátumok támogatása is szükségessé válna, csak egy beolvasást és kiírást implementáló modult kéne megírni.

A képek kezelését tovább egyszerűsíti a struktúra kialakítása, mely a képpontokat tároló pixelmátrixot nem csak bonyolult képletekkel lehet megindexelni – „$képpontok[sor \cdot oszlopok\_száma + oszlop]$” alakban, hanem lehetőség van kétszeres indirekció alkalmazására is, hogy a elég legyen egy koordinátákapár egy képpont eléréséhez – tehát „$képpontok[sor][oszlop]$” alakban is indexelhető legyen a mátrix.

A BMP formátum

Az eredeti, szabványos BMP formátum meglehetősen egyszerű kialakítású.

Két fejlécből, opcionálisan egy színtáblázatból, valamint a bittérképből (a tényleges pixeladatok „felsorolásából”) áll.

Mező	Méret (bájt)
Fájlfejléc	14

Szignatúra	2
Fájlméret	4
Fenntartott mező	4
Bittérkép kezdetének ofszetje	4
Információs fejléc	**40**
Információs fejléc mérete	4
Képszélesség	4
Képmagasság	4
Megjelenítés (monitor, …)	2
Bitmélység (1, 4, 8, 24)	2
Tömörítés (Ø, RLE4, RLE8)	4
Bittérkép mérete	4
Vízszintes felbontás (pixel/méter)	4
Függőleges felbontás (pixel/méter)	4
Színtáblázat (0 vagy méret)	4
Használt színek	4
Színtáblázat	színtáblázat×4
Kék	1
Zöld	1
Piros	1
Fenntartott mező	1
Bittérkép	-

• Sorok (4 bájtos padding-gel)
  ○ Pixelek (annyi bit jelent egy pixelt, amennyi bitmélység)
  a) Ha a bitmélység egyenlő eggyel, akkor 1-es vagy 0-s bit, annak megfelelően, hogy a színtáblázat egyetlen bejegyzésében szereplő színét vagy a feketét reprezentálják a pixel bitjei.
  b) Ha a bitmélység kisebb vagy egyenlő nyolccal, akkor a színtáblázat adott színének indexét kell érteni a pixelek bitjei alatt.
  c) Ha a bitmélység nagyobb mint nyolc, akkor konkrét RGB színkódok tárolódnak a bitekben, B-G-R sorrendben.
	-

A beolvasás lépései

A két fejlécet külön-külön egy segédfüggvénnyel olvassa be a program. Azért nem lehet egyetlen egy szabványos `fread()`-del beolvasni egy egész fejlécet egy

struktúrába, mert a bár a fordító a struktúrákban lévő mezőket nem rendezheti át, ám kitöltő bájtokat beszúrhat helyenként, hogy a struktúra kezelése a CPU számára a legoptimálisabb legyen; így viszont nem garantált, hogy pl. a fájlban a 3. bájton lévő fejléc elem a struktúra 3. bájtjaként lesz elérhető.

A fejlécek értelmezése és validálása után beolvassa a színtáblázatot, ha van, majd rögtön a fejlécben is meghatározott bittérkép kezdeti pozíciójától folytatódik a fájl értelmezése.

A sorokat egyenként olvassa be a program egy ideiglenes sorbufferbe, amit aztán a bitmélységnek megfelelő bitenként értelmezi és RGB színkódként eltárol a pixelmátrixban.

A kiírás lépései

A fájlba történő kiírás ugyanazokon az elemeken megy végig, mint a beolvasás, csak a másik irányból, és némi egyszerűsítéssel.

A program először létrehoz egy-egy struktúrát az információs- valamint a fájlfejlécnek, melynek attribútumait részben a kiírandó kép jellemző tulajdonságaival, részben pedig előre meghatározott konstansokkal állít be (pl. a kiírt kép bitmélysége mindig 24 bites), majd szintén segédfüggvények felhasználásával kiírja őket a fájlba.

Ezek után

Hibakezelés

A program mind a felhasználó, mind a környezet által generált kivételeket képes kezelni, mégpedig úgy, hogy a hibát generálni képes függvények státusz `int`-ekkel (vagy a lefoglalt területre mutató pointerrel; ilyen esetben `NULL` pointer jelzi a hibát) térnek vissza, tehát ha valahol generálódik egy kivétel, az mindig visszagörgethető lesz a legfelső hívásig, a `main`-beli függvényhívásig, amely a megfelelő felszabadítási munkálatok elvégzése után meghívja a hibakód-értelmező szubrutint, a `status_print`-et.

