МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: обработка изображений на языке Си

Студент гр. 0382 _____ Сергеев Д.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Сергеев Д.А.

Группа 0382

Тема работы: Обработка изображений на языке Си

Исходные данные:

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 весь следующий функционал по обработке bmp-файла

1. Отражение заданной области. Этот функционал определяется:
    - выбором оси относительно которой отражать (горизонтальная или вертикальная)
    - Координатами левого верхнего угла области

- Координатами правого нижнего угла области

2. Копирование заданной области. Функционал определяется:
    - Координатами левого верхнего угла области-источника
    - Координатами правого нижнего угла области-источника
    - Координатами левого верхнего угла области-назначения
3. Заменяет все пиксели одного заданного цвета на другой цвет. Функционал определяется:
    - Цвет, который требуется заменить
    - Цвет на который требуется заменить
4. Разделяет изображение на N*M частей. Реализация: либо провести линии заданной толщины, тем самым разделив изображение либо сохранение каждой части в отдельный файл. -- по желанию студента (можно и оба варианта). Функционал определяется:
    - Количество частей по “оси” Y
    - Количество частей по “оси” X
    - Толщина линии
    - Цвет линии
    - Либо путь куда сохранить кусочки

Содержание пояснительной записки:
«Содержание», «Введение», «Задание», «Ход выполнения работы», «Тестирование», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 05.04.2021

Дата сдачи реферата: 29.05.2021

Дата защиты реферата: 31.05.2021

Студент _____ Сергеев Д.А.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

В ходе выполнения работы была разработана программа на языке Си для обработки изображений в формате BMP, которая принимает на вход некоторые команды и исходное изображение, и в зависимости от команд либо отражает заданную область, либо копирует заданную область, либо заменяет все пиксели одного цвета на другой, либо разделяет изображение на части. Для считывания, записи и изменения содержимого файлов использовалась стандартная библиотека языка Си.

Для удобства пользования программа реализована в виде консольной утилиты.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – создать приложение на языке Си в виде консольной утилиты для обработки изображений формата BMP согласно запросам пользователя.

Для выполнения работы необходимо выполнить следующие задачи:

- Создание структур для работы с файлами
- Считывание и запись BMP файла
- Обработка запросов пользователя
- Изменение исходного изображения
- Обработка возможных ошибок

Для решения первой задачи была изучена структура BMP формата.
Для решения второй задачи были использованы стандартные методы библиотеки stdio.h.
Для решения третьей задачи была использована библиотека getopt.h и unistd.h.
Для решения четвертой задачи был обработан двумерный массив.
Для решения пятой задачи были отловлены возможные ошибки и краши.

1. ЗАДАНИЕ

Вариант 10

Программа **должна** иметь 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 весь следующий функционал по обработке bmp-файла:

1. Отражение заданной области. Этот функционал определяется:
    - выбором оси относительно которой отражать (горизонтальная или вертикальная)
    - Координатами левого верхнего угла области
    - Координатами правого нижнего угла области
2. Копирование заданной области. Функционал определяется:
    - Координатами левого верхнего угла области-источника
    - Координатами правого нижнего угла области-источника
    - Координатами левого верхнего угла области-назначения
3. Заменяет все пиксели одного заданного цвета на другой цвет. Функционал определяется:
    - Цвет, который требуется заменить
    - Цвет на который требуется заменить

4. Разделяет изображение на N*M частей. Реализация: либо провести линии заданной толщины, тем самым разделив изображение либо сохранение каждой части в отдельный файл. -- по желанию студента (можно и оба варианта). Функционал определяется:
    - Количество частей по “оси” Y
    - Количество частей по “оси” X
    - Толщина линии
    - Цвет линии

    Либо путь куда сохранить кусочки

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Считывание изображения

В соответствии с известной структурой файлов BMP формата создаются структуры BitmapFileHeader и BitmapInfoHeader, которые имеют соответствующие поля. Также создаётся двумерный массив объектов структуры Rgb(имеются 3 поля, соответствующие красной, зелёной и жёлтой компоненте), в этом массиве будет хранится непосредственно сама картинка. Считывание структур BitmapFileHeader и BitmapInfoHeader происходит по одному биту, а сама картинка считывается построчно.

2.2 Обработка изображения

2.2.1 Отражение заданной области

Функция изменяет массив пикселей таким образом, чтобы при вызове отражения по горизонтали правый верхний угол стал левый левым верхним, правый нижний угол стал левым нижним и так далее для остальных пикселей. А при вызове отражения по вертикали правый верхний угол стал правым нижним, правый нижний – правым верхним и так далее для остальных пикселей.

2.2.2 Копирование заданной области

Функция копирует определенную область исходного изображения, а далее начиная левого верхнего угла, заданного пользователем, заменяет исходное изображения на скопированную область.

2.2.3 Замена всех пикселей одного цвета на другой

Функция проходит по всему изображению и в случае совпадения цвета, который нужно заменить, и цвета пикселя, меняет цвет данного пикселя на второй цвет, заданный пользователем.

2.2.4 Разделение изображения на N*M частей

Функция в соответствии с заданными пользователями значения рисует полосы по вертикали и по горизонтали на равном расстоянии друг от друга, определенной толщины и цвета.

2.3 Вывод информации о программе

Данная функция при вызове печатает информацию о программе и о доступных командах.

2.4 Вывод информации о изображении

При вызове данной функции на терминал выводится информация о изображении, а именно содержимое структур BitmapFileHeader и BitmapInfoHeader.

