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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Работа с .BMP файлами

Студент гр. 9304 _____ Абдулрахман М.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Абдулрахман М.

Группа 9304

Тема работы: Работа с .BMP файлами

Исходные данные:

Некоторый набор аргументов, включающий в себя как минимум входной файл (должен располагаться на последнем месте) формата .BMP, 24 бита на цвет, беспалитровый.

Содержание пояснительной записки:

Аннотация

Введение

Основные теоретические положения.

Описание кода программы

Заключение

Список использованных источников.

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 01.03.2020

Дата сдачи реферата: 28.09.2020

Дата защиты реферата: 28.09.2020

Студент _____ Абдулрахман М.

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

Курсовая работа представляет собой программу, предназначенную для работы с файлами формата .BMP (версия формата: 24 бита на цвет, беспалитровый) с обеспечением базового функционала задания: рисования треугольника, определение максимального по площади прямоугольника, создание коллажа (1 файл на вход). Код программы написан на языке программирования C++, запуск программы подразумевается на операционных системах семейства Linux. При разработке кода программы активно использовались функции стандартных библиотек языка C++, основные управляющие конструкции языка C++. Также для каждой подзадачи были реализованы свои собственные функции. Взаимодействие с пользователем происходит посредством интерфейса командной строки (CLI). Для проверки работоспособности программы проводилось тестирование. Исходный код, скриншоты, показывающие корректную работу программы, и результаты тестирования представлены в приложениях.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы — разработка программы для работы с файлами формата .BMP (тип формата: 24 бита на цвет, беспалитровый), соответствующего заданию работы. Создание интерфейса командной строки для программы.

Для достижения поставленной цели требуется реализовать следующие задачи:

1. Изучение теоретического материала по написанию кода на языке C++.
2. Разработка программного кода в рамках полученного задания.
3. Написание программного кода.
4. Тестирование программного кода.

Полученное задание:

Программа **должна** иметь CLI или GUI.

Программа должна реализовывать весь следующий функционал по обработке bmp-файла

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 треугольника. Треугольник определяется
    - Координатами его вершин
    - Толщиной линий
    - Цветом линий
    - Треугольник может быть залит или нет
    - цветом которым он залит, если пользователем выбран залитый
- (2) Находит самый большой прямоугольник заданного цвета и перекрашивает его в другой цвет. Функционал определяется:
    - Цветом, прямоугольник которого надо найти
    - Цветом, в который надо его перекрасить
- (3) Создать коллаж размера N*M из одного либо нескольких фото -- на выбор студента (либо оба варианта по желанию). В случае с одним изображением коллаж представляет собой это же самое изображение повторяющееся N*M раз.
    - Количество изображений по “оси” Y
    - Количество изображений по “оси” X
    - Перечень изображений (если выбрана усложненная версия задания)

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

1.1. Основные управляющие конструкции языка C++.

Условный оператор if ... else. Форма оператора следующая:

```
if ( условие )
    оператор1
else
    оператор2
```

Если условие истинно, то выполнятся первый оператор, иначе второй. Если в условии отрабатываются несколько операторов, то они помещаются фигурные скобки. Вторая часть (else) может опускаться если нам не нужно обрабатывать ложное условие. Допускается вложение условных операторов.

Итерационный цикл - цикл управляемый счетчиком.

Структура следующая:

```
for ( инициализация; условие; операция ) {
     операторы
}
```

Первый параметр инициализирует счетчик, второй - условие при истиности которого выполняются операторы, третий - операция выполняемая перед новой итерацией.

Цикл с предусловием - цикл управляемый условием.

Структура следующая:

```
while ( условие ) {
     операторы
 }
```

Цикл выполняется пока условие истинно (условие проверяется в начале).

Оператор выбора switch. Форма оператора следующая:

```
switch (выражение) {
 case константа_1 : операторы ; break;
 ...
 case константа_n : операторы ; break;
 default : операторы ; break;
 }
```

Оператор switch вычисляет выражение и переходит к первому совпадающему значению после case. Далее выполняются операторы этого блока

и по команде break происходит выход из структуры. Если ни одно из значений не совпадает с константами из case, то выполняются операторы блока default. Отметим, что константы в case блоке определяются на этапе компиляции, поэтому они не могут содержать переменных и функций.

Операторы break и continue. Использование оператора break мы уже видели на примере структуры switch. Оператор break - это выход из цикла или конструкции switch.Оператор continue - переход на конец цикла (т.е. пропуск всех операторов от continue до конца структуры цикла).

1.2. Функции и операторы стандартных библиотек языка C++.

Stdio.h:

size_t fread (void * ptr, size_t size, size_t count, FILE * stream); - принимает на вход указатель на буфер, куда будет считана информация, размер блока памяти, количество блоков, указатель на файл, из которого считывается информация.

size_t fwrite (const void * ptr, size_t size, size_t count, FILE * stream); - принимает на вход указатель на буфер, куда будет считана информация, размер блока памяти, количество блоков, указатель на файл, в который записывается информация.

FILE * fopen (const char * filename, const char * mode); - принимает на вход строку — имя файла и строку, которая характеризует формат работы с файлом(чтение, запись и т. д.). Возвращает указатель на FILE.

int fclose (FILE * stream); - принимает на вход указатель на файл, закрывает его. Возвращает целое число.

String.h:

char * strcpy (char * destination, const char * source) - принимает на вход два аргумента: указатель на начало строки, куда будет скопирована строка-источник, указатель на начало строки-источника. Возвращает указатель на начало строки, куда была скопирована строка-источник.

int strcmp (const char * str1, const char * str2) - принимает на вход 2 аргумента: указатели на начало первой и второй строк. Возвращает нулевое значение строк, если они равны, значение меньше нуля, если код первого символа строки_1 меньше кода строки_2. В противном случае возвращает большее, чем ноль значение.

size_t strlen (const char * str) - принимает на вход один аргумент — указатель на начало строки. Возвращает её длину, без учёта символа конца строки.

Void* operator new[], void* operator new – операторы для выделения динамической памяти указанного типа.

Void* operator [] delete, void* operator delete – операторы для очистки динмаически выделенной памяти.

2. ОПИСАНИЕ КОДА ПРОГРАММЫ

2.1. Функция отрисовки треугольника

```
void BMP::draw_triangle(Triangle triangle){
    uint32_t x0 = triangle.X1, y0 = triangle.Y1, x1 = triangle.X2, y1 = triangle.Y2, x2 = triangle.X3, y2 = triangle.Y3;
    uint8_t B = triangle.color_B, G = triangle.color_G, R = triangle.color_R, A = ALPHA, line_w = triangle.thickness;
    bool filled = triangle.filled;
    uint8_t BF = triangle.filled_color_B, GF = triangle.filled_color_G, RF = triangle.filled_color_R;
//    triangle.filename

    uint32_t xmax_image = bmp_info_header.width;
    uint32_t ymax_image = bmp_info_header.height;
    uint32_t xmax, ymax, xmin, ymin;
    if ((x0==x1 &&y0==y1) || (x1==x2 && y1==y2) || (x0==x2 && y0==y2)){
        throw std::runtime_error("The coordinates do not form a triangle!");
```

```
        }
        if (x0<xmax_image && x1<xmax_image && x2<xmax_image &&
                y0<ymax_image && y1<ymax_image && y2<ymax_image){


            std::vector<uint32_t> line_1_pixels =
                    drawColoredLine(x0,y0,x1,y1,line_w, B,G,R,A);
            std::vector<uint32_t> line_2_pixels =
                    drawColoredLine(x1,y1,x2,y2,line_w, B,G,R,A);
            std::vector<uint32_t> line_3_pixels =
                    drawColoredLine(x2,y2,x0,y0,line_w, B,G,R,A);
            if (filled){
                // x0 x1 x2
                // y0 y1 y2

                xmax = max(max(x0,x1),x2);
                ymax = max(max(y0,y1),y2);
                xmin = min(min(x0,x1),x2);
                ymin = min(min(y0,y1),y2);

                for (uint32_t xi = xmin; xi<xmax; xi++){
                    for (uint32_t yi = ymin; yi<ymax; yi++){
                        if (inside_triangle(xi, yi, triangle)){
                            vector<uint32_t>::iterator
                                    itx1=
std::find(line_1_pixels.begin(), line_1_pixels.end(),xi),
                                    itx2=
std::find(line_2_pixels.begin(), line_2_pixels.end(),xi),
                                    itx3=
std::find(line_3_pixels.begin(), line_3_pixels.end(),xi);
                            vector<uint32_t>::iterator
                                    ity1=
std::find(line_1_pixels.begin(), line_1_pixels.end(),yi),
                                    ity2=
std::find(line_2_pixels.begin(), line_2_pixels.end(),yi),
                                    ity3=
std::find(line_3_pixels.begin(), line_3_pixels.end(),yi);
```

```
                                int index1 =
std::distance(line_1_pixels.begin(), itx1);
                                int index2 =
std::distance(line_2_pixels.begin(), itx2);
                                int index3 =
std::distance(line_3_pixels.begin(), itx3);

                                int index1y =
std::distance(line_1_pixels.begin(), ity1);
                                int index2y =
std::distance(line_2_pixels.begin(), ity2);
                                int index3y =
std::distance(line_3_pixels.begin(), ity3);


                                if (itx1 != line_1_pixels.end() &&
index1%2==0
                                        && ity1 != line_1_pixels.end() &&
index1y==index1+1
                                        && itx2 != line_2_pixels.end() &&
index2%2==0
                                        && ity2 != line_2_pixels.end() &&
index2y==index2+1
                                        && itx3 != line_3_pixels.end() &&
index3%2==0
                                        && ity3 != line_3_pixels.end() &&
index3y==index3+1){
                                    ;//do nothing
                                }else{
                                    set_pixel(xi,yi,BF,GF,RF,A); // inside
triangle and not borders
                                }
                            }
                        }
                    }
                }
                    // Adding the triangle to shapes of bmp
                    shapes.add_triangle(triangle);
```

```
            }else{
                throw std::runtime_error("The triangle does not fit in
the image!");
            }
     }
```

Метод класса BMP draw_triangle принимает на вход 1 аргумент**:** экземпляр структуры Triangle. Сама функция отрисовывает треугольник и сохраняет его параметры. Также устанавливает наличие треугольника на картинке через метод класса Shapes add_triangle(Triangle triangle)

Метод класса BMP drawColoredLine принимает на вход 9 аргументов, описывающих координаты точек начала и конца линии, ее цвет и толщину. Используется для отрисовки линий, а в данном случае: границ треугольника.

2.2. Функция отрисовки прямоугольника.

```
     void BMP::draw_rectangle(Rectangle rect){
         uint32_t x0 = rect.x_start, y0 = rect.y_start, w = rect.width,
h = rect.height;
         uint8_t B = rect.color_B, G = rect.color_G, R = rect.color_R, A
= ALPHA, line_w = rect.thickness;
         bool filled = rect.filled;
         uint8_t BF = rect.filled_color_B, GF = rect.filled_color_G, RF
= rect.filled_color_R;
         if (x0 + w > (uint32_t)bmp_info_header.width || y0 + h >
(uint32_t)bmp_info_header.height) {
             throw std::runtime_error("The rectangle does not fit in the
image!");
         }

         fill_region(x0, y0, w, line_w, B, G, R, A);
// top line
         fill_region(x0, (y0 + h - line_w), w, line_w, B, G, R, A);
// bottom line
         fill_region((x0 + w - line_w), (y0 + line_w), line_w, (h - (2 *
line_w)), B, G, R, A);  // right line
```

```
        fill_region(x0, (y0 + line_w), line_w, (h - (2 * line_w)), B,
G, R, A);                  // left line
        if (filled==1)
            fill_region(x0+line_w, y0+line_w, w-2*line_w, h-2*line_w,
BF, GF, RF, A);

        // Adding the triangle to shapes of bmp
        (shapes).add_rectangle(rect);
    }
```

Метод класса BMP draw_rectangle принимает на вход 1 аргумент**:** экземпляр структуры Rectangle. Сама функция отрисовывает прямоугольник и сохраняет его параметры. Также устанавливает наличие прямоугольника на картинке через метод класса Shapes add_rectangle(Rectangle rectangle)

Метод класса BMP drawColoredLine принимает на вход 9 аргументов, описывающих координаты точек начала и конца линии, ее цвет и толщину. Используется для отрисовки линий, а в данном случае: границ прямоугольника.

2.3. Функция создания коллажа.

```
    unsigned int getLargestBMP(std::vector<BMP> images, int* max_width,
int* max_height){
        unsigned int max = 0;
        unsigned int index = 0, max_index;
        *max_width = *max_height = 0;
        for(auto image:images){
            unsigned int value = image.bmp_info_header.width *
image.bmp_info_header.height;
            if (value>max){
                max = value;
                max_index = index;
            }
            if (image.bmp_info_header.width>*max_width){
                *max_width = image.bmp_info_header.width;
            }
            if (image.bmp_info_header.height>*max_height){
                *max_height = image.bmp_info_header.height;
            }
```

```
            index++;
        }
        return index;
    }

    BMP create_collage(int M, int N, std::vector<BMP> bmps){
        int max_width, max_height;
        getLargestBMP(bmps, &max_width, &max_height);
        BMP collage = BMP(max_width*M, max_height*N, false);
        int next = 0;
        int x, y;
        for (int i=0; i < M; i++){
            for (int j=0; j < N; j++){
               x = i*max_width; // x = 0 , y = 0 ,
               y = j*max_height;
               // center the image
                x += (max_width-bmps.at(next).bmp_info_header.width)/2;
                y += (max_height-
bmps.at(next).bmp_info_header.height)/2;

                collage.add_image(bmps[next++], x, y, max_width,
max_height);
                next = next % bmps.size(); // next 0 , 1 , 2 , 3 , 0 ,
1 , 2 , 3
            }
        }
        return collage;
    }
```

Функция create_collage принимает на вход 3 аргумента: значение количества изображений по высоте и ширине а также массив изображений, из которых составляется коллаж. Создаёт сам коллаж, возвращает объект класса BMP.

Функция getLargestBmp принимает на вход 3 аргументов: массив картинок (объектов класса BMP), указатели на переменные, хранящие значения

масимальной высоты и ширины. Находит параметры максимального по размеру изображения.

2.4. Описание аргументов и ключей к ним.

-tri — задаёт параметр для вызова функции рисования треугольника.

-A (P0) — задаёт координаты одной из вершин треугольника.

-B (P1) - задаёт координаты одной из вершин треугольника.

-C (P2) - задаёт координаты одной из вершин треугольника.

-t (thick) — задаёт ширину линии для отрисовки треугольника.

-c(color) — задаёт цвет линий треугольника .

-f (filled) — задаёт параметр, залит треугольник или нет.

-l (fillColor) — параметр цвета заливки треугольника.

-i (input) – имя входного файла.

rectangle — задаёт параметр для вызова функции поиска прямоугольника максимальной площади.

-s(start) – координаты стартовой точки.

-w(width) – ширина прямоугольника.

-h(height) – высота прямоугольника.

-t (thick) — задаёт ширину линии для отрисовки прямоугольника.

-c(color) — задаёт цвет линий прямоугольника.

-f (filled) — задаёт параметр, залит прямоугольник или нет.

-l (fillColor) — параметр цвета заливки прямоугольника.

-i (input) – имя входного файла.

collage — задаёт параметр для вызова функции по созданию коллажа из фото.

-m — количество изображений по высоте.

-n — количество изображений по длине.

-o (output) — имя выходного файла.

-i (images) — список имен входных файлов для создания коллажа.

