**Министр науки и высшего образования Российской̆ Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

Факультет информационных технологий и программирования

Лабораторная работа №14

**Выполнил студент группы № M3104**

Гурман Тимофей Владимирович

**Подпись:**

****

**Проверил:**

Повышев Владислав Вячеславович

Санкт-Петербург

2020Текст задания

Задания

Целью лабораторной работы является реализация [игры “Жизнь”](https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life) , позволяющая выводить поколение игры в монохромную картинку в [формате BMP](https://en.wikipedia.org/wiki/BMP_file_format). Плоскость “вселенной” игры ограничена положительными координатами.

Лабораторная работы должна быть выполнена в виде консольного приложения принимающего в качестве аргументов следующие параметры:

1. ***--input input\_file.bmp***

Где input\_file.bmp - монохромная картинка в формате bmp, хранящая начальную ситуация (первое поколение) игры

1. -***-output dir\_name***

Название директории для хранения поколений игры в виде монохромной картинки

1. ***--max\_iter N***

Максимальное число поколений которое может эмулировать программа. Необязательный параметр, по-умолчанию бесконечность

1. ***--dump\_freq N***

Частота с которой программа должно сохранять поколения виде картинки. Необязательный параметр, по-умолчанию равен 1

Программа должна предусматривать исключительные ситуации, которые могут возникать во время ее работы и корректно их обрабатывать.

Решение с комментариями

#define \_CRT\_SECURE\_NO\_WARNINGS  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <stdint.h>  
#include "Header.h"  
  
struct BITMAPFILEHEADER read\_BITMAPFILEHEADER(FILE\* F);  
struct BITMAPINFOHEADER read\_BITMAPINFOHEADER(FILE\* F);  
struct RGBQUAD\*\* pixelArray(FILE\* F, struct BITMAPINFOHEADER infoHeader);  
void create\_bmp(FILE\* F, struct BITMAPFILEHEADER header, struct BITMAPINFOHEADER infoHeader, struct RGBQUAD\*\* pixels);  
struct RGBQUAD\*\* conv\_to\_pixels(int\*\* in, struct BITMAPINFOHEADER infoHeader);  
int Check\_black(struct RGBQUAD pixel);  
int\*\* read\_files\_pixels(struct RGBQUAD\*\* pixels, struct BITMAPINFOHEADER infoHeader);  
  
char\* input\_file\_name;  
char\* output\_directory = "";  
  
int main(int argc, char\* argv[]) {  
  
 int max\_iter = INT32\_MAX, dump\_freq = 1;  
 int \_input = 0, \_output = 0;  
  
 for (int i = 1; i < argc; i += 2) {  
 if (strcmp(argv[i], "--input") == 0) {  
 input\_file\_name = argv[i + 1];  
 \_input = 1;  
 }  
 if (strcmp(argv[i], "--output") == 0) {  
 output\_directory = argv[i + 1];  
 \_output = 1;  
 }  
 if (strcmp(argv[i], "--max\_iter") == 0) {  
 sscanf(argv[i + 1], "%d", &max\_iter);  
 }  
 if (strcmp(argv[i], "--dump\_freq") == 0) {  
 sscanf(argv[i + 1], "%d", &dump\_freq);  
 }  
 }  
  
 FILE\* input\_file = fopen(input\_file\_name, "r");  
  
 if (input\_file == NULL) {  
 printf("Error with reading file");  
 return 0;  
 }  
  
 struct BITMAPFILEHEADER header = read\_BITMAPFILEHEADER(input\_file);  
 struct BITMAPINFOHEADER infoHeader = read\_BITMAPINFOHEADER(input\_file);  
 struct RGBQUAD\*\* pixelscolors = pixelArray(input\_file, infoHeader);  
  
 int\*\* life\_pixels = read\_files\_pixels(pixelscolors, infoHeader);  
 int\*\* life\_new\_pixels = read\_files\_pixels(pixelscolors, infoHeader);  
  
 fclose(input\_file);  
  
 int up, left, down, right, counter;  
 char s[50];  
 char way[100];  
  