A `status_print` arra épít, hogy a hibakódot generáló modulok illeszkednek az által elvárt hibakezelési elvhez, vagyis a modul rendelkezik egy legfeljebb ezer különböző kódot generál, amelyek mind a modul ofszetjétől helyezkednek el egy folytonosan az 1000 egészből álló számtartományban, továbbá a számokhoz definiálva van egy $(ofszet - 1000)$-rel indexelhető kód - hibaszöveg párosítást leíró sztringtömb.

A `status.h` ezenkívül általánosan felhasználható hibakódokat is nyújt a többi modul számára, így elkerülve a gyakran előforduló hibaesetek (memóriaallokáció, fájlkezelés stb.) újradefiniálását az egyes modulokban.

Fordítás

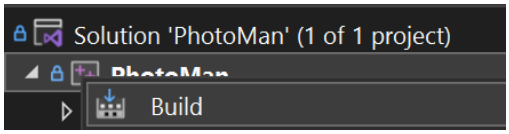
A fordítás x86-64-es MINGW-vel és MSVC-vel is történhet.

Mivel a kód nem függ külső könyvtárak meglététől, így a elégséges a fordítóban a forrásfájlok felsorolása is.

Pl.:

```
x86_64-w64-mingw32-gcc bmp.c cmd.c image.c main.c status.c -o ./photoman.exe
```

Legegyszerűbb viszont a Visual Studio-s PhotoMan Solution-t megnyitni, és az azon belül elhelyezett azonos nevű projektet build-elni.



Struktúrák dokumentációja

color_entry struktúrareferencia

Publikus attribútumok

`uint8_t blue`

`uint8_t green`

`uint8_t red`

`uint8_t reserved`

---

file_header_struct struktúrareferencia

Publikus attribútumok

`uint16_t signature`

`uint16_t __padding`

`uint32_t file_size`

`uint32_t reserved`

`uint32_t data_offset`

---

image_struct struktúrareferencia

Egy absztrakt képstruktúra, mely egy – a 32 bites előjel nélküli egész számábrázolási korlátjaitól eltekintve – tetszőlegesen nagy dimenziójú, legfeljebb 8 bites RGB komponensekkel rendelkező kép és jellemző paramétereinek (szélesség, magasság) egységbezárására alkalmas.

Publikus attribútumok

`uint32_t width`

`uint32_t height`

`Pixel* pixel_data`

`Pixel** pixels`

info_header_struct struktúrareferencia

Publikus attribútumok

```
uint32_t header_size
uint32_t width
uint32_t height
uint16_t planes
uint16_t bits_per_pixel
uint32_t compression
uint32_t image_size
uint32_t x_pixels_per_m
uint32_t y_pixels_per_m
uint32_t colors_used
uint32_t important_colors
```

pixel_struct struktúrareferencia

Egy legfeljebb 24 bites, vörös, zöld és kék RGB színkomponensekből álló színt leíró struktúra.

Publikus attribútumok

```
uint8_t blue
uint8_t green
uint8_t red
```

Modulok dokumentációja

bmp modulreferencia

A BMP formátumú képek kezelését megvalósító modul forrásfájlja.

Struktúrák

```
struct file_header_struct
struct info_header_struct
struct color_entry
```

Makródefiníciók

```
#define BMP_ERROR_OFFSET        2000
```

```
#define BMP_INVALID_SIGNATURE   2000
#define BMP_TOO_MANY_PLANES     2001
#define BMP_INVALID_COLORS      2002
#define BMP_SIGNATURE           19778
#define BMP_PLANES_VALUE        1
#define BMP_FILE_HEADER_SIZE    14
#define BMP_INFO_HEADER_SIZE    40
```

Típusdefiníciók

```
typedef size_t(*foperation)(void* buffer, size_t element_size, size_t
    element_count, FILE* stream)
```

Függvények

```
int bmp_load(Image** p_image, FILE* file)
int bmp_store(const Image** p_image, FILE* file)
```

Változók

```
const char* bmp_error_code_strings[]
```

Függvények dokumentációja

int bmp_load(Image** p_image, FILE* file)

Betölt egy szabványos BMP formátumú képet egy fájlból, melyet paraméterként ad vissza a hívónak.

A lefoglalt memóriaterület felszabadítása a hívó feladata.

Paraméterek

p_image	A képre mutató poitner helye.
file	A fájl.

