МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Математического Обеспечения и Применения ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений на зыке Си в формате BMP

Студент.гр. 0382 _____ Тихонов С.В.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2021

Студент: Тихонов С.В.

Группа: 0382

Тема работы: Обработка изображений на языке Си в формате BMP.

Исходные данные.

Программа получает на вход BMP файл и аргументы в виде ключей и числовых значений для некоторых ключей. Необходимо преобразовать картинку в зависимости от выбранных ключей и сохранить итог в отдельный файл. Поддержка операий ведётся через терминайльный интерфейс (CLI).

Предполагаемый объём пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания:

5.04.2021

Дата сдачи реферата:

24.05.2021

Дата защиты реферата:

25.05.2021

Студент.гр. 0382 _____ Тихонов С.В.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Аннотация

В курсовой работе была написана программа на языке Си, которая реализует обработку изображения в формате BMP в соответствие с заданными условиями. Для того, чтобы задать условия используется CLI и библиотека «getopt.h». В программе реализованы функции дл рисования квадрата (задаётся левый верхний угол, длина стороны, её цвет, нужна ли заливка, цвет заливки, если нужна), установка для одного из компонентов RGB определённого значения (-1<x<256), поворот части изображения или же всей картинки на 90/180/270 градусов. Возможно реализация нескольких команд за один запуск.

Содержание

ВВЕДЕНИЕ

Необходимо написать программу, для обработки BMP файла, выбранного пользователем.

Программа обеспечивает работу алгоритма на операционной системе на базе Linux. Написание производилось на операционной системе Ubuntu, для тестирования так же использовалось IDE Clion.

Результатом работы является программа, производящая обработку BMP файла и сохранения результата в отдельный BMP файл.

Программой используется динамическая память, которая отчищается перед завершением. Программа так же обрабатывает все случаи некорректного обращения, выводя сообщение об ошибки.

Цель работы.

Изучить основы обработки изображения в формате BMP и написать программ, производящую эти действия на языке Си.

Задачи.

Обеспечить бинарное считывание изображения с использованием соответствующих структур;

Реализовать вспомогательные функции производящих обработку BMP файлов;

Создать и сохранить копию исходного изображения примененными изменениями;

Реализовать поддержку CLI.

Задание.

Вариант 12.

Программа должна иметь поддержку CLI или GUI

Общие сведения

- 24 бита на цвет
- без сжатия
- файл всегда соответствует формату BMP (но стоит помнить, что версий у формата несколько)
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- обратите внимание на порядок записи пикселей
- все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна реализовывать следующий функционал по обработке bmp-файла

- (1) Рисование квадрата с диагоналями. Квадрат определяется:
    - Координатами левого верхнего угла
    - Размером стороны
    - Толщиной линий

    - Цветом линий
    - Может быть залит или нет (диагонали располагаются “поверх” заливки)
    - Цветом которым он залит, если пользователем выбран залитый
- (2) Фильтр rgb-компонент. Этот инструмент должен позволять для всего изображения либо установить в 0 либо установить в 255 значение заданной компоненты. Функционал определяется
    - Какую компоненту требуется изменить
    - В какой значение ее требуется изменить
- (3) Поворот изображения (части) на 90/180/270 градусов. Функционал определяется
    - Координатами левого верхнего угла области
    - Координатами правого нижнего угла области
    - Углом поворота

ВЫПОЛНЕНИЕ РАБОТЫ

Ход решения

Используются стандартные библиотеки языка Си, а именно <stdio.h>, <stdlib.h> ,<string.h>,<unistd.h>,<stdint.h>,<getopt.h>

На вход программе подаются ключи и аргументы к ним, если они нужны. По молчанию последним идёт название файла с результатом. Происходит считывание информации в специальные структуры для хранения информации об изображении. Динамически выделяется память для массива структур Rgb (структура состоит из 3 полей, каждое из них отвечает за определенный цвет) — этот двумерный массив и представляет из себ картинку.

Далее происходит обработка изображения в зависимости от полученных ключей от getopt_long. Выбор функций и записывание данных происходит с помощью функции «swich».Реализована структура «option Longopts[]» для хранения коротких версий ключей и информации, нужны ли переменные для данного ключа. В зависимости от полученных ключей происходит обработка изображения и сохранения его в отдельный файл, далее происходит очищение динамической памяти.

Функции.

1.main()

Функция осуществляет открытие файла в переменную ff, с помощью fopen. Затем записывает в bmih и в bmfh(объекты структур BITMAPINFOHEADER и BITMAPFILEHEADER) информацию из заголовка файла. Выделяет динамически память для массива arr. Далее в массив пикселей записывается исходное изображение и закрывается с помощью fclose. Впеременную opt считываются данные из getopt_long. Далее при помощи функции swich выбираются нужные функции и записываются данные. Затем происходит вызов нужных функций, сохранение результата и очищение памяти.