2.5. Дополнительные функции, описание структур

Описание структур для хранения информации о .BMP файле (поля структур отвечают стандартным полям файла .BMP, зарезервированные поля заполняются нулями), за хранение информации о изображении отвечают структуры BMPFileHeader, BMPInfoHeader. Структура BMPColorHeader является «палитрой», так как хранит значения основных цветов. В классе же BMP реализованы основные методы, а именно: открытие и закрытие файлов изображений, отрисовка треугольника и прямоугольника, отрисовка линии, заливка, покраска конкретного пикселя и получение информации о пикселе. Код структур:

```
struct BMPFileHeader {
    uint16_t file_type{ 0x4D42 }; //2 bytes         // File type
always BM which is 0x4D42 (stored as hex uint16_t in little endian)
    uint32_t file_size{ 0 };      //4 bytes         // Size of the
file (in bytes)
    uint16_t reserved1{ 0 };      //2 bytes         // Reserved,
always 0
    uint16_t reserved2{ 0 };      //2 bytes         // Reserved,
always 0
    uint32_t offset_data{ 0 };    //4 bytes         // Start
position of pixel data (bytes from the beginning of the file)
}; // 14 bytes

struct BMPInfoHeader {
    uint32_t size{ 0 }; //4 bytes                    // Size of
this header (in bytes)
    int32_t width{ 0 }; //4 bytes                    // width of
bitmap in pixels
    int32_t height{ 0 }; //4 bytes                    // width of
bitmap in pixels
                                                //       (if positive,
bottom-up, with origin in lower left corner)
```

```
                                                    //       (if negative,
top-down, with origin in upper left corner)
         uint16_t planes{ 1 };  //2 bytes                  // No. of
planes for the target device, this is always 1
         uint16_t bit_count{ 0 }; //2 bytes                 // No. of
bits per pixel
         uint32_t compression{ 0 };//4 bytes               // 0 or 3 -
uncompressed. THIS PROGRAM CONSIDERS ONLY UNCOMPRESSED BMP images
         uint32_t size_image{ 0 }; //4 bytes               // 0 - for
uncompressed images
         int32_t x_pixels_per_meter{ 0 }; // 4 bytes
         int32_t y_pixels_per_meter{ 0 }; // 4 bytes
         uint32_t colors_used{ 0 };      // 4 bytes         // No. color
indexes in the color table. Use 0 for the max number of colors allowed by
bit_count
         uint32_t colors_important{ 0 }; // 4 bytes          // No. of
colors used for displaying the bitmap. If 0 all colors are required
     }; // 40 bytes

     struct BMPColorHeader {
         uint32_t red_mask{ 0x00ff0000 };  //4 bytes         // Bit mask
for the red channel
         uint32_t green_mask{ 0x0000ff00 }; //4 bytes      // Bit mask
for the green channel
         uint32_t blue_mask{ 0x000000ff };  // 4 bytes      // Bit mask
for the blue channel
         uint32_t alpha_mask{ 0xff000000 }; // 4 bytes      // Bit mask
for the alpha channel
         uint32_t color_space_type{ 0x73524742 }; // 4 bytes // Default
"sRGB" (0x73524742)
         uint32_t unused[16]{ 0 };     // 4 bytes           // Unused
data for sRGB color space
     }; // 24 bytes
     #pragma pack(pop)

     struct BMP {
         BMPFileHeader file_header;
         BMPInfoHeader bmp_info_header;
```

```
        BMPColorHeader bmp_color_header;
        std::vector<uint8_t> data;
        struct Shapes shapes;

        BMP(const char *fname, char* shapes_fname);

        BMP(const char *fname);

        BMP(int32_t width, int32_t height, bool has_alpha = true);

        void read(const char *fname, char *shapes_fname);

        void read(const char *fname);

        void write(const char *fname);

        void write(const char *fname, char *shapes_fname);

        void fill_region(uint32_t x0, uint32_t y0, uint32_t w, uint32_t
h, uint8_t B, uint8_t G, uint8_t R, uint8_t A) ;

        void set_pixel(uint32_t x0, uint32_t y0, uint8_t B, uint8_t G,
uint8_t R, uint8_t A);

        void get_pixel(uint32_t x0, uint32_t y0, uint8_t* R, uint8_t*
G, uint8_t* B, uint8_t* A);

        void draw_rectangle(Rectangle rect);

        void draw_triangle(Triangle  triangle);

        std::vector<uint32_t> drawColoredLine(int x0, int y0, int x1,
int y1, float wd, uint8_t B, uint8_t G, uint8_t R, uint8_t A);

        void add_image(BMP current, int x, int y, int
image_panel_width, int image_panel_height);
```

```
    private:
        uint32_t row_stride{ 0 };

        void write_headers(std::ofstream &of);

        void write_headers_and_data(std::ofstream &of);


        // Add 1 to the row_stride until it is divisible with
align_stride
        uint32_t make_stride_aligned(uint32_t align_stride);

        // Check if the pixel data is stored as BGRA and if the color
space type is sRGB
        void check_color_header(BMPColorHeader &bmp_color_header);

        static bool inside_triangle(int x, int y, Triangle tri);

        static bool within_range(double x);

    };
```

Описание структуры Rectangle (хранит всю информацию о прямоугольнике, методы направлены на получение информации о фигуре):

```
    struct Rectangle{
    public:
        int x_start = 0, y_start = 0, x_end = 0, y_end =0;
        int thickness = 1;
        unsigned int color_R = 255;
        uint8_t color_G = 255;
        uint8_t color_B = 255;
        bool filled = 0;
        uint8_t filled_color_R = 255;
        uint8_t filled_color_G = 255;
        uint8_t filled_color_B = 255;
        unsigned int width = 0, height = 0;

        double get_area(){
```

```
            return width * height;
        }


        unsigned int get_max_X(){
            x_end = x_start + width;
            return max(max(x_start, (int)width),x_end);
        }

        unsigned int get_max_Y(){
            y_end = y_start + height;
            return max(max(y_start, (int)height),y_end);
        }

        static bool serialize(Rectangle rectangle, FILE* outfile){
            if (outfile == NULL){
                fprintf(stderr,"Serialization Error\n");
                exit(1);
            }
            int fwrite_output = fwrite(&rectangle.x_start, sizeof
(int), 1, outfile);
            fwrite_output *= fwrite(&rectangle.x_end, sizeof (int), 1,
outfile);
            fwrite_output *= fwrite(&rectangle.y_start, sizeof (int),
1, outfile);
            fwrite_output *= fwrite(&rectangle.y_end, sizeof (int), 1,
outfile);
            fwrite_output *= fwrite(&rectangle.width, sizeof (unsigned
int), 1, outfile);
            fwrite_output *= fwrite(&rectangle.height, sizeof (unsigned
int), 1, outfile);
            fwrite_output *= fwrite(&rectangle.filled, sizeof (bool),
1, outfile);
            fwrite_output *= fwrite(&rectangle.thickness, sizeof (int),
1, outfile);
            fwrite_output *= fwrite(&rectangle.color_B, sizeof
(uint8_t), 1, outfile);
```

```
            fwrite_output *= fwrite(&rectangle.color_R, sizeof
(uint8_t), 1, outfile);
            fwrite_output *= fwrite(&rectangle.color_G, sizeof
(uint8_t), 1, outfile);
            fwrite_output *= fwrite(&rectangle.filled_color_B, sizeof
(uint8_t), 1, outfile);
            fwrite_output *= fwrite(&rectangle.filled_color_G, sizeof
(uint8_t), 1, outfile);
            fwrite_output *= fwrite(&rectangle.filled_color_R, sizeof
(uint8_t), 1, outfile);

            if(fwrite_output==0)
                printf("Serialization Error\n");
            return fwrite_output;
        }

        static Rectangle deserialize(FILE* infile){
            if (infile == NULL){
                fprintf(stderr, "Deserialization Error\n");
                exit(1);
            }
            Rectangle rectangle;
            int fread_output = fread(&rectangle.x_start, sizeof (int),
1, infile);
            fread_output *= fread(&rectangle.x_end, sizeof (int), 1,
infile);
            fread_output *= fread(&rectangle.y_start, sizeof (int), 1,
infile);
            fread_output *= fread(&rectangle.y_end, sizeof (int), 1,
infile);
            fread_output *= fread(&rectangle.width, sizeof (unsigned
int), 1, infile);
            fread_output *= fread(&rectangle.height, sizeof (unsigned
int), 1, infile);
            fread_output *= fread(&rectangle.filled, sizeof (bool), 1,
infile);
            fread_output *= fread(&rectangle.thickness, sizeof (int),
1, infile);
```

```
            fread_output *= fread(&rectangle.color_B, sizeof (uint8_t),
1, infile);
            fread_output *= fread(&rectangle.color_R, sizeof (uint8_t),
1, infile);
            fread_output *= fread(&rectangle.color_G, sizeof (uint8_t),
1, infile);
            fread_output *= fread(&rectangle.filled_color_B, sizeof
(uint8_t), 1, infile);
            fread_output *= fread(&rectangle.filled_color_G, sizeof
(uint8_t), 1, infile);
            fread_output *= fread(&rectangle.filled_color_R, sizeof
(uint8_t), 1, infile);

            if(fread_output!=0){
                return rectangle;
            }else{
                printf("Deserialization Error\n");
                return Rectangle();
            }
        }

        // if (rect1==rect2)
        friend bool operator== (const Rectangle rect1, const Rectangle
rect2){
            return (rect1.x_start==rect2.x_start &&
                    rect1.x_end==rect2.x_end &&
                    rect1.y_start==rect2.y_start &&
                    rect1.y_end==rect2.y_end &&
                    rect1.width==rect2.width &&
                    rect1.height==rect2.height &&
                    rect1.filled==rect2.filled &&
                    rect1.color_B==rect2.color_B &&
                    rect1.color_G==rect2.color_G &&
                    rect1.color_R==rect2.color_R &&
                    rect1.filled_color_B==rect2.filled_color_B &&
                    rect1.filled_color_G==rect2.filled_color_G &&
                    rect1.filled_color_R==rect2.filled_color_R &&
                    rect1.thickness == rect2.thickness
```

```
                );
        }

        char* to_string(){
            char* arr = (char*)malloc(2000*sizeof (char));
            sprintf(arr,
"================================================\nRectangle Start (%d, %d),
End (%d, %d), Width = %d, Height = %d\nBorder Color = {%d, %d, %d}, Fill
Color = {%d, %d, %d}\nThickness = %d , Area
= %f\n================================================\n"
                    ,this->x_start, this->y_start, this->x_end,
this->y_end, width, height, color_R, color_G, color_B,
                     filled_color_R, filled_color_G, filled_color_B,
thickness, get_area());
            return arr;
        }
    };
```

Описание структуры Triangle (хранит всю информацию о треугольнике, методы направлены на получение информации о фигуре):

```
    struct Triangle{
    public:
        int X2 = 0, Y2 = 0, X1 = 0, Y1 =0, X3 = 0, Y3 =0;
        int thickness = 1;
        uint8_t color_R = 0;
        uint8_t color_G = 0;
        uint8_t color_B = 0;
        bool filled = 0;
        uint8_t filled_color_R = 0;
        uint8_t filled_color_G = 0;
        uint8_t filled_color_B = 0;
        unsigned int image_width = 0, image_height = 0;


        double get_area(){
            double A = sqrt((double)(X2-X1)*(X2-X1)+(Y2-Y1)*(Y2-Y1));
            double B = sqrt((double)(X2-X3)*(X2-X3)+(Y2-Y3)*(Y2-Y3));
            double C = sqrt((double)(X1-X3)*(X1-X3)+(Y1-Y3)*(Y1-Y3));
```

```
            double s = (A+B+C) / 2;
            return sqrt(s*(s-A)*(s-B)*(s-C));
        }


        unsigned int get_max_X(){
            return max(max(X1,X2),X3);
        }

        unsigned int get_max_Y(){
            return max(max(Y1, Y2),Y3);
        }

        static bool serialize(Triangle triangle, FILE* outfile){
            if (outfile == NULL){
                fprintf(stderr,"Serialization Error\n");
                exit(1);
            }
            int fwrite_output = fwrite(&triangle.X1, sizeof (int), 1,
outfile);
            fwrite_output *= fwrite(&triangle.X2, sizeof (int), 1,
outfile);
            fwrite_output *= fwrite(&triangle.X3, sizeof (int), 1,
outfile);
            fwrite_output *= fwrite(&triangle.Y1, sizeof (int), 1,
outfile);
            fwrite_output *= fwrite(&triangle.Y2, sizeof (int), 1,
outfile);
            fwrite_output *= fwrite(&triangle.Y3, sizeof (int), 1,
outfile);
            fwrite_output *= fwrite(&triangle.filled, sizeof (bool), 1,
outfile);
            fwrite_output *= fwrite(&triangle.thickness, sizeof (int),
1, outfile);
            fwrite_output *= fwrite(&triangle.color_B, sizeof
(uint8_t), 1, outfile);
            fwrite_output *= fwrite(&triangle.color_R, sizeof
(uint8_t), 1, outfile);
```

```
            fwrite_output *= fwrite(&triangle.color_G, sizeof
(uint8_t), 1, outfile);
            fwrite_output *= fwrite(&triangle.filled_color_B, sizeof
(uint8_t), 1, outfile);
            fwrite_output *= fwrite(&triangle.filled_color_G, sizeof
(uint8_t), 1, outfile);
            fwrite_output *= fwrite(&triangle.filled_color_R, sizeof
(uint8_t), 1, outfile);

            if(fwrite_output==0)
                printf("Serialization Error\n");

            return fwrite_output;
        }

        static Triangle deserialize(FILE* infile){
            if (infile == NULL){
                fprintf(stderr, "Deserialization Error\n");
                exit(1);
            }
            Triangle triangle;
            int fread_output = fread(&triangle.X1, sizeof (int), 1,
infile);
            fread_output *= fread(&triangle.X2, sizeof (int), 1,
infile);
            fread_output *= fread(&triangle.X3, sizeof (int), 1,
infile);
            fread_output *= fread(&triangle.Y1, sizeof (int), 1,
infile);
            fread_output *= fread(&triangle.Y2, sizeof (int), 1,
infile);
            fread_output *= fread(&triangle.Y3, sizeof (int), 1,
infile);
            fread_output *= fread(&triangle.filled, sizeof (bool), 1,
infile);
            fread_output *= fread(&triangle.thickness, sizeof (int), 1,
infile);
```

```
            fread_output *= fread(&triangle.color_B, sizeof (uint8_t),
1, infile);
            fread_output *= fread(&triangle.color_R, sizeof (uint8_t),
1, infile);
            fread_output *= fread(&triangle.color_G, sizeof (uint8_t),
1, infile);
            fread_output *= fread(&triangle.filled_color_B, sizeof
(uint8_t), 1, infile);
            fread_output *= fread(&triangle.filled_color_G, sizeof
(uint8_t), 1, infile);
            fread_output *= fread(&triangle.filled_color_R, sizeof
(uint8_t), 1, infile);

            if(fread_output!=0){
                return triangle;
            }else{
                printf("Deserialization Error\n");
                return Triangle();
            }
        }

        friend bool operator== (const Triangle tri1, const Triangle
tri2){
            if (    tri1.X1==tri2.X1 &&
                    tri1.X2==tri2.X2 &&
                    tri1.X3==tri2.X3 &&
                    tri1.Y1==tri2.Y1 &&
                    tri1.Y2==tri2.Y2 &&
                    tri1.Y3==tri2.Y3 &&
                    tri1.filled==tri2.filled &&
                    tri1.color_B==tri2.color_B &&
                    tri1.color_G==tri2.color_G &&
                    tri1.color_R==tri2.color_R &&
                    tri1.filled_color_B==tri2.filled_color_B &&
                    tri1.filled_color_G==tri2.filled_color_G &&
                    tri1.filled_color_R==tri2.filled_color_R &&
                    tri1.thickness == tri2.thickness
                    ){
```

```
                return true;
            }
            return false;
        }


        char* to_string(){
            std::string str1;
            char* arr = (char*)malloc(2000*sizeof(char));
            sprintf(arr,
"=============================================\nTriangle A (%d, %d), B
(%d, %d), C = (%d, %d)\nBorder Color = {%d, %d, %d}, Fill Color =
{%d, %d, %d}\nThickness = %d , Area
= %f\n=====================================================\n"
                    ,this->X1, this->Y1, this->X2, this->Y2,this->X3,
this->Y3, color_R, color_G, color_B,
                     filled_color_R, filled_color_G, filled_color_B,
thickness, get_area());
            return arr;
        }

    };
```