Visszatérési érték

Sikeres lefutás esetén NO_ERROR-ral, egyébként a validálások, az allokációk vagy egy I/O mûvelet által okozott hibakóddal tér vissza.

int bmp_store(const Image** p_image, FILE* file)

Kiment egy szabványos BMP formátumú képet egy fájlba, melyet paraméterként vesz át.

Paraméterek

p_image	A képre mutató poitner helye.
file	A fájl.

Visszatérési érték

Sikeres lefutás esetén NO_ERROR-ral, egyébként az allokációk vagy egy I/O mûvelet által okozott hibakóddal tér vissza.

Változók dokumentációja

const char* bmp_error_code_strings[]

```
Kezdő érték:= {
    "Hibas fajlalairas.",
    "Nem tamogatott megjelenitesi beallitas.",
    "Hibas bitmelyseg."
```

```
}
```

---

cmd modulreferencia

Parancssori argumentumok kezelését és validálását végző függvényeket megvalósító modul forrásfájlja.

Makródefiníciók

```
#define CMD_ERROR_OFFSET        3000
#define CMD_TOO_FEW_ARGUMENTS   3000
#define CMD_UNKNOWN_CMD_SWITCH  3001
```

Függvények

```
int cmd_check_argc(int argc, int desired)
bool cmd_find_argument(const char* argv[], const char* arg)
int cmd_parse_manip_switch(Image* image, const char* sw)
```

Változók

```
const char* cmd_error_code_strings[]
```

Függvények dokumentációja

int cmd_check_argc(int argc, int desired)

Megvizsgálja, hogy a parancssorból érkező argumentumok számossága megfelel-e az elvártnak.

Paraméterek

argc	A parancssori argumentumok száma.
desired	Az elvárt számosság.

Visszatérési érték

Az ellenőrzés eredményét tükröző státusszal tér vissza, vagyis eltérés esetén CMD_TOO_FEW_ARGUMENTS hibakóddal, egyébként pedig NO_ERROR-ral.

bool cmd_find_argument(const char* argv[], const char* arg)

Megvizsgálja, hogy megtalálható-e az argumentumértéket reprezentáló sztring az argumentumvektorban.

Paraméterek

argv	Az argumentumvektor.
arg	A keresett sztring.

Visszatérési érték

Amennyiben megtalálható a sztring a vektorban, logikai igazzal, egyébként logikai hamissal tér vissza.

int cmd_parse_manip_switch(Image* image, const char* sw)

Értelmezi a sztringként megadott kapcsolót, és amennyiben lehetséges, végrehajtja az ahhoz társított mûveletet a megadott képen.

Paraméterek

image	A feldolgozandó kép.
sw	A mûvelet parancssori kapcsolóját tartalmazó sztring.

Visszatérési érték

Sikeres lefutás esetén NO_ERROR-ral, egyébként a mûveletekhez tartozó státuszjellel/hibakóddal, vagy CMD_UNKNOWN_CMD_SWITCH-csel tér vissza, amennyiben nem ismeret vagy hibás a megadott kapcsoló.

Változók dokumentációja

```
const char* cmd_error_code_strings[]
    Kezdő érték:= {
        "Tul keves argumentum.",
        "Ismeretlen parancssori kapcsolo."
    }
```

---

image modulreferencia

Absztrakt képek kezelését megvalósító modul forrásfájlja.

Struktúrák

```
struct pixel_struct
```

```
struct image_struct
```

Makródefiníciók

```
#define IMAGE_ERROR_OFFSET      1000
```

```
#define IMAGE_BAD_PARAMETER     1000
```

Függvények

```
Image* image_create(uint32_t width, uint32_t height)
```

```
void image_destroy(Image* image)
```

```
int image_scale(Image* image, float horizontal, float vertical)
```

```
int image_mirror_x(Image* image)
```

```
int image_mirror_y(Image* image)
```

```
int image_blur(Image* image, int value)
```

```
int image_exposure(Image* image, int value)
```

Változók

```
const char* image_error_code_strings[]
```

Függvények dokumentációja

```
int image_blur(Image* image, int value)
```

Elhomályosít vagy élesít egy képet megadott intenzitással.

Paraméterek

image	A feldolgozandó kép.
value	Az mûvelet intenzitása. A mûvelet pozitív értékek esetén elhomályosítás, negatív értékek esetén élesítés.