2.Rgb square()

Функция рисует квадрат проводит диагонали, и если нужно, делает заливку области (рисование сторон и диагоналей происходит поверх заливки). На вход функции поступают координаты верхнего левого угла, длины стороны, цвет линий, нужна ли заливка, цвет заливки. Функция возвращает Rgb структуру, это картинка с нарисованным квадратом в

нужном месте.

3.Rgb set_color()

Функция выставляет определённое значение одного из RGB компонента для всей картинки. На вход функции поступает информация о том, какой компонент нужно изменить, и на какое значение. Функция возвращает Rgb структуру, это картинка с одним изменённым RGB компонентом.

4.Rgb turn()

Функция производит поворот участка или всего изображения на 90/180/270 градусов. На вход принимает массив arr координаты левого верхнего угла, правого нижнего, угол поворота, bmih, bmfh. Функция создаёт новый массив пикселей, размер которого определяется разницей в длине и высоте между правым нижним и левым верхнем углом. В данный массив записывается информация в нужном порядке и затем она переписывается в исходный массив, а данный массив отправляется на освобождение памяти. Если происходит поворот всей картинки, то функция сама сохраняет её в отдельный файл, посл чего завершает выполнение работы программы. Если происходит поворот только части картинки, то повторяющееся место закрашивается белым, а сам массив возвращается обратно.

5. void printmenu()

Функция ничего не принимает и не возвращает, печатает информацию о возможностях программы, и как нужно выводить данные.

6 void printinfo()

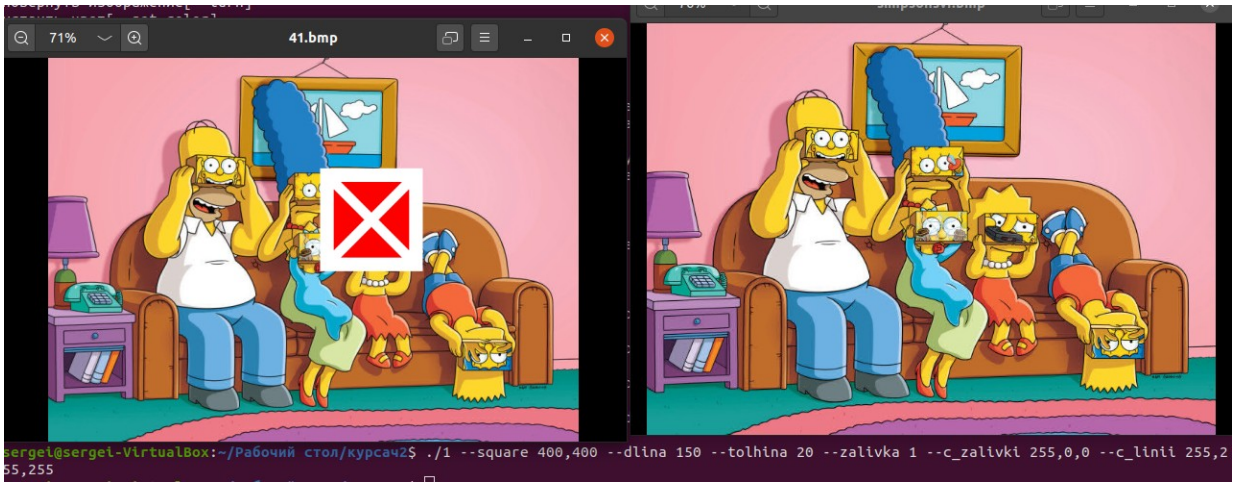
Функция принимает на вход bmih и bmfh и печатает информацию об изображение. Данная функция ничего не возвращает.