2.5 Запись изображения

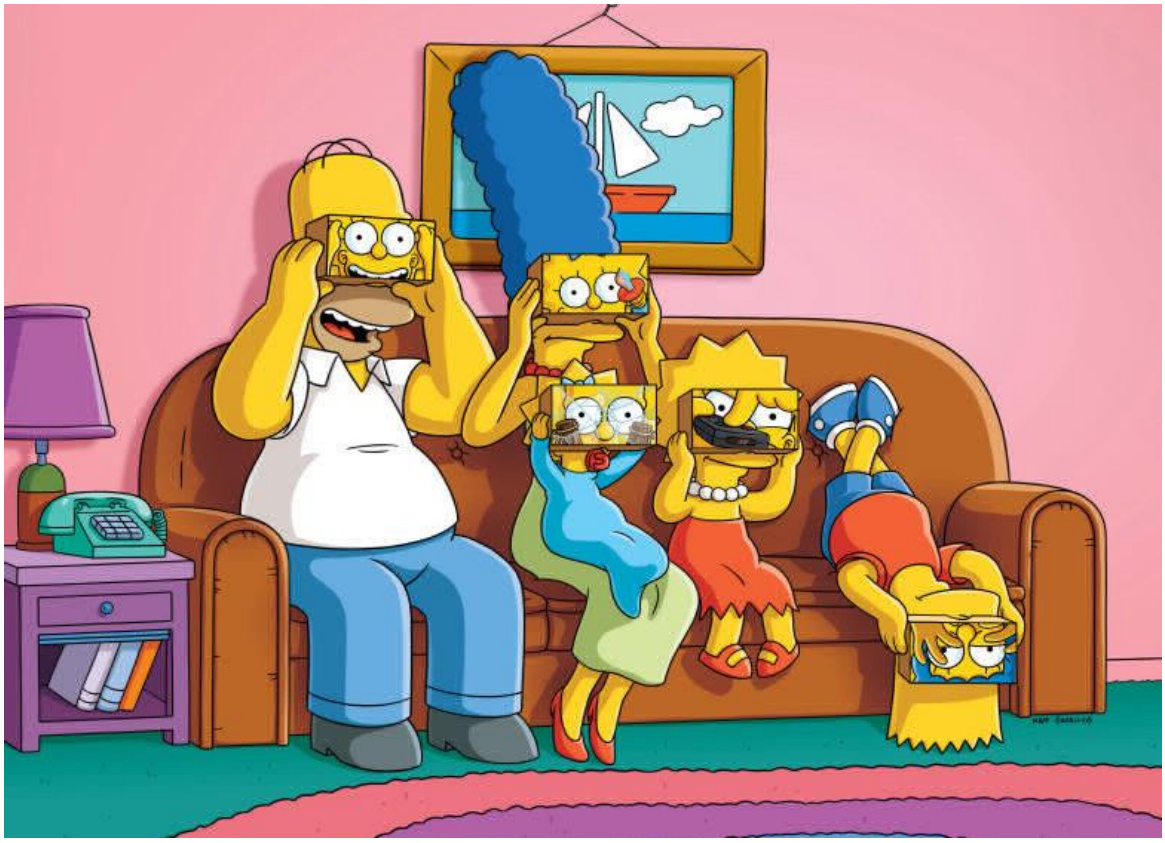
Создаётся новый файл с заданным или по умолчанию именем, в который сначала записывается информация о формате изображения (BitmapFileHeader) и информация о самом изображении (BitmapInfoHeader) по одному биту, а далее и сам массив пикселей по одной строке.

2.6 Интерфейс

В данной программе реализован CLI (Command Line Interface), для этого используется функция getopt_long(), создан массив структур option long option[] и строка optstring. Далее обрабатывается каждая команда, и с помощью оператора switch() исполняется определенная функция. Интерфейс поддерживает как длинные так и короткие ключи.

3. ТЕСТИРОВАНИЕ

Исходное изображение:



3.1 Информация об изображении и информация о программе

Ввод: ./cw –h –i simpsonsvr.bmp

Результат:

