

**Министерство науки и высшего образования
Российской Федерации**

**Федеральное государственное автономное
образовательное учреждение высшего образования**

**«Национальный исследовательский университет
ИТМО»**

**Факультет информационных технологий и
программирования**

Лабораторная работа № 14

Игра «Жизнь»

Выполнил студент группы № М3109

Бабич Артём Антонович

Подпись:



Проверил:

Повышев Владислав Вячеславович

Санкт-Петербург
2020

Текст задания

Целью лабораторной работы является реализация игры “Жизнь”, позволяющая выводить поколение игры в монохромную картинку в формате BMP. 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 ситуацию (первое поколение) игры
2. **--output dir_name**
Название директории для хранения поколений игры в виде монохромной картинки
3. **--max_iter N**
Максимальное число поколений которое может эмулировать программа. Необязательный параметр, по-умолчанию бесконечность
4. **--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

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

int **arr_malloc(unsigned long h, unsigned long w)
{
    int **ARR = (int **) malloc(h * sizeof(int *));
    for (int i = 0; i < h; i++)
    {
        ARR[i] = (int *) malloc(w * sizeof(int));
    }
    return ARR;
}

int neighborCounter(int **area, int x, int y) // Функция посчёта кол-ва
соседей
{
    int sum = 0;
    for (int i = x - 1; i <= x + 1; i++)
    {
        for (int j = y - 1; j <= y + 1; j++)
        {
            sum += area[i][j];
        }
    }
    if (area[x][y] == 1)
    {
        sum--;
    }
    return sum;
}

unsigned char *transl_to_str(int **matrix, unsigned long h, unsigned long w)
{
    unsigned char *info_on_pix = (unsigned char *) malloc(h * w * 3);
    int m = 0;
    for (int i = h - 1; i >= 0; i--)
    {
        for (int j = 0; j < w; j++)
        {
            if (matrix[i][j] == 1)
            {
                info_on_pix[m] = 0;
                info_on_pix[m + 1] = 0;
                info_on_pix[m + 2] = 0;
            } else
            {
                info_on_pix[m] = 255;
                info_on_pix[m + 1] = 255;
                info_on_pix[m + 2] = 255;
            }
            m += 3;
        }
    }
    return info_on_pix;
}

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

```

```

setlocale(LC_ALL, "Rus");
FILE *image; // задаём основные параметры
long dump_freq = 1;
long max_iter = 1;
char *dirName;
//-----
unsigned long h; // инициализируем параметры основного изображения
unsigned long w;
unsigned long size;
unsigned long image_offset;
unsigned char bmp_header[54];
//-----
int **cur_gen; // инициализируем массивы для текущего и следующего
поколения
int **next_gen;
//-----
for (int i = 1; i < argc; i += 2) // считывание аргументов командной
строки
{
    if (strcmp(argv[i], "--input") == 0)
    {
        image = fopen(argv[i + 1], "rb"); // открываем для считывания
        информации в бинарном виде
        if (image == NULL)
        {
            printf("Файл не получилось открыть :(");
            return (0);
        }
    }
    else if (strcmp(argv[i], "--output") == 0)
    {
        dirName = argv[i + 1];
    }
    else if (strcmp(argv[i], "--max_iter") == 0)
    {
        max_iter = strtol(argv[i + 1], NULL, 10);
    }
    else if (strcmp(argv[i], "--dump_freq") == 0)
    {
        dump_freq = strtol(argv[i + 1], NULL, 10);
    }
}
//-----
fread(bmp_header, sizeof(unsigned char), 54, image); // считываем
заголовок bmp файла
// -----
image_offset = bmp_header[0xD] << 24 | bmp_header[0xC] << 16 |
bmp_header[0xB] << 8 | bmp_header[0xA]; // считываем из заголовка параметров
изображения путем побитового прибавления значений байтов со смещением
size = bmp_header[0x5] << 24 | bmp_header[0x4] << 16 | bmp_header[0x3] <<
8 | bmp_header[0x2];
w = bmp_header[0x15] << 24 | bmp_header[0x14] << 16 | bmp_header[0x13] <<
8 | bmp_header[0x12];
h = bmp_header[0x19] << 24 | bmp_header[0x18] << 16 | bmp_header[0x17] <<
8 | bmp_header[0x16];
//-----
printf("offset - %lu\n", image_offset); // выводим основные параметры bmp
изображения
printf("size - %lu\n", size);
printf("height - %lu\n", h);

```