Описание структуры Shapes (содержит массивы треугольников и прямоугольников, методы для добавления новых фигур а также методы, которые позволяют получить треугольник или прямоугольник максимального размера):

```
    struct Shapes{
        vector<Rectangle> rectangles;
        vector<Triangle> triangles;
        char* filename;// ""
        Shapes(char* filename){
            this->filename = filename;
        }

        Shapes(){
            char* nothing = (char*)malloc(sizeof (char));
            strcat(nothing, "");
```

```
    this->filename = nothing;
}

static char* get_shapes_filename(char* filename){
    char* str = (char*)malloc(sizeof (char));
    return strcat(strcpy(str,filename), ".data");
    // filename = "file.bmp";
    // strcat(filename,".data"); // file.bmp.data
}

void add_rectangle(Rectangle rect){
    if (rectangles.size()==0){
        rectangles.push_back(rect);
    }else{
        for (Rectangle rex :rectangles)
            if(rex==rect)
                return;
        rectangles.push_back(rect);
    }
}

void add_triangle(Triangle tri){
    if (triangles.size()==0){
        triangles.push_back(tri);
    }else{
        for (Triangle trx : triangles)
            if(trx==tri)
                return;
        triangles.push_back(tri);
    }
}

Rectangle get_largest_rectangle(int* index){
    Rectangle largest_rect;
    int counter=0;
    for (auto r:rectangles){
        if (r.get_area()>largest_rect.get_area()){
            largest_rect = r;
```

```
                (*index) = counter;
            }
            (counter)++;
        }
        return largest_rect;
    }

    Triangle get_largest_triangle(int* index){
        Triangle largest_triangle;
        int counter=0;
        for (auto t:triangles){
            if (t.get_area()>largest_triangle.get_area()){
                largest_triangle = t;
                (*index) = counter;
            }
            (counter)++;
        }
        return largest_triangle;
    }


    static bool serialize(Shapes shapes, FILE* outfile){
        if (outfile == NULL){
            fprintf(stderr,"Serialization Error\n");
            exit(1);
        }
        int tris_size = shapes.triangles.size();
        int rect_size = shapes.rectangles.size();
        int fwrite_output = fwrite(&tris_size, sizeof (int), 1,
outfile);
        for (Triangle tri : shapes.triangles){
            fwrite_output *= Triangle::serialize(tri, outfile);
        }
        fwrite_output *= fwrite(&rect_size, sizeof (int), 1,
outfile);
        for (Rectangle rect : shapes.rectangles){
            fwrite_output *= Rectangle::serialize(rect, outfile);
        }
```

```
            if(fwrite_output==0)
                printf("Serialization Error\n");

            return fwrite_output;
        }

        static Shapes deserialize(FILE* infile){
            if (infile == NULL){
                fprintf(stderr, "Deserialization Error\n");
                exit(1);
            }
            Shapes shapes;
            int tris_size;
            int fread_output = fread(&tris_size, sizeof (int), 1,
infile);
            for (int i = 0; i< tris_size; i++){
                Triangle t = Triangle::deserialize(infile);
                if (t.get_area() !=0){
                    shapes.add_triangle(t);
                    fread_output*=1;
                }else{
                    fread_output*=0;
                }
            }
            int rect_size;
            fread_output *= fread(&rect_size, sizeof (int), 1, infile);
            for (int i = 0; i< rect_size; i++){
                Rectangle r = Rectangle::deserialize(infile);
                if (r.get_area()!=0){
                    shapes.add_rectangle(r);
                    fread_output*=1;
                }else{
                    fread_output*=0;
                }
            }

            if(fread_output!=0){
```

```
                return shapes;
            }else{
                printf("Deserialization Error\n");
                return Shapes();
            }
        }

        int write2file(char* filename){
            FILE* outfile = fopen(filename, "w");
            if (outfile == NULL){
                fprintf(stderr, "\nError open file\n");
                exit(1);
            }
            struct Shapes shapes = *this;
            printf("Storing shapes to data file....\n");
            int fwrite_output = Shapes::serialize(shapes, outfile);
            if(fwrite_output!=0){
                printf("Data to file %s successfully stored!\n",
filename);
            }else{
                printf("error writing to file !\n");
            }

            fclose(outfile);
            return fwrite_output;
        }

        static Shapes read4file(char* filename){
            FILE* infile = fopen(filename, "r");
            if (infile == NULL){
                fprintf(stderr, "\nError: can't open
file %s\n",filename);
                exit(1);
            }
            struct Shapes shapes(filename);
            shapes = Shapes::deserialize(infile);
            int fread_output = 0;
            if (strcmp(shapes.filename,"")==0){
```

```
                fread_output = 1;
            }else{
                printf("Deserialization Error\n");
            }

            if(fread_output!=0){
                printf("Data from file %s successfully read!\n",
filename);
                fclose(infile);
                return shapes;
            }else{
                printf("error reading file !\n");
                // assign "" to filename of shapes
                return Shapes();
            }
        }

        void to_string(){
            printf("We have %d shapes: %d Rectangles and %d
Triangles\n", int(this->triangles.size()+this->rectangles.size()),
                   int(rectangles.size()), int(triangles.size()));
            for(auto tri: triangles){
                printf("%s",tri.to_string());
            }

            for(auto rect: rectangles){
                printf("%s",rect.to_string());
            }
        }

    };
```

ЗАКЛЮЧЕНИЕ

Для успешного достижения поставленной цели — написания программы для работы с файлами формата .BMP, соответствующей заданию курсовой работы, были выполнены соответствующие задачи:

1. Изучен теоретический материал по теме курсовой работы.
2. Разработан программный код.
3. Реализован программный код.
4. Проведено тестирование программы.

Исходный код программы представлен в приложении А, результаты тестирования - в приложении Б.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Язык программирования СИ / Керниган Б., Ритчи Д. СПб.: Издательство "Невский Диалект", 2001. 352 с.

2. Основы программирования на языках Си и C++ [Электронный ресурс URL: http://cplusplus.com