ТЕСТИРОВАНИЕ

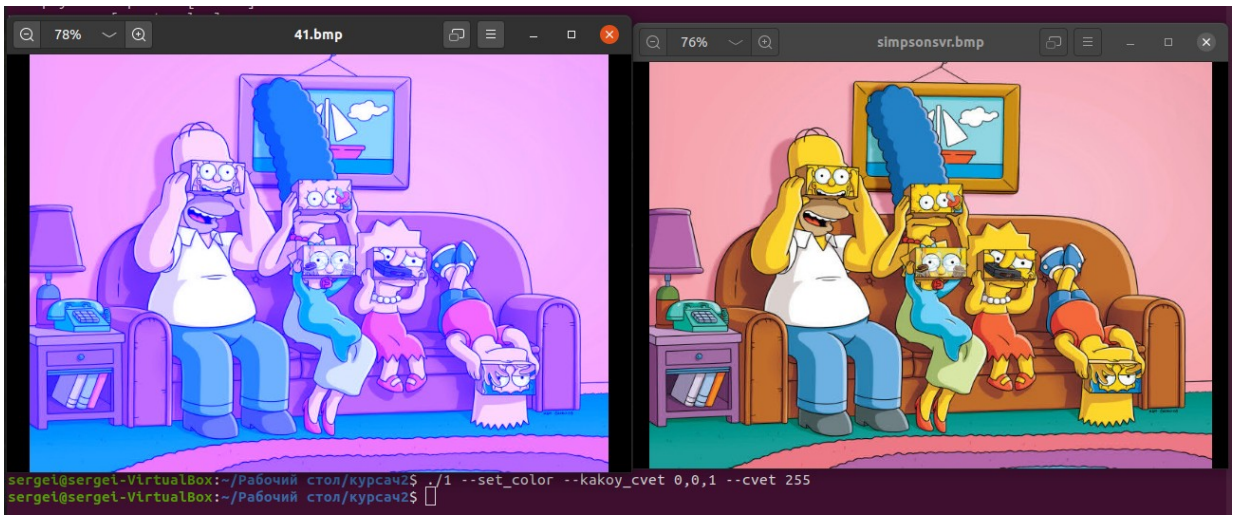
Вывод меню

```
sergei@sergei-VirtualBox:~/Рабочий стол/курсач2$ ./1 --menu
нарисовать квадрат [--square], затем через заптую нужно подать координаты верхнего угла [y,x]
повернуть изображение[--turn]
устаить цвет[--set_color]
координата y для верхнего левого угла участка поворота[--l_visota]
координата x для верхнего левого угла участка поворота[--l_shirina]
длина стороны для рисования квадрата [--dlina]
толщина стороны для риования квадрата[--tolhina]
нужно ли заливать облость под квадратом, принимает значения 1 или 0 [--zalivka]
каким цветом заливать область под квадратом, принимает 3 значения через запятую [--c_zalivki]
цвет линии для стороны квадрата[--c_linii]
какой цветовой компонетнт нужно поменять принимает 3 значени через запятую 0 или 1[--kakoy_cvet]
на какой цвет нужно поменять значение, принимает число от 0 до 255 [--cvet]координата y для правого нижнего угла участка поворота[--p_visota]
координата x для правого нижнего угла участка поворота[--p_shirina]
на какой угол нужно повернуть участок изображения 90/180/270[--ugol]
печатает информацию об изображение[--info]
название результата работы програмы(формат должен быть bmp)вводится последним[--name]
печатает эту информацию[--menu]
```

