



МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное
учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт	ИВТИ
Кафедра	ВМСС

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(бакалаврская работа)

Направление 09.03.01 Информатика и вычислительная техника
(код и наименование)

Направленность (профиль) Вычислительные машины, комплексы,
системы и сети

Форма обучения очная
(очная/очно-заочная/заочная)

Тема: Разработка программных инструментов для работы с
расположением объектов на плоскости

Студент	А-12-19	Винокуров Р.Н.
	группа	подпись фамилия и инициалы

Научный руководитель	К.Т.Н.	доцент	Гольцов А.Г.
	уч. степень	должность	подпись фамилия и инициалы

«Работа допущена к защите»

Зав. кафедрой	К.Т.Н.	доцент	Вишняков С.В.
	уч. степень	звание	подпись фамилия и инициалы

Дата _____

Москва, 2023

АННОТАЦИЯ

В данной выпускной квалификационной работе выполнены проектирование и разработка программного решения задачи ввода, хранения, и отображения объектов на двумерном изображении. Результатом выполненной работы является графический формат, позволяющий совмещать векторную и растровую графику, а также инструменты для работы с файлами данного формата для ОС Windows. Набор инструментов состоит из двух приложений, написанных на языке C++ с использованием IDE Microsoft Visual Studio: консольной программы-редактора и оконного приложения, демонстрирующего возможности разработанного формата. Для упрощения разработки данного программного комплекса, совместно используемые двумя приложениями функции, были вынесены в отдельную DLL-библиотеку, подключаемую к обоим приложениям. Работа состоит из 3 разделов, 120 страниц, 35 рисунков, 3 таблиц и 2 приложений.

СОДЕРЖАНИЕ

Определения.....	4
Обозначения и сокращения	5
Введение.....	6
1. ПОСТАНОВКА ЗАДАЧИ	8
1.1. Описание предметной области	8
1.2. Функции программы	8
1.3. Вид приложения и среда разработки	9
2. РАЗРАБОТКА ПРИЛОЖЕНИЯ	13
2.1. Разработка формата хранения векторных данных.....	13
2.2. Разработка библиотеки для работы с векторной графикой	19
2.3. Разработка редактора файлов векторной графики.....	27
2.4. Разработка демонстрационного приложения.....	34
3. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ	53
3.1. Реализация программы-редактора	53
3.2. Реализация демонстрационной программы	54
3.3. Тестирование программы-редактора	55
3.4. Тестирование демонстрационной программы	65
ЗАКЛЮЧЕНИЕ.....	77
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	79
Приложение А. Задание на выпускную квалификационную работу ..	81
Приложение Б. Листинг программы.....	84

ОПРЕДЕЛЕНИЯ

Графический примитив – простейшая геометрическая фигура (прямоугольник, окружность, многоугольник и т.д.), составляющая цельный элемент векторного рисунка и хранящаяся в памяти в виде отдельной информационной структуры, отображаемой на экране дисплея.

Прицеливание – проверка попадания координат клика мыши в область вокруг или внутри некоторой фигуры.

Сглаживание – технология, используемая в компьютерной графике для устранения или минимизации эффекта ступенчатости на границах объектов и линий.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ОС — операционная система

ВМ — вычислительная машина

BMP — Bitmap Picture

DLL — Dynamic Link Library

ASCII — American Standard Code for Information Interchange

RGB — Red, Green, Blue

API — Application Programming Interface

GDI — Graphics Device Interface

APU — Accelerated Processing Unit

IDE — Integrated Development Environment

ВВЕДЕНИЕ

Сегодня компьютерная графика окружает нас повсюду: от графического интерфейса современных операционных систем до технологий виртуальной реальности. Несмотря на все разнообразие компьютерной графики, которая нас окружает, можно выделить два основных подхода к формату представления графических данных в вычислительной технике: векторный и растровый, каждый из которых обладает своими преимуществами и недостатками. Основные преимущества растровой графики — это возможность представления в таком формате реальных изображений с большой точностью и детализацией, а минусы — невозможность масштабирования отображаемых данных без потери качества и большой объем занимаемого дискового пространства. Ситуация в случае с векторной графикой ровно противоположная — она плохо подходит для представления реальных фотографий, зато отлично масштабируется без потери качества и занимает малый объем памяти. Совмещение этих двух форматов представления графических данных является перспективной областью исследования, поскольку позволит использовать преимущества каждого из них.

В рамках выполнения выпускной квалификационной работы необходимо решить следующие задачи:

- а) Разработать формат файлов, хранящий примитивы векторного слоя графики, накладываемого на растровую подложку.
- б) Разработать DLL-библиотеку, содержащую набор методов для чтения, отображения на экране, и записи данных разработанного формата.
- в) Разработать консольный редактор файлов разработанного формата, позволяющий создавать и изменять данные слоя векторной графики и использующий методы DLL-библиотеки из предыдущего пункта.

г) Разработать оконное графическое приложение, отображающее слой векторной графики из файла разработанного формата на растровой подложке из BMP-файла и использующее методы созданной ранее DLL-библиотеки.

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Описание предметной области

Предметной областью разрабатываемого приложения является: ввод, хранение, и отображение объектов на двумерном изображении (к примеру, отображение границ земельных участков на карте или расположение точек и зон на медицинском изображении). Таким образом, необходимо разработать редактор для подготовки и изменения таких данных, формат хранения и библиотеку доступа для использования в программах, где требуется их отображение, а также тестовое приложение, демонстрирующее возможности разработанного формата.

1.2. Функции программы

Подробно перечислим функции каждой из программ разрабатываемого набора. Функции программы-редактора:

- а) Создание файлов разработанного формата
- б) Добавление примитивов в файл разработанного формата
- в) Редактирование примитивов файла разработанного формата
- г) Удаление примитивов из файла разработанного формата
- д) Вывод на экран данных файла разработанного формата

Для удобства пользователя при редактировании примитивов файла разработанного формата реализуется вывод всех примитивов, содержащихся в файле на экран, и запрашивается ввод номера редактируемого примитива пользователем.

Теперь перечислим функции демонстрационной программы:

- а) Отображение слоя векторной графики поверх растровой подложки на экране
- б) Включение/отключение сглаживания графических данных векторного слоя
- в) Масштабирование отображаемых данных
- г) Вывод данных о примитиве при попадании координат клика мыши в область прицеливания какой-либо из фигур

1.3. Вид приложения и среда разработки

Поскольку функции программы-редактора — добавление и редактирование данных в файлах разработанного формата, в ней реализовано изменение данных в файле векторного формата на основании вводимых с клавиатуры пользователем данных. Программу, имплементирующую подобный набор функций, удобно реализовать в виде консольного приложения, поэтому выбор пал именно на данную форму приложений. При запуске приложение должно выводить пользователю список пунктов консольного меню, в ходе работы с которым выбирается производимое с файлом действие и вводятся все необходимые данные для выполнения этого действия. Выбор пунктов меню должен осуществляться нажатием тех или иных клавиш пользователем. Пример того как должен выглядеть диалог пользователя с разработанным приложением приведен на рис. 1.



```

-----Выбор действия-----
1 - создать новый файл
2 - редактировать уже существующий файл
3 - завершить работу программы
4 - удалить из существующего файла структуру(-ы)
Выберите действие:
Введите название создаваемого файла:
newfile
-----Добавление примитивов-----
Клавиши и их соответствие примитивам
A - установить параметры базиса
B - точка
C - линия
D - ломаная
E - окружность
F - многоугольник
G - в главное меню
Выберите опцию:
Введите refwidth:512
Введите refheight:256
Введите basex:300
Введите basey:100
-----Добавление примитивов-----
Клавиши и их соответствие примитивам
A - установить параметры базиса
B - точка
C - линия
D - ломаная
E - окружность
F - многоугольник
G - в главное меню
Выберите опцию:
Введите координату точки по горизонтали:10
Введите координату точки по вертикали:10
Введите размер области прицеливания:5
Введите интенсивность цветового канала R:0
Введите интенсивность цветового канала G:0
Введите интенсивность цветового канала B:0
-----Добавление примитивов-----
Клавиши и их соответствие примитивам
A - установить параметры базиса
B - точка
C - линия
D - ломаная
E - окружность
F - многоугольник
G - в главное меню
Выберите опцию:
-----Выбор действия-----
1 - создать новый файл
2 - редактировать уже существующий файл
3 - завершить работу программы
4 - удалить из существующего файла структуру(-ы)
Выберите действие:

```

10

В представленном на рис. 1 диалоге производятся следующие действия: создание нового файла с именем “newfile” и добавление в него базиса, а также примитива точки.

Демонстрационное приложение должно реализовывать отображение графических данных на экране, поэтому наиболее удобной формой его реализации является оконное приложение. Для отрисовки графических данных на экране была выбрана библиотека GDI+ Windows API. Данный выбор можно обосновать простотой использования GDI+ для реализации несложной компьютерной графики (в отличие от альтернатив), а также необходимостью реализации в приложении сглаживания, которое поддерживается методами данной библиотеки. Поскольку интерфейс данного приложения является оконным, при запуске оно сразу открывает файлы растровой и векторной графики (чтобы приложение получило их при запуске, можно реализовать передачу аргументов к примеру, через интерфейс командной строки) и отображает слой векторной графики поверх слоя растровой на экране компьютера пользователя. Остальные функции приложения реализуются в ходе тех или иных воздействий от пользователя (клик мыши – прицеливание, нажатие клавиш – изменение масштаба, включение/выключение сглаживания), обрабатываемых основным окном приложения. Пример первоначального (т.е. до поступления каких-либо пользовательских воздействий) внешнего вида окна приложения приведен на рис. 2.

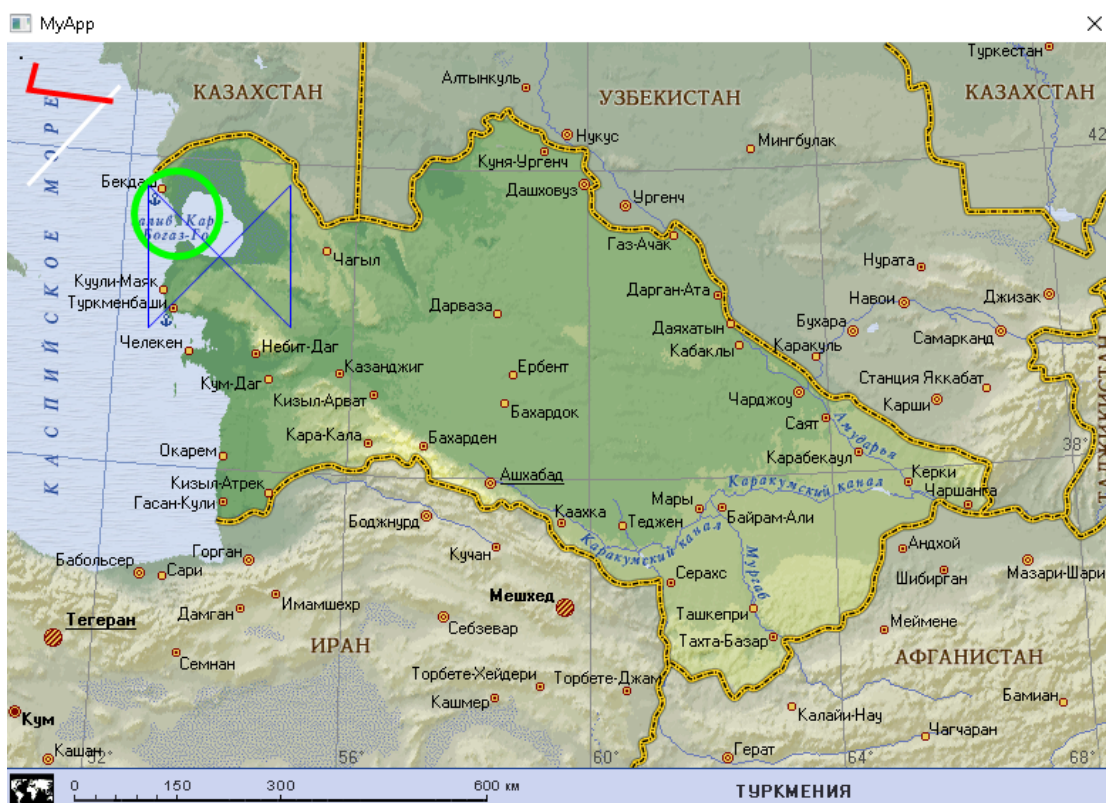


Рис. 2 Первоначальный вид окна демонстрационного приложения

Отдельным модулем является DLL-библиотека, содержащая методы доступа к графическим данным файла, содержащего векторную графику. Методы данной библиотеки использованы при разработке обоих приложений и позволяют читать, записывать, и отрисовывать на экране данные разработанного формата.