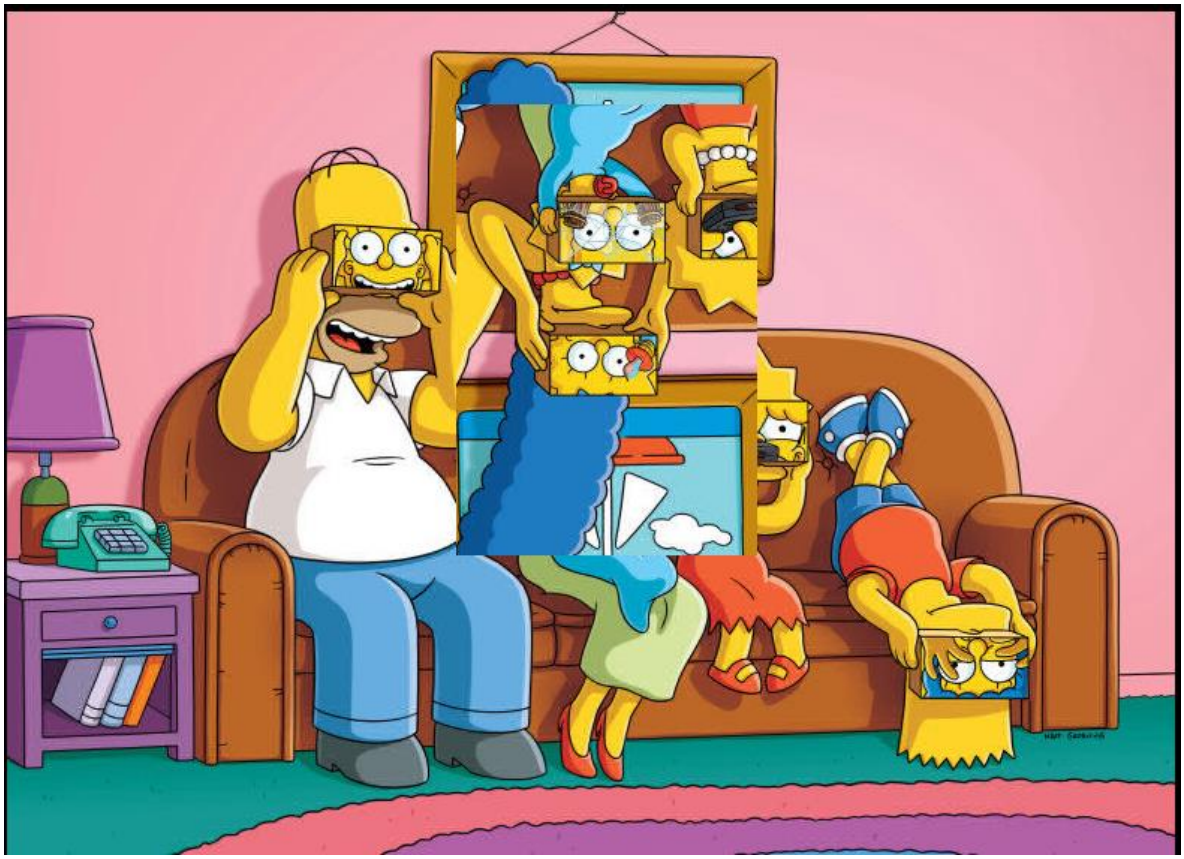
```
dimasergeev02@dimasergeev02-VirtualBox:~/kurs final$ ./cw -h -i simpsonsvr.bmp
////////////////////////////////////////////////////////////////////////
This program supports uncompressed, 24 bit per color BMP format images.
Please, read infromation below and run program again.
////////////////////////////////////////////////////////////////////////
Information about possible commands:     -h,  --help

Image information:       -i,  --info

Name of output file:     -o,  --output
(Example: -o filename.bmp)
Default: output.bmp

Change all pixels of one color to another color:         -s,  --set
Input requirements:      replaceable color, replacing color.
(Example: -s [r] [g] [b] [r1] [g1] [b1] )

Divide the image into NxM parts:         -l,  --lines
Input requirements:      number of horizontal and vertical segments,line thickness, color (for each parameter).
(Example: -l [Ox] [Oy] [thick] [red] [green] [blue])

Reflect segment horizontally or vertical:        -r,  --reflection
Input requirements:      axis of reflection, coordinates of left upper and rigth lower corners.
(Example: -r [axis_of_reflection] [LeftUpperx] [LeftUppery] [RightLowerx] [RightLowery])

Copy segment of image and insert it in certain place:    -c,  --copy
Input requirements:      coordinates of left upper and right lower corners of copied segment, coordinates of left upper corner of place of insetation.
(Example: -c [LeftUpperx] [LeftUppery] [RightLowerx] [RightLowery] [LeftUpperxi] [LeftUpperyi])

///////////////// BITMAPFILEHEADER information ///////////////
signature:      4d42 (19778)
filesize:       141a62 (1317474)
reserved1:      0 (0)
reserved2:      0 (0)
pixelArrOffset: 36 (54)
///////////////// BITMAPINFOHEADER information ///////////////
headerSize:     28 (40)
width:          30c (780)
height:         233 (563)
planes:         1 (1)
bitsPerPixel:   18 (24)
compression:    0 (0)
imageSize:      0 (0)
xPixelsPerMeter:        2e23 (11811)
yPixelsPerMeter:        2e23 (11811)
colorsInColorTable:     0 (0)
importantColorCount:    0 (0)
dimasergeev02@dimasergeev02-VirtualBox:~/kurs final$
```