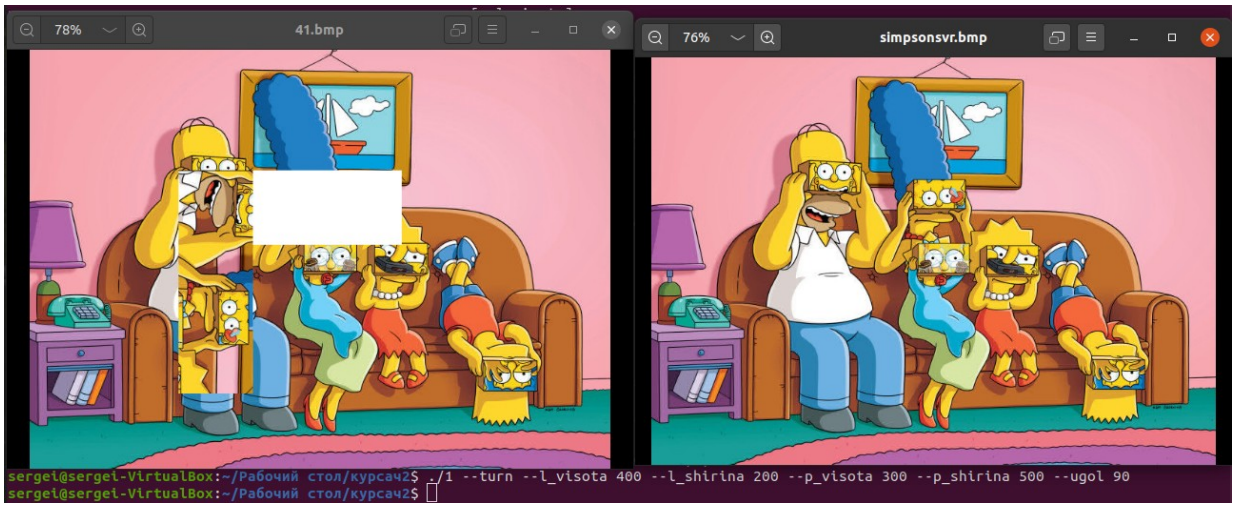
Рисование квадрата



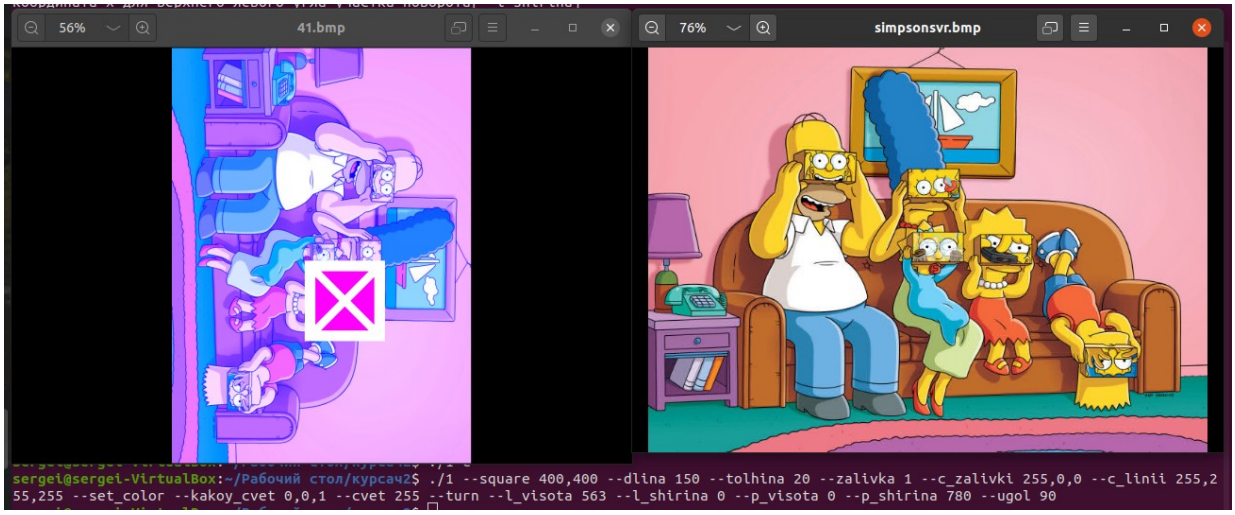
Установка для одного из RGB компанента определённого значени



Поворот части картинки



Рисование квадрата установка цвета и поворот всего изображения



ЗАКЛЮЧЕНИЕ

Были изучены основы обработки BMP изображения на языке Си.

Разработана программа, производящая считывание, затем обработку и сохранение изображения в формате BMP в отдельный файл. Программой используется динамическая память, которая очищается перед завершением. Программа также обрабатывает все случаи некорректного обращения, выводя сообщение о соответствующей ошибке. Взаимодействие с пользователем происходит с помощью CLI.

СПИСОК ИСТОЧНИКОВ

cplusplus.com - The C++ Resources Network

https://ru.wikipedia.org/wiki/BMP

https://www.gnu.org/software/libc/manual/html_node/Getopt.html

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

название файла: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <getopt.h>

#pragma pack (push, 1)
typedef struct
{
   unsigned short signature;
   unsigned int filesize;
   unsigned short reserved1;
   unsigned short reserved2;
   unsigned int pixelArrOffset;
} BitmapFileHeader;

typedef struct
{
   unsigned int headerSize;
   unsigned int width;
   unsigned int height;
   unsigned short planes;
   unsigned short bitsPerPixel;
   unsigned int compression;
   unsigned int imageSize;
   unsigned int xPixelsPerMeter;
   unsigned int yPixelsPerMeter;
   unsigned int colorsInColorTable;
   unsigned int importantColorCount;
} BitmapInfoHeader;
char *outputName="41.bmp";
typedef struct
```

```
{
    unsigned char b;
    unsigned char g;
    unsigned char r;
} Rgb;

#pragma pack(pop)


Rgb square(Rgb **arr, int a, int s, int c, int f, int r, int g, int b, int z, int zr, int zg, int zb){
    int shi=s;
    int vi=a;
    int tolhina=c;
    int dlina=f;
    if(z){
        for(int d=vi; d>a-f; d--){
            for (int i=shi; i<s+f; i++){
                arr[d][i].g = zg;
                arr[d][i].b = zb;
                arr[d][i].r = zr;
            }
        }
    }
    arr[a][s].r=r;
    arr[a][s].g=g;
    arr[a][s].b=b;
    for(int i=a; i>a-c; i--){
        for(int d=s; d<s+f; d++){
            arr[i][d]=arr[a][s];
        }
    }
    shi=shi+f;
    for(int i=a; i>vi-f; i--){
        for(int d=shi; d>shi-c; d--){
            arr[i][d]=arr[a][s];
```