Для реализации описанного выше набора программ и DLL-библиотеки использовался язык программирования Си++ в среде Microsoft Visual Studio 2010. Языком программирования для разработки был выбран язык Си++ в силу его высокой производительности[1] и возможности низкоуровневой работы с памятью, что важно для реализации чтения и записи файлов. Оптимальной средой разработки для ОС Windows, поддерживающей язык Си++ и большой набор библиотек для программирования в данной операционной системе является Microsoft Visual Studio, из-за чего данная среда была выбрана как основная среда программирования.

2. РАЗРАБОТКА ПРИЛОЖЕНИЯ

В данном разделе подробно описаны разработанные способы решения задач, приведены блок-схемы алгоритмов, а также обоснован выбор используемых способов решения.

2.1. Разработка формата хранения векторных данных

Опишем структуру разработанного формата хранения векторных примитивов. Файл разработанного формата состоит из двух частей: базиса и структур векторных примитивов. Базис содержит данные для пересчета координат векторных примитивов, хранящихся в файле векторного формата в координаты примитивов на растровом полотне при отображении. Отметим, что для отображения векторных данных на экране в файле обязательно наличие базиса, поскольку базис связывает координаты векторных примитивов файла векторного слоя и координаты на растровом полотне.

Приведем более подробное описание полей структуры базиса:

а) Подпись (размер - 4 байта) = FE DE AE BE в hex-формате. Уникальный набор байт, идентифицирующий файл разработанного формата.

б) Коэффициенты масштабирования *refwidth* и *refheight* (размер каждого из полей в файле – 4 байта). Два данных коэффициента описывают размер в пикселях хранящегося в файле слоя векторной графики для масштабирования в растр.

в) Координаты точки начала отсчета на растровом полотне *basex* и *basey* (размер каждого из полей в файле – 4 байта). Два данных поля описывают соответствие координаты (0,0) в векторном файле координате на растровом полотне в пикселях растра.

Более точно смысл полей базиса отражают нижеприведенные соотношения.

Для горизонтальных координат отображения:

$$x = basex + \frac{width}{refwidth} * x_0 \quad (1)$$

Для вертикальных координат отображения:

$$y = basey + \frac{height}{refheight} * y_0 \quad (2)$$

Формулы (1) и (2) отражают связь между координатами по каждой из осей x и y на растровом полотне, полями структуры базиса $basex, basey, refwidth, refheight$, шириной и высотой растровой подложки $width$ и $height$, и координатами точки x_0, y_0 внутри векторного примитива.

Важно отметить, что при нецелом отношении размеров изображения и коэффициентов масштабирования конечный результат вычисления округляется до целого значения в **меньшую** сторону.

Таким образом, суммарный размер базисной структуры файла разработанного формата $4+4+4+4+4=20$ байт. Сразу после данной структуры в файле следуют данные о каждом из графических примитивов векторного слоя. Каждая информационная структура описывает один из примитивов векторного слоя и делится на 2 части: идентификатор (подпись) примитива и непосредственные числовые данные о геометрических размерах фигуры, а также ее цвете, толщине линии для ее рисования, и области прицеливания (если нужно). Общее представление об информационных структурах векторных примитивов можно получить на основании данных,

представленных в табл. 1. Строки под заголовком «данные об области прицеливания» принимают только 2 значения («Да» / «Нет»), отражающих, присутствует ли в данной структуре поле, описывающее размеры области прицеливания для примитива.

Таблица 1

Некоторые данные об информационных структурах векторных
примитивов

Название примитива	Идентификатор (в символьном виде)	Идентификатор (в hex-форме)	Размер структуры, в байтах	Данные об области прицеливания
Точка	POIN	50 4F 49 4E	16	Да
Линия	LINE	4C 49 4E 45	25	Да
Ломаная	BRLI	42 52 4C 49	89	Нет
Окружность	CIRC	43 49 52 43	21	Да
Многоугольник	POLY	50 4F 4C 59	89	Нет

Остановимся подробнее на каждом примитиве и опишем поля, используемые в его информационной структуре. Первый из примитивов, точка, содержит следующий набор полей:

а) Идентификатор информационной структуры точки, 4 символа POIN в таблице ASCII. Размер поля – 4 байта.

б) Координаты точки по горизонтали и вертикали. Два целых неотрицательных числа. Размер каждого из полей – 4 байта.

в) Размер области прицеливания точки. Целое неотрицательное число. Размер поля – 1 байт.

г) Цвет точки в модели 24-битной модели RGB, 3 однобайтных целых неотрицательных числа.

Для краткости в дальнейших описаниях опустим поле идентификатора и сосредоточимся на непосредственно информационных полях.

Второй из используемых примитивов, линия, содержит следующий набор полей:

а) Координаты точки начала линии по горизонтали и вертикали. Два целых неотрицательных числа. Размер каждого из полей – 4 байта.

б) Координаты точки конца линии по горизонтали и вертикали. Два целых неотрицательных числа. Размер каждого из полей – 4 байта.

в) Толщина линии. Целое неотрицательное число. Размер поля – 1 байт.

г) Размер области прицеливания линии. Целое неотрицательное число. Размер поля – 1 байт.

д) Цвет точки в 24-битной модели RGB, 3 однобайтных целых неотрицательных числа.

Третий из используемых примитивов, ломаная, содержит следующий набор полей:

а) Толщина ломаной. Целое неотрицательное число. Размер поля – 1 байт.

б) Цвет точки в 24-битной модели RGB, 3 однобайтных целых неотрицательных числа.

в) Количество точек ломаной. Целое неотрицательное число. Размер поля – 1 байт.

г) Массив точек ломаной. Массив 20 целых неотрицательных 4-байтных чисел. Размер поля – 80 байт.

Четвертый примитив, окружность, содержит следующий набор полей:

а) Координаты центра окружности по вертикали и горизонтали. Два целых неотрицательных числа. Размер каждого из полей – 4 байта.

б) Радиус окружности. Целое неотрицательное число. Размер поля – 4 байта.

в) Толщина линии окружности. Целое неотрицательное число. Размер поля – 1 байт.

г) Размер области прицеливания окружности. Целое неотрицательное число. Размер поля – 1 байт.

д) Цвет точки в 24-битной модели RGB, 3 однобайтных целых неотрицательных числа.

Пятый примитив, многоугольник, содержит аналогичный ломаной набор полей, отличие заключается исключительно в наборе символов идентификатора данной информационной структуры.

Перед тем как завершить описание разработанного формата, необходимо сделать еще одно важное замечание: по поводу порядка следования байтов в представлении многобайтных полей файла. Поскольку разработка велась для ОС Windows, условимся, что в разработанном формате используется порядок следования байтов little-endian, характерный для архитектуры x86[2].

Пример структуры файла разработанного формата приведен на рис. 3.

Байты 0-19	ID=FEDEAEBEh (4 байта)		refwidth (4 байта)		refheight (4 байта)		basex (4 байта)				basey (4 байта)				
Байты 20-39	ID="POIN" (4 байта)		x (4 байта)		y (4 байта)		aim_area (1 байт)	red (1 байт)	green (1 байт)	blue (1 байт)	ID="LINE" (4 байта)				
Байты 40-59	x (4 байта)		y (4 байта)		x1 (4 байта)		y1 (4 байта)				thickness (1 байт)	aim_area (1 байт)	red (1 байт)	green (1 байт)	
Байты 60-79	blue (1 байт)	ID="BRLI" (4 байта)		thickness (1 байт)	red (1 байт)	green (1 байт)	blue (1 байт)	count (1 байт)	arr (80 байт)						
Байты 80-99															
Байты 100-119															
Байты 120-139															
Байты 140-159															
Байты 160-179		radius (4 байта)		thickness (1 байт)	aim_area (1 байт)	red (1 байт)	green (1 байт)	blue (1 байт)	ID="POLY" (4 байта)		thickness (1 байт)	red (1 байт)	green (1 байт)	blue (1 байт)	count (1 байт)
Байты 180-199	arr (80 байт)														
Байты 200-219															
Байты 220-239															
Байты 240-259															

Рис. 3. Пример структуры файла разработанного формата

На данном рисунке представлен набор полей файла, содержащего базис, и по одной информационной структуре на каждый тип векторного примитива. Общий размер приведенного на рисунке файла, содержащего векторный слой графики – 260 байт.

2.2. Разработка библиотеки для работы с векторной графикой

Как уже было упомянуто выше, методы DLL-библиотеки должны реализовывать следующий набор функций:

- а) запись векторных данных в файл разработанного формата
- б) чтение векторных данных из файла разработанного формата
- в) отображение векторных примитивов файла на экране
- г) прицеливание для отображаемых на экране векторных примитивов
- д) масштабирование отображаемых на экране данных

В соответствии с этим в DLL-библиотеке были реализованы функции, решающие каждую из перечисленных выше задач.

Но прежде чем реализовывать функции, работающие с векторными примитивами, необходимо реализовать объявления типов структур векторных примитивов, с которыми работают функции DLL-библиотеки. Все типы структур векторных примитивов были реализованы в соответствии с описанием содержимого файла разработанного формата, приведенным выше. Для пояснения данного утверждения приведем пример описания типа структуры, соответствующей прямой линии:

```
typedef struct line          //описание структуры примитива прямой линии
{
```

```

uint32_t x;           //координата начала прямой по оси абсцисс
uint32_t y;           //координата начала прямой по оси ординат
uint32_t x1;          //координата конца прямой по оси абсцисс
uint32_t y1;          //координата конца прямой по оси ординат
unsigned char thickness; //толщина прямой
unsigned char aim_area; //область прицеливания прямой
unsigned char red;      //интенсивность цветового канала R модели RGB
unsigned char green;    //интенсивность цветового канала G модели RGB
unsigned char blue;     //интенсивность цветового канала B модели RGB
} line_;

```

Данное описание полностью соответствует набору полей, которые были описаны в пункте 2.1 настоящей работы. Для хранения координат начала и конца прямой используется тип `uint32_t`, реализующий 32-битное неотрицательное целое число, причем размер поля данного типа всегда равен ровно 4 байтам независимо от аппаратных особенностей используемой ВМ, в отличие от стандартного типа `int` языка Си++[3]. Для хранения остальных однобайтных неотрицательных полей используется тип `unsigned char`, как правило хранящий однобайтное беззнаковое целое число[4]. По аналогии с данным типом структур описаны типы структур для хранения векторных примитивов точки и окружности.

Структуры ломаной и многоугольника описываются с помощью массива 32-битных беззнаковых целых чисел. Описание типа структуры для хранения ломаной выглядит следующим образом:

```

typedef struct polyline //описание структуры примитива ломаной линии
{
    unsigned char thickness; //толщина ломаной
    unsigned char red;       //интенсивность цветового канала R модели RGB
    unsigned char green;     //интенсивность цветового канала G модели RGB
    unsigned char blue;      //интенсивность цветового канала B модели RGB
    unsigned char count;     //число точек ломаной
    uint32_t arr[20];        //массив точек ломаной
} polyline_;

```

Аналогичный набор полей содержит тип, описывающий многоугольник. Создание двух типов структур с аналогичным набором полей, но разными названиями преследовало своей целью предотвратить возможную путаницу при разработке программы, а также при потенциальном добавлении полей в одну из структур при совершенствовании или доработке программы.