Вывод программа работает верно

3.2 Отражение заданной области

Ввод: ./cw -r vertical 300 500 500 200 simpsonsvr.bmp

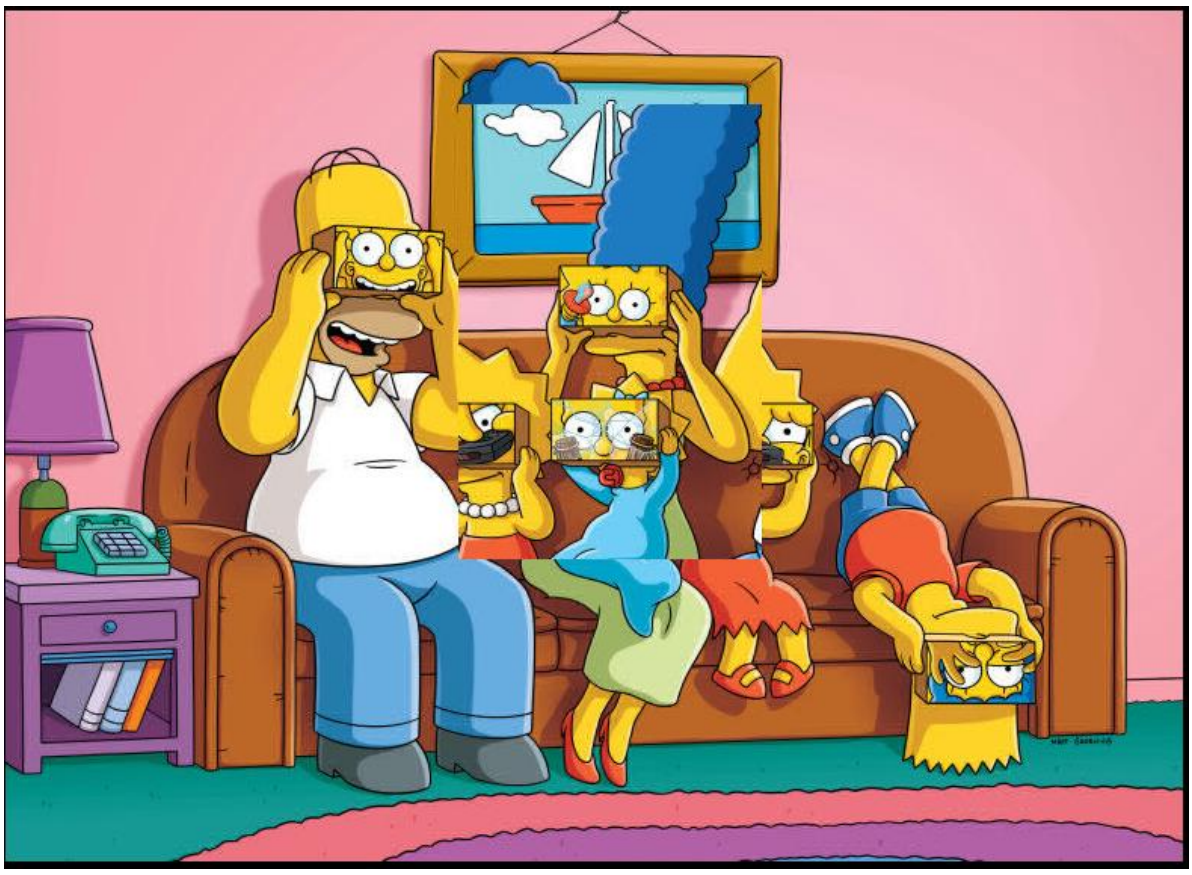
Результат:



Вывод: программа работает верно

Ввод: ./cw -r horizontal 300 500 500 200 simpsonsvr.bmp

Результат:

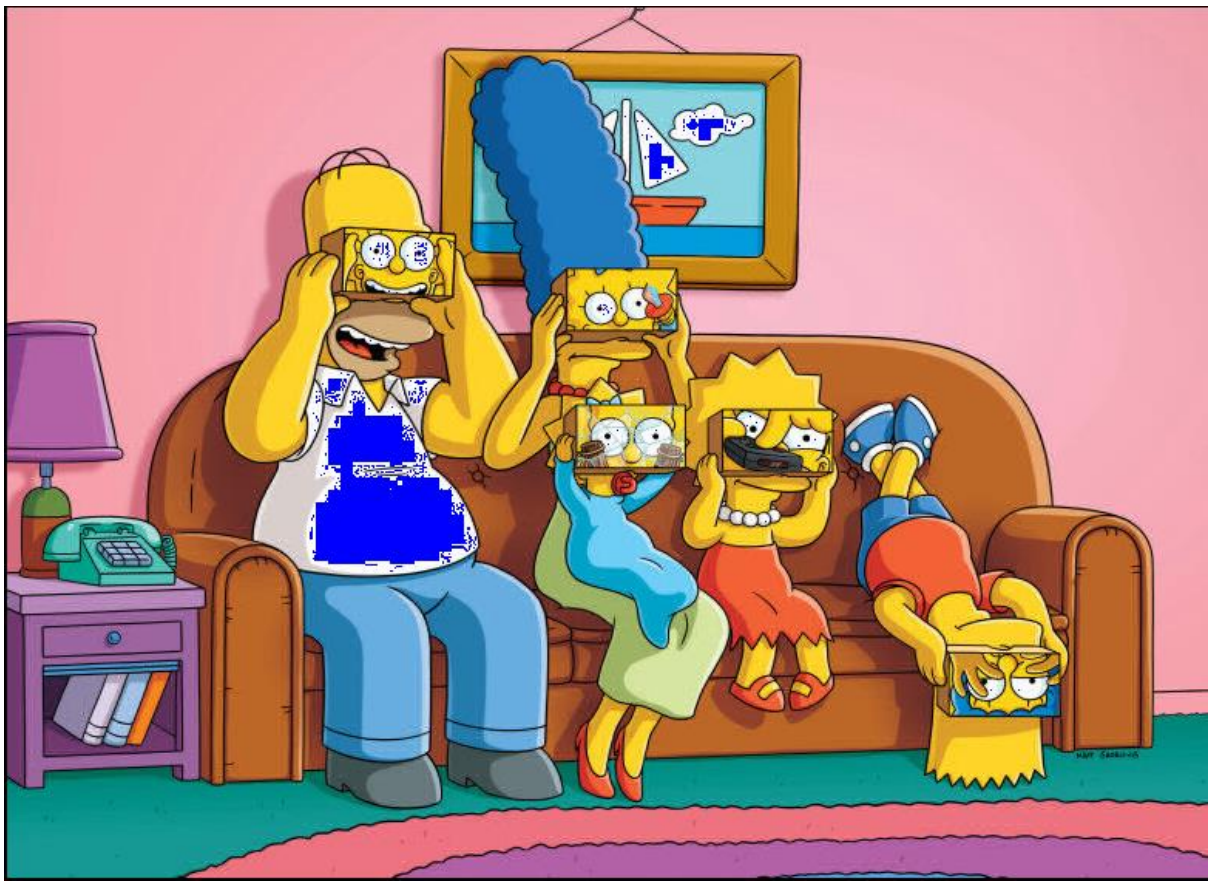


Вывод: программа работает верно

3.3 Замена всех пикселей одного цвета на другой

Ввод: ./cw -s 255 255 255 0 0 255 simpsonsvr.bmp

Результат:

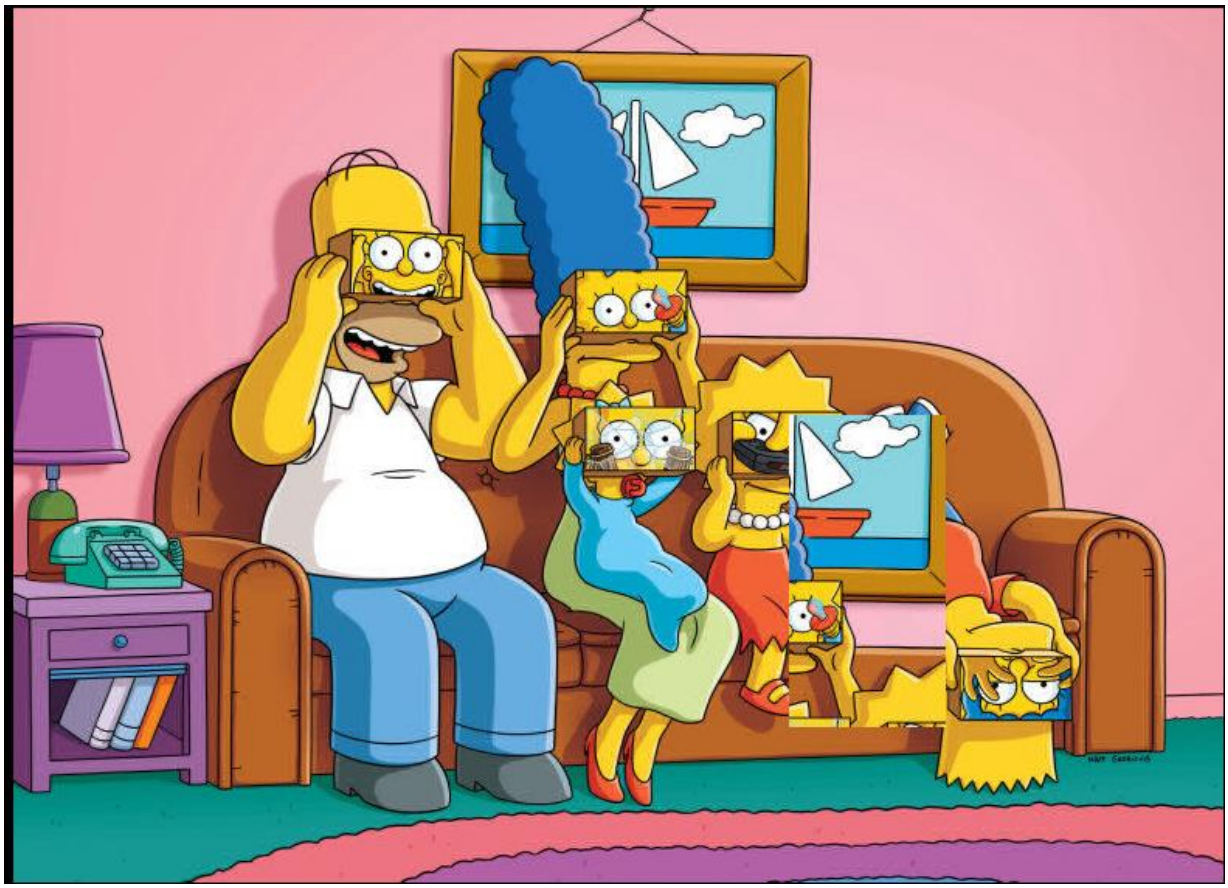


Вывод: программа работает верно

3.4 Копирование заданной области

Ввод: ./cw --copy 400 500 500 300 500 300 simpsonsvr.bmp

Результат:

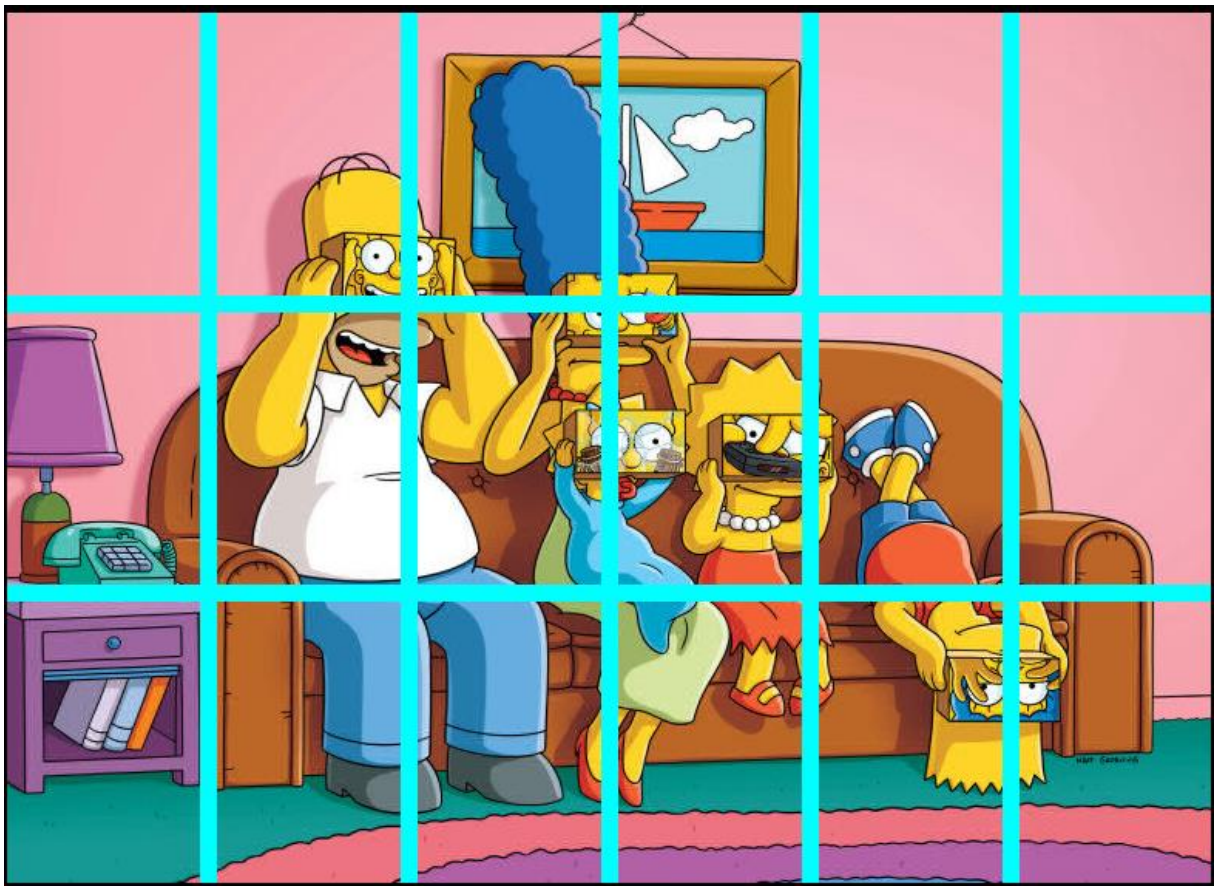


Вывод: программа работает верно.

3.5 Разделение изображения на N*M частей

Ввод: ./cw --lines 6 3 10 0 255 255 simpsonsvr.bmp

Результат:



Вывод: программа работает правильно

4. ЗАКЛЮЧЕНИЕ

В ходы выполнения работы была создана программа для обработки BMP изображений в соответствии с командами пользователя. Для этого был реализован CLI интерфейс, а также выполнены все поставленные задачи.

5. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б. и Ритчи Д. Язык программирования Си. М.: Вильямс, 1978 288 с.
2. Cplusplus URL: http://cplusplus.com
3. https://firststeps.ru/linux/r.php?10
4. https://firststeps.ru/linux/r.php?11

ПРИЛОЖЕНИЕ А

Исходный код приложения

Название файла: cw.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <unistd.h>
#include <stdint.h>
#include <getopt.h>


unsigned int H;
unsigned int W;
char* optstring="hircslo:";
char* outputFile="output.bmp";
char* inputFile;
int flag = 2;

const struct option long_options[] = {
        {"help",no_argument,NULL,'h'},
        {"reflection",no_argument,NULL,'r'},
        {"info",no_argument,NULL,'i'},
        {"copy",no_argument,NULL,'c'},
        {"set",no_argument,NULL,'s'},
        {"lines",no_argument,NULL,'l'},
        {"output",no_argument,NULL,'o'},
        {NULL,0,NULL,0}
};

#pragma pack (push, 1)
typedef struct{
    uint16_t signature;
    uint32_t filesize;
```

```
    uint16_t reserved1;
    uint16_t reserved2;
    uint32_t pixelArrOffset;
} BitmapFileHeader;

#pragma pack(pop)

typedef struct{
    uint32_t headerSize;
    uint32_t width;
    uint32_t height;
    uint16_t planes;
    uint16_t bitsPerPixel;
    uint32_t compression;
    uint32_t imageSize;
    uint32_t xPixelsPerMeter;
    uint32_t yPixelsPerMeter;
    uint32_t colorsInColorTable;
    uint32_t importantColorCount;
} BitmapInfoHeader;

BitmapFileHeader bmfh;
BitmapInfoHeader bmif;

typedef struct
{
    uint8_t b;
    uint8_t g;
    uint8_t r;
} Rgb;

Rgb** arr;

void printFileHeader(BitmapFileHeader header){
```

```
    printf("/////////////// BITMAPFILEHEADER information
///////////////\n");
    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("/////////////// BITMAPINFOHEADER information
///////////////\n");
    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 setColor (int r,int g,int b,int r1,int g1,int b1)
{
    for (int i=0;i<H;i++)
    {
        for (int j=0;j<W;j++)
        {
            if (arr[i][j].r==r && arr[i][j].g==g &&
arr[i][j].b==b)
            {
                    arr[i][j].r=r1;
                    arr[i][j].g=g1;
                    arr[i][j].b=b1;
               }
        }
    }
}

void cutPicture(int m,int n,int wid,int r,int g,int b)
{
    int ny=H/n;
    for (int i=1;i<=(n-1);i++)
    {
        for (int j=0;j<W;j++)
        {
            for (int k=-wid/2;k<wid/2+1;k++) {
                arr[i * ny + k][j].b = b;
                arr[i * ny + k][j].g = g;
                arr[i * ny + k][j].r = r;
            }
```

```
        }
    }
    int mx=W/m;
    for (int i=1;i<=(m-1);i++)
    {
        for (int j=0;j<H;j++)
        {
            for (int k=-wid/2;k<wid/2+1;k++) {
                arr[j][i*mx+k].b = b;
                arr[j][i*mx+k].g = g;
                arr[j][i*mx+k].r = r;
            }
        }
    }
}

void copySegment(int x1, int y1, int x2, int y2,int xi,int yi)
{
    for (int i=y1;i>=y2;i--)
    {
        for (int j=x1;j<=x2;j++)
        {
            arr[yi+(i-y1)][xi+(j-x1)]=arr[i][j];
        }
    }
}

void pictureReflection(char* way, int xl, int yl, int xr, int yr)
{
    if (strcmp(way,"vertical")==0)
    {
        Rgb tmp;
        for (int i=yl;i>=(yl+yr)/2;i--)
        {
            for (int j=xl;j<=xr;j++)
```