```
        }
    }
    vi=vi-f;
    for(int i=vi; i<vi+c; i++){
        for(int d=shi; d>shi-f; d--){
            arr[i][d]=arr[a][s];
        }
    }
    shi=shi-f;
    for(int i=vi; i<vi+f; i++){
        for(int d=shi; d<shi+c; d++){
            arr[i][d]=arr[a][s];
        }
    }
    for(int i=a-c, d=s+c; i>(a-f+c-1); i--, d++){
        for(int m=0; m<c/2+1; m++) {
            arr[i][d] = arr[a][s];
            arr[i + m][d] = arr[a][s];
            arr[i - m][d] = arr[a][s];
        }
    }
    for(int i=a-c, d=s+f-c; i>(a-f+c-1); i--, d--){
        for(int m=0; m<c/2+1; m++){
            arr[i][d]=arr[a][s];
            arr[i+m][d]=arr[a][s];
            arr[i-m][d]=arr[a][s];
        }
    }
    return **arr;
}
Rgb set_color(Rgb **arr, int cr, int cg, int cb, int cvet, unsigned int H,unsigned int W){
    if(cr){
        for(int i=0; i<H; i++){
            for(int d=0; d<W; d++){
                arr[i][d].r=cvet;
```

```
            }
        }
    }
    if(cg){
        for(int i=0; i<H; i++){
            for(int d=0; d<W; d++){
                arr[i][d].g=cvet;
            }
        }
    }
    if(cb){
        for(int i=0; i<H; i++){
            for(int d=0; d<W; d++){
                arr[i][d].b=cvet;
            }
        }
    }
    return **arr;
}

Rgb turn(Rgb **arr, int lv, int ls, int pv, int ps, int ugol, BitmapInfoHeader bmih,
BitmapFileHeader bmfh){
    int v=lv-pv;
    int s=ps-ls;
    if(ugol==90) {
        Rgb **brr = malloc(s * sizeof(Rgb *));
        for (int i = 0; i < (s + 1); i++) {
            brr[i] = malloc(v * sizeof(Rgb) + (v * 3) % 4);
        }
        for (int av = pv, bv = 0; av < lv; bv++, av++) {
            for (int as = ps, bs = 0; as > ls; bs++, as--) {
                brr[bs][bv] = arr[av][as];
            }
        }
        if (s == bmih.width && v == bmih.height) {
```

```
    FILE *ff = fopen(outputName, "wb");

    bmih.height = s + 1;
    bmih.width = v + 1;
    fwrite(&bmfh, 1, sizeof(BitmapFileHeader), ff);
    fwrite(&bmih, 1, sizeof(BitmapInfoHeader), ff);
    unsigned int w = (v + 1) * sizeof(Rgb) + ((v + 1) * 3) % 4;
    for (int i = 0; i < s + 1; i++) {
        fwrite(brr[i], 1, w, ff);
    }
    fclose(ff);
    return **arr;
} else {
    for (int av = lv, bv = s; bv > 0; bv--, av--) {
        for (int as = ls, bs = 0; bs < v; bs++, as++) {
            if (av > 0 && as < bmih.width) {
                arr[av][as] = brr[bv][bs];
            }
        }
    }
    for (int i = lv - s; i > pv; i--) {
        for (int j = ls; j < ps; j++) {
            if (i > 0 && j < bmih.width) {
                arr[i][j].r = 255;
                arr[i][j].g = 255;
                arr[i][j].b = 255;
            }
        }
    }
    for (int i = lv; i > pv; i--) {
        for (int j = ls + v; j < ps; j++) {
            if (i > 0 && j < bmih.width) {
                arr[i][j].r = 255;
                arr[i][j].g = 255;
                arr[i][j].b = 255;
```

```
            }
        }
    }
        for (int i = 0; i < s; i++) {
            free(brr[i]);
        }
        free(brr);
    }
}
if(ugol==180) {
    Rgb **brr = malloc(v * sizeof(Rgb *));
    for (int i = 0; i < (v + 1); i++) {
        brr[i] = malloc(s * sizeof(Rgb) + (s * 3) % 4);
    }

    for (int av = pv, bv = v; av < lv; bv--, av++) {
        for (int as = ls, bs = s; as < ps; bs--, as++) {
            brr[bv][bs] = arr[av][as];
        }
    }
    if (s == bmih.width && v == bmih.height) {
        FILE *ff = fopen(outputName, "wb");

        bmih.height = v;
        bmih.width = s;
        fwrite(&bmfh, 1, sizeof(BitmapFileHeader), ff);
        fwrite(&bmih, 1, sizeof(BitmapInfoHeader), ff);
        unsigned int w = (s) * sizeof(Rgb) + ((s) * 3) % 4;
        for (int i = 0; i < v; i++) {
            fwrite(brr[i], 1, w, ff);
        }
        fclose(ff);
        return **arr;
    } else {
        for (int av = lv, bv = v; av > pv; bv--, av--) {
```