Также отдельным типом структур является тип `basisd_`, описывающий структуру, содержащую данные базиса файла разработанного формата. Его описание выглядит следующим образом:

```
typedef struct basisdescr          //описание структуры базиса
{
    uint32_t refwidth;
    uint32_t refheight;
    int32_t basex;
    int32_t basey;
} basisd_;
```

Дополнительно в библиотеке реализовано перечисление `pctype_t` для идентификации каждого из вышеописанных типов и объявление типа структуры элемента двусвязного списка для хранения данных векторных примитивов, а также регионов для областей прицеливания в унифицированном виде. Объявление типа данной структуры выглядит следующим образом:

```
typedef struct datatype {          //элемент двусвязного списка
    void* data;                    //указатель на данные векторного примитива
    pctype_t type;                  /*тип примитива, на данные которого
    указывает data*/
    Gdiplus::Region* r;             //регион области прицеливания фигуры
    struct datatype* next;          //указатель на следующий элемент списка
    struct datatype* prev;         //указатель на предыдущий элемент списка
} datatype_t;
```

Отметим, что указатель `data` указывает на одну из вышеописанных структур векторных примитивов. Поскольку сами структуры векторных примитивов не содержат идентификаторов своих типов, для различения того на какой тип указывает указатель `data` используется вышеупомянутое перечисление `pctype_t`.

Теперь перейдем непосредственно к функциям, реализованным в разработанной DLL-библиотеке. Опишем функции, выполняющие каждую из задач, которые должна решать разработанная библиотека. Первой из задач, перечисленных в начале данного раздела работы была упомянута запись векторных данных в файл разработанного формата. Данное действие

осуществляет набор функций: `addbasisdescr`, `addpoint`, `addline`, `addbrokenline`, `addcircle`, `addpolyline`, соответственно реализующих добавление базиса, точки, линии, ломаной, окружности, и многоугольника в файл разработанного формата. Так как информационные структуры данных объектов не содержат их идентификаторов, перед записью данных в файл каждая из функций добавляет перед записываемыми данными 4-байтный идентификатор объекта, соответствующий тому, запись какого из примитивов осуществляет функция.

Следующей задачей, которую должны решать функции разработанной DLL-библиотеки должно быть чтение файлов разработанного формата. Данная задача решается с помощью универсальной функции `stread`, принимающей указатель на открытый файловый поток `fp`, указатель на сохраняемые данные и возвращающей тип прочитанной структуры `ptype_t`. Данная функция читает ровно одну структуру из файла разработанного формата по текущей позиции указателя открытого файлового потока `fp`, записывает адрес прочитанных данных в виде указателя на данные, на который указывает указатель `data`, и возвращает тип прочитанной структуры. В случае, если первые 4 байта по текущему указателю в открытом файловом потоке не соответствуют ни одному из примитивов или базису, функция возвращает элемент перечисления `PTYPE_NULL`, указывающий либо на достижение окончания читаемого файла, либо на несоответствие данных в нем требованиям разработанного формата. Весомым преимуществом именно такой реализации функции чтения файлов разработанного формата можно выдвинуть удобство реализации чтения файлов с ее использованием в цикле, выполнение которого продолжается до того момента пока функция не возвратит `PTYPE_NULL` в качестве типа прочитанного примитива.

Далее в списке задач, решаемых разработанной DLL-библиотекой следует отображение векторных примитивов, прочитанных из файла разработанного формата на экране. Для этого используется набор функций, имеющих следующие названия: `ppoint`, `pline`, `pbrokenline`, `pcircle`, `ppolyline`, соответственно реализующих вывод на экран точки, прямой линии, ломаной

линии, окружности, и многоугольника. Данные функции принимают указатель на графический контекст (тип Graphics*) и числовые данные для рисования, а возвращают тип void. Для рисования примитива они используют перо, цвет и толщина которого соответствуют данным, указанным при вызове. Проиллюстрируем это примером функции pline, отображающей на экране прямую линию:

```
/* Функция pline рисует линию на графическом контексте graphics, соединяющую
точки (x, y) и (x1, y1) с толщиной и цветом, заданными значениями thickness,
red, green и blue. */
/* Функция принимает девять аргументов: указатель на графический контекст
graphics, координаты начальной точки (x, y), координаты конечной точки (x1,
y1), толщину линии thickness и значения цвета red, green и blue. */
extern "C" NDLLPROJ_API void pline(Graphics* graphics, uint32_t x, uint32_t
y, uint32_t x1, uint32_t y1, char thickness, char red, char green, char blue)
{
    Pen* pen = new Pen(Color(255, red, green, blue), thickness); /* Создаем
объект Pen с цветом, заданным значениями red, green и blue, и толщиной,
заданной значением thickness */
    graphics->DrawLine(pen, (int)x, (int)y, (int)x1, (int)y1); /* Рисуем
линию на графическом контексте graphics, соединяющую точки (x, y) и (x1, y1),
используя объект Pen */
    delete pen; // Освобождаем память, занятую объектом Pen
}
```

После использования локального объекта пера в функции необходимо удалить его, чтобы избежать утечки объектов GDI, которая является достаточно опасным явлением, в особо запущенных случаях приводящим к полному зависанию операционной системы[5].

Следующей задачей, которую должны решать функции разрабатываемой библиотеки является прицеливание для отображаемых на экране векторных примитивов. Реализуется оно единственной функцией IsInside, работающей на основании данных о регионах прицеливания из двусвязного списка, описание типа структуры элемента которого было приведено ранее. Данная функция принимает 3 аргумента: 2 числа x и y типа uint32_t, являющихся координатами клика мыши в окне соответственно по оси абсцисс и ординат, и указатель на голову двусвязного списка head, а возвращает указатель на элемент двусвязного списка, описывающий примитив, в который произошло попадание. Сама функция реализует

следующий простой алгоритм: обходит элементы списка с помощью указателя на следующий элемент `next`, и для каждого элемента списка проверяет попадание координат клика мыши `x,y` в регион прицеливания, содержащийся внутри структуры соответствующего элемента двусвязного списка. Для проверки попадания используется функция GDI+ `IsVisible`, проверяющая, лежит ли соответствующая точка, описанная в виде объекта `Point` внутри указанного региона и возвращающая результат в виде типа `bool`[6]. Если для какого-либо из элементов списка данная функция возвращает значение `true`, цикл обхода принудительно завершается командой `break` и возвращается указатель на элемент, в область прицеливания которого произошло попадание. Если же цикл обхода дошел до конца двусвязного списка, что означает что попадание в область прицеливания ни для одного из примитивов так и не произошло, функция возвращает указатель `NULL`, что означает отсутствие попадания при прицеливании. Отметим, что вышеописанный алгоритм имеет одну важную особенность, которую надо учитывать при работе с векторным слоем графики: если координаты клика мыши соответствуют нескольким элементам двусвязного списка, элементом, в область прицеливания которого произошло попадание будет считаться первый по порядку следования в списке элемент, в область прицеливания которого произошло попадание. Блок-схема алгоритма работы функции `IsInside` представлена на рис. 4.

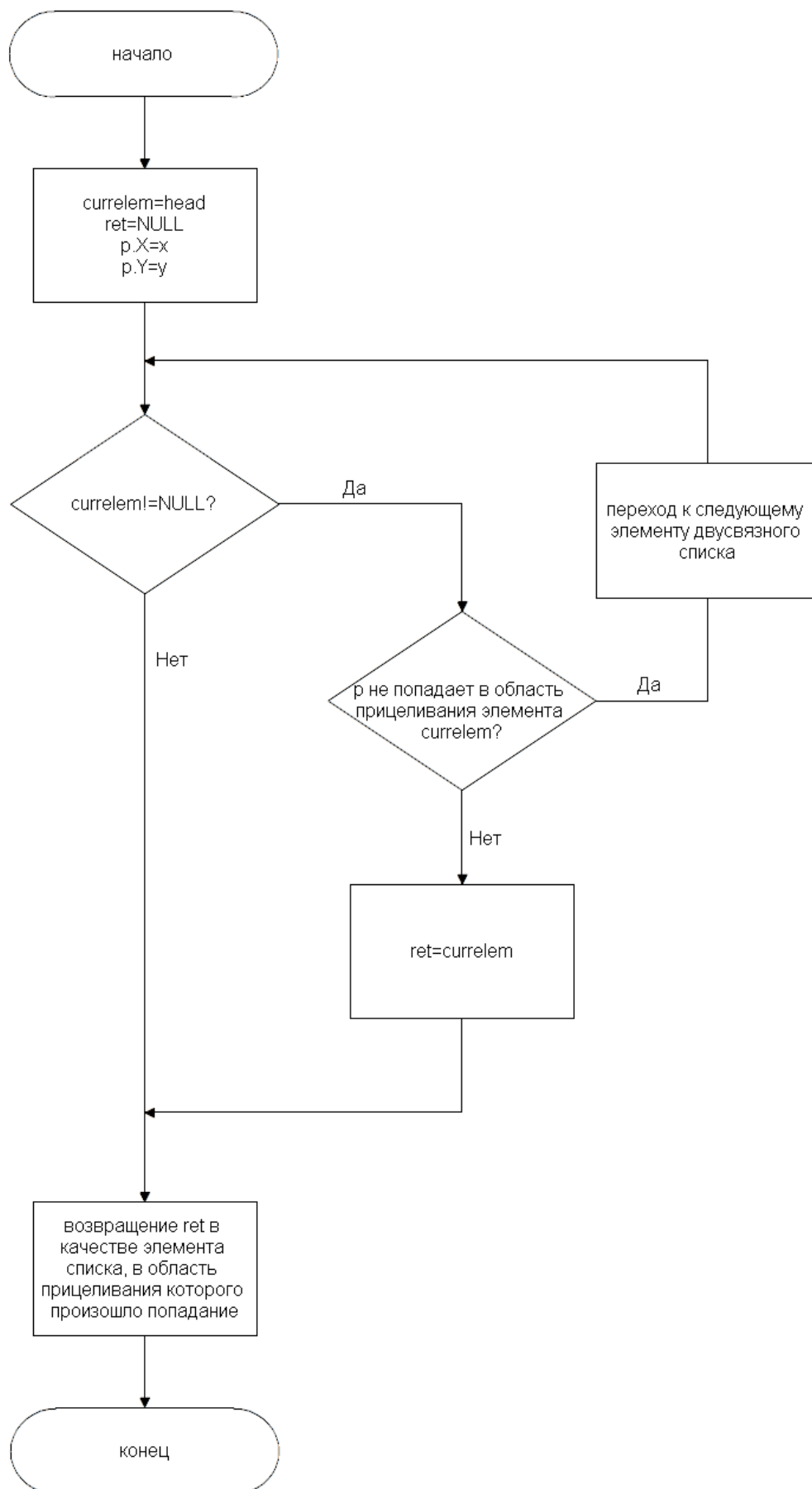


Рис.4. Схема алгоритма работы функции `IsInside`

Последней задачей, которую должны решать функции разработанной DLL-библиотеки является масштабирование отображаемых на экране данных. Реализуется это с помощью функции `PaintBitmapInHDC`, возвращающей тип `void` и принимающей следующий набор аргументов:

а) **hWnd** – дескриптор окна, в котором будет отображаться изображение (тип `HWND`)

б) **hWndScroll** – дескриптор окна, содержащего горизонтальный ползунок прокрутки (тип `HWND`)

в) **hWndScroll1** – дескриптор окна, содержащего вертикальный ползунок прокрутки (тип `HWND`)

г) **hDC** – контекст устройства окна, в котором будет отображаться изображение (тип `HDC`)

д) **memBit** – контекст устройства, содержащий изображение (тип `HDC`)

е) **state** – состояние отображения изображения (0 – без масштабирования, 1 – масштаб 200% по каждой оси, 2 – масштаб 300% по каждой оси, -1 – масштаб 50% по каждой оси, -2 – масштаб 33% по каждой оси; тип `int`)

ж) **x_** – координата x верхнего левого угла области изображения, которая будет отображаться в окне (тип `int`)

и) **y_** – координата y верхнего левого угла области изображения, которая будет отображаться в окне (тип `int`)

к) **width** – ширина области изображения, которая будет отображаться в окне в пикселях (тип `int`)

л) **height** – высота области изображения, которая будет отображаться в окне в пикселях (тип `int`)

Данная функция осуществляется отрисовку графических данных из контекста `memBit` в контексте окна приложения `hDC`, масштабируя нужным образом окно `hWnd` и настраивая видимость ползунков прокрутки `hWndScroll` и `hWndScroll1` в соответствии со значением `state` (`state>0` – ползунки видимы, иначе скрыты). Для сжатия/растяжения отображаемых данных используется

функция `SetStretchBltMode` в режиме `ColorOnColor`, минимизирующем искажение цветов пикселей цветного изображения при сжатии[7]. Оригинальным размером окна (т.е. соответствующим `state=0`) считается размер окна в пикселях, заданный величинами `width` и `height`. При этом для отображения данных в данной функции используется стандартный режим отображения `MM_TEXT`, в котором координатные оси направлены вправо вниз, а единицами измерения являются пиксели устройства отображения[7]. Таким образом, величины `x_` и `y_` задают координаты верхнего левого угла области изображения, которая будет отображаться в окне в пикселях устройства отображения.

