**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

*Название работы: Игра «Жизнь».*

**Выполнил студент группы № M3117**

Жовнир Артём Владимирович

**Подпись:**

**Проверил:**

Повышев Владислав Вячеславович

Санкт-Петербург

2022**Текст 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.

Правила игры:

* Место действия игры — размеченная на клетки плоскость, которая может быть безграничной, ограниченной или замкнутой.
* Каждая клетка на этой поверхности [имеет восемь соседей](https://ru.wikipedia.org/wiki/%D0%9E%D0%BA%D1%80%D0%B5%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%9C%D1%83%D1%80%D0%B0), окружающих её, и может находиться в двух состояниях: быть «живой» (заполненной) или «мёртвой» (пустой).
* Распределение живых клеток в начале игры называется первым поколением. Каждое следующее поколение рассчитывается на основе предыдущего по таким правилам:
  + в пустой (мёртвой) клетке, с которой соседствуют три живые клетки, зарождается жизнь;
  + если у живой клетки есть две или три живые соседки, то эта клетка продолжает жить; в противном случае (если живых соседей меньше двух или больше трёх) клетка умирает («от одиночества» или «от перенаселённости»).
* Игра прекращается, если
  + на поле не останется ни одной «живой» клетки;
  + конфигурация на очередном шаге в точности (без сдвигов и поворотов) повторит себя же на одном из более ранних шагов (складывается периодическая конфигурация)
  + при очередном шаге ни одна из клеток не меняет своего состояния (частный случай предыдущего правила, складывается стабильная конфигурация)

Игрок [не принимает активного участия в игре](https://ru.wikipedia.org/wiki/Zero_Player_Game). Он лишь расставляет или генерирует начальную конфигурацию «живых» клеток, которые затем изменяются согласно правилам. Несмотря на простоту правил, в игре может возникать огромное разнообразие форм

**Решение с комментариями**

#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
  
typedef struct{  
 int height, width; // размер картинки  
} fieldProperties;  
  
typedef struct{  
 int black, white; // цвета  
} pixelInfo;  
  
typedef struct{  
 char width[6];  
 char height[6];  
} header;  
  
int cast(unsigned char \*bytes) {  
 return (int) ((bytes[3] << 16) + (bytes[2] << 8) + (bytes[1] << 4) + bytes[0]); // сдвиг битов  
}  
  
void gameLife(int \*\*firstGen, fieldProperties bitmap) {  
 int \*\*newGen = (int \*\*) malloc(sizeof(int \*) \* bitmap.height); //выделяем память   
 for (int i = 0; i < bitmap.height; i++)  
 newGen[i] = (int \*) malloc(bitmap.width \* sizeof(int));  
 int nbrsCounter;  
 for (int y = 0; y < bitmap.height; y++)  
 for (int x = 0; x < bitmap.width; x++) {  
 nbrsCounter = 0;  
 for (int i = x - 1; i <= x + 1; i++) {  
 for (int j = y - 1; j <= y + 1; j++) {  
 if (i == x && j == y)  
 continue;  
 nbrsCounter += firstGen[(j + bitmap.height) % bitmap.height][(i + bitmap.width) % bitmap.width];  
 }  
 }  
 if (firstGen[y][x] == 1) {  
 if (nbrsCounter < 2 || nbrsCounter > 3)// вычисление стоящих рядом квадратов  
 newGen[y][x] = 0;  
 else  
 newGen[y][x] = 1;  
 } else {  
 if (nbrsCounter == 3)  
 newGen[y][x] = 1;  
 else  
 newGen[y][x] = 0;  
 }  
 }  
 for (int y = 0; y < bitmap.height; y++)  
 for (int x = 0; x < bitmap.width; x++)  
 firstGen[y][x] = newGen[y][x];  
 for (int i = 0; i < bitmap.height; i++)  
 free(newGen[i]);  
 free(newGen);  
}  
  
  
int main(int argc, char \*argv[]) {  
 int max\_iter = 100, dump\_freq = 1;  
 char \*inputFile;  
 char \*outputFile;  
 for (int i = 1; i < argc; i += 2) {  
 if (strcmp(argv[i], "--input") == 0)  
 inputFile = argv[i + 1];  
  
 if (strcmp(argv[i], "--output") == 0)  
 outputFile = argv[i + 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 = fopen(inputFile,"rb");  
  
 header infoFromFile;  
 fread(&infoFromFile, 1, 54, input);  
 fieldProperties bitmap;  
 bitmap.height = cast((unsigned char \*) infoFromFile.height);  
 bitmap.width = cast((unsigned char \*) infoFromFile.width);  
 pixelInfo \*\*pixels = (pixelInfo \*\*) malloc(bitmap.height \* sizeof(pixelInfo \*));  
 for (int i = 0; i < bitmap.height; i++)  
 pixels[i] = (pixelInfo \*) malloc(bitmap.width \* sizeof(pixelInfo));  
 for (int i = bitmap.height - 1; i >= 0; i--)  
 for (int j = 0; j < bitmap.width; j++) {  
 pixels[i][j].black = getc(input);  
 pixels[i][j].white = getc(input);  
 }  
 int \*\*firstGen = (int \*\*) malloc(bitmap.height \* sizeof(int \*));  
 for (int i = 0; i < bitmap.height; i++) {  
 firstGen[i] = (int \*) malloc(bitmap.width \* sizeof(int));  
 }  
 for (int i = 0; i < bitmap.height; i++)  
 for (int j = 0; j < bitmap.width; j++) {  
 firstGen[i][j] = (pixels[i][j].black == 0 && pixels[i][j].white == 0)?1:0;  
 }  
 for (int i = 0; i < bitmap.height; i++)  
 free(pixels[i]);  
 free(pixels);  
  
 for (int i = 1; i <= max\_iter; i++) {  
 gameLife(firstGen, bitmap);  
 if (i % dump\_freq != 0)  
 continue;  
 char filePath[256];  
 char fileName[4];  
 sprintf(fileName, "%d", i);  
 strcpy(filePath, outputFile);  
 strcat(filePath, "/");  
 strcat(filePath, fileName);  
 strcat(filePath, ".bmp");  
 FILE \*output = fopen(filePath, "wb");  
 fwrite(&infoFromFile, 1, 54, output);  
 int n = 0;  
 unsigned char \*newGen = (unsigned char \*) malloc(bitmap.width \* bitmap.height \* 4);  
 for (int tmp\_iterator = bitmap.height - 1; tmp\_iterator >= 0; tmp\_iterator--)  
 for (int j = 0; j < bitmap.width; j++)  
 for (int k = 0; k < 3; k++) {  
 if (firstGen[tmp\_iterator][j] == 0)  
 newGen[n] = 255;  
 else  
 newGen[n] = 0;  
 n++;  
 }  
 fwrite(newGen, 1, bitmap.width \* bitmap.height \* 3, output);  
 free(newGen);  
 fclose(output);  
 }  
 for (int i = 0; i < bitmap.height; i++)  
 free(firstGen[i]);  
 free(firstGen); // очистка памяти  
 return 0;