 for (int k = 0; k < max\_iter; k++) {  
  
 for (int i = 0; i < infoHeader.biHeight; i++) {  
 up = i - 1;  
 down = i + 1;  
 if (i == 0)  
 up = infoHeader.biHeight - 1;  
 if (i == infoHeader.biHeight - 1)  
 down = 0;  
  
 for (int j = 0; j < infoHeader.biWidth; j++) {  
 left = j - 1;  
 right = j + 1;  
 if (j == 0)  
 left = infoHeader.biWidth - 1;  
 if (j == infoHeader.biWidth - 1)  
 right = 0;  
  
 counter = life\_pixels[up][left] + life\_pixels[up][j] + life\_pixels[up][right] + life\_pixels[i][left] + life\_pixels[i][right] + life\_pixels[down][left] + life\_pixels[down][j] + life\_pixels[down][right];  
  
 if (life\_pixels[i][j] == 1) {  
 if (counter < 2 || counter > 3)  
 life\_new\_pixels[i][j] = 0;  
 else  
 life\_new\_pixels[i][j] = 1;  
 }  
 else {  
 if (counter == 3)  
 life\_new\_pixels[i][j] = 1;  
 else  
 life\_new\_pixels[i][j] = 0;  
 }  
  
 }  
 }  
  
 for (int i = 0; i < infoHeader.biHeight; i++) {  
 for (int j = 0; j < infoHeader.biWidth; j++) {  
 life\_pixels[i][j] = life\_new\_pixels[i][j];  
 }  
 }  
  
 if (k % dump\_freq == 0) {  
  
 memset(s, 0, 50);  
 memset(way, 0, 100);  
  
 sprintf(s, "/%d.bmp", k);  
 strcpy(way, output\_directory);  
 strcat(way, s);  
  
 input\_file = fopen(way, "w");  
 create\_bmp(input\_file, header, infoHeader, conv\_to\_pixels(life\_pixels, infoHeader));  
 fclose(input\_file);  
 }  
 }  
  
 return 0;  
}  
  
struct BITMAPFILEHEADER read\_BITMAPFILEHEADER(FILE\* F) {  
 struct BITMAPFILEHEADER header;  
  
 fread(&header.bfType, 1, 2, F);  
 fread(&header.bfSize, 4, 1, F);  
 fread(&header.bfReserved1, 2, 1, F);  
 fread(&header.bfReserved2, 2, 1, F);  
 fread(&header.bfOffBits, 4, 1, F);  
  
 return header;  
}  
  
struct BITMAPINFOHEADER read\_BITMAPINFOHEADER(FILE\* F) {  
 struct BITMAPINFOHEADER infoHeader;  
  
 fread(&infoHeader.biSize, 4, 1, F);  
 fread(&infoHeader.biWidth, 4, 1, F);  
 fread(&infoHeader.biHeight, 4, 1, F);  
 fread(&infoHeader.biPlanes, 2, 1, F);  
 fread(&infoHeader.biBitCount, 2, 1, F);  
 fread(&infoHeader.biCompression, 4, 1, F);  
 fread(&infoHeader.biSizeImage, 4, 1, F);  
 fread(&infoHeader.biXPelsPerMeter, 4, 1, F);  
 fread(&infoHeader.biYPelsPerMeter, 4, 1, F);  
 fread(&infoHeader.biClrUsed, 4, 1, F);  
 fread(&infoHeader.biClrImportant, 4, 1, F);  
  
 return infoHeader;  
}  
  
struct RGBQUAD\*\* pixelArray(FILE\* F, struct BITMAPINFOHEADER infoHeader) {  
 struct RGBQUAD\*\* pixels = (struct RGBQUAD\*\*)malloc(infoHeader.biHeight \* sizeof(struct RGBQUAD\*));  
 for (int i = 0; i < infoHeader.biHeight; ++i)  
 pixels[i] = (struct RGBQUAD\*)malloc(infoHeader.biWidth \* sizeof(struct RGBQUAD));  
  
 for (int i = infoHeader.biHeight - 1; i >= 0; i--) {  
 for (int j = 0; j < infoHeader.biWidth; j++) {  
 pixels[i][j].rgbBlue = getc(F);  
 pixels[i][j].rgbGreen = getc(F);  
 pixels[i][j].rgbRed = getc(F);  
 }  
 if (infoHeader.biWidth % 4 != 0) {  
 getc(F);  
 }  
 }  
  