Перед тем как завершить описание реализации DLL-библиотеки отметим еще одну важную деталь. Для того, чтобы использовать функции разработанной библиотеки в приложениях был создан заголовочный файл, содержащий прототипы всех вышеописанных функций. Также в данный файл были вынесены объявления типов для хранения структур векторных примитивов. Данный подход позволяет сразу использовать функции разработанной библиотеки в исходном коде разрабатываемых приложений, не прибегая к функциям `LoadLibrary` и `GetProcAddress`. Для того чтобы использовать данные функции в приложении достаточно с помощью директивы препроцессора `#include` подключить к программе заголовочный файл, содержащий прототипы, и указать линковщику подключаемую к программе DLL-библиотеку.

2.3. Разработка редактора файлов векторной графики

Согласно п.1.2 данной работы, программа-редактор файлов разработанного формата должна выполнять следующий набор функций:

- а) Создание файлов разработанного формата
- б) Добавление примитивов в файл разработанного формата

- в) Редактирование примитивов файла разработанного формата
- г) Удаление примитивов из файла разработанного формата
- д) Вывод на экран данных файла разработанного формата

Перед тем как перейти к рассмотрению конкретных реализаций каждой из перечисленных функций, уточним реализацию интерфейса разработанного приложения. Для объединения всех вышеперечисленных функций в данном приложении реализуется консольное многоуровневое интерактивное меню. Наиболее общим образом алгоритм работы подобного меню можно описать так: при запуске приложения на экран выводится список пунктов меню и клавиш, которые соответствуют выбору каждого из пунктов данного меню. Пользователь производит нажатие на клавиатуре клавиши, соответствующей выбранному пункту меню, после чего приложение требует ввод пользователем данных для выполнения выбранного действия. После ввода всех необходимых данных и выполнения приложением требуемых действий, консольное приложение снова выводит на экран список пунктов меню и предлагает пользователю выбрать один из них. Данный диалог продолжается до тех пор, пока пользователь не выберет в интерактивном меню пункт, соответствующий завершению работы программы. В интерактивном меню разработанного приложения реализован следующий набор пунктов:

- а) создать новый файл
- б) редактировать уже существующий файл
- в) завершить работу программы
- г) удалить из существующего файла структуру(-ы)

Как видно из приведенного описания, пункты интерактивного меню не совсем соответствуют набору функций, выполняемых программой-редактором. Уточним отношение между перечисленным ранее набором функций программы-редактора и пунктами интерактивного меню. Создание нового файла соответствует функциям а) и б) разрабатываемой программы редактора, поскольку при создании нового файла редактор сразу предлагает

пользователю начать заполнять новый файл данными. Редактирование уже существующего файла соответствует функциям в) и д), поскольку при редактировании файла приложение выводит на экран список примитивов файла и предлагает пользователю выбрать тот из примитивов, который он хочет отредактировать. Аналогичным образом, удаление структур из файла соответствует функциям г) и д) программы-редактора. Завершение работы программы не соответствует ни одной из функций программы-редактора, поскольку является пунктом, необходимым для реализации самой логики работы интерактивного консольного меню.

Теперь более подробно рассмотрим алгоритм работы приложения при выборе каждого из пунктов интерактивного меню. Начнем с первого пункта, создания нового файла. Выбор данного пункта меню осуществляется нажатием на клавиатуре цифровой клавиши «1». При выборе данного пункта приложение запрашивает у пользователя имя создаваемого файла. После ввода имени файла пользователю предлагается новое интерактивное меню, которое осуществляет выбор добавляемого в файл примитива и имеет аналогичную меню выбора действия логику работы: пользователю предлагается список примитивов (включающий также и базис файла), после выбора пункта которого пользователь осуществляет ввод данных о примитиве. Меню выбора примитивов по аналогии интерактивному меню выбора действия также содержит пункт завершения, после выбора которого пользователь возвращается в меню выбора действия. Пример создания файла в консольном интерфейсе приложения приведен на рис. 5.

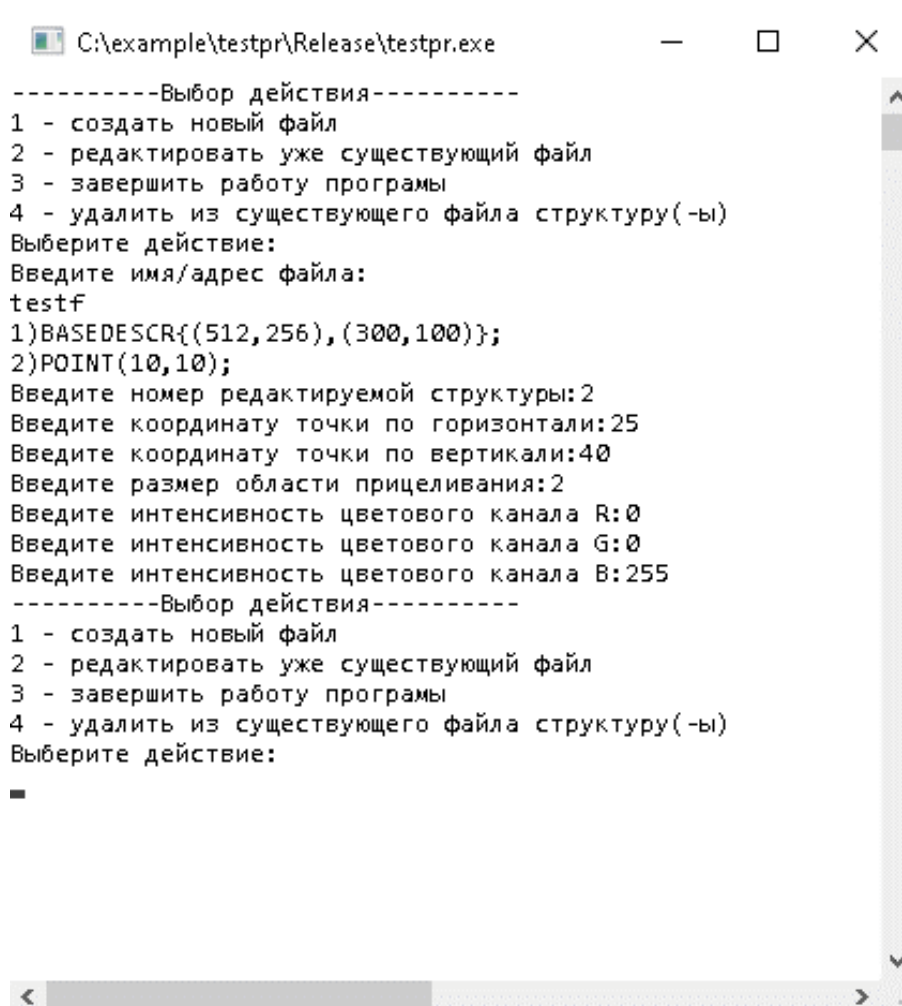
```
C:\example\testpr\Release\testpr.exe

-----Выбор действия-----
1 - создать новый файл
2 - редактировать уже существующий файл
3 - завершить работу программы
4 - удалить из существующего файла структуру(-ы)
Выберите действие:
Введите название создаваемого файла:
testf
-----Добавление примитивов-----
Клавиши и их соответствие примитивам
A - установить параметры базиса
B - точка
C - линия
D - ломаная
E - окружность
F - многоугольник
G - в главное меню
Выберите опцию:
Введите refwidth:512
Введите refheight:256
Введите basex:300
Введите basey:100
-----Добавление примитивов-----
Клавиши и их соответствие примитивам
A - установить параметры базиса
B - точка
C - линия
D - ломаная
E - окружность
F - многоугольник
G - в главное меню
Выберите опцию:
Введите координату точки по горизонтали:10
Введите координату точки по вертикали:10
Введите размер области прицеливания:4
Введите интенсивность цветового канала R:255
Введите интенсивность цветового канала G:255
Введите интенсивность цветового канала B:0
-----Добавление примитивов-----
Клавиши и их соответствие примитивам
A - установить параметры базиса
B - точка
C - линия
D - ломаная
E - окружность
F - многоугольник
G - в главное меню
Выберите опцию:
-----Выбор действия-----
1 - создать новый файл
2 - редактировать уже существующий файл
3 - завершить работу программы
4 - удалить из существующего файла структуру(-ы)
Выберите действие:
```

Рис. 5. Создание файла в консольном интерфейсе приложения

Следующий пункт интерактивного меню выбора действия – редактирование уже существующего файла. Выбор данного пункта меню осуществляется нажатием на клавиатуре цифровой клавиши «2». Аналогично первому пункту, пользователь вводит имя открываемого файла (только в отличие от создания файла разработанного формата, файл открывается не для записи, а для чтения и изменения). После открытия файла программа выводит на экран список уже добавленных в файл примитивов и запрашивает у

пользователя номер редактируемого примитива. Потом приложение запрашивает у пользователя значения полей редактируемого примитива, после ввода которых приложение находит редактируемый примитив в файле и перезаписывает его структуру. Пример редактирования файла в консольном интерфейсе приложения приведен на рис. 6.