```
            for (int as = ls, bs = 0; as < ps; bs++, as++) {
                arr[av][as] = brr[bv][bs];
            }
        }

        return **arr;
    }
}
if(ugol==270){
    Rgb **brr = malloc(s * sizeof(Rgb *));
    for (int i = 0; i < (s+1); i++) {
        brr[i] = malloc(v * sizeof(Rgb) + (v * 3) % 4);
    }
    for (int av = lv, bv = 0; av > pv; bv++, av--) {
        for (int as = ls, bs = 0; as < ps; bs++, as++) {
            brr[bs][bv] = arr[av][as];
        }
    }
    if(s==bmih.width && v==bmih.height){
        FILE *ff = fopen(outputName, "wb");

        bmih.height = s+1;
        bmih.width = v+1;
        fwrite(&bmfh, 1, sizeof(BitmapFileHeader),ff);
        fwrite(&bmih, 1, sizeof(BitmapInfoHeader),ff);
        unsigned int w = (v+1) * sizeof(Rgb) + ((v+1)*3)%4;
        for(int i=0; i<s+1; i++){
            fwrite(brr[i],1,w,ff);
        }
        fclose(ff);
        return **arr;
    }
    else{
        for (int av = lv, bv = s; bv > 0; bv--, av--) {
            for (int as = ls, bs = 0; bs < v; bs++, as++) {
```

```
                if( av>0 && as<bmih.width) {
                    arr[av][as] = brr[bv][bs];
                }
            }
        }
        for(int i=lv-s; i>pv; i--){
            for(int j=ls; j<ps; j++) {
                if (i > 0 && j < bmih.width) {
                    arr[i][j].r = 255;
                    arr[i][j].g = 255;
                    arr[i][j].b = 255;
                }
            }
        }
        for (int i = lv; i > pv; i--) {
            for (int j = ls + v; j < ps; j++) {
                if (i > 0 && j < bmih.width) {
                    arr[i][j].r = 255;
                    arr[i][j].g = 255;
                    arr[i][j].b = 255;
                }
            }
        }
        for (int i = 0; i < s; i++) {
            free(brr[i]);
        }
        free(brr);
    }
  }
  return **arr;
}
void printFileHeader(BitmapFileHeader header){
  printf("signature:\t%x (%hu)\n", header.signature, header.signature);
  printf("filesize:\t%x (%u)\n", header.filesize, header.filesize);
  printf("reserved1:\t%x (%hu)\n", header.reserved1, header.reserved1);
```

```
    printf("reserved2:\t%x (%hu)\n", header.reserved2, header.reserved2);
    printf("pixelArrOffset:\t%x (%u)\n", header.pixelArrOffset, header.pixelArrOffset);
}

void printInfoHeader(BitmapInfoHeader header){
    printf("headerSize:\t%x (%u)\n", header.headerSize, header.headerSize);
    printf("width:     \t%x (%u)\n", header.width, header.width);
    printf("height:    \t%x (%u)\n", header.height, header.height);
    printf("planes:    \t%x (%hu)\n", header.planes, header.planes);
    printf("bitsPerPixel:\t%x (%hu)\n", header.bitsPerPixel, header.bitsPerPixel);
    printf("compression:\t%x (%u)\n", header.compression, header.compression);
    printf("imageSize:\t%x (%u)\n", header.imageSize, header.imageSize);
    printf("xPixelsPerMeter:\t%x (%u)\n", header.xPixelsPerMeter, header.xPixelsPerMeter);
    printf("yPixelsPerMeter:\t%x (%u)\n", header.yPixelsPerMeter, header.yPixelsPerMeter);
    printf("colorsInColorTable:\t%x (%u)\n", header.colorsInColorTable,
header.colorsInColorTable);
    printf("importantColorCount:\t%x (%u)\n", header.importantColorCount,
header.importantColorCount);
}
void print_menu(){
    printf("нарисовать квадрат [--square], затем через заптую нужно подать координаты
верхнего угла [y,x]\n");
    printf("повернуть изображение[--turn]\n");
    printf("устаить цвет[--set_color]\n");
    printf("координата y для верхнего левого угла участка поворота[--l_visota]\n");
    printf("координата x для верхнего левого угла участка поворота[--l_shirina]\n");
    printf("длина стороны для рисования квадрата [--dlina]\n");
    printf("толщина стороны для риования квадрата[--tolhina]\n");
    printf("нужно ли заливать облость под квадратом, принимает значения 1 или 0 [--
zalivka]\n");
    printf("каким цветом заливать область под квадратом, принимает 3 значения через
запятую [--c_zalivki]\n");
    printf("цвет линии для стороны квадрата[--c_linii]\n");
    printf("какой цветовой компонетнт нужно поменять принимает 3 значени через
запятую 0 или 1[--kakoy_cvet]\n");
    printf("на какой цвет нужно поменять значение, принимает число от 0 до 255 [--cvet]");
```