```
            {
                tmp=arr[i][j];
                arr[i][j]=arr[yr+(yl-i)][j];
                arr[yr+(yl-i)][j]=tmp;
            }
        }
    }
    if (strcmp(way,"horizontal")==0)
    {
        Rgb tmp;
        for (int j=xl;j<=(xr+xl)/2;j++)
        {
            for (int i=yl;i>=yr;i--)
            {
                tmp=arr[i][j];
                arr[i][j]=arr[i][xr-(j-xl)];
                arr[i][xr-(j-xl)]=tmp;
            }
        }
    }
}

int checkRGB(int r,int g,int b)
{
          if (r<0 || r>255)
          {
               return 0;
          }
          if (g<0 || g>255)
          {
               return 0;
          }
          if (b<0 || b>255)
          {
               return 0;
```

```
            }
            return 1;
}

void printHelp()
{

printf("////////////////////////////////////////////////////////////
///////////////\n");
    printf("This program supports uncompressed, 24 bit per color
BMP format images.\n");
    printf("Please, read infromation below and run program
again.\n");

printf("////////////////////////////////////////////////////////////
///////////////\n");
    printf("Information about possible commands:\t -h,  --help
\n");
    printf("\n");
    printf("Image information:\t -i,  --info \n");
    printf("\n");
    printf("Name of output file:\t -o,  --output \n");
    printf("(Example: -o filename.bmp) \n");
    printf("Default: output.bmp\n");
    printf("\n");
    printf("Change all pixels of one color to another color:\t -s,
--set \n");
    printf("Input requirements:\t replaceable color, replacing
color.\n");
    printf("(Example: -s [r] [g] [b] [r1] [g1] [b1] ) \n");
    printf("\n");
    printf("Divide the image into NxM parts:\t -l,  --lines\n");
    printf("Input requirements:\t number of horizontal and
vertical segments,line thickness, color (for each parameter).\n");
```

```
    printf("(Example: -l [Ox] [Oy] [thick] [red] [green]
[blue])\n");
    printf("\n");
    printf("Reflect segment horizontally or vertical:\t -r,  --
reflection \n");
    printf("Input requirements:\t axis of reflection, coordinates
of left upper and rigth lower corners.\n");
    printf("(Example: -r [axis_of_reflection] [LeftUpperx]
[LeftUppery] [RightLowerx] [RightLowery]) \n");
    printf("\n");
    printf("Copy segment of image and insert it in certain
place:\t -c,  --copy \n");
    printf("Input requirements:\t coordinates of left upper and
right lower corners of copied segment, coordinates of left upper
corner of place of insetation.\n");
    printf("(Example: -c [LeftUpperx] [LeftUppery] [RightLowerx]
[RightLowery] [LeftUpperxi] [LeftUpperyi]) \n");
    printf("\n");
}

void createOutput()
{
    FILE *ff = fopen(outputFile, "wb");
    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);
    free(arr);
}

int main(int argc, char ** argv){
     if(argc == 1)
```

```
     {
          printHelp();
        printf("Please try again\n");
        return 0;
    }
    inputFile=malloc(strlen(argv[argc-1])*sizeof(char));
    strcpy(inputFile,argv[argc-1]);
    FILE *f = fopen(inputFile, "rb");
    if (f == NULL){
            printf("Please add a file.\n");
            return 0;
    }
    fread(&bmfh,1,sizeof(BitmapFileHeader),f);
    fread(&bmif,1,sizeof(BitmapInfoHeader),f);
    if(bmfh.signature!=0x4d42)
    {
          printf("Unsupported image format.\n");
          printf("Try again!\n");
          return 0;
     }
     if(bmif.bitsPerPixel!=24)
     {
          printf("The color depth of this image is not
supported.\n");
        printf("Requires 24 bits per color.\n");
          printf("Try again!\n");
        return 0;
     }
     if(bmif.colorsInColorTable!=0)
     {
          printf("The color table of this image is not
supported.\n");
          printf("Try again!\n");
          return 0;
     }
```

```
      if(bmif.compression!=0)
      {
          printf("Please use an uncompressed image.\n");
         return 0;
      }

    H = bmif.height;
    W = bmif.width;

    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);
    }
     fclose(f);
    int option_index;
    int
opt=getopt_long(argc,argv,optstring,long_options,&option_index);
    if (opt == -1)
    {
        printf("Please add the command.\n");
          return 0;
    }
    while(opt!=-1)
    {
        switch (opt)
        {
            case 'h':
            {
                    flag+=1;
                    if (flag>argc)
                    {
                         printf("The arguments do not match the
command.\n");
                    return 0;
```

```
                }
            printHelp();
            break;
        }
        case 'i':
        {
                flag+=1;
                if (flag>argc)
                {
                     printf("The arguments do not match the
command.\n");
                return 0;
                }
            printFileHeader(bmfh);
            printInfoHeader(bmif);
            break;
        }
        case 'r':
        {
                flag+=6;
                if (flag>argc)
                {
                     printf("The arguments do not match the
command.\n");
                return 0;
                }
            char* par[5];
            int long_add=0;
            if (option_index!=0)
            {
                long_add=1;
            }
            for (int k=-1+long_add;k<4+long_add;++k)
            {
                par[k+1-long_add]=(argv[optind+k+!long_add]);
```