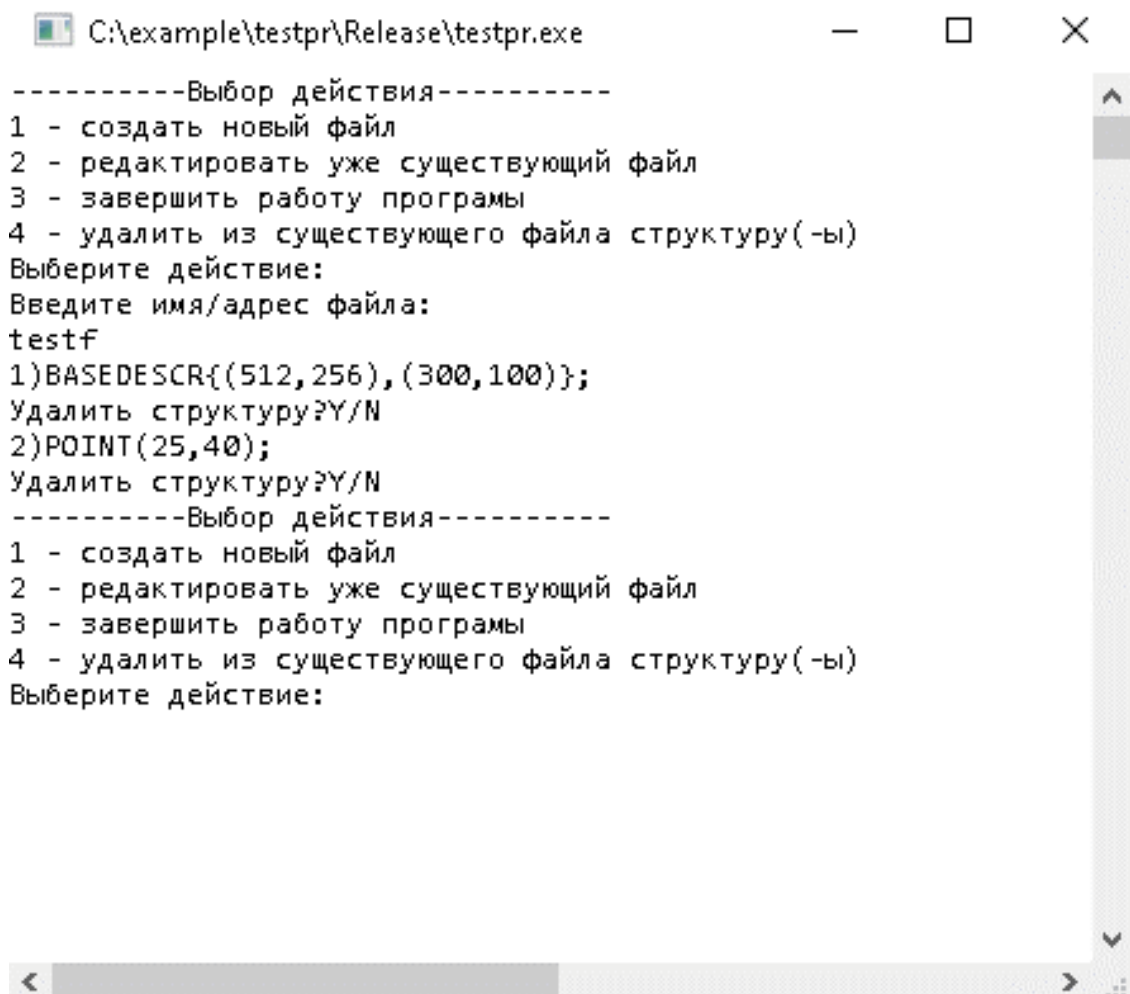
```
C:\example\testpr\Release\testpr.exe
-----Выбор действия-----
1 - создать новый файл
2 - редактировать уже существующий файл
3 - завершить работу программы
4 - удалить из существующего файла структуру(-ы)
Выберите действие:
Введите имя/адрес файла:
testf
1)BASEDESCR{(512,256),(300,100)};
2)POINT(10,10);
Введите номер редактируемой структуры:2
Введите координату точки по горизонтали:25
Введите координату точки по вертикали:40
Введите размер области прицеливания:2
Введите интенсивность цветового канала R:0
Введите интенсивность цветового канала G:0
Введите интенсивность цветового канала B:255
-----Выбор действия-----
1 - создать новый файл
2 - редактировать уже существующий файл
3 - завершить работу программы
4 - удалить из существующего файла структуру(-ы)
Выберите действие:
■
```

Рис. 6. Редактирование файла в консольном интерфейсе приложения

Пункт завершения работы программы (соответствует цифровой клавише «3») не нуждается в подробном описании: при нажатии пользователем клавиши, соответствующей данному пункту, работа приложения завершается.

Удаление структуры из файла разработанного формата (соответствует цифровой клавише «4») работает следующим образом: программа запрашивает у пользователя имя изменяемого файла, читает данные из файла, выводит на экран по одному из примитивов файла и спрашивает пользователя, нужно ли его удалить. При этом программа создает временный файл, в

который по мере выбора пользователем удалить или оставить каждый из примитивов записываются только те примитивы файла, которые необходимо сохранить. После того, как пользователь выбирает удалить или оставить последний примитив, программа удаляет изменяемый файл, а временный файл переименовывает так, чтобы его название соответствовало только что удаленному файлу. Пример удаления примитива из файла в интерфейсе разработанного приложения приведен на рис. 7.



```
C:\example\testpr\Release\testpr.exe

-----Выбор действия-----
1 - создать новый файл
2 - редактировать уже существующий файл
3 - завершить работу программы
4 - удалить из существующего файла структуру(-ы)
Выберите действие:
Введите имя/адрес файла:
testf
1)BASEDESCR{(512,256),(300,100)};
Удалить структуру?Y/N
2)POINT(25,40);
Удалить структуру?Y/N
-----Выбор действия-----
1 - создать новый файл
2 - редактировать уже существующий файл
3 - завершить работу программы
4 - удалить из существующего файла структуру(-ы)
Выберите действие:
```

Рис. 7. Удаление примитива из файла в консольном интерфейсе приложения

Обобщает работу всех пунктов интерактивного консольного меню блок-схема алгоритма работы редактора файлов, приведенная на рис.8.

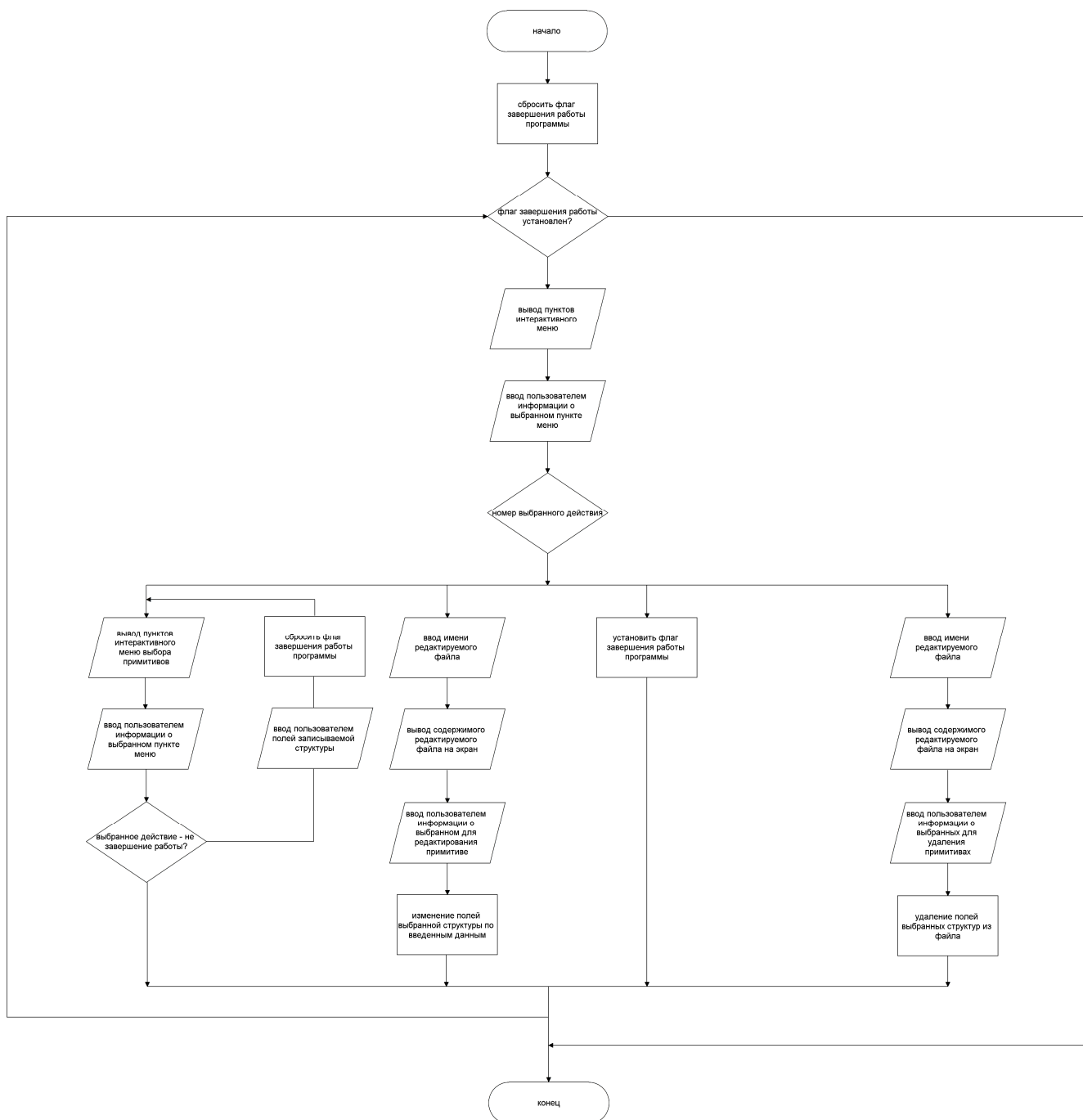


Рис. 8. Схема алгоритма работы редактора файлов

2.4. Разработка демонстрационного приложения

Разрабатываемое в данном пункте приложение обладает оконным интерфейсом, поэтому требует особого подхода при разработке. В первую очередь нужно определиться с интерфейсом основного окна приложения. В качестве основного окна использовано окно с заголовком «MyApp» и следующим набором стилей: `WS_OVERLAPPED`, `WS_CAPTION`, `WS_SYSMENU`. В соответствии с данным набором стилей основное окно приложения имеет заголовок, границу, а также меню заголовка. Функция `CreateWindow`, создающая объект окна, вызывается в функции `WinMain`. `WinMain` – начальная точка входа для базирующейся на Windows прикладной программы [8]. Вызов функции `CreateWindow`, реализующий вышеприведенное описание, выглядит следующим образом:

```
hWnd = CreateWindow(  
TEXT("MyApp"), // имя класса окна  
TEXT("MyApp"), // заголовок окна  
WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU, // стиль окна  
CW_USEDEFAULT, // начальная позиция по оси x  
CW_USEDEFAULT, // начальная позиция по оси y  
CW_USEDEFAULT, // начальный размер по оси x  
CW_USEDEFAULT, // начальный размер по оси y  
NULL, // дескриптор родительского окна  
NULL, // дескриптор меню окна  
hInstance, // дескриптор экземпляра программы  
NULL); // параметры создания
```

Также как правило в функции `WinMain` реализуется цикл выборки и обработки сообщений, который необходим для того чтобы основное окно могло получать и обрабатывать сообщения мыши, клавиатуры, и оконных органов управления. Цикл выборки и обработки сообщений в функции `WinMain` разработанного приложения выглядит следующим образом:

```
// Цикл выполняется, пока не получено сообщение WM_QUIT  
while(GetMessage(&msg, NULL, 0, 0))  
{  
    /* Преобразование виртуальных кодов клавиш в символьные коды для сообщений  
    WM_KEYDOWN и WM_KEYUP */  
    TranslateMessage(&msg);
```

```

// Передача сообщения в оконную процедуру соответствующего окна
DispatchMessage(&msg);
}

```

Поскольку разработанное приложение работает с файлами, надо также предусмотреть в нем получение имен или адресов открываемых файлов. Для реализации данного действия в разработанном приложении было реализовано получение имен открываемых файлов через аргументы командной строки, в силу простоты данного варианта реализации. Код, реализующий данное действие также находится в функции WinMain и выглядит следующим образом:

```

// Преобразование командной строки в массив аргументов в формате Unicode
LPWSTR* lpArgv = CommandLineToArgvW(GetCommandLineW(), &argc);

// Выделение памяти для массива аргументов в формате ASCII
argv = (char**)malloc(argc * sizeof(char*));

// Инициализация счетчика итераций
int size, i = 0;

/* Цикл для преобразования каждого аргумента из формата Unicode в формат
ASCII */
for (; i < argc; ++i)
{
    // Вычисление размера аргумента в символах, включая нулевой символ
    size = wcslen(lpArgv[i]) + 1;

    // Выделение памяти для аргумента в формате ASCII
    argv[i] = (char*)malloc(size);

    // Преобразование аргумента из формата Unicode в формат ASCII
    wcstombs(argv[i], lpArgv[i], size);
}

/* Освобождение памяти, выделенной для массива аргументов в формате
Unicode */
LocalFree(lpArgv);