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

BMP.H:

```
#IFNDEF BMP_H
#DEFINE BMP_H
#PRAGMA ONCE

#IFNDEF STRUCTURES_H
#INCLUDE <STRUCTURES.H>
#ENDIF

USING NAMESPACE STD;

#PRAGMA PACK(PUSH, 1)
STRUCT BMPFILEHEADER {
    UINT16_T FILE_TYPE{ 0X4D42 }; //2 BYTES         // FILE TYPE ALWAYS
BM WHICH IS 0X4D42 (STORED AS HEX UINT16_T IN LITTLE ENDIAN)
    UINT32_T FILE_SIZE{ 0 };      //4 BYTES         // SIZE OF THE FILE
(IN BYTES)
    UINT16_T RESERVED1{ 0 };      //2 BYTES         // RESERVED, ALWAYS 0
    UINT16_T RESERVED2{ 0 };      //2 BYTES         // RESERVED, ALWAYS 0
    UINT32_T OFFSET_DATA{ 0 };    //4 BYTES         // START POSITION OF
PIXEL DATA (BYTES FROM THE BEGINNING OF THE FILE)
}; // 14 BYTES

STRUCT BMPINFOHEADER {
    UINT32_T SIZE{ 0 }; //4 BYTES                    // SIZE OF THIS
HEADER (IN BYTES)
    INT32_T WIDTH{ 0 }; //4 BYTES                    // WIDTH OF BITMAP
IN PIXELS
    INT32_T HEIGHT{ 0 }; //4 BYTES                    // WIDTH OF BITMAP
IN PIXELS
                                                //       (IF POSITIVE,
BOTTOM-UP, WITH ORIGIN IN LOWER LEFT CORNER)
                                                //       (IF NEGATIVE, TOP-
DOWN, WITH ORIGIN IN UPPER LEFT CORNER)
```

```
    UINT16_T PLANES{ 1 };  //2 BYTES                    // NO. OF PLANES
FOR THE TARGET DEVICE, THIS IS ALWAYS 1
    UINT16_T BIT_COUNT{ 0 }; //2 BYTES                  // NO. OF BITS PER
PIXEL
    UINT32_T COMPRESSION{ 0 };//4 BYTES                // 0 OR 3 -
UNCOMPRESSED. THIS PROGRAM CONSIDERS ONLY UNCOMPRESSED BMP IMAGES
    UINT32_T SIZE_IMAGE{ 0 }; //4 BYTES                // 0 - FOR
UNCOMPRESSED IMAGES
    INT32_T X_PIXELS_PER_METER{ 0 }; // 4 BYTES
    INT32_T Y_PIXELS_PER_METER{ 0 }; // 4 BYTES
    UINT32_T COLORS_USED{ 0 };      // 4 BYTES          // NO. COLOR
INDEXES IN THE COLOR TABLE. USE 0 FOR THE MAX NUMBER OF COLORS ALLOWED BY
BIT_COUNT
    UINT32_T COLORS_IMPORTANT{ 0 }; // 4 BYTES          // NO. OF COLORS
USED FOR DISPLAYING THE BITMAP. IF 0 ALL COLORS ARE REQUIRED
}; // 40 BYTES

STRUCT BMPCOLORHEADER {
    UINT32_T RED_MASK{ 0X00FF0000 };  //4 BYTES         // BIT MASK FOR
THE RED CHANNEL
    UINT32_T GREEN_MASK{ 0X0000FF00 }; //4 BYTES      // BIT MASK FOR THE
GREEN CHANNEL
    UINT32_T BLUE_MASK{ 0X000000FF };  // 4 BYTES      // BIT MASK FOR
THE BLUE CHANNEL
    UINT32_T ALPHA_MASK{ 0XFF000000 }; // 4 BYTES      // BIT MASK FOR
THE ALPHA CHANNEL
    UINT32_T COLOR_SPACE_TYPE{ 0X73524742 }; // 4 BYTES // DEFAULT "SRGB"
(0X73524742)
    UINT32_T UNUSED[16]{ 0 };     // 4 BYTES           // UNUSED DATA FOR
SRGB COLOR SPACE
}; // 24 BYTES
#PRAGMA PACK(POP)

STRUCT BMP {
    BMPFILEHEADER FILE_HEADER;
    BMPINFOHEADER BMP_INFO_HEADER;
    BMPCOLORHEADER BMP_COLOR_HEADER;
    STD::VECTOR<UINT8_T> DATA;
```

```
    STRUCT SHAPES SHAPES;

    BMP(CONST CHAR *FNAME, CHAR* SHAPES_FNAME);

    BMP(CONST CHAR *FNAME);

    BMP(INT32_T WIDTH, INT32_T HEIGHT, BOOL HAS_ALPHA = TRUE);

    VOID READ(CONST CHAR *FNAME, CHAR *SHAPES_FNAME);

    VOID READ(CONST CHAR *FNAME);

    VOID WRITE(CONST CHAR *FNAME);

    VOID WRITE(CONST CHAR *FNAME, CHAR *SHAPES_FNAME);

    VOID FILL_REGION(UINT32_T X0, UINT32_T Y0, UINT32_T W, UINT32_T H,
UINT8_T B, UINT8_T G, UINT8_T R, UINT8_T A) ;

    VOID SET_PIXEL(UINT32_T X0, UINT32_T Y0, UINT8_T B, UINT8_T G,
UINT8_T R, UINT8_T A);

    VOID GET_PIXEL(UINT32_T X0, UINT32_T Y0, UINT8_T* R, UINT8_T* G,
UINT8_T* B, UINT8_T* A);

    VOID DRAW_RECTANGLE(RECTANGLE RECT);

    VOID DRAW_TRIANGLE(TRIANGLE  TRIANGLE);

    STD::VECTOR<UINT32_T> DRAWCOLOREDLINE(INT X0, INT Y0, INT X1, INT Y1,
FLOAT WD, UINT8_T B, UINT8_T G, UINT8_T R, UINT8_T A);

    VOID ADD_IMAGE(BMP CURRENT, INT X, INT Y, INT IMAGE_PANEL_WIDTH, INT
IMAGE_PANEL_HEIGHT);


PRIVATE:
    UINT32_T ROW_STRIDE{ 0 };
```

```
    VOID WRITE_HEADERS(STD::OFSTREAM &OF);

    VOID WRITE_HEADERS_AND_DATA(STD::OFSTREAM &OF);


    // ADD 1 TO THE ROW_STRIDE UNTIL IT IS DIVISIBLE WITH ALIGN_STRIDE
    UINT32_T MAKE_STRIDE_ALIGNED(UINT32_T ALIGN_STRIDE);

    // CHECK IF THE PIXEL DATA IS STORED AS BGRA AND IF THE COLOR SPACE
TYPE IS SRGB
    VOID CHECK_COLOR_HEADER(BMPCOLORHEADER &BMP_COLOR_HEADER);

    STATIC BOOL INSIDE_TRIANGLE(INT X, INT Y, TRIANGLE TRI);

    STATIC BOOL WITHIN_RANGE(DOUBLE X);

};


#ENDIF

BMP.CPP:
#INCLUDE<BMP.H>

BOOL BMP::WITHIN_RANGE(DOUBLE X){
    RETURN 0 <= X && X <= 1;
}

BOOL BMP::INSIDE_TRIANGLE(INT X, INT Y, TRIANGLE TRI){
    INT X0 = TRI.X1, Y0 = TRI.Y1, X1 = TRI.X2, Y1 = TRI.Y2,
            X2 = TRI.X3, Y2 = TRI.Y3;
    DOUBLE DET = (Y1-Y2)*(X0-X2) + (X2-X1)*(Y0-Y2);
    DOUBLE FACTOR_ALPHA = (Y1-Y2)*(X-X2)+(X2-X1)*(Y-Y2);
    DOUBLE FACTOR_BETA = (Y2-Y0)*(X-X2)+(X0-X2)*(Y-Y2);
    DOUBLE ALPHA = FACTOR_ALPHA/DET;
    DOUBLE BETA = FACTOR_BETA/DET;
    DOUBLE GAMMA = 1.0 - ALPHA - BETA;
```

```
    RETURN (X==X0 && Y==Y0) || (X==X1 && Y==Y1) || (X==X1 && Y==Y1) ||
            (WITHIN_RANGE(ALPHA) && WITHIN_RANGE(BETA) &&
WITHIN_RANGE(GAMMA));
}


BMP::BMP(CONST CHAR *FNAME) {
    READ(FNAME);
}

BMP::BMP(CONST CHAR *FNAME, CHAR* SHAPES_FNAME) {
    READ(FNAME, SHAPES_FNAME);
}


BMP::BMP(INT32_T WIDTH, INT32_T HEIGHT, BOOL HAS_ALPHA) {
    IF (WIDTH <= 0 || HEIGHT <= 0) {
        THROW STD::RUNTIME_ERROR("THE IMAGE WIDTH AND HEIGHT MUST BE
POSITIVE NUMBERS.");
    }

    BMP_INFO_HEADER.WIDTH = WIDTH;
    BMP_INFO_HEADER.HEIGHT = HEIGHT;
    IF (HAS_ALPHA) {
        BMP_INFO_HEADER.SIZE = SIZEOF(BMPINFOHEADER) +
SIZEOF(BMPCOLORHEADER);
        FILE_HEADER.OFFSET_DATA = SIZEOF(BMPFILEHEADER) +
SIZEOF(BMPINFOHEADER) + SIZEOF(BMPCOLORHEADER);

        BMP_INFO_HEADER.BIT_COUNT = 32;
        BMP_INFO_HEADER.COMPRESSION = 3;
        ROW_STRIDE = WIDTH * 4;
        DATA.RESIZE(ROW_STRIDE * HEIGHT);
        FILE_HEADER.FILE_SIZE = FILE_HEADER.OFFSET_DATA + DATA.SIZE();
    }
    ELSE {
        BMP_INFO_HEADER.SIZE = SIZEOF(BMPINFOHEADER);
```

```
        FILE_HEADER.OFFSET_DATA = SIZEOF(BMPFILEHEADER) +
SIZEOF(BMPINFOHEADER);

        BMP_INFO_HEADER.BIT_COUNT = 24;
        BMP_INFO_HEADER.COMPRESSION = 0;
        ROW_STRIDE = WIDTH * 3;
        DATA.RESIZE(ROW_STRIDE * HEIGHT);

        UINT32_T NEW_STRIDE = MAKE_STRIDE_ALIGNED(4);
        FILE_HEADER.FILE_SIZE = FILE_HEADER.OFFSET_DATA +
STATIC_CAST<UINT32_T>(DATA.SIZE()) + BMP_INFO_HEADER.HEIGHT * (NEW_STRIDE
- ROW_STRIDE);
    }
}


VOID BMP::READ(CONST CHAR *FNAME) {
    // INOUT FILE STREAM
    STD::IFSTREAM INP( FNAME, STD::IFSTREAM::BINARY ); //OPEN BINARY FILE
    IF (INP) {
        INP.READ((CHAR*)&FILE_HEADER, SIZEOF(FILE_HEADER)); // READ THE
FILE AND EXTRACTS SIZEOF(FILE_HEADER) FROM FILE_HEADER
        IF(FILE_HEADER.FILE_TYPE != 0X4D42) { // IT'S BMP OR NOT
            THROW STD::RUNTIME_ERROR("ERROR! UNRECOGNIZED FILE FORMAT.");
        }
        INP.READ((CHAR*)&BMP_INFO_HEADER, SIZEOF(BMP_INFO_HEADER));

        // THE BMPCOLORHEADER IS USED ONLY FOR TRANSPARENT IMAGES
        IF(BMP_INFO_HEADER.BIT_COUNT == 32) {
            // CHECK IF THE FILE HAS BIT MASK COLOR INFORMATION
            IF(BMP_INFO_HEADER.SIZE >= (SIZEOF(BMPINFOHEADER) +
SIZEOF(BMPCOLORHEADER))) {
                INP.READ((CHAR*)&BMP_COLOR_HEADER,
SIZEOF(BMP_COLOR_HEADER));
                // CHECK IF THE PIXEL DATA IS STORED AS BGRA AND IF THE
COLOR SPACE TYPE IS SRGB
                CHECK_COLOR_HEADER(BMP_COLOR_HEADER);
            } ELSE {
```

```
                // STANDARD OUTPUT STREAM FOR ERRORS
                STD::CERR << "ERROR! THE FILE \"" << FNAME << "\" DOES
NOT SEEM TO CONTAIN BIT MASK INFORMATION\N";
                THROW STD::RUNTIME_ERROR("ERROR! UNRECOGNIZED FILE
FORMAT.");
            }
        }

        // JUMP TO THE PIXEL DATA LOCATION
        INP.SEEKG(FILE_HEADER.OFFSET_DATA, INP.BEG);// INP.BEG BEGINNING
OF STREAM

        // ADJUST THE HEADER FIELDS FOR OUTPUT.
        // SOME EDITORS WILL PUT EXTRA INFO IN THE IMAGE FILE, WE ONLY
SAVE THE HEADERS AND THE DATA.
        IF(BMP_INFO_HEADER.BIT_COUNT == 32) {
            BMP_INFO_HEADER.SIZE = SIZEOF(BMPINFOHEADER) +
SIZEOF(BMPCOLORHEADER);
            FILE_HEADER.OFFSET_DATA = SIZEOF(BMPFILEHEADER) +
SIZEOF(BMPINFOHEADER) + SIZEOF(BMPCOLORHEADER);
        } ELSE {
            BMP_INFO_HEADER.SIZE = SIZEOF(BMPINFOHEADER);
            FILE_HEADER.OFFSET_DATA = SIZEOF(BMPFILEHEADER) +
SIZEOF(BMPINFOHEADER);
        }
       FILE_HEADER.FILE_SIZE = FILE_HEADER.OFFSET_DATA;

        BOOL SCANNING_BOTTOM_UP = TRUE;
        // PIXLES START FROM BOTTOM LEFT CORNER
        IF (BMP_INFO_HEADER.HEIGHT < 0) {
            BMP_INFO_HEADER.HEIGHT = - BMP_INFO_HEADER.HEIGHT;
            SCANNING_BOTTOM_UP = FALSE;
//            THROW STD::RUNTIME_ERROR("THE PROGRAM CAN TREAT ONLY BMP
IMAGES WITH THE ORIGIN IN THE BOTTOM LEFT CORNER!");
        }

        DATA.RESIZE(BMP_INFO_HEADER.WIDTH * BMP_INFO_HEADER.HEIGHT *
BMP_INFO_HEADER.BIT_COUNT / 8);
```

```
        // HERE WE CHECK IF WE NEED TO TAKE INTO ACCOUNT ROW PADDING
        IF (BMP_INFO_HEADER.WIDTH % 4 == 0) {

            INP.READ((CHAR*)DATA.DATA(), DATA.SIZE());
            IF (!SCANNING_BOTTOM_UP){
                STD::REVERSE(DATA.BEGIN(), DATA.END());
                UINT8_T R, G, B, A;
                FOR (INT I=0; I<BMP_INFO_HEADER.WIDTH; I++){
                    FOR (INT J=0; J<BMP_INFO_HEADER.HEIGHT; J++){
                        GET_PIXEL(I,J,&R,&G,&B,&A);
                        THIS->SET_PIXEL(I, J, R, G, B, A);
                     }
                }
            }
            FILE_HEADER.FILE_SIZE += STATIC_CAST<UINT32_T>(DATA.SIZE());
        }
        ELSE {
            ROW_STRIDE = BMP_INFO_HEADER.WIDTH *
BMP_INFO_HEADER.BIT_COUNT / 8;
            UINT32_T NEW_STRIDE = MAKE_STRIDE_ALIGNED(4);
            STD::VECTOR<UINT8_T> PADDING_ROW(NEW_STRIDE - ROW_STRIDE);

            FOR (INT Y = 0; Y < BMP_INFO_HEADER.HEIGHT; ++Y) {
                INP.READ((CHAR*)(DATA.DATA() + ROW_STRIDE * Y),
ROW_STRIDE);
                INP.READ((CHAR*)PADDING_ROW.DATA(), PADDING_ROW.SIZE());
            }
            IF (!SCANNING_BOTTOM_UP){
                STD::REVERSE(DATA.BEGIN(), DATA.END());
                UINT8_T R, G, B, A;
                FOR (INT I=0; I<BMP_INFO_HEADER.WIDTH; I++){
                    FOR (INT J=0; J<BMP_INFO_HEADER.HEIGHT; J++){
                        GET_PIXEL(I,J,&R,&G,&B,&A);
                        THIS->SET_PIXEL(I, J, R, G, B, A);
                    }
                }
            }
```

```
            FILE_HEADER.FILE_SIZE += STATIC_CAST<UINT32_T>(DATA.SIZE()) +
BMP_INFO_HEADER.HEIGHT * STATIC_CAST<UINT32_T>(PADDING_ROW.SIZE());
        }
    }
    ELSE {
        THROW STD::RUNTIME_ERROR("UNABLE TO OPEN THE INPUT IMAGE FILE.");
    }
}

VOID BMP::READ(CONST CHAR *FNAME, CHAR *SHAPES_FNAME) {
    // INOUT FILE STREAM
    STD::IFSTREAM INP{ FNAME, STD::IOS_BASE::BINARY }; //OPEN BINARY FILE
    IF (INP) {
        INP.READ((CHAR*)&FILE_HEADER, SIZEOF(FILE_HEADER)); // READ THE
FILE AND EXTRACTS SIZEOF(FILE_HEADER) FROM FILE_HEADER
        IF(FILE_HEADER.FILE_TYPE != 0X4D42) { // IT'S BMP OR NOT
            THROW STD::RUNTIME_ERROR("ERROR! UNRECOGNIZED FILE FORMAT.");
        }
        INP.READ((CHAR*)&BMP_INFO_HEADER, SIZEOF(BMP_INFO_HEADER));

        // THE BMPCOLORHEADER IS USED ONLY FOR TRANSPARENT IMAGES
        IF(BMP_INFO_HEADER.BIT_COUNT == 32) {
            // CHECK IF THE FILE HAS BIT MASK COLOR INFORMATION
            IF(BMP_INFO_HEADER.SIZE >= (SIZEOF(BMPINFOHEADER) +
SIZEOF(BMPCOLORHEADER))) {
                INP.READ((CHAR*)&BMP_COLOR_HEADER,
SIZEOF(BMP_COLOR_HEADER));
                // CHECK IF THE PIXEL DATA IS STORED AS BGRA AND IF THE
COLOR SPACE TYPE IS SRGB
                CHECK_COLOR_HEADER(BMP_COLOR_HEADER);
            } ELSE {
                // STANDARD OUTPUT STREAM FOR ERRORS
                STD::CERR << "ERROR! THE FILE \"" << FNAME << "\" DOES
NOT SEEM TO CONTAIN BIT MASK INFORMATION\N";
                THROW STD::RUNTIME_ERROR("ERROR! UNRECOGNIZED FILE
FORMAT.");
            }
        }
```

```
        // JUMP TO THE PIXEL DATA LOCATION
        INP.SEEKG(FILE_HEADER.OFFSET_DATA, INP.BEG);// INP.BEG BEGINNING
OF STREAM

        // ADJUST THE HEADER FIELDS FOR OUTPUT.
        // SOME EDITORS WILL PUT EXTRA INFO IN THE IMAGE FILE, WE ONLY
SAVE THE HEADERS AND THE DATA.
        IF(BMP_INFO_HEADER.BIT_COUNT == 32) {
            BMP_INFO_HEADER.SIZE = SIZEOF(BMPINFOHEADER) +
SIZEOF(BMPCOLORHEADER);
            FILE_HEADER.OFFSET_DATA = SIZEOF(BMPFILEHEADER) +
SIZEOF(BMPINFOHEADER) + SIZEOF(BMPCOLORHEADER);
        } ELSE {
            BMP_INFO_HEADER.SIZE = SIZEOF(BMPINFOHEADER);
            FILE_HEADER.OFFSET_DATA = SIZEOF(BMPFILEHEADER) +
SIZEOF(BMPINFOHEADER);
        }
        FILE_HEADER.FILE_SIZE = FILE_HEADER.OFFSET_DATA;

        BOOL SCANNING_BOTTOM_UP = TRUE;
        // PIXLES START FROM BOTTOM LEFT CORNER
        IF (BMP_INFO_HEADER.HEIGHT < 0) {
            BMP_INFO_HEADER.HEIGHT = - BMP_INFO_HEADER.HEIGHT;
            SCANNING_BOTTOM_UP = FALSE;

//            THROW STD::RUNTIME_ERROR("THE PROGRAM CAN TREAT ONLY BMP
IMAGES WITH THE ORIGIN IN THE BOTTOM LEFT CORNER!");
        }

        DATA.RESIZE(BMP_INFO_HEADER.WIDTH * BMP_INFO_HEADER.HEIGHT *
BMP_INFO_HEADER.BIT_COUNT / 8);

        // HERE WE CHECK IF WE NEED TO TAKE INTO ACCOUNT ROW PADDING
        IF (BMP_INFO_HEADER.WIDTH % 4 == 0) {

            INP.READ((CHAR*)DATA.DATA(), DATA.SIZE());
            IF (!SCANNING_BOTTOM_UP){
```

```
                STD::REVERSE(DATA.BEGIN(), DATA.END());
                UINT8_T R, G, B, A;
                FOR (INT I=0; I<BMP_INFO_HEADER.WIDTH; I++){
                    FOR (INT J=0; J<BMP_INFO_HEADER.HEIGHT; J++){
                        GET_PIXEL(I,J,&R,&G,&B,&A);
                        THIS->SET_PIXEL(I, J, R, G, B, A);
                    }
                }
            }
            FILE_HEADER.FILE_SIZE += STATIC_CAST<UINT32_T>(DATA.SIZE());
        }
        ELSE {
            ROW_STRIDE = BMP_INFO_HEADER.WIDTH *
BMP_INFO_HEADER.BIT_COUNT / 8;
            UINT32_T NEW_STRIDE = MAKE_STRIDE_ALIGNED(4);