```
    printf("координата y для правого нижнего угла участка поворота[--p_visota]\n");
    printf("координата x для правого нижнего угла участка поворота[--p_shirina]\n");
    printf("на какой угол нужно повернуть участок изображения 90/180/270[--ugol]\n");
    printf("печатает информацию об изображение[--info]\n");
    printf("название результата работы програмы(формат должен быть bmp)вводится последним[--name]\n");
    printf("печатает эту информацию[--menu]\n");
}
int main(int argc, char*argv[]){
    char *opts = "vidopqlkmgwu:stc?!";
    static struct option longOpts[] = {
            {"square",      1,0,'s'},
            {"turn",     0,0,'t'},
            {"set_color",       0,0,'c'},
            {"l_visota",     1,0,'v'},
            {"l_shirina",    1,0,'i'},
            {"dlina",  1,0,'d'},
            {"tolhina",1,0,'o'},
            {"zalivka", 1,0,'p'},
            {"c_zalivki",   1,0,'q'},
            {"c_linii",    1,0,'l'},
            {"kakoy_cvet",      1,0,'k'},
            {"cvet",       1,0,'m'},
            {"p_visota", 1,0,'g'},
            {"p_shirina", 1,0,'w'},
            {"ugol", 1,0,'u'},
            {"name", 1,0,'n'},
            {"menu", 0,0,'?'},
            {"info", 0,0,'!'},
            {0,0,0,0}};
    int opt, longIndex;
    opterr=0;
    int s=0;
    int t=0;
    int sc=0;
```

```
int to,c, lv, ls, d,  cr, cg, cb,z, zr,zg,zb,pv,ps,u, lr, lg, lb,lv1,ls1;
FILE *f = fopen("simpsonsvr.bmp", "rb");
BitmapFileHeader bmfh;
BitmapInfoHeader bmif;
fread(&bmfh,1,sizeof(BitmapFileHeader),f);
fread(&bmif,1,sizeof(BitmapInfoHeader),f);
unsigned int H = bmif.height;
unsigned int W = bmif.width;
if(bmif.compression != 0){
   printf("Unknown compression method\n");
   return 1;
}
if(bmif.importantColorCount != 0){
   printf("Version not supported\n");
   return 1;
}
Rgb **arr = malloc(H*sizeof(Rgb*));
for(int i=0; i<H; i++){
   arr[i] = malloc(W * sizeof(Rgb) + (W*3)%4);
   fread(arr[i],1,W * sizeof(Rgb) + (W*3)%4,f);
}
opt=getopt_long(argc, argv, opts, longOpts, &longIndex);
while (opt!=-1){
   switch (opt) {
      case's':
         sscanf(optarg,"%d,%d",&lv1,&ls1);
         if(lv1>W || lv1<0 || ls1>H ||ls1<0){
            printf("некоректные данные для верхнего левого угла квадрата");
            for(int i=0; i<H; i++){
               free(arr[i]);
            }
            free(arr);
            return 0;
         }
         s=1;
```

```
            break;
        case't':
            t=1;
            break;
        case'c':
            sc=1;
            break;
        case'v':
            sscanf(optarg,"%d",&lv);
            if(lv>W || lv<0){
                printf("некоректные данные для координаты высоты левого угла. Значение должно быть от 0, до %d\n", H);
                for(int i=0; i<H; i++){
                    free(arr[i]);
                }
                free(arr);
                return 0;
            }
            break;
        case'i':
            sscanf(optarg,"%d", &ls);
            if(lv>H || lv<0){
                printf("некоректные данные для координаты ширины левого угла. Значение должно быть от 0, до %d\n", W);
                for(int i=0; i<H; i++){
                    free(arr[i]);
                }
                free(arr);
                return 0;
            }
            break;
        case'd':
            sscanf(optarg,"%d",&d);
            if(d<0 || d>W){
                printf("некоректные данные для длины стороны квадрата\n");
```