```

printf("width - %lu\n", w);
printf("\n");
// -----
cur_gen = arr_malloc(h, w * 3); // выделение памяти для текущего и
следующего поколения
next_gen = arr_malloc(h, w * 3);
fseek(image, image_offset, SEEK_SET);
//-----
char buffer[3]; // записываем значения пикселей в двумерный массив
for (int i = h - 1; i >= 0; i--)
{
    for (int j = 0; j < w; j++)
    {
        buffer[0] = fgetc(image);
        buffer[1] = fgetc(image);
        buffer[2] = fgetc(image);

        if (buffer[0] != 0 || buffer[1] != 0 || buffer[2] != 0)
        {
            cur_gen[i][j] = 1;
        }
        else
        {
            cur_gen[i][j] = 0;
        }
    }
}
// -----
char fileName[10];
char directory[256];
char *pixelInfo;
// -----
for (unsigned long i = 0; i < h; i++) // копирование информации о
начальном поколении для последующего редактирования
{
    for (unsigned long j = 0; j < w; j++)
    {
        next_gen[i][j] = cur_gen[i][j];
    }
}
//-----
int countOfNeighbors; // цикл для создания новых поколений
for (int gameIteration = 0; gameIteration < max_iter; gameIteration++)
{
    for (unsigned long i = 1; i < h - 1; i++) // изменение след.
поколения в соотв. с правилами игры
    {
        for (unsigned long j = 1; j < w - 1; j++)
        {
            countOfNeighbors = neighborCounter(cur_gen, i, j);
            if (cur_gen[i][j] == 0 && countOfNeighbors == 3)
            {
                next_gen[i][j] = 1;
            }
            else if (cur_gen[i][j] == 1)
            {
                if (countOfNeighbors < 2 || countOfNeighbors > 3)
                {
                    next_gen[i][j] = 0;
                }
            }
        }
    }
}

```

```

    }
    for (unsigned long i = 0; i < h; i++) // обновление сгенерированного
поколения
    {
        for (unsigned long j = 0; j < w; j++)
        {
            cur_gen[i][j] = next_gen[i][j];
        }
    }
    pixelInfo = transl_to_str(cur_gen, h, w); // перевод информации о
пикселях из двумерного массива в строку
    if (gameIteration % dump_freq == 0) // если номер поколения соотв.
частоте
    {
        // адрес сохраняемого файла
        sprintf(fileName, "%d", gameIteration); // приведение номера
итерации к строковому типу для задания имени файла
        strcpy(directory, dirName); // название директории, считанное с
командной строки
        strcat(directory, "/");
        strcat(directory, fileName);
        strcat(directory, ".bmp");
        FILE *new_bmp = fopen(directory, "wb"); // создание файла
поколения
        if (new_bmp != NULL)
        {
            printf("Файл %d был создан\n", gameIteration);
        }
        else
        {
            printf("Файл %4d не совсем получилось открыть\n",
gameIteration);
        }
        fseek(new_bmp, 0, SEEK_SET); // запись в файл поколения значения
заголовка и значений пикселей
        fwrite bmp_header, 1, 54, new_bmp);
        fwrite(pixelInfo, 1, 3 * w * h, new_bmp);
    }
    free(cur_gen);
    free(next_gen);
    free(pixelInfo);
    return 0;
}

```

Данная программа реализует клеточный автомат «Жизнь» со следующими основными правилами:

- Внутри трех рядом стоящих клеток появляется «жизнь» (новая клетка);
- Если у клетки менее двух или более трех соседей – ее «жизнь» завершается (удаление клетки из игры);

Результат работы программы – последовательность .bmp файлов, содержащих поколения игры;