            STD::VECTOR<UINT8_T> PADDING_ROW(NEW_STRIDE - ROW_STRIDE);

            FOR (INT Y = 0; Y < BMP_INFO_HEADER.HEIGHT; ++Y) {
                INP.READ((CHAR*)(DATA.DATA() + ROW_STRIDE * Y),
ROW_STRIDE);
                INP.READ((CHAR*)PADDING_ROW.DATA(), PADDING_ROW.SIZE());
            }
            IF (!SCANNING_BOTTOM_UP){
                STD::REVERSE(DATA.BEGIN(), DATA.END());
                UINT8_T R, G, B, A;
                FOR (INT I=0; I<BMP_INFO_HEADER.WIDTH; I++){
                    FOR (INT J=0; J<BMP_INFO_HEADER.HEIGHT; J++){
                        GET_PIXEL(I,J,&R,&G,&B,&A);
                        THIS->SET_PIXEL(I, J, R, G, B, A);
                    }
                }
            }
            FILE_HEADER.FILE_SIZE += STATIC_CAST<UINT32_T>(DATA.SIZE()) +
BMP_INFO_HEADER.HEIGHT * STATIC_CAST<UINT32_T>(PADDING_ROW.SIZE());
        }
        STRUCT SHAPES SHAPES = SHAPES::READ4FILE(SHAPES_FNAME);
        IF (STRCMP(SHAPES.FILENAME, "")){
            PRINTF("ERROR READING SHAPES FILE\N");
```

```
            EXIT(1);
        }ELSE{
            THIS->SHAPES = SHAPES;
        }
    }
    ELSE {
        THROW STD::RUNTIME_ERROR("UNABLE TO OPEN THE INPUT IMAGE FILE.");
    }
}



VOID BMP::WRITE(CONST CHAR *FNAME) {
    STD::OFSTREAM OF{ FNAME, STD::OFSTREAM::BINARY };
    IF (OF) {
        IF (BMP_INFO_HEADER.BIT_COUNT == 32) {
            WRITE_HEADERS_AND_DATA(OF);
        }
        ELSE IF (BMP_INFO_HEADER.BIT_COUNT == 24) {
            IF (BMP_INFO_HEADER.WIDTH % 4 == 0) {
                WRITE_HEADERS_AND_DATA(OF);
            }
            ELSE {
                UINT32_T NEW_STRIDE = MAKE_STRIDE_ALIGNED(4);
                STD::VECTOR<UINT8_T> PADDING_ROW(NEW_STRIDE -
ROW_STRIDE);

                WRITE_HEADERS(OF);

                FOR (INT Y = 0; Y < BMP_INFO_HEADER.HEIGHT; ++Y) {
                    OF.WRITE((CONST CHAR*)(DATA.DATA() + ROW_STRIDE * Y),
ROW_STRIDE);
                    OF.WRITE((CONST CHAR*)PADDING_ROW.DATA(),
PADDING_ROW.SIZE());
                }
            }
        }
        ELSE {
```

```
            THROW STD::RUNTIME_ERROR("THE PROGRAM CAN TREAT ONLY 24 OR 32
BITS PER PIXEL BMP FILES");
        }
    }
    ELSE {
        THROW STD::RUNTIME_ERROR("UNABLE TO OPEN THE OUTPUT IMAGE
FILE.");
    }
}


VOID BMP::WRITE(CONST CHAR *FNAME, CHAR *SHAPES_FNAME) {
    STD::OFSTREAM OF{ FNAME, STD::OFSTREAM::BINARY };
    IF (OF) {
        IF (BMP_INFO_HEADER.BIT_COUNT == 32) {
            WRITE_HEADERS_AND_DATA(OF);
        }
        ELSE IF (BMP_INFO_HEADER.BIT_COUNT == 24) {
            IF (BMP_INFO_HEADER.WIDTH % 4 == 0) {
                WRITE_HEADERS_AND_DATA(OF);
            }
            ELSE {
                UINT32_T NEW_STRIDE = MAKE_STRIDE_ALIGNED(4);
                STD::VECTOR<UINT8_T> PADDING_ROW(NEW_STRIDE -
ROW_STRIDE);

                WRITE_HEADERS(OF);

                FOR (INT Y = 0; Y < BMP_INFO_HEADER.HEIGHT; ++Y) {
                    OF.WRITE((CONST CHAR*)(DATA.DATA() + ROW_STRIDE * Y),
ROW_STRIDE);
                    OF.WRITE((CONST CHAR*)PADDING_ROW.DATA(),
PADDING_ROW.SIZE());
                }
            }
        }
        ELSE {
```

```
            THROW STD::RUNTIME_ERROR("THE PROGRAM CAN TREAT ONLY 24 OR 32
BITS PER PIXEL BMP FILES");
        }
    }
    ELSE {
        THROW STD::RUNTIME_ERROR("UNABLE TO OPEN THE OUTPUT IMAGE
FILE.");
    }
    IF (SHAPES.WRITE2FILE(SHAPES_FNAME)){
        PRINTF("SHAPES ARE SUCCESSFULLY STORED!\N");
    }ELSE{
        PRINTF("ERROR IN STORING SHAPES IN FILE!\N");
    }
}


VOID BMP::FILL_REGION(UINT32_T X0, UINT32_T Y0, UINT32_T W, UINT32_T H,
UINT8_T B, UINT8_T G, UINT8_T R, UINT8_T A) {
    IF (X0 + W > (UINT32_T)BMP_INFO_HEADER.WIDTH || Y0 + H >
(UINT32_T)BMP_INFO_HEADER.HEIGHT) {
        THROW STD::RUNTIME_ERROR("THE REGION DOES NOT FIT IN THE
IMAGE!");
    }

    UINT32_T CHANNELS = BMP_INFO_HEADER.BIT_COUNT / 8;
    FOR (UINT32_T Y = Y0; Y < Y0 + H; ++Y) {
        FOR (UINT32_T X = X0; X < X0 + W; ++X) {
            DATA[CHANNELS * (Y * BMP_INFO_HEADER.WIDTH + X) + 0] = B;
            DATA[CHANNELS * (Y * BMP_INFO_HEADER.WIDTH + X) + 1] = G;
            DATA[CHANNELS * (Y * BMP_INFO_HEADER.WIDTH + X) + 2] = R;
            IF (CHANNELS == 4) {
                DATA[CHANNELS * (Y * BMP_INFO_HEADER.WIDTH + X) + 3] = A;
            }
        }
    }
}
```

```
VOID BMP::SET_PIXEL(UINT32_T X0, UINT32_T Y0, UINT8_T B, UINT8_T G,
UINT8_T R, UINT8_T A) {
    IF (X0 > (UINT32_T)BMP_INFO_HEADER.WIDTH || Y0 >
(UINT32_T)BMP_INFO_HEADER.HEIGHT) {
        THROW STD::RUNTIME_ERROR("THE POINT IS OUTSIDE THE IMAGE
BOUNDARIES!");
    }

    UINT32_T CHANNELS = BMP_INFO_HEADER.BIT_COUNT / 8;
    DATA[CHANNELS * (Y0 * BMP_INFO_HEADER.WIDTH + X0) + 0] = B;
    DATA[CHANNELS * (Y0 * BMP_INFO_HEADER.WIDTH + X0) + 1] = G;
    DATA[CHANNELS * (Y0 * BMP_INFO_HEADER.WIDTH + X0) + 2] = R;
    IF (CHANNELS == 4) {
        DATA[CHANNELS * (Y0 * BMP_INFO_HEADER.WIDTH + X0) + 3] = A;
    }
}


VOID BMP::GET_PIXEL(UINT32_T X0, UINT32_T Y0, UINT8_T* R, UINT8_T* G,
UINT8_T* B, UINT8_T* A) {
    IF (X0 > (UINT32_T)BMP_INFO_HEADER.WIDTH || Y0 >
(UINT32_T)BMP_INFO_HEADER.HEIGHT) {
        THROW STD::RUNTIME_ERROR("THE POINT IS OUTSIDE THE IMAGE
BOUNDARIES!");
    }

    UINT32_T CHANNELS = BMP_INFO_HEADER.BIT_COUNT / 8;
    *B = DATA[CHANNELS * (Y0 * BMP_INFO_HEADER.WIDTH + X0) + 0];
    *G = DATA[CHANNELS * (Y0 * BMP_INFO_HEADER.WIDTH + X0) + 1];
    *R = DATA[CHANNELS * (Y0 * BMP_INFO_HEADER.WIDTH + X0) + 2];
    IF (CHANNELS == 4) {
        *A = DATA[CHANNELS * (Y0 * BMP_INFO_HEADER.WIDTH + X0) + 3];
    }
}

VOID BMP::ADD_IMAGE(BMP CURRENT, INT X, INT Y, INT IMAGE_PANEL_WIDTH, INT
IMAGE_PANEL_HEIGHT){
    IF (CURRENT.BMP_INFO_HEADER.WIDTH <= THIS->BMP_INFO_HEADER.WIDTH
```

```
        && CURRENT.BMP_INFO_HEADER.HEIGHT <=
THIS->BMP_INFO_HEADER.HEIGHT){
        UINT8_T R, G, B, A;

        FOR (INT I=0; I<CURRENT.BMP_INFO_HEADER.WIDTH; I++){
            FOR (INT J=0; J<CURRENT.BMP_INFO_HEADER.HEIGHT; J++){
                CURRENT.GET_PIXEL(I,J,&R,&G,&B,&A);
                THIS->SET_PIXEL(I+X, J+Y, B, G, R, A);
            }
        }
    }ELSE{
        PRINTF("ERROR: THE IMAGE DOESN'T FIT...");
        EXIT(EXIT_FAILURE);
    }
}

VOID BMP::DRAW_RECTANGLE(RECTANGLE RECT){
    UINT32_T X0 = RECT.X_START, Y0 = RECT.Y_START, W = RECT.WIDTH, H =
RECT.HEIGHT;
    UINT8_T B = RECT.COLOR_B, G = RECT.COLOR_G, R = RECT.COLOR_R, A =
ALPHA, LINE_W = RECT.THICKNESS;
    BOOL FILLED = RECT.FILLED;
    UINT8_T BF = RECT.FILLED_COLOR_B, GF = RECT.FILLED_COLOR_G, RF =
RECT.FILLED_COLOR_R;
    IF (X0 + W > (UINT32_T)BMP_INFO_HEADER.WIDTH || Y0 + H >
(UINT32_T)BMP_INFO_HEADER.HEIGHT) {
        THROW STD::RUNTIME_ERROR("THE RECTANGLE DOES NOT FIT IN THE
IMAGE!");
    }

    FILL_REGION(X0, Y0, W, LINE_W, B, G, R, A);
// TOP LINE
    FILL_REGION(X0, (Y0 + H - LINE_W), W, LINE_W, B, G, R, A);
// BOTTOM LINE
    FILL_REGION((X0 + W - LINE_W), (Y0 + LINE_W), LINE_W, (H - (2 *
LINE_W)), B, G, R, A);  // RIGHT LINE
    FILL_REGION(X0, (Y0 + LINE_W), LINE_W, (H - (2 * LINE_W)), B, G, R,
A);                 // LEFT LINE
```

```
    IF (FILLED==1)
        FILL_REGION(X0+LINE_W, Y0+LINE_W, W-2*LINE_W, H-2*LINE_W, BF, GF,
RF, A);

    // ADDING THE TRIANGLE TO SHAPES OF BMP
    (SHAPES).ADD_RECTANGLE(RECT);
}


VOID BMP::DRAW_TRIANGLE(TRIANGLE TRIANGLE){
    UINT32_T X0 = TRIANGLE.X1, Y0 = TRIANGLE.Y1, X1 = TRIANGLE.X2, Y1 =
TRIANGLE.Y2, X2 = TRIANGLE.X3, Y2 = TRIANGLE.Y3;
    UINT8_T B = TRIANGLE.COLOR_B, G = TRIANGLE.COLOR_G, R =
TRIANGLE.COLOR_R, A = ALPHA, LINE_W = TRIANGLE.THICKNESS;
    BOOL FILLED = TRIANGLE.FILLED;
    UINT8_T BF = TRIANGLE.FILLED_COLOR_B, GF = TRIANGLE.FILLED_COLOR_G,
RF = TRIANGLE.FILLED_COLOR_R;
//    TRIANGLE.FILENAME

    UINT32_T XMAX_IMAGE = BMP_INFO_HEADER.WIDTH;
    UINT32_T YMAX_IMAGE = BMP_INFO_HEADER.HEIGHT;
    UINT32_T XMAX, YMAX, XMIN, YMIN;
    IF ((X0==X1 &&Y0==Y1) || (X1==X2 && Y1==Y2) || (X0==X2 && Y0==Y2)){
        THROW STD::RUNTIME_ERROR("THE COORDINATES DO NOT FORM A
TRIANGLE!");
    }
    IF (X0<XMAX_IMAGE && X1<XMAX_IMAGE && X2<XMAX_IMAGE &&
            Y0<YMAX_IMAGE && Y1<YMAX_IMAGE && Y2<YMAX_IMAGE){


        STD::VECTOR<UINT32_T> LINE_1_PIXELS =
                DRAWCOLOREDLINE(X0,Y0,X1,Y1,LINE_W, B,G,R,A);
        STD::VECTOR<UINT32_T> LINE_2_PIXELS =
                DRAWCOLOREDLINE(X1,Y1,X2,Y2,LINE_W, B,G,R,A);
        STD::VECTOR<UINT32_T> LINE_3_PIXELS =
                DRAWCOLOREDLINE(X2,Y2,X0,Y0,LINE_W, B,G,R,A);
        IF (FILLED){
            // X0 X1 X2
```

```
            // Y0 Y1 Y2

            XMAX = MAX(MAX(X0,X1),X2);
            YMAX = MAX(MAX(Y0,Y1),Y2);
            XMIN = MIN(MIN(X0,X1),X2);
            YMIN = MIN(MIN(Y0,Y1),Y2);

            FOR (UINT32_T XI = XMIN; XI<XMAX; XI++){
                FOR (UINT32_T YI = YMIN; YI<YMAX; YI++){
                    IF (INSIDE_TRIANGLE(XI, YI, TRIANGLE)){
                        VECTOR<UINT32_T>::ITERATOR
                                ITX1= STD::FIND(LINE_1_PIXELS.BEGIN(),
LINE_1_PIXELS.END(),XI),
                                ITX2= STD::FIND(LINE_2_PIXELS.BEGIN(),
LINE_2_PIXELS.END(),XI),
                                ITX3= STD::FIND(LINE_3_PIXELS.BEGIN(),
LINE_3_PIXELS.END(),XI);
                        VECTOR<UINT32_T>::ITERATOR
                                ITY1= STD::FIND(LINE_1_PIXELS.BEGIN(),
LINE_1_PIXELS.END(),YI),
                                ITY2= STD::FIND(LINE_2_PIXELS.BEGIN(),
LINE_2_PIXELS.END(),YI),
                                ITY3= STD::FIND(LINE_3_PIXELS.BEGIN(),
LINE_3_PIXELS.END(),YI);
                        INT INDEX1 = STD::DISTANCE(LINE_1_PIXELS.BEGIN(),
ITX1);
                        INT INDEX2 = STD::DISTANCE(LINE_2_PIXELS.BEGIN(),
ITX2);
                        INT INDEX3 = STD::DISTANCE(LINE_3_PIXELS.BEGIN(),
ITX3);

                        INT INDEX1Y =
STD::DISTANCE(LINE_1_PIXELS.BEGIN(), ITY1);
                        INT INDEX2Y =
STD::DISTANCE(LINE_2_PIXELS.BEGIN(), ITY2);
                        INT INDEX3Y =
STD::DISTANCE(LINE_3_PIXELS.BEGIN(), ITY3);
```

```
                        IF (ITX1 != LINE_1_PIXELS.END() && INDEX1%2==0
                                && ITY1 != LINE_1_PIXELS.END() &&
INDEX1Y==INDEX1+1
                                && ITX2 != LINE_2_PIXELS.END() &&
INDEX2%2==0
                                && ITY2 != LINE_2_PIXELS.END() &&
INDEX2Y==INDEX2+1
                                && ITX3 != LINE_3_PIXELS.END() &&
INDEX3%2==0
                                && ITY3 != LINE_3_PIXELS.END() &&
INDEX3Y==INDEX3+1){
                            ;//DO NOTHING
                        }ELSE{
                            SET_PIXEL(XI,YI,BF,GF,RF,A); // INSIDE
TRIANGLE AND NOT BORDERS
                        }
                    }
                }
            }
        }
            // ADDING THE TRIANGLE TO SHAPES OF BMP
            SHAPES.ADD_TRIANGLE(TRIANGLE);

        }ELSE{
            THROW STD::RUNTIME_ERROR("THE TRIANGLE DOES NOT FIT IN THE
IMAGE!");
        }
}


STD::VECTOR<UINT32_T> BMP::DRAWCOLOREDLINE(INT X0, INT Y0, INT X1, INT
Y1, FLOAT WD, UINT8_T B, UINT8_T G, UINT8_T R, UINT8_T A)
{                                   /* PLOT AN ANTI-ALIASED LINE OF
WIDTH WD */
   INT DX = ABS(INT(X1-X0)), SX = X0 < X1 ? 1 : -1;
   INT DY = ABS(INT(Y1-Y0)), SY = Y0 < Y1 ? 1 : -1;
```

```
   INT ERR = DX-DY, E2, X2, Y2;                              /* ERROR VALUE
E_XY */
   FLOAT ED = DX+DY == 0 ? 1 : STD::SQRT((FLOAT)DX*DX+(FLOAT)DY*DY);
   UINT32_T X,Y;
   STD::VECTOR<UINT32_T> PIXELS;


   FOR (WD = (WD+1)/2; ; ) {                                 /* PIXEL LOOP
*/
      X= X0; Y=Y0;
      SET_PIXEL(X, Y, B, G, R, A);
      PIXELS.PUSH_BACK(X);
      PIXELS.PUSH_BACK(Y);
      E2 = ERR; X2 = X0;
      IF (2*E2 >= -DX) {   /* X STEP */
         FOR (E2 += DY, Y2 = Y0; E2 < ED*WD && (Y1 != Y2 || DX > DY); E2
+= DX)
          {
            X = X0;
            Y = Y2;

            SET_PIXEL(X, Y, B, G, R, A);
            PIXELS.PUSH_BACK(X);
            PIXELS.PUSH_BACK(Y);
            Y2 += SY;
         }
         IF (X0 == X1) BREAK;
         E2 = ERR; ERR -= DY; X0 += SX;
      }
      IF (2*E2 <= DY) {    /* Y STEP */
         FOR (E2 = DX-E2; E2 < ED*WD && (X1 != X2 || DX < DY); E2 += DY)
         {
            X = X2;
            Y = Y0;
            SET_PIXEL(X, Y, B, G, R, A);
            PIXELS.PUSH_BACK(X);
            PIXELS.PUSH_BACK(Y);
            X2 += SX;
```

```
            }
            IF (Y0 == Y1) BREAK;
            ERR += DX; Y0 += SY;
        }
    }
    RETURN PIXELS;
}

VOID BMP::WRITE_HEADERS(STD::OFSTREAM &OF) {
    OF.WRITE((CONST CHAR*)&FILE_HEADER, SIZEOF(FILE_HEADER));
    OF.WRITE((CONST CHAR*)&BMP_INFO_HEADER, SIZEOF(BMP_INFO_HEADER));
    IF(BMP_INFO_HEADER.BIT_COUNT == 32) {
        OF.WRITE((CONST CHAR*)&BMP_COLOR_HEADER,
SIZEOF(BMP_COLOR_HEADER));
    }
}

VOID BMP::WRITE_HEADERS_AND_DATA(STD::OFSTREAM &OF) {
    WRITE_HEADERS(OF);
    OF.WRITE((CONST CHAR*)DATA.DATA(), DATA.SIZE());
}

// ADD 1 TO THE ROW_STRIDE UNTIL IT IS DIVISIBLE WITH ALIGN_STRIDE
UINT32_T BMP::MAKE_STRIDE_ALIGNED(UINT32_T ALIGN_STRIDE) {
    UINT32_T NEW_STRIDE = ROW_STRIDE;
    WHILE (NEW_STRIDE % ALIGN_STRIDE != 0) {
        NEW_STRIDE++;
    }
    RETURN NEW_STRIDE;
}

// CHECK IF THE PIXEL DATA IS STORED AS BGRA AND IF THE COLOR SPACE TYPE
IS SRGB
VOID BMP::CHECK_COLOR_HEADER(BMPCOLORHEADER &BMP_COLOR_HEADER) {
    BMPCOLORHEADER EXPECTED_COLOR_HEADER;
    IF(EXPECTED_COLOR_HEADER.RED_MASK != BMP_COLOR_HEADER.RED_MASK ||
        EXPECTED_COLOR_HEADER.BLUE_MASK != BMP_COLOR_HEADER.BLUE_MASK ||
```

```
        EXPECTED_COLOR_HEADER.GREEN_MASK != BMP_COLOR_HEADER.GREEN_MASK
||
        EXPECTED_COLOR_HEADER.ALPHA_MASK != BMP_COLOR_HEADER.ALPHA_MASK)
{
        THROW STD::RUNTIME_ERROR("UNEXPECTED COLOR MASK FORMAT! THE
PROGRAM EXPECTS THE PIXEL DATA TO BE IN THE BGRA FORMAT");
    }
    IF(EXPECTED_COLOR_HEADER.COLOR_SPACE_TYPE !=
BMP_COLOR_HEADER.COLOR_SPACE_TYPE) {
        THROW STD::RUNTIME_ERROR("UNEXPECTED COLOR SPACE TYPE! THE
PROGRAM EXPECTS SRGB VALUES");
    }
}
```