```

После рассмотрения содержимого функции WinMain, отражающего внешний вид основного окна разработанного приложения, а также способ получения сообщений приложением, можно перейти к рассмотрению реализации каждой из функций разработанного приложения. Отметим, что основой оконного приложения Windows кроме функции WinMain является также оконная функция WndProc, в которой описывается алгоритм обработки получаемых приложением сообщений, что является основой алгоритма

работы разработанного приложения. Именно поэтому дальнейшее повествование будет сосредоточено на алгоритмах, реализуемых внутри данной функции. Согласно п.1.2, демонстрационное приложение должно выполнять следующий набор функций:

а) Отображение слоя векторной графики поверх растровой подложки на экране

б) Включение/отключение сглаживания графических данных векторного слоя

в) Масштабирование отображаемых данных

г) Вывод данных о примитиве при попадании координат клика мыши в область прицеливания какой-либо из фигур

Разберем реализацию каждой из перечисленных функций в разработанном приложении. Первой такой функцией является отображение векторной графики поверх растровой подложки на экране. Основой реализации данной функции внутри разработанного приложения являются функции `readfile` и `vgraph`. Первая из них реализует чтение файла векторной графики и заполнение двусвязного списка данными векторных примитивов, а вторая реализует отображение векторной графики на основе двусвязного списка в контексте устройства `hdc`. При этом функция `readfile` также заполняет поле данных об области прицеливания двусвязного списка, предварительно рассчитывая его размеры на основе данных о векторных примитивах (более подробно то, как рассчитывается размер области прицеливания будет рассмотрено позднее). Функция `readfile` принимает следующие 2 аргумента: строку с именем открываемого файла (тип `char*`) и указатель на голову двусвязного списка (тип `datatype_t**`). Функции не передается сам указатель на вершину списка, а именно указатель на этот указатель по той причине, что при заполнении списка указатель на вершину списка будет изменяться, а, чтобы функция реализовывала изменение внешней переменной

ей необходимо передавать указатель на эту переменную[9]. Разработанная функция производит следующий набор действий: открывает файл, имя которого было указано в переданной строке с помощью функции `fopen`, производит чтение файла в цикле с помощью функции `strread` разработанной DLL-библиотеки (напомним, что условием завершения цикла является возвращение функцией `strread` элемента `PType_NULL` перечисления `pType_t`), формирует на основе прочитанных данных элемент двусвязного списка и добавляет его в список с помощью макроса `LIST_ADD` (содержится в заголовочном файле `list_helper.h`). Отметим, что сама функция `readfile` возвращает логическое значение (тип `bool`), указывающее на успешность операции чтения файла. Если файл не удалось открыть, либо в нем нет данных/они не соответствуют требованиям разработанного формата данная функция возвращает `false`, в противном случае – `true`.

Теперь подробно рассмотрим вторую из вышеупомянутых функций, `vgraph`. Функция `vgraph` отображает в контексте устройства векторный слой графики, сохраненный функцией `readfile` в двусвязном списке. В соответствии с этим функция `vgraph` возвращает тип `void` и принимает следующий набор аргументов:

- а) Указатель на голову двусвязного списка, тип `datatype_t*` (поскольку изменение списка в данной функции не производится)
- б) Контекст устройства, в котором будет отображен векторный слой графики, тип `HDC`
- в) Разрешение отображаемого изображения по горизонтали, тип `int` (используется для масштабирования отображаемых данных на основе базиса)
- г) Разрешение отображаемого изображения по вертикали, тип `int` (используется для масштабирования отображаемых данных на основе базиса)
- д) Флаг использования сглаживания при отображении векторного слоя графики, тип `bool`

Алгоритм работы данной функции обобщенно можно описать следующим образом: сначала функция на основании контекста отображения

получает контекст для рисования (тип Graphics), в котором с помощью функции GDI+ SetSmoothingMode устанавливает режим сглаживания в соответствии с флагом сглаживания. Далее функция с помощью указателей на следующий элемент обходит список, последовательно отображая каждый из элементов списка с помощью функций ppoint,pline,pcircle,pbrokenline,ppolyline разработанной DLL-библиотеки. Отметим, что для элемента, которому соответствует тип PTYPE_BASE (т.е. для элемента базиса) данная функция заполняет глобальные переменные scalex, scaley, basex, basey используемые для пересчета координат отображения для функций рисования. Если basex и basey просто заполняются значениями информационной структуры, то scalex и scaley вычисляются согласно формулам (3) и (4).

$$scalex = \frac{width}{refwidth} \quad (3)$$

Здесь width – ширина изображения, отображаемого на экране (ширина растровой подложки), refwidth – опорная ширина изображения, содержащаяся в структуре базиса векторного слоя графики.

$$scaley = \frac{height}{refheight} \quad (4)$$

Здесь height – высота изображения, отображаемого на экране (высота растровой подложки), refheight – опорная высота изображения, содержащаяся в структуре базиса векторного слоя графики.

С учетом соотношений (3) и (4) приведенные в пункте 2.1 формулы (1) и (2) можно переписать следующим образом:

$$x = basex + scalex * x_0 \quad (5)$$

$$y = basey + scaley * y_0 \quad (6)$$

Обобщенно преобразование, соответствующее формулам (5) и (6) реализует функция `recount`, которая выглядит следующим образом:

```
//пересчет координаты с учетом базиса
//coord - пересчитываемая координата
//scale - коэффициент масштабирования
//base - базовый коэффициент пересчета
//возвращает пересчитанную координату
int recount(int coord, float scale, int base)
{
    return (int)(coord*scale+base);
}
```

Заметим, что итоговый результат преобразования приводится к типу `int` (не может же значение координаты точки в пикселях быть дробным!). Важно понимать, что в результате подобного приведения значение округляется в меньшую сторону.

При вызове функций `ppoint`, `pline`, `pcircle`, `pbrokenline`, `ppolyline` к координатам отрисовки примитивов применяется функция `recount`. Проиллюстрируем это примером вызова функции `ppoint` внутри функции `vragph`:

```
ppoint(g, recount(ptrp->x, scalex, basex), recount(ptrp->y, scaley, basey), ptrp->red, ptrp->green, ptrp->blue);
```

Здесь координаты точки на экране, прочитанные из файла векторного слоя перед вызовом функции `ppoint` пересчитываются с помощью функции `vgraph` в соответствии с глобальными переменными базиса `scalex`, `basex`, `scaley`, `basey`.

После подробного рассмотрения функций `readfile` и `vgraph` перейдем непосредственно к тому, как в оконной функции приложения реализуется отображение векторных данных поверх растровой подложки. Сразу после создания окна (т.е. при получении сообщения `WM_CREATE`) производится попытка загрузки слоя растровой графики в объект типа `HBITMAP` с помощью функции Windows API `LoadImageA`, а также попытка прочесть файл со слоем векторной графики с помощью ранее описанной функции `readfile`. Если хотя бы одно из этих действий осуществить не удастся (что можно понять по

возвращаемым значениям данных функций), программа выводит окно с сообщением об ошибке с помощью MessageBox и завершает свою работу через вызов функции PostQuitMessage. Если же прочесть данные из файлов векторной и растровой графики все-таки удалось, работа программы продолжается, и при отрисовке окна на экране (т.е. при поступлении сообщения WM_PAINT) производится описанный ниже набор действий. Во-первых, создается промежуточный контекст устройства отображения с помощью CreateCompatibleDC. Создаваемый данной функцией контекст памяти совместим с контекстом памяти окна приложения, что позволяет использовать его в качестве буфера для отображения данных в окне разрабатываемого приложения. После этого в данный контекст с помощью SelectObject выбирается ранее загруженный в объект типа HBITMAP растровый рисунок. Далее с помощью ранее описанной функции vgraph в буферном контексте памяти производится отрисовка векторного слоя графики и данные из буферного контекста памяти переносятся уже в контекст памяти окна приложения вызовом функции PaintBitmapinHDC. Завершается отрисовка данных на экране удалением объекта HBITMAP, содержащего загруженное растровое изображение, буферного контекста памяти для отрисовки графических данных memBit1, а также освобождением контекста окна приложения hdc. Таким образом, алгоритм отрисовки графических данных в разработанном приложении соответствует следующему исходному коду:

```
hdc = BeginPaint(hWnd, &ps);           /* Получение контекста устройства для окна
hWnd */
memBit1 = CreateCompatibleDC(hdc); /* Создание совместимого контекста
устройства */
hBMP = (HBITMAP)LoadImageA( NULL, (LPCSTR)argv[1], IMAGE_BITMAP, 0, 0,
LR_LOADFROMFILE);                     /*Загрузка изображения из файла в виде
битовой карты*/
holdBMP = (HBITMAP)SelectObject(memBit1,hBMP); /* Выбор загруженной битовой
карты в контексте устройства memBit1 */
vgraph(head,memBit1,bm.bmWidth,bm.bmHeight,AliasFlag);/* Вызов функции vgraph
для отрисовки графики на битовой карте */

PaintBitmapinHDC(hWnd,hwndScroll,hwndScroll1,hdc,memBit1,state,x_,y_,bm.bmWid
th,bm.bmHeight); /* Вызов функции PaintBitmapinHDC для отрисовки битовой
карты в контексте устройства hdc */
```



```

SelectObject(memBit1, hOldBMP);      /* Возврат предыдущего объекта контекст
устройства memBit1 */
DeleteObject(hBMP);                  // Удаление объекта HBITMAP
DeleteDC(memBit1);                   // Удаление контекста устройства memBit1

//Завершение рисования и освобождение контекста устройства hdc
EndPaint(hWnd, &ps);
ReleaseDC(hWnd, hdc);

```

Перейдем к следующей функции, которую должно реализовывать разрабатываемое приложение: включение/выключение сглаживания отображаемых данных. Переключение режима сглаживания осуществляется нажатием на клавиатуре клавиши «А». При нажатии какой-либо клавиши окну приложения отправляется сообщение WM_KEYDOWN, в параметре которого wParam содержится виртуальный код, идентифицирующий нажатую клавишу[7]. Соответственно, в обработчике данного сообщения производится проверка содержащегося в wParam кода и, если произошло нажатие клавиши «А», производится изменение значения глобальной логической переменной AliasFlag. Чтобы изменения отобразились на экране, необходимо вызвать перерисовку содержимого окна, что осуществляется последовательным вызовом функций InvalidateRect и UpdateWindow. Сам глобальный флаг сглаживания AliasFlag передается в качестве аргумента функции vgraph и определяет, будет ли данная функция производить сглаживание отображаемых данных при отрисовке векторного слоя графики.

