Programozói dokumentáció

A konvertálás menete

A program a bekért képfájlt beolvassa, és a folyamathoz szükséges adatokat eltárolja egy struktúrában. A program ezután a kép minden pixelének r, g és b értékeiből kiszámol egy szürkeárnyalatos értéket, majd ehhez az értékhez hozzárendel egy, az azt legjobban reprezentáló ASCII karaktert. A kapott ASCII karakterekből álló kétdimenziós tömböt ezután (az argumentumokat is figyelembe véve) rendezve beírjuk egy fájlba.

Adatszerkezetek

Pixel struktúra:

```
typedef struct Pixel {
	unsigned char red;
	unsigned char green;
	unsigned char blue;
} Pixel;
```

Egy pixel 0 és 255 közé eső red, green és blue értékeit tárolja

BMP struktúra:

```
typedef struct BMP {
	unsigned short signature;
	unsigned int offset;
	unsigned int width;
	unsigned int height;
	unsigned short bits_per_pixel;
	unsigned int size;
	Pixel** bitmap;
} BMP;
```

A beolvasott kép szükséges adatait tárolja.

A struktúra tulajdonságainál nagy szerepet játszik a méretük, hiszen binárisan olvassuk be a file-t. Ha a beolvasott byte-ok és a tulajdonság mérete nem egyezik, az hibával jár az adatok eltolódása miatt.

Minden tulajdonsághoz tartozik egy, a BMP formátum által meghatározott offset, ami megmondja, hogy az adott adat melyik indexnél (vagyis hányadik byte-on) kezdődik.

A struktúra tulajdonságai:

- signature: Minden BMP képfájl a „BM”, vagyis a „42” és „4D” bájtokkal kezdődik, ez határozza meg, hogy egy BMP típusú képről beszélünk.
    - Kezdés: 0, Méret: 2 byte
- offset: Megadja, hogy hányadik byte-tól kezdődik a bitmap leírása. A program által támogatott formátumnál ez 54, (hiszen palettával rendelkező képnél ez eltolódik)
    - Kezdés: 10, Méret: 4 byte
- width: A kép szélessége. Kezdés: 18, Méret: 4 byte
- height: A kép magassága. Kezdés: 22, Méret: 4 byte
- bits_per_pixel: Másnéven színmélység, megadja, hogy hány bit vonatkozzon egy pixelre. A program által támogatott formátumnál ez 24 (8 piros, 8 zöld, 8 kék)
    - Kezdés: 28, Méret: 2 byte
- size: A bitmap mérete.

    - Méret: 4 byte, Kezdés: 34 lenne, de a program több képet kezelhet, ha ezt az adatot a szélesség és magasság adatokból számoljuk ki: minden pixel 3 byte, és minden sor végén 2 byte-nyi 0 van padding-ként, vagyis a teljes bitmap mérete: size = width*height*3 + height * 2 byte.
- bitmap: Egy 2 dimenziós Pixelekből álló tömb, melynek első indexe a Pixel sora, második pedig az oszlopa.
    - Kezdés: az offset tulajdonság értéke, Méret: az size tulajdonság értéke

A program felépítése modulokra, függvényekre bontva

Header modul – IMGtoASCII.h

Include-ok:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "debugmalloc.h"
```

Ebben a fájlban találhatóak a struktúrák, valamint a függvények deklarációi.

Függvény modul – Argument_handling.c

```
int handleArgument(char* arg, char** ascii_p, int* gamma_p, int* compression_p,
int* displayOnConsole)
```

A handleArgument függvény egy argumentum kezelésére szolgál. Az argumentum első karaktere alapján beállítja az értékeket a főprogram számára. Esetek:

- 'i': megfordítja a char tömböt, melyre az ascii_p paramétere mutat.
- 'g': beolvassa a közvetlenül a karakter után írt számot a gamma_p által mutatott változóba.
- 'c': beolvassa a közvetlenül a karakter után írt számot a compression_p által mutatott változóba.
- 'd': 1-re állítja a displayOnConsole által mutatott számot.

Ha az egyik típusba beleesik az argumentum, a függvény 0-t, ha egyik esetbe sem esik bele, 1-et ad vissza.

```
void terminate(char** ascii_p, BMP* BMPimg_p, FILE** imgp_p, FILE** txtp_p)
```

Felszabadítja az argumentumként kapott adatokat és bezárja a nyitott fájlokat. Ezek sorra: az ascii karaktereket tartalmazó tömb, a bitmap, valamint az olvasásra és írásra használt file-ok.

A program több fájlt használ és több elemnek is foglal memóriát. Ezek lezárása és felszabadítása minden egyes kivétel kezelésekor rengeteg helyet foglalna, függvénnyel sokkal egyszerűbb. és átláthatóbb.

Függvény modul – Image_processing.c

```
int calculate_index(Pixel p, int ascii_length)
```

Egy pixel r, g és b értékeiből egy szürkeárnyalatos értéket számol, ami alapján visszaadja az ascii tömbben lévő, a pixelt legjobban reprezentáló ascii karakter indexét.

Számolás menete: először minden színértéket beszorzunk egy hozzá rendelt konstanssal, ami a világosságukat reprezentálja, majd ezeket összeadjuk. A konstansok összege 1, így a végeredmény is 0 és 255 közé fog esni. A kapott értéket átskálázzuk 0 és az ascii tömbünk

hossza közti értékre, majd vesszük az alsó egészrészét, hogy a kapott szám így indexként funkcionálhasson.

A lefele kerekítés miatt a legutolsó elem a többivel ellentétben csak a 255-ös értéknél kapna értéket, ezért, hogy a skála egyenletes maradjon, a végére kell illeszteni még egy space-t

```
int compress(Pixel** bitmap, int cur_i, int cur_j, int compression, int ascii_length)
```

A compress függvény egy pixel feldolgozása helyett a compression értékének négyzetével egyenlő pixel indexét átlagolja, így kisebbé téve a képet.

Egy olyan négyzet pixeleit átlagolja, melynek oldalhossza compression nagyságú, bal felső sarka pedig az [i][j]-edik elem.

Természetesen a függvény használja az előzőleg megemlített calculate_index()-et, és az az által visszaadott értékekkel számol.