COMMANDS.H:

```
#IFNDEF COMMANDS_H
#DEFINE COMMANDS_H


#IFNDEF UTILS_H
#INCLUDE<UTILS.H>
#ENDIF
#IF !DEFINED(BMP_H)
#INCLUDE<BMP.H>
#ENDIF

TRIANGLE DRAW_TRIANGLE_COMMAND(INT ARGC, CHAR *ARGV[]){//, STRUCT BMP BMP

    PRINTF("DRAWING A TRIANGLE\N");

    TRIANGLE TRIANGLE;
    INT OPT= 0;

    STATIC STRUCT OPTION LONG_OPTIONS[] = {
        {"P0", REQUIRED_ARGUMENT, 0,  'A' }, // -A   --P0
        {"P1", REQUIRED_ARGUMENT,       0, 'B' },
        {"P2", REQUIRED_ARGUMENT, 0,  'C' },
        {"THICK",    REQUIRED_ARGUMENT, 0,  'T' },
```

```
        {"COLOR",   REQUIRED_ARGUMENT, 0,  'C' }, // --COLOR -C
        {"FILLED",   NO_ARGUMENT, 0,  'F' },
        {"FILLCOLOR",   REQUIRED_ARGUMENT, 0,  'L' },
        {"INPUT",   REQUIRED_ARGUMENT, 0,  'I' },
        {0,           0,                 0,  0   }
    };

    INT LONG_INDEX =0;
    BOOL FILL_COLORED=FALSE;
    CHAR* FILENAME = NULL;
    UNSIGNED INT WIDTH=1, HEIGHT=1;
    BMP BMP(WIDTH, HEIGHT,FALSE);
    TRY {
        IF ((OPT = GETOPT_LONG(ARGC, ARGV,"A:B:C:T:C:FL:I:",
                               LONG_OPTIONS, &LONG_INDEX ))==-1){THROW
EXCEPTION();}
        IF(ARGC < 18) THROW EXCEPTION();
        DO{
            SWITCH (OPT) {
            CASE 'A' :
                    TRIANGLE.X1 = STOI(OPTARG);
                    TRIANGLE.Y1 = STOI(ARGV[OPTIND]);
                    BREAK;
            CASE 'B' :

                TRIANGLE.X2 = STOI(OPTARG);
                TRIANGLE.Y2 = STOI(ARGV[OPTIND]);
                BREAK;
            CASE 'C' :

                TRIANGLE.X3 = STOI(OPTARG);
                TRIANGLE.Y3 = STOI(ARGV[OPTIND]);
                BREAK;
            CASE 'T' :
                    TRIANGLE.THICKNESS = STOI(OPTARG);
                    BREAK;
            CASE 'C' :
                    TRIANGLE.COLOR_R = STOI(OPTARG);
```

```
                    TRIANGLE.COLOR_G = STOI(ARGV[OPTIND]);
                    TRIANGLE.COLOR_B = STOI(ARGV[OPTIND+1]);
                    BREAK;
            CASE 'F':
                    TRIANGLE.FILLED = TRUE;
                    BREAK;
            CASE 'L':
                IF (TRIANGLE.FILLED){
                   TRIANGLE.FILLED_COLOR_R = STOI(ARGV[OPTIND-1]);
                   TRIANGLE.FILLED_COLOR_G = STOI(ARGV[OPTIND]);
                   TRIANGLE.FILLED_COLOR_B = STOI(ARGV[OPTIND+1]);
                }
                FILL_COLORED = TRUE;
                 BREAK;
            CASE 'I':
                IF (IS_BMP_FILE(OPTARG)){
                    FILENAME = OPTARG;
                }ELSE{
                    PRINTF("ERROR: IMAGE FILE NAME IS NOT
RECOGNIZED...\N");
                    THROW EXCEPTION();
                }
                //
                IF (FEXISTS(FILENAME)){
                    BMP =  BMP(FILENAME);
                    WIDTH = BMP.BMP_INFO_HEADER.WIDTH;
                    HEIGHT = BMP.BMP_INFO_HEADER.HEIGHT;
                }ELSE{
                    TRY {
                        WIDTH = STOI(ARGV[OPTIND]);
                        IF (WIDTH<TRIANGLE.GET_MAX_X()) THROW
EXCEPTION();
                        HEIGHT = STOI(ARGV[OPTIND+1]);
                        IF (HEIGHT<TRIANGLE.GET_MAX_Y()) THROW
EXCEPTION();
                    } CATCH (EXCEPTION CONST & E) {
                            WIDTH = TRIANGLE.GET_MAX_X()*2, HEIGHT =
TRIANGLE.GET_MAX_Y()*2;
```

```
                            PRINTF("WARNING: WIDTH AND HEIGHT SHOULD BE
UNSIGNED INTEGERS.\NDEFAULT VALUES: WIDTH = %D , HEIGHT = %D\N", WIDTH,
HEIGHT);
                        }
                    }

                    PRINTF("CURRENT VALUES: WIDTH = %D, HEIGHT = %D\N",
WIDTH, HEIGHT);
                    BREAK;

                DEFAULT: PRINT_USAGE_TRI(); EXIT(EXIT_FAILURE);
                }
            }WHILE ((OPT = GETOPT_LONG(ARGC, ARGV,"A:B:C:T:C:FL:I:",
                       LONG_OPTIONS, &LONG_INDEX )) != -1);
            IF (FILENAME==NULL || STRCMP(FILENAME,"")==0) THROW EXCEPTION();


            IF(TRIANGLE.FILLED && !FILL_COLORED){
               PRINTF("NOTE: YOU SHOULD SPECIFY FILL COLOR FOR THE TRIANGLE "
                      "BUT IT WILL BE ASSIGNED TO BORDER COLOR\N");
               TRIANGLE.FILLED_COLOR_R = TRIANGLE.COLOR_R;
               TRIANGLE.FILLED_COLOR_G = TRIANGLE.COLOR_G;
               TRIANGLE.FILLED_COLOR_B = TRIANGLE.COLOR_B;
               // THE USER HAVE TO SELECT FILLED
            }
        } CATCH (EXCEPTION CONST & E) {
                PRINTF("COMMAND OPTIONS ERROR \N");
                PRINT_USAGE_TRI();
                EXIT(EXIT_FAILURE);

        }

            CHAR* SHAPES_FILENAME = SHAPES::GET_SHAPES_FILENAME(FILENAME);
            BMP = BMP(WIDTH, HEIGHT, FALSE);
            IF (FEXISTS(STRING2ARRAY(FILENAME)) &&
FEXISTS(STRING2ARRAY(SHAPES_FILENAME))){
                BMP = BMP(STRING2ARRAY(FILENAME),
STRING2ARRAY(SHAPES_FILENAME));
```

```
        }
        BMP.DRAW_TRIANGLE(TRIANGLE);
        BMP.WRITE(FILENAME, SHAPES_FILENAME);
    RETURN TRIANGLE;
}

RECTANGLE DRAW_RECTANGLE_COMMAND(INT ARGC, CHAR *ARGV[]){

    PRINTF("DRAWING A RECTANGLE\N");

    RECTANGLE RECT;
    INT OPT= 0;


    STATIC STRUCT OPTION LONG_OPTIONS[] = {
        {"START", REQUIRED_ARGUMENT, 0,  'S' },
        {"WIDTH", REQUIRED_ARGUMENT, 0,  'W' },
        {"HEIGHT", REQUIRED_ARGUMENT, 0,  'H' },
        {"THICK",    REQUIRED_ARGUMENT, 0,  'T' },
        {"COLOR",   REQUIRED_ARGUMENT, 0,  'C' },
        {"FILLED",   NO_ARGUMENT, 0,  'F' },
        {"FILLCOLOR",   REQUIRED_ARGUMENT, 0,  'L' },
    {"INPUT",   REQUIRED_ARGUMENT, 0,  'I' },
        {0,           0,                 0,  0   }
    };

    INT LONG_INDEX =0;
    BOOL FILL_COLORED=FALSE;
    CHAR* FILENAME = NULL;
    UNSIGNED INT WIDTH=1, HEIGHT=1;
    BMP BMP(WIDTH, HEIGHT,FALSE);
    TRY {
        IF((OPT = GETOPT_LONG(ARGC, ARGV,"S:W:H:T:C:FL:I:",
                              LONG_OPTIONS, &LONG_INDEX ))==-1)THROW
EXCEPTION();
        IF (ARGC < 16) THROW EXCEPTION();
        DO{
            SWITCH (OPT) {
```

```
                CASE 'S' :
                        RECT.X_START = STOI(OPTARG);
                        RECT.Y_START = STOI(ARGV[OPTIND]);
                        BREAK;
                CASE 'W' :

                    RECT.WIDTH = STOI(OPTARG);
                    RECT.X_END = RECT.X_START+RECT.WIDTH;
                    BREAK;
                CASE 'H' :
                    RECT.HEIGHT = STOI(OPTARG);
                    RECT.Y_END = RECT.Y_START + RECT.HEIGHT;
                    BREAK;
                CASE 'T' :
                        RECT.THICKNESS = STOI(OPTARG);
                        BREAK;
                CASE 'C' :
                        RECT.COLOR_R = STOI(OPTARG);
                        RECT.COLOR_G = STOI(ARGV[OPTIND]);
                        RECT.COLOR_B = STOI(ARGV[OPTIND+1]);
                        BREAK;
                CASE 'F':
                        RECT.FILLED = TRUE;
                        BREAK;
                CASE 'L':
                    IF(RECT.FILLED){
                       RECT.FILLED_COLOR_R = STOI(ARGV[OPTIND-1]);
                       RECT.FILLED_COLOR_G = STOI(ARGV[OPTIND]);
                       RECT.FILLED_COLOR_B = STOI(ARGV[OPTIND+1]);
                   }
                   FILL_COLORED = TRUE;
                    BREAK;
                CASE 'I':
                    IF (IS_BMP_FILE(OPTARG)){
                        FILENAME = OPTARG;
                    }ELSE{
                        PRINTF("ERROR: IMAGE FILE NAME IS NOT
RECOGNIZED...\N");
```

```
                        THROW EXCEPTION();
                    }
                    IF (FEXISTS(FILENAME)){
                        BMP =  BMP(FILENAME);
                        WIDTH = BMP.BMP_INFO_HEADER.WIDTH;
                        HEIGHT = BMP.BMP_INFO_HEADER.HEIGHT;
                    }ELSE{
                        TRY {
                            WIDTH = STOI(ARGV[OPTIND]);
                            IF (WIDTH<RECT.GET_MAX_X()) THROW
EXCEPTION();
                            HEIGHT = STOI(ARGV[OPTIND+1]);
                            IF (HEIGHT<RECT.GET_MAX_Y()) THROW
EXCEPTION();
                        } CATCH (EXCEPTION CONST & E) {
                                WIDTH = RECT.GET_MAX_X()*2, HEIGHT =
RECT.GET_MAX_Y()*2;
                                PRINTF("WARNING: WIDTH AND HEIGHT SHOULD
BE UNSIGNED INTEGERS.\NDEFAULT VALUES: WIDTH = %D , HEIGHT = %D\N",
WIDTH, HEIGHT);
                        }
                    }

                    PRINTF("CURRENT VALUES: WIDTH = %D, HEIGHT = %D\N",
WIDTH, HEIGHT);
                    BREAK;

                DEFAULT: PRINT_USAGE_RECT(); EXIT(EXIT_FAILURE);
            }
        } WHILE ((OPT = GETOPT_LONG(ARGC, ARGV,"S:W:H:T:C:FL:I:",
                    LONG_OPTIONS, &LONG_INDEX )) != -1);
        IF (FILENAME==NULL || STRCMP(FILENAME,"")==0) THROW EXCEPTION();


        IF(RECT.FILLED && !FILL_COLORED){
           PRINTF("NOTE: YOU SHOULD SPECIFY FILL COLOR FOR THE TRIANGLE "
                  "BUT IT WILL BE ASSIGNED TO BORDER COLOR\N");
           RECT.FILLED_COLOR_R = RECT.COLOR_R;
```

```
            RECT.FILLED_COLOR_G = RECT.COLOR_G;
            RECT.FILLED_COLOR_B = RECT.COLOR_B;
            // THE USER HAVE TO SELECT FILLED
        }
    }CATCH (EXCEPTION CONST & E) {
            PRINTF("COMMAND OPTIONS ERROR \N");
            PRINT_USAGE_RECT();
            EXIT(EXIT_FAILURE);
    }

    CHAR* SHAPES_FILENAME = SHAPES::GET_SHAPES_FILENAME(FILENAME);
    BMP = BMP(WIDTH, HEIGHT, FALSE);
    IF (FEXISTS(STRING2ARRAY(FILENAME)) &&
FEXISTS(STRING2ARRAY(SHAPES_FILENAME))){
        BMP = BMP(STRING2ARRAY(FILENAME), STRING2ARRAY(SHAPES_FILENAME));
    }
    BMP.DRAW_RECTANGLE(RECT);
    BMP.WRITE(FILENAME, SHAPES_FILENAME);
    RETURN RECT;
}

VOID REPAINT_COMMAND(INT ARGC, CHAR *ARGV[]){
    PRINTF("REPAINTING A SHAPE!\N");
        UINT8_T NEWR = 0; // 0-255
        UINT8_T NEWG = 0;
        UINT8_T NEWB = 0;
        BOOL IS_RECTANGLE = FALSE;
        CHAR* FILENAME;
        INT OPT = 0;

        TRY{
            STATIC STRUCT OPTION LONG_OPTIONS[] = {
                {"SHAPE", REQUIRED_ARGUMENT, 0,  'S' },
                {"COLOR", REQUIRED_ARGUMENT, 0,  'C' },
                {"INPUT", REQUIRED_ARGUMENT, 0,  'I' },
                {0,           0,                 0,  0   }
            };
```

```
            INT LONG_INDEX = 0;
            IF ((OPT = GETOPT_LONG(ARGC, ARGV,"S:C:I:",
                                    LONG_OPTIONS, &LONG_INDEX ))==-1)
THROW EXCEPTION();
            IF (ARGC < 10) THROW EXCEPTION();
                DO{
                    SWITCH (OPT) {
                        CASE 'S' : // REPAINT -S CIRCLE
                            IF (STRCMP(OPTARG,"RECT")==0) IS_RECTANGLE =
TRUE;
                            ELSE IF (STRCMP(OPTARG,"TRI")==0)
IS_RECTANGLE = FALSE;
                            ELSE {PRINTF("ONLY RECT AND TRI IS
AVAILIBILE\N");THROW EXCEPTION();}
                            BREAK;
                        CASE 'C' :// REPAINT -S RECT -C 150 40 0
                            NEWR = STOI(OPTARG);
                            NEWG = STOI(ARGV[OPTIND]);
                            NEWB = STOI(ARGV[OPTIND+1]);
                            BREAK;
                    CASE 'I' :
                        IF (IS_BMP_FILE(OPTARG) && FEXISTS(OPTARG)){
                            FILENAME = OPTARG;
                        }ELSE{
                            PRINTF("ERROR: IMAGE FILE NAME IS NOT
RECOGNIZED OR NOT FOUND...\N");
                            THROW EXCEPTION();
                        }
                        BREAK;
                    DEFAULT: THROW EXCEPTION();
                    }
            }WHILE ((OPT = GETOPT_LONG(ARGC, ARGV,"S:C:I:",
                                LONG_OPTIONS, &LONG_INDEX )) != -1);
            IF (!IS_BMP_FILE(FILENAME) || !FEXISTS(FILENAME))
{PRINTF("ERROR: FILE NOT RECOGNIZED\N");THROW EXCEPTION();}

            CHAR* SHAPES_FILENAME =
SHAPES::GET_SHAPES_FILENAME(FILENAME);
```

```
            BMP BMP = BMP(STRING2ARRAY(FILENAME),
STRING2ARRAY(SHAPES_FILENAME));

            INT INDEX;
            IF (IS_RECTANGLE && BMP.SHAPES.RECTANGLES.SIZE()>0){
               BMP.SHAPES.GET_LARGEST_RECTANGLE(&INDEX);

               RECTANGLE RECT = BMP.SHAPES.RECTANGLES.AT(INDEX);
               RECT.COLOR_R = NEWR;
               RECT.COLOR_G = NEWG;
               RECT.COLOR_B = NEWB;
               BMP.DRAW_RECTANGLE(RECT);

BMP.SHAPES.RECTANGLES.ERASE(BMP.SHAPES.RECTANGLES.BEGIN()+INDEX);
            }ELSE IF(BMP.SHAPES.TRIANGLES.SIZE()>0){
               BMP.SHAPES.GET_LARGEST_TRIANGLE(&INDEX);
               TRIANGLE TRI = BMP.SHAPES.TRIANGLES.AT(INDEX);
               TRI.COLOR_R = NEWR;
               TRI.COLOR_G = NEWG;
               TRI.COLOR_B = NEWB;
               BMP.DRAW_TRIANGLE(TRI);

BMP.SHAPES.TRIANGLES.ERASE(BMP.SHAPES.TRIANGLES.BEGIN()+INDEX);
           }ELSE{
                THROW STD::RUNTIME_ERROR("SHAPE NOT FOUND \N");

            }
            BMP.WRITE(FILENAME, SHAPES_FILENAME);
        }CATCH (EXCEPTION CONST & E) {
            PRINTF("COMMAND OPTIONS ERROR\N");
            PRINT_USAGE_REPAINT();
            EXIT(EXIT_FAILURE);
        }
}

UNSIGNED INT GETLARGESTBMP(STD::VECTOR<BMP> IMAGES, INT* MAX_WIDTH, INT*
MAX_HEIGHT){
    UNSIGNED INT MAX = 0;
```

```
    UNSIGNED INT INDEX = 0, MAX_INDEX;
    *MAX_WIDTH = *MAX_HEIGHT = 0;
    FOR(AUTO IMAGE:IMAGES){
        UNSIGNED INT VALUE = IMAGE.BMP_INFO_HEADER.WIDTH *
IMAGE.BMP_INFO_HEADER.HEIGHT;
        IF (VALUE>MAX){
            MAX = VALUE;
            MAX_INDEX = INDEX;
        }
        IF (IMAGE.BMP_INFO_HEADER.WIDTH>*MAX_WIDTH){
            *MAX_WIDTH = IMAGE.BMP_INFO_HEADER.WIDTH;
        }
        IF (IMAGE.BMP_INFO_HEADER.HEIGHT>*MAX_HEIGHT){
            *MAX_HEIGHT = IMAGE.BMP_INFO_HEADER.HEIGHT;
        }
        INDEX++;
    }
    RETURN INDEX;
}

BMP CREATE_COLLAGE(INT M, INT N, STD::VECTOR<BMP> BMPS){
    INT MAX_WIDTH, MAX_HEIGHT;
    GETLARGESTBMP(BMPS, &MAX_WIDTH, &MAX_HEIGHT);
    BMP COLLAGE = BMP(MAX_WIDTH*M, MAX_HEIGHT*N, FALSE);
    INT NEXT = 0;
    INT X, Y;
    FOR (INT I=0; I < M; I++){
        FOR (INT J=0; J < N; J++){
           X = I*MAX_WIDTH; // X = 0 , Y = 0 ,
           Y = J*MAX_HEIGHT;
           // CENTER THE IMAGE
            X += (MAX_WIDTH-BMPS.AT(NEXT).BMP_INFO_HEADER.WIDTH)/2;
            Y += (MAX_HEIGHT-BMPS.AT(NEXT).BMP_INFO_HEADER.HEIGHT)/2;

            COLLAGE.ADD_IMAGE(BMPS[NEXT++], X, Y, MAX_WIDTH, MAX_HEIGHT);
            NEXT = NEXT % BMPS.SIZE(); // NEXT 0 , 1 , 2 , 3 , 0 , 1 ,
2 , 3
        }
```

```
    }
    RETURN COLLAGE;
}

VOID COLLAGE_COMMAND(INT ARGC, CHAR *ARGV[]){
    PRINTF("CREATING A COLLAGE OF IMAGES!\N");
        STD::VECTOR<STD::STRING> IMAGES;
        UNSIGNED INT M, N;
        INT OPT = 0;
        CHAR* OUTPUT_FILENAME;

        TRY{
            STATIC STRUCT OPTION LONG_OPTIONS[] = {
                {"M", REQUIRED_ARGUMENT, 0,  'M' },
                {"N", REQUIRED_ARGUMENT, 0,  'N' },
                {"OUTPUT", REQUIRED_ARGUMENT, 0,  'O' },
                {"IMAGES", REQUIRED_ARGUMENT, 0,  'I' },
                {0,           0,                 0,  0   }
            };

            INT LONG_INDEX = 0;
            IF ((OPT = GETOPT_LONG(ARGC, ARGV,"M:N:O:I:",
                                   LONG_OPTIONS, &LONG_INDEX )) == -1)
THROW EXCEPTION();
            IF (ARGC<9) THROW EXCEPTION();
            DO{
                SWITCH (OPT) {
                CASE 'M' : M = STOI(OPTARG);
                    BREAK;
                CASE 'N' : N = STOI(OPTARG);
                    BREAK;
                CASE 'O':
                    OUTPUT_FILENAME = OPTARG;
                    BREAK;
                CASE 'I' ://COLLAGE -M 4 -N 6 -I IMAGE1.BMP IMAGE2.BMP
IMAGE3.BMP IMAGE4.BMP -O OUTPUT.BMP
                    FOR (INT I=OPTIND-1; I<ARGC; I++){
                        IF(ARGV[I][0]=='-') BREAK; // END OF ARGUMENTS
```

```
                        IF (IS_BMP_FILE(ARGV[I]) && FEXISTS(ARGV[I])){
                            IMAGES.PUSH_BACK(ARGV[I]);
                        }
                    }
                BREAK;
                DEFAULT:
                    THROW EXCEPTION();
                }
            }WHILE ((OPT = GETOPT_LONG(ARGC, ARGV,"M:N:O:I:",
                        LONG_OPTIONS, &LONG_INDEX )) != -1);

            STD::VECTOR<BMP> BMPS;
            IF (IMAGES.SIZE()<=0) {PRINTF("NO VALID IMAGES
FOUND! :(\N");THROW EXCEPTION();}

            FOR (STD::STRING STR : IMAGES){
                CHAR* FILENAME = STRING2ARRAY(STR);// CHAR* STR //
STD::STRING STR2 ="SDFSDFS";

                TRY {
                    BMPS.PUSH_BACK(BMP(FILENAME));
                } CATCH (EXCEPTION CONST & E) {
                    PRINTF("UNRECOGNIZED %S IMAGE FILE", FILENAME);
                    THROW E;
                }
            }
            BMP COLLAGE = CREATE_COLLAGE(M, N, BMPS);
            PRINTF("SAVING COLLAGE IN FILE %S ...\N", OUTPUT_FILENAME);
            COLLAGE.WRITE(OUTPUT_FILENAME);
            PRINTF("SAVED SUCCESSFULLY!\N");

        }CATCH (EXCEPTION CONST & E) {
            PRINTF("COMMAND OPTIONS ERROR\N");
            PRINT_USAGE_COLLAGE();
            EXIT(EXIT_FAILURE);
        }
}
```

```
VOID ERROR_COMMAND(){
    PRINTF("AVAILABLE COMMANDS ARE:\N1- DRAW TRIANGLE\N");
    PRINT_USAGE_TRI();
    PRINTF("2- DRAW RECTANGLE\N");
    PRINT_USAGE_RECT();
    PRINTF("3- REPAINT RECTANGLE OR TRIANGLE\N");
    PRINT_USAGE_REPAINT();
    PRINTF("4- CREATE COLLAGE OF IMAGES\N");
    PRINT_USAGE_COLLAGE();
    PRINTF("5- GET INFO OF IMAGE AND SHAPES\N");
    PRINT_USAGE_INFO();
    PRINTF("RE-RUN THE PROGRAM WITH AVAILABLE COMMANDS :)\N");
}


VOID INFO_COMMAND(INT ARGC, CHAR *ARGV[]){
    CHAR* FILENAME = ARGV[ARGC-1];
    IF (!IS_BMP_FILE(FILENAME)){
        PRINTF("ERROR: YOU SHOULD SPECIFY AN IMAGE FILE....\N");
        EXIT(EXIT_FAILURE);
    }
    SHAPES SHAPES =
SHAPES::READ4FILE(SHAPES::GET_SHAPES_FILENAME(FILENAME));
    IF (SHAPES.TRIANGLES.SIZE()>0 || SHAPES.RECTANGLES.SIZE()>0){
        SHAPES.TO_STRING();
    }ELSE{
        PRINTF("THE IMAGE DOESNT CONTAIN ANY RECOGNIZABLE
SHAPES.....\N");
    }
}

#ENDIF // COMMANDS_H
```