```
                }
                if (strcmp(par[0],"horizontal")!=0 &&
strcmp(par[0],"vertical")!=0)
                {
                        printf("Please use one of avaible way of
reflection(horizontal or vertical)\n");
                        return 0;
                    }
                     if (atoi(par[1])<0 || atoi(par[2])>H-1)
                {
                        printf("Coordinates of left upper corner
is out of image.\nPlease check parameters of image(use -i(--info))
and try again.\n");
                        return 0;
                    }
                    if (atoi(par[3])>W-1 || atoi(par[4])<0)
                    {
                        printf("Coordinates of right lower corner
is out of image.\nPlease check parameters of image(use -i(--info))
and try again.\n");
                        return 0;
                    }
                    if (atoi(par[1])>atoi(par[3]) ||
atoi(par[2])<atoi(par[4]))
                    {
                        printf("Left upper corner is not left
upper corner or right lower corner in not right lower
corner\nCheck entered coordinates and try again\n");
                        return 0;
                    }
                //printf("%s %s %s %s
%s\n",par[0],par[1],par[2],par[3],par[4]);

pictureReflection(par[0],atoi(par[1]),atoi(par[2]),atoi(par[3]),at
oi(par[4]));
```

```
                break;
            }
            case 'c':
            {
                    flag+=7;
                    if (flag>argc)
                    {
                         printf("The arguments do not match the
command.\n");
                    return 0;
                    }
                int par[6];
                int long_add=0;
                if (option_index!=0)
                {
                    long_add=1;
                }
                for (int k=-1+long_add;k<5+long_add;++k)
                {
                    par[k+1-long_add]=
atoi(argv[optind+k+!long_add]);
                }
                if (par[0]<0 || par[1]>H-1)
                {
                         printf("Coordinates of left upper corner
is out of image.\nPlease check parameters of image(use -i(--info))
and try again.\n");
                         return 0;
                    }
                    if (par[2]>W-1 || par[3]<0)
                    {
                         printf("Coordinates of right lower corner
is out of image.\nPlease check parameters of image(use -i(--info))
and try again.\n");
                         return 0;
```

```
                    }
                    if (par[4]<0 || par[5]>H-1)
                {
                        printf("Coordinates of left upper corner
of inserted image is out of default image.\nPlease check
parameters of image(use -i(--info)) and try again.\n");
                        return 0;
                    }
                    if (par[0]>par[2] || par[1]<par[3])
                    {
                        printf("Left upper corner is not left
upper corner or right lower corner in not right lower
corner\nCheck entered coordinates and try again\n");
                        return 0;
                    }
                    if ((W-par[4]<par[2]-par[0]) || (par[5]<par[1]-
par[3]))
                    {
                        printf("Copied image doesn't fit in
entered coordinates\nPlease check parameters of image(use -i(--
info)) and try again.\n");
                        return 0;
                    }
                //printf("%d %d %d %d %d
%d\n",par[0],par[1],par[2],par[3],par[4],par[5]);

copySegment(par[0],par[1],par[2],par[3],par[4],par[5]);
                break;
            }
            case 's':
            {
                    flag+=7;
                    if (flag>argc)
                    {
```

```
                        printf("The arguments do not match the
command.\n");
                    return 0;
                    }
                int long_add=0;
                if (option_index!=0)
                {
                    long_add=1;
                }
                int par[6];
                for (int k=-1+long_add;k<5+long_add;++k)
                {
                    par[k+1-
long_add]=atoi((argv[optind+k+!long_add]));
                }
                if (checkRGB(par[3],par[4],par[5])==0)
                    {
                        printf("Please set color components from 0
to 255 and try again.\n");
                        return 0;
                    }
                    if (checkRGB(par[0],par[1],par[2])==0)
                    {
                        printf("Please set color components from 0
to 255 and try again.\n");
                        return 0;
                    }
                //printf("%s %s",par[0],par[1]);

setColor(par[0],par[1],par[2],par[3],par[4],par[5]);
                break;
            }
            case 'l':
            {
                    flag+=7;
```

```
                    if (flag>argc)
                    {
                         printf("The arguments do not match the
command.\n");
                    return 0;
                    }
                int long_add=0;
                if (option_index!=0)
                {
                    long_add=1;
                }
                int par[6];
                for (int k=-1+long_add;k<5+long_add;++k)
                {
                    par[k+1-
long_add]=atoi((argv[optind+k+!long_add]));
                }
                if (par[0]<0 || par[1]<0)
                {
                         printf("Please enter values>0 and try
again.\n");
                         return 0;
                    }
                    if (par[0]>30 || par[1]>30)
                    {
                         printf("Please use values<30 because there
is a possibility of error.\n");
                         return 0;
                    }
                    if (par[2]<0 || par[2]>W || par[2]>H)
                    {

                         printf("Please, change value of line's
width. For example, use 5 or 10.\n");
                         return 0;
```

```
                    }
                    if (checkRGB(par[3],par[4],par[5])==0)
                    {
                        printf("Please set color components from 0
to 255 and try again.\n");
                        return 0;
                    }
                //printf("%d %d %d",par[0],par[1],par[2]);

cutPicture(par[0],par[1],par[2],par[3],par[4],par[5]);
                break;
            }
            case 'o':
            {
                    flag+=2;
                    if (flag>argc)
                    {
                        printf("The arguments do not match the
command.\n");
                    return 0;
                    }
                outputFile=optarg;
                break;
            }
            default:
            {
                break;
            }
        }

opt=getopt_long(argc,argv,optstring,long_options,&option_index);
    }
    createOutput();
    return 0;
}
```