```
void setGamma(char** ascii_pointer, int gamma)
```

A függvény a gamma érték alapján átírja az ascii tömböt, hogy a program nagyobb valószínűséggel válasszon világosabb/sötétebb pixeleket.

Nulla érték esetén nem csinál semmit.

Pozitív érték esetén készít egy gamma hosszúságú tömböt, melynek minden eleme az ascii_pointer által mutatott tömb utolsó eleme, maja ezt az ascii_pointer által mutatott tömb végéhez fűzi.

Negatív érték esetén a gamma hosszúságú tömb az ascii_pointer által mutatott tömb első elemével töltődik fel, és az új tömböt az eredeti elejére szúrjuk.

Ebben a függvényben van deklarálva az alap ascii tömb is, melyben 71 ascii karakter van általánosan (néhány különc betűtípus máshogy viselkedik) sorba rendezve sötéttől világosig.

A program main részeinek részletes magyarázata

Fájlok megnyitása (10-16. sor)

Filepointerek létrehozása, fájlok megnyitása írásra és bináris olvasásra.

Header adatok kinyerése a képből (18-54. sor)

A program legkevésbé stílusos része. A .bmp filelformátum adatai minden file esetén ugyanott helyezkednek el (a kezelt kivételeket leszámítva), ezt nevezzük offsetnek.

Általánosan egy adat kinyerésének lépései:

- fseek (*miben, mennyi byte-ot, mettől)
- fread (*hova, egy elem mérete byte-ban, kiolvasott elemek száma, *honnan)
- Feltétel a felhasználó hibáinak vagy a nem támogatott formátumoknak a szűrésére.

Az fseek() függvény minden esetben a fájl elejétől keres, és a formátum által meghatározott offsetek-re ugrik. Innen aztán az fread() függvénnyel beolvasunk pontosan annyi byte-ot, amennyit a formátum az adott adatra fenntart.

Prebitmap (56-60. sor)

A bináris fájlkezelés miatt az adatokat a program több részben dolgozza fel, hogy a fejlesztő számára is érthetőbb legyen.

A prebitmap változó egy segédtömb, melybe beleraktuk a bittérkép adatait változtatások nélkül. Felépítése: b g r b g r … b g r 0 0 b g r b g r … b g r 0 0 b g r … 0 0

Érthetőbben: a kép pixeleinek kék, zöld és piros értékei (0-255) egymás mellett elválaszát nélkül, valamint a sorok végén 2 db 1 byte-os 0 érték.

Ha a bináris fájlban csak előre fele haladnánk, a kép pixelein a bal alsó saroktól indulnánk, és a jobb felsőhöz érkeznénk. Vagyis a sorok fordított sorrendben helyezkednek el. Az utolsó sor van legelöl, a legelső pedig a fájl végén. Ugyanakkor egy sorban mind a kép, mind a bináris fájl balról jobbra halad.

Bitmap feltöltése (62-91. sor)

A prebitmap adatait feldolgozzuk, és fogyasztható formátumban eltároljuk a BMP struktúra bitmapjében. A BMP struktúra bitmap-je az adatszerkezetek pontban már leírt módon épül fel.

Argumentumok kezelése (93-124. sor)

Az ascii tömb, valamint a gamma, compression és displayOnConsole alapértékeinek beállítrása. Amennyiben létezik argumentum, a handleArgument() függvénnyel megvizsgáljuk, ami aztán elvégzi helyettünk a munkát.

Ha túl sok az argumentum, vagy a kép nem tömöríthető a megadott adatok alapján, ez a blokk megállítja a programot.

Bitmap pixeleihez rendelt ascii karakterek console-ra és txt-be írása (126-140. sor)

A dupla ciklus belsejében végigiterálunk a bitmap adatain. A compress függvénnyel egyszerre a szükséges mennyiségűt vizsgáljuk, majd hozzájuk rendelünk egy-egy indexet.

A meglévő, motanra formázott ascii tömbünk indexedik eleme lesz a karakter, ami reprezentálja a jelenlegi pixelcsoportunkat a végleges képben.

A program lényeges része lefutott, már csak ki kell írnunk a fájlba valamint a console-ra a kapott karaktert.

Végül a program utolsó része tájékoztat minket a sikeres konvertálásról, felszabadítja a memóriát és lezárja a file-okat

Visszatérési értékek

A program visszatérési értékei az inputoktól és a szabad memória mennyiségétől függően több visszatérési értéket is felvehet:

- return 0; - sikeres lefutás
- return 1; - argumentummal kapcsolatos probléma
- return 2; - fájl létrehozásával/megnyitásával kapcsolatos probléma
- return 3; - kompatibilitási probléma
- return 4; - fileméret túl nagy