MAIN.CPP

```
#INCLUDE<COMMANDS.H>
```

```
INT MAIN(INT ARGC, CHAR *ARGV[]) {

    IF (ARGV[1]==NULL) {ERROR_COMMAND();RETURN 0;}
    IF (STRCMP("TRI", ARGV[1])==0){ // THE 1ST COMMAND IS TRI
        DRAW_TRIANGLE_COMMAND(ARGC, ARGV);
    }ELSE IF(STRCMP("RECT", ARGV[1])==0){ // THE 2ND COMMAND IS RECT
        DRAW_RECTANGLE_COMMAND(ARGC, ARGV);
    }ELSE IF(STRCMP("REPAINT", ARGV[1])==0){ // THE 3RD COMMAND IS
REPAINT
        REPAINT_COMMAND(ARGC, ARGV);
    }ELSE IF(STRCMP("COLLAGE", ARGV[1])==0){ // THE 4TH COMMAND IS
COLLAGE
        COLLAGE_COMMAND(ARGC, ARGV);
    }ELSE IF (STRCMP("INFO", ARGV[1])==0){
        INFO_COMMAND(ARGC,ARGV);
    }ELSE{
        ERROR_COMMAND();
    }
    RETURN 0;
}
STRUCTURES.H
#IFNDEF STRUCTURES_H
#DEFINE STRUCTURES_H

#INCLUDE <UNISTD.H>
#INCLUDE <STDIO.H>
#INCLUDE <STDLIB.H>
#INCLUDE <GETOPT.H>
#INCLUDE <STRING.H>
#INCLUDE <IOSTREAM>
#INCLUDE <VECTOR>
#INCLUDE <FSTREAM>
#INCLUDE <IOSTREAM>
#INCLUDE <ALGORITHM>
#INCLUDE <CMATH>

#DEFINE ALPHA 0 // THIS DEACTIVATES ALPHA CHANNEL OF ALL PROCESSED IMAGES
```

```
USING NAMESPACE STD;


STRUCT RECTANGLE{
PUBLIC:
    INT X_START = 0, Y_START = 0, X_END = 0, Y_END =0;
    INT THICKNESS = 1;
    UNSIGNED INT COLOR_R = 255;
    UINT8_T COLOR_G = 255;
    UINT8_T COLOR_B = 255;
    BOOL FILLED = 0;
    UINT8_T FILLED_COLOR_R = 255;
    UINT8_T FILLED_COLOR_G = 255;
    UINT8_T FILLED_COLOR_B = 255;
    UNSIGNED INT WIDTH = 0, HEIGHT = 0;

    DOUBLE GET_AREA(){
        RETURN WIDTH * HEIGHT;
    }



    UNSIGNED INT GET_MAX_X(){
        X_END = X_START + WIDTH;
        RETURN MAX(MAX(X_START, (INT)WIDTH),X_END);
    }

    UNSIGNED INT GET_MAX_Y(){
        Y_END = Y_START + HEIGHT;
        RETURN MAX(MAX(Y_START, (INT)HEIGHT),Y_END);
    }

    STATIC BOOL SERIALIZE(RECTANGLE RECTANGLE, FILE* OUTFILE){
        IF (OUTFILE == NULL){
            FPRINTF(STDERR,"SERIALIZATION ERROR\N");
            EXIT(1);
        }
```

```
        INT FWRITE_OUTPUT = FWRITE(&RECTANGLE.X_START, SIZEOF (INT), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&RECTANGLE.X_END, SIZEOF (INT), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&RECTANGLE.Y_START, SIZEOF (INT), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&RECTANGLE.Y_END, SIZEOF (INT), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&RECTANGLE.WIDTH, SIZEOF (UNSIGNED INT),
1, OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&RECTANGLE.HEIGHT, SIZEOF (UNSIGNED INT),
1, OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&RECTANGLE.FILLED, SIZEOF (BOOL), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&RECTANGLE.THICKNESS, SIZEOF (INT), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&RECTANGLE.COLOR_B, SIZEOF (UINT8_T), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&RECTANGLE.COLOR_R, SIZEOF (UINT8_T), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&RECTANGLE.COLOR_G, SIZEOF (UINT8_T), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&RECTANGLE.FILLED_COLOR_B, SIZEOF
(UINT8_T), 1, OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&RECTANGLE.FILLED_COLOR_G, SIZEOF
(UINT8_T), 1, OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&RECTANGLE.FILLED_COLOR_R, SIZEOF
(UINT8_T), 1, OUTFILE);

        IF(FWRITE_OUTPUT==0)
            PRINTF("SERIALIZATION ERROR\N");
        RETURN FWRITE_OUTPUT;
    }

    STATIC RECTANGLE DESERIALIZE(FILE* INFILE){
        IF (INFILE == NULL){
            FPRINTF(STDERR, "DESERIALIZATION ERROR\N");
            EXIT(1);
```

```
        }
        RECTANGLE RECTANGLE;
        INT FREAD_OUTPUT = FREAD(&RECTANGLE.X_START, SIZEOF (INT), 1,
INFILE);
        FREAD_OUTPUT *= FREAD(&RECTANGLE.X_END, SIZEOF (INT), 1, INFILE);
        FREAD_OUTPUT *= FREAD(&RECTANGLE.Y_START, SIZEOF (INT), 1,
INFILE);
        FREAD_OUTPUT *= FREAD(&RECTANGLE.Y_END, SIZEOF (INT), 1, INFILE);
        FREAD_OUTPUT *= FREAD(&RECTANGLE.WIDTH, SIZEOF (UNSIGNED INT), 1,
INFILE);
        FREAD_OUTPUT *= FREAD(&RECTANGLE.HEIGHT, SIZEOF (UNSIGNED INT),
1, INFILE);
        FREAD_OUTPUT *= FREAD(&RECTANGLE.FILLED, SIZEOF (BOOL), 1,
INFILE);
        FREAD_OUTPUT *= FREAD(&RECTANGLE.THICKNESS, SIZEOF (INT), 1,
INFILE);
        FREAD_OUTPUT *= FREAD(&RECTANGLE.COLOR_B, SIZEOF (UINT8_T), 1,
INFILE);
        FREAD_OUTPUT *= FREAD(&RECTANGLE.COLOR_R, SIZEOF (UINT8_T), 1,
INFILE);
        FREAD_OUTPUT *= FREAD(&RECTANGLE.COLOR_G, SIZEOF (UINT8_T), 1,
INFILE);
        FREAD_OUTPUT *= FREAD(&RECTANGLE.FILLED_COLOR_B, SIZEOF
(UINT8_T), 1, INFILE);
        FREAD_OUTPUT *= FREAD(&RECTANGLE.FILLED_COLOR_G, SIZEOF
(UINT8_T), 1, INFILE);
        FREAD_OUTPUT *= FREAD(&RECTANGLE.FILLED_COLOR_R, SIZEOF
(UINT8_T), 1, INFILE);

        IF(FREAD_OUTPUT!=0){
            RETURN RECTANGLE;
        }ELSE{
            PRINTF("DESERIALIZATION ERROR\N");
            RETURN RECTANGLE();
        }
    }

    // IF (RECT1==RECT2)
```

```
    FRIEND BOOL OPERATOR== (CONST RECTANGLE RECT1, CONST RECTANGLE
RECT2){
        RETURN (RECT1.X_START==RECT2.X_START &&
                RECT1.X_END==RECT2.X_END &&
                RECT1.Y_START==RECT2.Y_START &&
                RECT1.Y_END==RECT2.Y_END &&
                RECT1.WIDTH==RECT2.WIDTH &&
                RECT1.HEIGHT==RECT2.HEIGHT &&
                RECT1.FILLED==RECT2.FILLED &&
                RECT1.COLOR_B==RECT2.COLOR_B &&
                RECT1.COLOR_G==RECT2.COLOR_G &&
                RECT1.COLOR_R==RECT2.COLOR_R &&
                RECT1.FILLED_COLOR_B==RECT2.FILLED_COLOR_B &&
                RECT1.FILLED_COLOR_G==RECT2.FILLED_COLOR_G &&
                RECT1.FILLED_COLOR_R==RECT2.FILLED_COLOR_R &&
                RECT1.THICKNESS == RECT2.THICKNESS
                );
    }

    CHAR* TO_STRING(){
        CHAR* ARR = (CHAR*)MALLOC(2000*SIZEOF (CHAR));
        SPRINTF(ARR,
"==========================================\NRECTANGLE START (%D, %D),
END (%D, %D), WIDTH = %D, HEIGHT = %D\NBORDER COLOR = {%D, %D, %D}, FILL
COLOR = {%D, %D, %D}\NTHICKNESS = %D , AREA
= %F\N==================================================\N"
               ,THIS->X_START, THIS->Y_START, THIS->X_END, THIS->Y_END,
WIDTH, HEIGHT, COLOR_R, COLOR_G, COLOR_B,
                FILLED_COLOR_R, FILLED_COLOR_G, FILLED_COLOR_B,
THICKNESS, GET_AREA());
        RETURN ARR;
    }
};

STRUCT TRIANGLE{
PUBLIC:
    INT X2 = 0, Y2 = 0, X1 = 0, Y1 =0, X3 = 0, Y3 =0;
    INT THICKNESS = 1;
```

```
    UINT8_T COLOR_R = 0;
    UINT8_T COLOR_G = 0;
    UINT8_T COLOR_B = 0;
    BOOL FILLED = 0;
    UINT8_T FILLED_COLOR_R = 0;
    UINT8_T FILLED_COLOR_G = 0;
    UINT8_T FILLED_COLOR_B = 0;
    UNSIGNED INT IMAGE_WIDTH = 0, IMAGE_HEIGHT = 0;


    DOUBLE GET_AREA(){
        DOUBLE A = SQRT((DOUBLE)(X2-X1)*(X2-X1)+(Y2-Y1)*(Y2-Y1));
        DOUBLE B = SQRT((DOUBLE)(X2-X3)*(X2-X3)+(Y2-Y3)*(Y2-Y3));
        DOUBLE C = SQRT((DOUBLE)(X1-X3)*(X1-X3)+(Y1-Y3)*(Y1-Y3));
        DOUBLE S = (A+B+C) / 2;
        RETURN SQRT(S*(S-A)*(S-B)*(S-C));
    }


    UNSIGNED INT GET_MAX_X(){
        RETURN MAX(MAX(X1,X2),X3);
    }

    UNSIGNED INT GET_MAX_Y(){
        RETURN MAX(MAX(Y1, Y2),Y3);
    }

    STATIC BOOL SERIALIZE(TRIANGLE TRIANGLE, FILE* OUTFILE){
        IF (OUTFILE == NULL){
            FPRINTF(STDERR,"SERIALIZATION ERROR\N");
            EXIT(1);
        }
        INT FWRITE_OUTPUT = FWRITE(&TRIANGLE.X1, SIZEOF (INT), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&TRIANGLE.X2, SIZEOF (INT), 1, OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&TRIANGLE.X3, SIZEOF (INT), 1, OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&TRIANGLE.Y1, SIZEOF (INT), 1, OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&TRIANGLE.Y2, SIZEOF (INT), 1, OUTFILE);
```

```
        FWRITE_OUTPUT *= FWRITE(&TRIANGLE.Y3, SIZEOF (INT), 1, OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&TRIANGLE.FILLED, SIZEOF (BOOL), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&TRIANGLE.THICKNESS, SIZEOF (INT), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&TRIANGLE.COLOR_B, SIZEOF (UINT8_T), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&TRIANGLE.COLOR_R, SIZEOF (UINT8_T), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&TRIANGLE.COLOR_G, SIZEOF (UINT8_T), 1,
OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&TRIANGLE.FILLED_COLOR_B, SIZEOF
(UINT8_T), 1, OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&TRIANGLE.FILLED_COLOR_G, SIZEOF
(UINT8_T), 1, OUTFILE);
        FWRITE_OUTPUT *= FWRITE(&TRIANGLE.FILLED_COLOR_R, SIZEOF
(UINT8_T), 1, OUTFILE);

        IF(FWRITE_OUTPUT==0)
            PRINTF("SERIALIZATION ERROR\N");

        RETURN FWRITE_OUTPUT;
    }

    STATIC TRIANGLE DESERIALIZE(FILE* INFILE){
        IF (INFILE == NULL){
            FPRINTF(STDERR, "DESERIALIZATION ERROR\N");
            EXIT(1);
        }
        TRIANGLE TRIANGLE;
        INT FREAD_OUTPUT = FREAD(&TRIANGLE.X1, SIZEOF (INT), 1, INFILE);
        FREAD_OUTPUT *= FREAD(&TRIANGLE.X2, SIZEOF (INT), 1, INFILE);
        FREAD_OUTPUT *= FREAD(&TRIANGLE.X3, SIZEOF (INT), 1, INFILE);
        FREAD_OUTPUT *= FREAD(&TRIANGLE.Y1, SIZEOF (INT), 1, INFILE);
        FREAD_OUTPUT *= FREAD(&TRIANGLE.Y2, SIZEOF (INT), 1, INFILE);
        FREAD_OUTPUT *= FREAD(&TRIANGLE.Y3, SIZEOF (INT), 1, INFILE);