Visszatérési érték

Sikeres lefutás esetén NO_ERROR-ral, hibás paraméterezés esetén IMAGE_BAD_PARAMETER-rel, memóriafoglalási hiba esetén pedig MEMORY_ERROR-ral tér vissza.

```
Image* image_create(uint32_t width, uint32_t height)
```

Készít egy dinamikusan foglalt absztrakt képet tároló sturktúrát, mely elkészítésekor egy width × height dimenziójú kép tárolására alkalmas.

A lefoglalt memóriaterület felszabadítása a hívó feladata.

Paraméterek

width	A kép szélessége.
height	A kép magassága.

Visszatérési érték

Sikeres lefutás esetén a dinamikusan foglalt stuktúrára mutató pointer, foglalási hiba esetén pedig NULL-pointer.

```
void image_destroy(Image* image)
```

Felszabadít egy dinamikusan foglalt absztrakt képet tároló struktúrát annak minden dinamikusan foglalt memóriaterületével együtt.

Paraméterek

image	A felszabadítandó kép sturktúrára mutató pointer.

```
int image_exposure(Image* image, int value)
```

Megnöveli, illetve lecsökkenti egy kép fényerejét megadott intenzitással.

Paraméterek

image	A feldolgozandó kép.
value	Az mûvelet intenzitása.

Visszatérési érték

Minden esetben NO_ERROR státusszal tér vissza.

```
int image_mirror_x(Image* image)
```

Tükröz egy képet az x tengelyre.

Paraméterek

image	A feldolgozandó kép.

Visszatérési érték

Minden esetben NO_ERROR státusszal tér vissza.

```
int image_mirror_y(Image* image)
```

Tükröz egy képet az y tengelyre.

Paraméterek

image	A feldolgozandó kép.

Visszatérési érték

Minden esetben NO_ERROR státusszal tér vissza.

```
int image_scale(Image* image, float horizontal, float vertical)
```

Fel- vagy leskáláz egy képet megadott függőleges és vízszintes paraméterek szerint.

Az adott tengely szerint skálázás egynél nagyobb értékeknél a kép nagyítását, egynél kisebb értékekre pedig a kép kicsinyítését idézi elő.

A függvény újrafoglalhat dinamikusan memóriaterületet, ilyenkor a korábbi területeket felszabadítja, viszont az újonnan foglaltak felszabadítása továbbra is a hívó feladata marad.

Paraméterek

image	A feldolgozandó kép.
horizontal	A vízszintes skálázás értéke. Mindig pozitív.
vertical	A függőleges skálázás értéke. Mindig pozitív.

Visszatérési érték

Sikeres lefutás esetén NO_ERROR-ral, hibás paraméter esetén IMAGE_BAD_PARAMETER-rel, memóriafoglalási hiba esetén pedig MEMORY_ERROR-ral tér vissza.

Változók dokumentációja

```
const char* image_error_code_strings[]
```

```
Kezdő érték:= {
    "Hibas parameter."
}
```

---

main modulreferencia

A főprogram.

Függvények

```
int main(int argc, char* argv[])
```

Függvények dokumentációja

```
int main(int argc, char* argv[])
```

A program belépési pontja. Itt történik az argumentumok validálása (részben),

az erőforrások kezelése (allokátorok, deallokátorok vezérlése), az argumentumokban meghatározott mûveletek meghívása.

Paraméterek

argc	Argumentumok száma beleértve a futtatható bináris nevét.
argv	A NULL-terminált arugmentumvektor.

Visszatérési érték

A program visszatérési kódja, mely sikeres lefutás esetén nulla, egyébként egy külső (az operációs rendszer által generált) vagy belső (a program által generált) hibakód.

status modulreferencia

A hibakezelő modul forrásfájlja.

Makródefiníciók

```
#define STATUS_ERROR_OFFSET     0000
#define NO_ERROR                0000
#define MEMORY_ERROR            0001
#define IO_ERROR                0002
```

Függvények

```
void status_print(int code)
```

Változók

```
const char* status_error_code_strings[]
```

Függvények dokumentációja

void status_print(int code)

A támogatott modulok és a saját hibakódjait értelmezni és szövegesen megjeleníteni képes függvény.

Paraméterek

code	A konzolban megjelenítendő hibakód.

Változók dokumentációja

const char* status_error_code_strings[]

```
Kezdő érték:= {
    "Nincs hiba.",
    "Nincs eleg memoria.",
    "I/O hiba."
```

Felhasznált források

http://www.ece.ualberta.ca/~elliott/ee552/studentAppNotes/2003_w/misc/bmp_file_format/bmp_file_format.htm

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