```
            for(int i=0; i<H; i++){
                free(arr[i]);
            }
            free(arr);
            return 0;
        }
        break;
    case'o':
        sscanf(optarg,"%d",&to);
        if(to<0){
            printf("некоректная толщина\n");
            for(int i=0; i<H; i++){
                free(arr[i]);
            }
            free(arr);
            return 0;
        }
        break;
    case'q':
        sscanf(optarg, "%d,%d,%d", &zr,&zg,&zb);
        if(zr<0 || zr> 255 || zg<0 || zg>255 || zb<0 || zg>255){
            printf("данные о цвете заливки некоректны, введите 3 значения череза
запятую, значени от 0 до 255\n");
            for(int i=0; i<H; i++){
                free(arr[i]);
            }
            free(arr);
            return 0;
        }
        break;
    case'p':
        sscanf(optarg,"%d",&z);
        if(z!=1 && z!=0){
            printf("данные некоректны, принимаются значени 0 или 1\n");
            for(int i=0; i<H; i++){
```

```
                free(arr[i]);
            }
            free(arr);
            return 0;
        }
        break;
    case 'l':
        sscanf(optarg,"%d,%d,%d", &lr,&lg,&lb);
        if(lr<0 || lr> 255 || lg<0 || lg>255 || lb<0 || lg>255){
            printf("данные о цвете линии некоректны, введите 3 значения череза запятую,
значени от 0 до 255\n");
            for(int i=0; i<H; i++){
                free(arr[i]);
            }
            free(arr);
            return 0;
        }
        break;
    case'k':
        sscanf(optarg, "%d,%d,%d", &cr,&cg,&cb);
        if((cr!=1 && cr!=0)||(cg!=1 && cg!=0)||(cb!=1 && cb!=0)){
            printf("данные о том, какой цвет нужно изменить некоректны, введите 3
значения через запятую, значения 1 или 0\n");
            for(int i=0; i<H; i++){
                free(arr[i]);
            }
            free(arr);
            return 0;
        }
        break;
    case'm':
        sscanf(optarg,"%d", &c);
        if(c<0 || c>255){
            printf("данные о цвете  некоректны, введите  значение от 0 до 255\n");
            for(int i=0; i<H; i++){
```

```
            free(arr[i]);
        }
        free(arr);
        return 0;
    }
    break;
case'g':
    sscanf(optarg,"%d",&pv);
    if(pv<0 || pv> H) {
        printf("высота правого нижнего угла некоректна\n");
        for (int i = 0; i < H; i++) {
            free(arr[i]);
        }
        free(arr);
        return 0;
    }
    break;
case'w':
    sscanf(optarg,"%d",&ps);

    if(ps<0 || ps> W) {
        printf("ширина правого нижнего угла некоректна\n");
        for (int i = 0; i < H; i++) {
            free(arr[i]);
        }
        free(arr);
        return 0;
    }
    break;
case'u':
    sscanf(optarg,"%d",&u);

    if(u!=90 && u!=180 && u!=270 ) {
        printf("угол не верный, введите 90/180/270\n");
        for (int i = 0; i < H; i++) {
```

```
                free(arr[i]);
            }
            free(arr);
            return 0;
        }
        break;
    case 'n':
        sscanf(optarg,"%s",outputName);
    case '?':
        print_menu();
        for(int i=0; i<H; i++){
            free(arr[i]);
        }
        free(arr);
        return 0;
    case'!':
        printFileHeader(bmfh);
        printInfoHeader(bmif);
        for(int i=0; i<H; i++){
            free(arr[i]);
        }
        free(arr);
        return 0;
    default:
        break;;
  }
  opt=getopt_long(argc, argv, opts, longOpts, &longIndex);
}
if(s) {
  square(arr, lv1, ls1, to, d, lr, lg, lb, z, zr, zg, zb);
}
if(sc){
  set_color(arr, cr,cg,cb,c,H,W);
}
if(t) {
```

```
        if(lv==H && ps!=W){
            printf("нельзя брать дл поворота масимальную высоту и не максимальную длину одновременно\n");
            for (int i = 0; i < H; i++) {
                free(arr[i]);
            }
            free(arr);
            return 0;
        }
        turn(arr, lv, ls, pv, ps, u, bmif, bmfh);
        if(ps-ls==W && lv-pv==H){
            return 0;
        }
    }
    FILE *ff = fopen(outputName, "wb");

    bmif.height = H;
    bmif.width = W;
    fwrite(&bmfh, 1, sizeof(BitmapFileHeader),ff);
    fwrite(&bmif, 1, sizeof(BitmapInfoHeader),ff);
    unsigned int w = (W) * sizeof(Rgb) + ((W)*3)%4;
    for(int i=0; i<H; i++){
        fwrite(arr[i],1,w,ff);
    }
    fclose(ff);
    for(int i=0; i<H; i++){
        free(arr[i]);
    }
    free(arr);

    return 0;
}
```