Следующей задачей, которую должно реализовывать разработанное приложение в списке из п.1.2 обозначено масштабирование отображаемых данных. Как уже было написано выше, масштабирование при отображении данных из буферного контекста памяти memBit1 осуществляется функцией PaintBitmapinHDC на основании значения глобальной переменной state. Соответственно, при изменении масштаба отображения осуществляется следующий набор действий: обработчик сообщения WM_KEYDOWN сбрасывает (т.е. устанавливает в 0) значения глобальных переменных x_ и y_, идентифицирующие соответствие координат левого верхнего угла окна координатам на отображаемом битовом образе (задается положением

ползунков), с помощью функции `SetScrollPos` сбрасывает положение вертикального и горизонтального ползунков, изменяет в соответствии с нажатой клавишей значение глобальной переменной `state` и вызывает перерисовку окна последовательным вызовом функций `InvalidateRect` и `UpdateWindow`. Работоспособность ползунков обеспечивается обработкой сообщений `WM_VSCROLL` и `WM_HSCROLL`, поступающих соответственно от вертикального и горизонтального ползунков окна. Младшее слово параметра `wparam` данных сообщений содержит идентификатор произведенного с ползунком действия, а старшее слово – какой-либо дополнительный аргумент, обычно это новое положение ползунка после данного действия. Обработчики данных сообщений, содержащиеся в функции `WndProc` обрабатывают три вида действий с ползунками. Первое из них, `SB_THUMBPOSITION`, соответствует тому, что пользователь захватил ползунок и установил его в новое положение, завершив его перемещение[7]. В обработчике сообщения данному событию соответствует следующий алгоритм: обработчик получает из старшего слова параметра `wparam` новую позицию ползунка, с помощью `SetScrollPos` устанавливает ползунок в новую позицию, изменяет одну из глобальных переменных `x_`, `y_`, соответствующих левому верхнему углу окна, и вызывает перерисовку окна последовательным вызовом функций `InvalidateRect` и `UpdateWindow`. Другие два события, `SB_LINEUP` и `SB_LINEDOWN` соответствуют изменению положения полосы прокрутки с помощью стрелок, присутствующих у данного элемента управления[7]. Обычно в данной ситуации просто производится уменьшение или увеличение положения ползунка на какое-либо константное значение в соответствии с направлением, соответствующим нажатой стрелке. При этом необходимо производить проверку, что новое положение ползунка не выходит за границы установленного для данного ползунка диапазона значений. Если положение ползунка выходит за установленный для него диапазон значений, новым его положением устанавливается ближайшее крайнее значение диапазона допустимых положений ползунка. В соответствии с

вышеприведенными соображениями в обработчиках сообщений SB_LINEUP и SB_LINEDOWN реализован следующий алгоритм: получение текущего положения ползунка с помощью функции GetScrollPos, получение диапазона положений ползунка с помощью GetScrollRange, изменение значения положения ползунка в нужную сторону, проверка нового значения (если оно не укладывается в диапазон значений, производится его корректировка), установка нового положения ползунка с помощью SetScrollPos, а также сохранение нового положения верхнего левого угла экрана в одной из переменных x_, y_, и перерисовка окна последовательным вызовом функций InvalidateRect и UpdateWindow. Описанный выше набор действий можно проиллюстрировать следующим куском кода обработчика сообщения WM_HSCROLL:

```
// Если нажата кнопка "Стрелка вправо"
if (LOWORD(wParam) == SB_LINEDOWN)
{
    int nPos = GetScrollPos(hwndScroll, SB_CTL);          /* Получение
текущей позиции скроллбара */
    int nMin, nMax;
    GetScrollRange(hwndScroll, SB_CTL, &nMin, &nMax);    /* Получение
минимальной и максимальной позиции скроллбара */
    nPos += 10;
    // Увеличение позиции на 10
    nPos = (nPos > nMax) ? nMax : nPos;                  /* Если позиция
больше максимальной, то установка в максимальную */
    SetScrollPos(hwndScroll, SB_CTL, nPos, false);       /* Установка новой
позиции скроллбара */
    x_ = nPos;                                           /* Установка
нового значения смещения по оси X */

    // Вызов функций перерисовки окна
    InvalidateRect(hwnd, NULL, TRUE);
    UpdateWindow(hwnd);
}
```

Наконец, перейдем к рассмотрению реализации последней из функций разработанного приложения: вывод данных о примитиве при попадании координат клика мыши в область прицеливания какой-либо из фигур. Во-первых, отметим, что не для всех примитивов необходим какой-либо математический расчет их областей прицеливания (согласно таблице 1 для

ломаной и многоугольника расчет размеров областей прицеливания не требуется). Во-вторых, напомним, что расчет размеров областей прицеливания производится функцией `readfile` во время чтения файла и полученный регион прицеливания сохраняется как поле региона внутри двусвязного списка векторных примитивов. Последовательно рассмотрим расчет областей прицеливания для различных примитивов и соответствующий данному расчету алгоритм прицеливания для данных примитивов. Начнем с самого простого примитива, точки. В качестве области прицеливания точки используется обычный прямоугольный регион, создаваемый с помощью функции `CreateRectRgn` на основе координат левого верхнего и правого нижнего углов прямоугольника по следующим формулам:

$$x_0 = x - aim_area \quad (7)$$

Здесь: x_0 – координата по оси абсцисс левого верхнего угла прямоугольного региона области прицеливания, x – координата по оси абсцисс точки, aim_area – размер области прицеливания.

$$y_0 = y - aim_area \quad (8)$$

Здесь: y_0 – координата по оси ординат левого верхнего угла прямоугольного региона области прицеливания, y – координата по оси ординат точки, aim_area – размер области прицеливания.

$$x_1 = x + aim_area \quad (9)$$

Здесь: x_1 – координата по оси абсцисс правого нижнего угла прямоугольного региона области прицеливания, x – координата по оси абсцисс точки, aim_area – размер области прицеливания.

$$y_1 = y + aim_area \quad (10)$$

Здесь: y_1 – координата по оси ординат левого верхнего угла прямоугольного региона области прицеливания, y – координата по оси ординат точки, aim_area – размер области прицеливания.

Таким образом, формулы (7)-(10) описывают область прицеливания для точки как квадратный регион со стороной равной $2*aim_area$, в центре которого лежит точка. Обобщенно алгоритм прицеливания для примитива точки можно описать следующим образом: функция `stread` читает структуру примитива точки из файла разработанного формата, после чего по формулам (7)-(10) рассчитываются координаты левого верхнего и правого нижнего углов формируемого прямоугольного региона. Далее на основании полученных координат с помощью функции `CreateRectRgn` формируется прямоугольный регион и вместе с другими данными о примитиве добавляется в двусвязный список с помощью макроса `LIST_ADD`, содержащегося в заголовочном файле `list_helper.h`. Когда пользователь производит нажатие левой кнопки мыши в рабочей области окна, посылается сообщение `WM_LBUTTONDOWN`, в параметре которого `lparam` содержатся координаты нажатия клавиши мыши. Соответственно, обработчик данного сообщения в оконной функции `WndProc` получает координаты клика мыши, умножает или делит их на коэффициент текущего масштаба (определяется глобальной переменной `state`), добавляет к ним глобальные переменные `x_` и `y_`, определяющие координаты верхнего левого угла окна, задаваемые положением ползунков, производит обратное производимому функцией `rescount` преобразование, реализуемое с помощью функции `rev_rescount`, после чего проверяет, попали ли координаты клика мыши в область прицеливания какой-либо из фигур с помощью функции `IsInside`, и если попадание произошло, выводит на экран данные о примитиве, в область прицеливания которого произошло попадание. Вышеописанный алгоритм иллюстрирует блок-схема, приведенная на рис.9.



Рис.9 Схема алгоритма прицеливания для примитива точки

При дальнейшем описании алгоритмов прицеливания для примитивов мы будем опускать сам алгоритм прицеливания (поскольку у всех примитивов он общий) и сосредоточимся на том, как формируется область прицеливания для того или иного примитива.

Теперь рассмотрим, как формируется область прицеливания для следующего примитива – прямой линии. В качестве области прицеливания для прямой в программе использован регион в форме квадрата, центром которого является середина прямой, а сторона которого равна $2 \cdot \text{aim_area}$. Координаты середины прямой вычисляются по следующим формулам:

$$x_{\text{сер}} = \frac{x_{\text{нач}} + x_{\text{кон}}}{2} \quad (11)$$

Здесь: $x_{\text{сер}}$ – координата середины прямой по оси абсцисс, $x_{\text{нач}}$ – координата начала прямой по оси абсцисс, $x_{\text{кон}}$ – координата конца прямой по оси абсцисс.

$$y_{\text{сер}} = \frac{y_{\text{нач}} + y_{\text{кон}}}{2} \quad (12)$$

Здесь: $y_{\text{сер}}$ – координата середины прямой по оси ординат, $y_{\text{нач}}$ – координата начала прямой по оси ординат, $y_{\text{кон}}$ – координата конца прямой по оси ординат.

Соответственно, координаты левого верхнего и правого нижнего углов квадрата с центром в полученной точке вычисляются по формулам (7)-(10).

Проиллюстрируем это следующим куском кода:

```
//Рассчитываем координаты середины прямой
xcenter = (x1 + x) / 2;
ycenter = (y1 + y) / 2;

//Создание квадратного региона с центром в полученной точке
region = CreateRectRgn(xcenter-aim_area, ycenter-aim_area, xcenter+aim_area, ycenter+aim_area);
```

Обобщенный алгоритм прицеливания для примитива прямой линии представлен на рис. 10.



Рис.10 Схема алгоритма прицеливания для примитива прямой линии

Перейдем к следующему примитиву, для которого было реализовано прицеливание – ломаной линии. Область прицеливания ломаной формируется на основе многоугольника, содержащего все ее вершины. Соответственно, попадание внутрь данного многоугольника считается попаданием в область прицеливания ломаной. Поскольку область прицеливания в данном случае формируется исключительно на основе данных о самом векторном примитиве, файл векторной графики не содержит области прицеливания для данного примитива. Отметим два важных нюанса реализации области прицеливания для данного примитива. Во-первых, формирование региона многоугольника осуществляется с помощью функции GDI CreatePolygonRgn, формирующей регион многоугольника на основании массива объектов POINT. Поэтому, перед формированием региона нужно привести прочитанный из файла массив точек к данному виду: исключить недействительные точки на основании значения count, а также сгруппировать отдельные координаты по оси абсцисс и ординат в элементы массива объектов POINT. Во-вторых, отметим, что алгоритм прицеливания для примитива многоугольника полностью аналогичен вышеописанному. Таким образом, на рис. 11 представлен алгоритм прицеливания сразу для двух примитивов: ломаной и многоугольника.

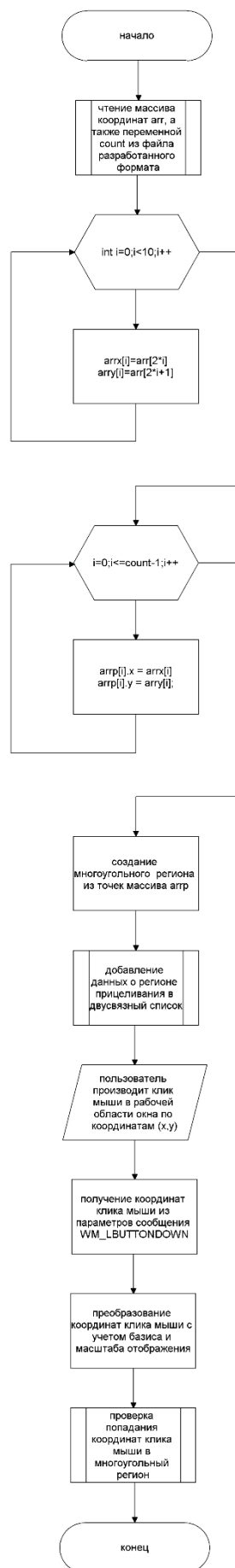


Рис.11 Схема алгоритма прицеливания для примитивов ломаной линии и многоугольника

Перейдем к последнему примитиву, прицеливание для которого было реализовано в разработанной программе – окружности. В качестве области прицеливания для окружности используется окружность с центром в той же точке, и радиусом, равным сумме радиуса примитива окружности и величины aim_area. Само формирование окружности производится с помощью функции CreateEllipticRgn на основе координат левого верхнего и правого нижнего углов прямоугольника, в который вписан формируемый эллипс[10]. Формированию области прицеливания в исходном коде соответствует следующая строчка:

```
//создание объекта HRGN  
region = CreateEllipticRgn(xcenter-radius-  
aim_area,ycenter+radius+aim_area,xcenter+radius+aim_area,ycenter-radius-  
aim_area);
```

Полностью алгоритм работы прицеливания для примитива окружности проиллюстрирован в виде блок-схемы на рис. 12.