        FREAD_OUTPUT *= FREAD(&TRIANGLE.FILLED, SIZEOF (BOOL), 1,
INFILE);
```

```
        FREAD_OUTPUT *= FREAD(&TRIANGLE.THICKNESS, SIZEOF (INT), 1,
INFILE);
        FREAD_OUTPUT *= FREAD(&TRIANGLE.COLOR_B, SIZEOF (UINT8_T), 1,
INFILE);
        FREAD_OUTPUT *= FREAD(&TRIANGLE.COLOR_R, SIZEOF (UINT8_T), 1,
INFILE);
        FREAD_OUTPUT *= FREAD(&TRIANGLE.COLOR_G, SIZEOF (UINT8_T), 1,
INFILE);
        FREAD_OUTPUT *= FREAD(&TRIANGLE.FILLED_COLOR_B, SIZEOF (UINT8_T),
1, INFILE);
        FREAD_OUTPUT *= FREAD(&TRIANGLE.FILLED_COLOR_G, SIZEOF (UINT8_T),
1, INFILE);
        FREAD_OUTPUT *= FREAD(&TRIANGLE.FILLED_COLOR_R, SIZEOF (UINT8_T),
1, INFILE);

        IF(FREAD_OUTPUT!=0){
            RETURN TRIANGLE;
        }ELSE{
            PRINTF("DESERIALIZATION ERROR\N");
            RETURN TRIANGLE();
        }
    }

    FRIEND BOOL OPERATOR== (CONST TRIANGLE TRI1, CONST TRIANGLE TRI2){
        IF (    TRI1.X1==TRI2.X1 &&
                TRI1.X2==TRI2.X2 &&
                TRI1.X3==TRI2.X3 &&
                TRI1.Y1==TRI2.Y1 &&
                TRI1.Y2==TRI2.Y2 &&
                TRI1.Y3==TRI2.Y3 &&
                TRI1.FILLED==TRI2.FILLED &&
                TRI1.COLOR_B==TRI2.COLOR_B &&
                TRI1.COLOR_G==TRI2.COLOR_G &&
                TRI1.COLOR_R==TRI2.COLOR_R &&
                TRI1.FILLED_COLOR_B==TRI2.FILLED_COLOR_B &&
                TRI1.FILLED_COLOR_G==TRI2.FILLED_COLOR_G &&
                TRI1.FILLED_COLOR_R==TRI2.FILLED_COLOR_R &&
                TRI1.THICKNESS == TRI2.THICKNESS
```

```
                ){
            RETURN TRUE;
        }
        RETURN FALSE;
    }


    CHAR* TO_STRING(){
        STD::STRING STR1;
        CHAR* ARR = (CHAR*)MALLOC(2000*SIZEOF(CHAR));
        SPRINTF(ARR,
"=============================================\NTRIANGLE A (%D, %D), B
(%D, %D), C = (%D, %D)\NBORDER COLOR = {%D, %D, %D}, FILL COLOR =
{%D, %D, %D}\NTHICKNESS = %D , AREA
= %F\N=====================================================\N"
               ,THIS->X1, THIS->Y1, THIS->X2, THIS->Y2,THIS->X3,
THIS->Y3, COLOR_R, COLOR_G, COLOR_B,
                FILLED_COLOR_R, FILLED_COLOR_G, FILLED_COLOR_B,
THICKNESS, GET_AREA());
        RETURN ARR;
    }

};

STRUCT SHAPES{
    VECTOR<RECTANGLE> RECTANGLES;
    VECTOR<TRIANGLE> TRIANGLES;
    CHAR* FILENAME;// ""
    SHAPES(CHAR* FILENAME){
        THIS->FILENAME = FILENAME;
    }

    SHAPES(){
        CHAR* NOTHING = (CHAR*)MALLOC(SIZEOF (CHAR));
        STRCAT(NOTHING, "");
        THIS->FILENAME = NOTHING;
    }
```

```
STATIC CHAR* GET_SHAPES_FILENAME(CHAR* FILENAME){
    CHAR* STR = (CHAR*)MALLOC(SIZEOF (CHAR));
    RETURN STRCAT(STRCPY(STR,FILENAME), ".DATA");
    // FILENAME = "FILE.BMP";
    // STRCAT(FILENAME,".DATA"); // FILE.BMP.DATA
}

VOID ADD_RECTANGLE(RECTANGLE RECT){
    IF (RECTANGLES.SIZE()==0){
        RECTANGLES.PUSH_BACK(RECT);
    }ELSE{
        FOR (RECTANGLE REX :RECTANGLES)
            IF(REX==RECT)
                RETURN;
        RECTANGLES.PUSH_BACK(RECT);
    }
}

VOID ADD_TRIANGLE(TRIANGLE TRI){
    IF (TRIANGLES.SIZE()==0){
        TRIANGLES.PUSH_BACK(TRI);
    }ELSE{
        FOR (TRIANGLE TRX : TRIANGLES)
            IF(TRX==TRI)
                RETURN;
        TRIANGLES.PUSH_BACK(TRI);
    }
}

RECTANGLE GET_LARGEST_RECTANGLE(INT* INDEX){
    RECTANGLE LARGEST_RECT;
    INT COUNTER=0;
    FOR (AUTO R:RECTANGLES){
        IF (R.GET_AREA()>LARGEST_RECT.GET_AREA()){
            LARGEST_RECT = R;
            (*INDEX) = COUNTER;
        }
        (COUNTER)++;
```

```
    }
    RETURN LARGEST_RECT;
}

TRIANGLE GET_LARGEST_TRIANGLE(INT* INDEX){
    TRIANGLE LARGEST_TRIANGLE;
    INT COUNTER=0;
    FOR (AUTO T:TRIANGLES){
        IF (T.GET_AREA()>LARGEST_TRIANGLE.GET_AREA()){
            LARGEST_TRIANGLE = T;
            (*INDEX) = COUNTER;
        }
        (COUNTER)++;
    }
    RETURN LARGEST_TRIANGLE;
}


STATIC BOOL SERIALIZE(SHAPES SHAPES, FILE* OUTFILE){
    IF (OUTFILE == NULL){
        FPRINTF(STDERR,"SERIALIZATION ERROR\N");
        EXIT(1);
    }
    INT TRIS_SIZE = SHAPES.TRIANGLES.SIZE();
    INT RECT_SIZE = SHAPES.RECTANGLES.SIZE();
    INT FWRITE_OUTPUT = FWRITE(&TRIS_SIZE, SIZEOF (INT), 1, OUTFILE);
    FOR (TRIANGLE TRI : SHAPES.TRIANGLES){
        FWRITE_OUTPUT *= TRIANGLE::SERIALIZE(TRI, OUTFILE);
    }
    FWRITE_OUTPUT *= FWRITE(&RECT_SIZE, SIZEOF (INT), 1, OUTFILE);
    FOR (RECTANGLE RECT : SHAPES.RECTANGLES){
        FWRITE_OUTPUT *= RECTANGLE::SERIALIZE(RECT, OUTFILE);
    }

    IF(FWRITE_OUTPUT==0)
        PRINTF("SERIALIZATION ERROR\N");

    RETURN FWRITE_OUTPUT;
```

```
}

STATIC SHAPES DESERIALIZE(FILE* INFILE){
    IF (INFILE == NULL){
        FPRINTF(STDERR, "DESERIALIZATION ERROR\N");
        EXIT(1);
    }
    SHAPES SHAPES;
    INT TRIS_SIZE;
    INT FREAD_OUTPUT = FREAD(&TRIS_SIZE, SIZEOF (INT), 1, INFILE);
    FOR (INT I = 0; I< TRIS_SIZE; I++){
        TRIANGLE T = TRIANGLE::DESERIALIZE(INFILE);
        IF (T.GET_AREA() !=0){
            SHAPES.ADD_TRIANGLE(T);
            FREAD_OUTPUT*=1;
        }ELSE{
            FREAD_OUTPUT*=0;
        }
    }
    INT RECT_SIZE;
    FREAD_OUTPUT *= FREAD(&RECT_SIZE, SIZEOF (INT), 1, INFILE);
    FOR (INT I = 0; I< RECT_SIZE; I++){
        RECTANGLE R = RECTANGLE::DESERIALIZE(INFILE);
        IF (R.GET_AREA()!=0){
            SHAPES.ADD_RECTANGLE(R);
            FREAD_OUTPUT*=1;
        }ELSE{
            FREAD_OUTPUT*=0;
        }
    }

    IF(FREAD_OUTPUT!=0){
        RETURN SHAPES;
    }ELSE{
        PRINTF("DESERIALIZATION ERROR\N");
        RETURN SHAPES();
    }
}
```

```
INT WRITE2FILE(CHAR* FILENAME){
    FILE* OUTFILE = FOPEN(FILENAME, "W");
    IF (OUTFILE == NULL){
        FPRINTF(STDERR, "\NERROR OPEN FILE\N");
        EXIT(1);
    }
    STRUCT SHAPES SHAPES = *THIS;
    PRINTF("STORING SHAPES TO DATA FILE....\N");
    INT FWRITE_OUTPUT = SHAPES::SERIALIZE(SHAPES, OUTFILE);
    IF(FWRITE_OUTPUT!=0){
        PRINTF("DATA TO FILE %S SUCCESSFULLY STORED!\N", FILENAME);
    }ELSE{
        PRINTF("ERROR WRITING TO FILE !\N");
    }

    FCLOSE(OUTFILE);
    RETURN FWRITE_OUTPUT;
}

STATIC SHAPES READ4FILE(CHAR* FILENAME){
    FILE* INFILE = FOPEN(FILENAME, "R");
    IF (INFILE == NULL){
        FPRINTF(STDERR, "\NERROR: CAN'T OPEN FILE %S\N",FILENAME);
        EXIT(1);
    }
    STRUCT SHAPES SHAPES(FILENAME);
    SHAPES = SHAPES::DESERIALIZE(INFILE);
    INT FREAD_OUTPUT = 0;
    IF (STRCMP(SHAPES.FILENAME,"")==0){
        FREAD_OUTPUT = 1;
    }ELSE{
        PRINTF("DESERIALIZATION ERROR\N");
    }

    IF(FREAD_OUTPUT!=0){
        PRINTF("DATA FROM FILE %S SUCCESSFULLY READ!\N", FILENAME);
        FCLOSE(INFILE);
```

```
            RETURN SHAPES;
        }ELSE{
            PRINTF("ERROR READING FILE !\N");
            // ASSIGN "" TO FILENAME OF SHAPES
            RETURN SHAPES();
        }
    }

    VOID TO_STRING(){
        PRINTF("WE HAVE %D SHAPES: %D RECTANGLES AND %D TRIANGLES\N",
INT(THIS->TRIANGLES.SIZE()+THIS->RECTANGLES.SIZE()),
               INT(RECTANGLES.SIZE()), INT(TRIANGLES.SIZE()));
        FOR(AUTO TRI: TRIANGLES){
            PRINTF("%S",TRI.TO_STRING());
        }

        FOR(AUTO RECT: RECTANGLES){
            PRINTF("%S",RECT.TO_STRING());
        }
    }

};




#ENDIF


UTILS.H
#IFNDEF UTILS_H
#DEFINE UTILS_H

#IFNDEF STRUCTURES_H
    #INCLUDE<STRUCTURES.H>
#ENDIF
```

```
// CHECKS IF FILE EXISTS OR NOT
BOOL FEXISTS(CONST CHAR *FILENAME){
    STD::IFSTREAM IFILE(FILENAME);
    RETURN !(!IFILE);
}

// CONVERTS THE STRING OR CONST ARRAY TO DYNAMIC ARRAY AND INITIALIZES IT
WITH THE CONTENT OF INPUT STRING
CHAR* STRING2ARRAY(STD::STRING STR){
    CHAR* ARR = (CHAR*) MALLOC(STR.LENGTH()*SIZEOF (CHAR));
    INT I=0;
    FOR(; I<(INT)STR.LENGTH();I++){
        ARR[I] = STR[I];
    }
    ARR[I]='\0';
    RETURN ARR;
}

BOOL IS_BMP_FILE(CHAR* FILENAME){
    IF (FILENAME != NULL && STRCMP(FILENAME, "")!=0){
        STD::STRING STR = FILENAME;
        IF (STR.SIZE()>4){// "1.BMP"
            RETURN STRCMP(STRING2ARRAY(STR.SUBSTR(STR.SIZE()-4)),".BMP")
== 0;
        }
    }
    RETURN FALSE;
}

// PRINT HELP OF EACH COMMAND
VOID PRINT_USAGE_TRI() {
    PRINTF("DRAWING TRIANGLE COMMAND USAGE:\NTRI -A(--P0) X Y -B(--P1) X
Y -C(--P2) X Y "
           "-T(--THICK) VALUE -C(--COLOR) R G B [-F(--FILLED) -L(--
FILLCOLOR) R G B] -I(--INPUT) FILENAME.BMP [WIDTH HEIGHT]\N");
}
```

```
VOID PRINT_USAGE_RECT() {
    PRINTF("DRAWING RECTANGLE COMMAND USAGE:\NRECT -S(--START) X Y -W(--
WIDTH) WIDTH -H(--HEIGHT) HEIGHT "
           "-T(--THICK) VALUE -C(--COLOR) R G B [-F(--FILLED) -L(--
FILLCOLOR) R G B] -I(--INPUT) FILENAME.BMP [WIDTH HEIGHT]\N");
}

VOID PRINT_USAGE_REPAINT() {
    PRINTF("REPAINTING SHAPE COMMAND USAGE:\NREPAINT -S(HAPE) [RECT|TRI]
--(C)OLOR R G B -I(NPUT) FILENAME.BMP\N");
}


VOID PRINT_USAGE_COLLAGE() {
    PRINTF("CREATING A COLLAGE OF IMAGES COMMAND USAGE:\NCOLLAGE -M(--M)
NUMBER -N(--N) NUMBER -O(--OUTPUT) OUTPUT.BMP -I(--INPUT) IMAGE1.BMP
IMAGE2.BMP IMAGE3.BMP ........\N");
}

VOID PRINT_USAGE_INFO(){
    PRINTF("GETTING INFO OF IMAGE AND SHAPES COMMAND USAGE:\NINFO
IMAGE.BMP\N");
}


#ENDIF
```

ПРИЛОЖЕНИЕ Б

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