 return pixels;  
}  
  
void create\_bmp(FILE\* F, struct BITMAPFILEHEADER header, struct BITMAPINFOHEADER infoHeader, struct RGBQUAD\*\* pixels) {  
 fwrite(&header.bfType, 1, 2, F);  
 fwrite(&header.bfSize, 4, 1, F);  
 fwrite(&header.bfReserved1, 2, 1, F);  
 fwrite(&header.bfReserved2, 2, 1, F);  
 fwrite(&header.bfOffBits, 4, 1, F);  
 fwrite(&infoHeader.biSize, 4, 1, F);  
 fwrite(&infoHeader.biWidth, 4, 1, F);  
 fwrite(&infoHeader.biHeight, 4, 1, F);  
 fwrite(&infoHeader.biPlanes, 2, 1, F);  
 fwrite(&infoHeader.biBitCount, 2, 1, F);  
 fwrite(&infoHeader.biCompression, 4, 1, F);  
 fwrite(&infoHeader.biSizeImage, 4, 1, F);  
 fwrite(&infoHeader.biXPelsPerMeter, 4, 1, F);  
 fwrite(&infoHeader.biYPelsPerMeter, 4, 1, F);  
 fwrite(&infoHeader.biClrUsed, 4, 1, F);  
 fwrite(&infoHeader.biClrImportant, 4, 1, F);  
  
 for (int i = infoHeader.biHeight - 1; i >= 0; i--) {  
 for (int j = 0; j < infoHeader.biWidth; j++) {  
 putc(pixels[i][j].rgbBlue, F);  
 putc(pixels[i][j].rgbGreen, F);  
 putc(pixels[i][j].rgbRed, F);  
 }  
 if (infoHeader.biWidth % 4 != 0) {  
 putc(0, F);  
 }  
 }  
}  
  
struct RGBQUAD\*\* conv\_to\_pixels(int\*\* in, struct BITMAPINFOHEADER infoHeader) {  
 struct RGBQUAD\*\* pixels = (struct RGBQUAD\*\*)malloc(infoHeader.biHeight \* sizeof(struct RGBQUAD\*));  
 for (int i = 0; i < infoHeader.biHeight; ++i)  
 pixels[i] = (struct RGBQUAD\*)malloc(infoHeader.biWidth \* sizeof(struct RGBQUAD));  
  
 for (int i = infoHeader.biHeight - 1; i >= 0; i--) {  
 for (int j = 0; j < infoHeader.biWidth; j++) {  
 if (in[i][j] == 1) {  
 pixels[i][j].rgbBlue = 0;  
 pixels[i][j].rgbGreen = 0;  
 pixels[i][j].rgbRed = 0;  
 }  
 else  
 {  
 pixels[i][j].rgbBlue = 255;  
 pixels[i][j].rgbGreen = 255;  
 pixels[i][j].rgbRed = 255;  
 }  
  
 }  
 }  
  
 return pixels;  
}  
  
int Check\_black(struct RGBQUAD pixel) {  
 if (pixel.rgbBlue == 0 && pixel.rgbGreen == 0 && pixel.rgbRed == 0) {  
 return 1;  
 }  
  
 return 0;  
}  
  
int\*\* read\_files\_pixels(struct RGBQUAD\*\* pixels, struct BITMAPINFOHEADER infoHeader) {  
 int\*\* pixels\_array = (int\*\*)malloc(infoHeader.biHeight \* sizeof(int\*));  
 for (int i = 0; i < infoHeader.biHeight; ++i)  
 pixels\_array[i] = (int\*)malloc(infoHeader.biWidth \* sizeof(int));  
  
 for (int i = 0; i < infoHeader.biHeight; i++) {  
 for (int j = 0; j < infoHeader.biWidth; j++) {  
 pixels\_array[i][j] = (Check\_black(pixels[i][j]) == 1 ? 1 : 0);  
 }  
 }  
  
 return pixels\_array;  
}