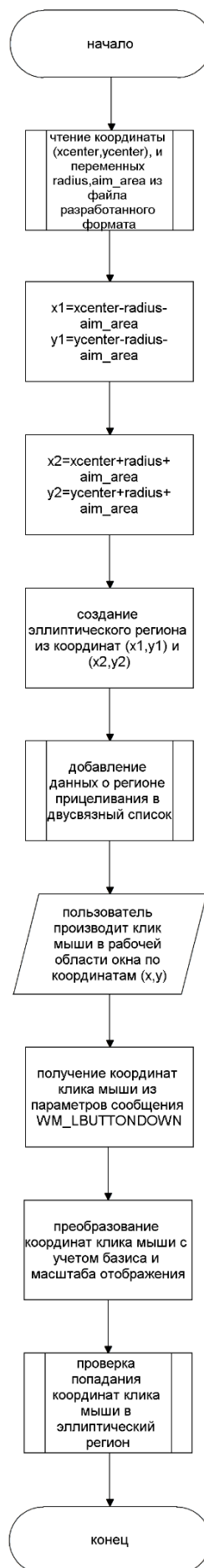


Рис.12 Схема алгоритма прицеливания для примитива окружности

3. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ

3.1. Реализация программы-редактора

Приложение реализовано средствами языка программирования C++ с использованием структурного подхода, отладка, компиляция и компоновка программы производились средствами IDE Microsoft Visual Studio для платформы Win32. Приложение имеет следующую структуру:

- Основной модуль `testpr.cpp` содержит функцию `main`, описывающую интерфейс интерактивного консольного меню, открытия файла, чтения и редактирования его данных. Реализует функции создания файла с данными векторного слоя графики и изменения таких файлов.

- Вспомогательный модуль `ndllproj.h` содержит описания структур векторных примитивов, используемых библиотекой, а также прототипов функций DLL-библиотеки `ndllproj.dll`, что необходимо для использования функций данной библиотеки в исходном коде модуля `testpr.cpp`.

- Вспомогательный модуль `ndllproj.cpp` содержит описание функций работы с файлами векторной графики и содержит функции для их чтения, редактирования, а также отображения графических данных из файлов такого формата на экране. Часть функций данного модуля используется основным модулем `testpr.cpp`

- Вспомогательный модуль `list_helper.h` содержит описание макросов для работы с двусвязным списком, в частности добавления и удаления элементов из него. Подключается в вспомогательный модуль `ndllproj.cpp`, для поддержки работы функций которого необходимы макросы для работы с элементами двусвязного списка.

Результатами компиляции выступают два объектных файла – результат компиляции основного модуля и результат компиляции вспомогательного

модуля `ndllproj.cpp`. Результатом сборки основного модуля выступает исполняемый файл редактора файла разработанного формата, а результатом сборки вспомогательного модуля выступает DLL-библиотека, необходимая для работы редактора. Приложение не требует установки, достаточно наличие на компьютере пользователя исполнимого файла и DLL-библиотеки, содержащей методы для работы со слоем векторной графики.

3.2. Реализация демонстрационной программы

Приложение реализовано средствами языка программирования C++ с использованием структурного подхода, отладка, компиляция и компоновка программы производились средствами IDE Microsoft Visual Studio для платформы Win32. Приложение имеет следующую структуру:

- Основной модуль `example1.cpp` содержит функцию `WinMain`, описывающую точку входа в оконное приложение, и оконную функцию `WndProc`, описывающую основной алгоритм работы приложения. Реализует функции отображения векторного слоя графики поверх слоя растровой, масштабирование отображаемых данных, а также прицеливание.

- Вспомогательный модуль `ndllproj.h` содержит описания структур векторных примитивов, используемых библиотекой, а также прототипов функций DLL-библиотеки `ndllproj.dll`, что необходимо для использования функций данной библиотеки в исходном коде модуля `example1.cpp`.

- Вспомогательный модуль `ndllproj.cpp` содержит описание функций работы с файлами векторной графики и содержит функции для их чтения, редактирования, а также отображения графических данных из файлов такого формата на экране. Часть функций данного модуля используется основным модулем `example1.cpp`

- Вспомогательный модуль `list_helper.h` содержит описание макросов для работы с двусвязным списком, в частности добавления и удаления элементов из него. Подключается в вспомогательный модуль `ndllproj.cpp`, а также в основной модуль `example1.cpp`, для поддержки работы функций которых необходимы макросы для работы с элементами двусвязного списка.

Результатами компиляции выступают два объектных файла – результат компиляции основного модуля и результат компиляции вспомогательного модуля `ndllproj.cpp`. Результатом сборки основного модуля выступает исполняемый файл демонстрационного приложения, а результатом сборки вспомогательного модуля выступает DLL-библиотека, необходимая для работы приложения. Приложение не требует установки, достаточно наличие на компьютере пользователя исполнимого файла и DLL-библиотеки, содержащей методы для работы со слоем векторной графики, а также установленного пакета Microsoft Visual C++ 10.0.

3.3. Тестирование программы-редактора

Тестирование проводится в соответствии с ГОСТ 19.301-79 «Программа и методика испытаний».

Объект испытания: файл `testpr.exe`.

Цель испытания: проверка работоспособности при различных входных данных.

Средства проверки: персональный компьютер с процессором AMD A6-7310 APU, графической картой AMD Radeon R4 Graphics, 4Гб ОЗУ, ОС Windows 10 Home в стандартной комплектации ПО, 200 Гб свободной памяти.

Порядок испытаний: последовательная проверка работоспособности различных команд в составе приложения.

Методы испытаний: испытания проводятся методом функционального тестирования в нормальных и исключительных случаях.

Результаты тестирования программы-редактора представлены в табл. 2.

Таблица 2

Результаты тестирования программы-редактора

Условие	Объект тестирования	Входные данные	Выходные данные
Нормальная работа	Создание файла	Нормальные входные данные при выборе опции создания файла (Рис.13)	Нормальный вывод при успешном создании файла (Рис. 13)
	Редактирование примитива файла	Нормальные входные данные при выборе опции редактирования примитива файла (Рис.14)	Нормальный вывод при успешном редактировании примитива файла (Рис. 14)
	Завершение работы программы	Выбор опции завершения работы программы	Нормальное завершение работы программы
	Удаление примитива из файла	Нормальные входные данные при выборе опции удаления примитива файла (Рис.15)	Нормальный вывод при успешном удалении примитива файла (Рис. 15)
Выбор пункта, отсутствующего в интерактивном меню	Выбор действия	Нажатие клавиши «5» (пункт, который отсутствует в меню)	Вывод сообщения об ошибке: «В меню отсутствует пункт с таким номером!» (Рис.16)

Продолжение Таблицы 2

Условие	Объект тестирования	Входные данные	Выходные данные
Выбор пункта редактирования файла и ввод имени несуществующего файла	Ввод имени файла, примитив которого редактируется	Нажатие клавиши «2» (выбор пункта редактирования файла) и ввод «f» (имя несуществующего файла)	Вывод сообщения об ошибке: «Не удалось открыть файл!» (Рис.17)
Выбор пункта редактирования файла и ввод номера несуществующего примитива	Ввод номера примитива, подлежащего редактированию	Нажатие клавиши «2» (выбор пункта редактирования файла), ввод «newfile», и ввод «3» (номер редактируемого примитива)	Вывод сообщения об ошибке: «Неверный номер редактируемого примитива!» (Рис.18)
Выбор пункта удаления примитива файла и ввод имени несуществующего файла	Ввод имени файла, примитив которого удаляется	Нажатие клавиши «4» (выбор пункта удаления примитив файла) и ввод «f» (имя несуществующего файла)	Вывод сообщения об ошибке: «Не удалось открыть файл!» (Рис.19)
Выбор пункта удаления примитива и неверный ввод при выборе операции с примитивом	Выбор удалить или сохранить примитив файла	Нажатие клавиши «2» (выбор пункта редактирования файла), ввод «newfile», и ввод «h» (неверный ответ на вопрос о редактировании примитива)	Вывод сообщения об ошибке: «Неверный ввод! Файл не изменен!» (Рис.20)

Продолжение Таблицы 2

Условие	Объект тестирования	Входные данные	Выходные данные
Выбор пункта редактирования файла и ввод имени пустого файла	Ввод имени файла, примитив которого редактируется	Нажатие клавиши «2» (выбор пункта редактирования файла) и ввод «myfile» (имя пустого файла)	Вывод сообщения об ошибке: «Файл пуст!» (Рис.21)
Выбор пункта удаления примитивов файла и ввод имени пустого файла	Ввод имени файла, примитивы которого удаляются	Нажатие клавиши «4» (выбор пункта редактирования файла) и ввод «myfile» (имя пустого файла)	Вывод сообщения об ошибке: «Файл пуст!» (Рис.22)

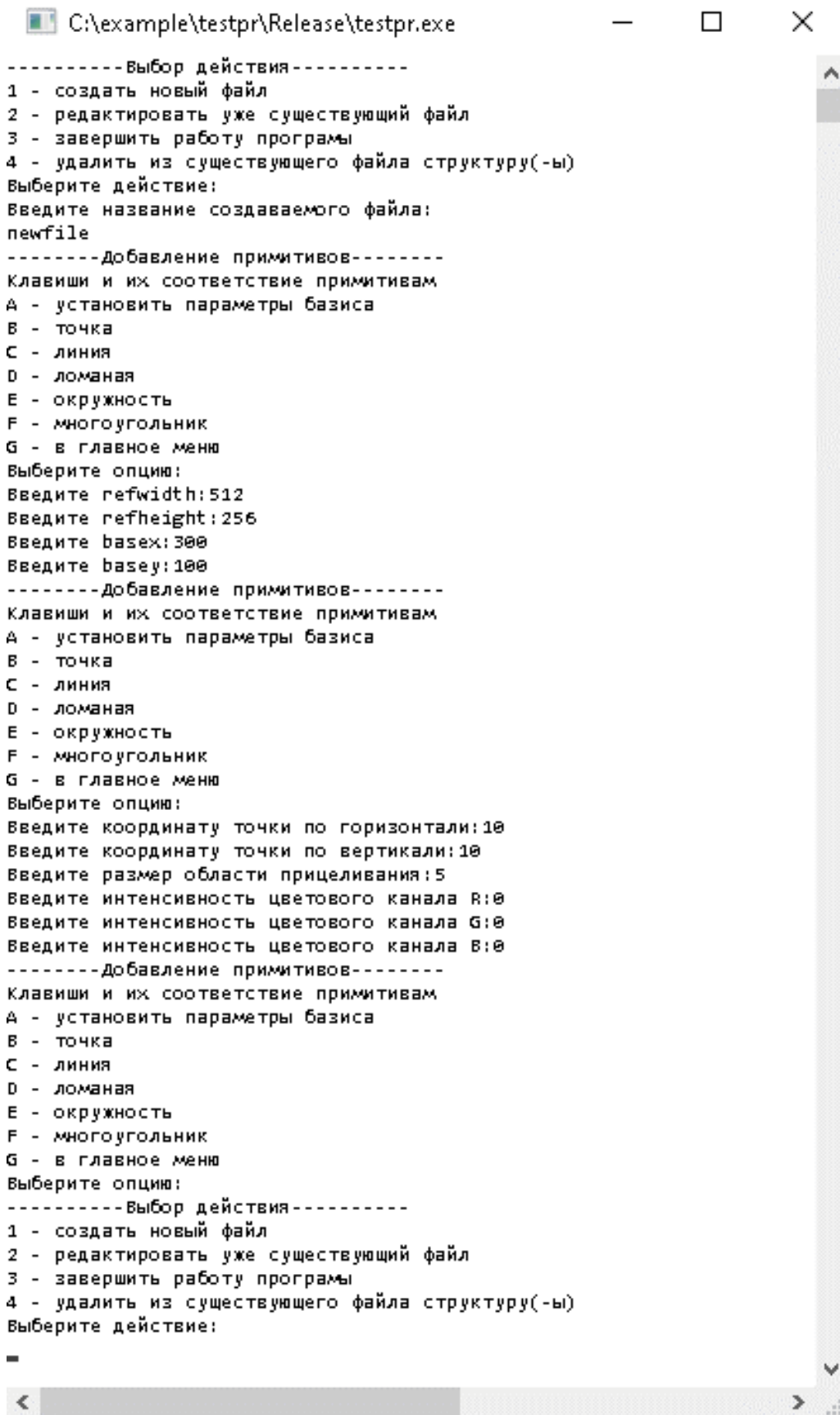
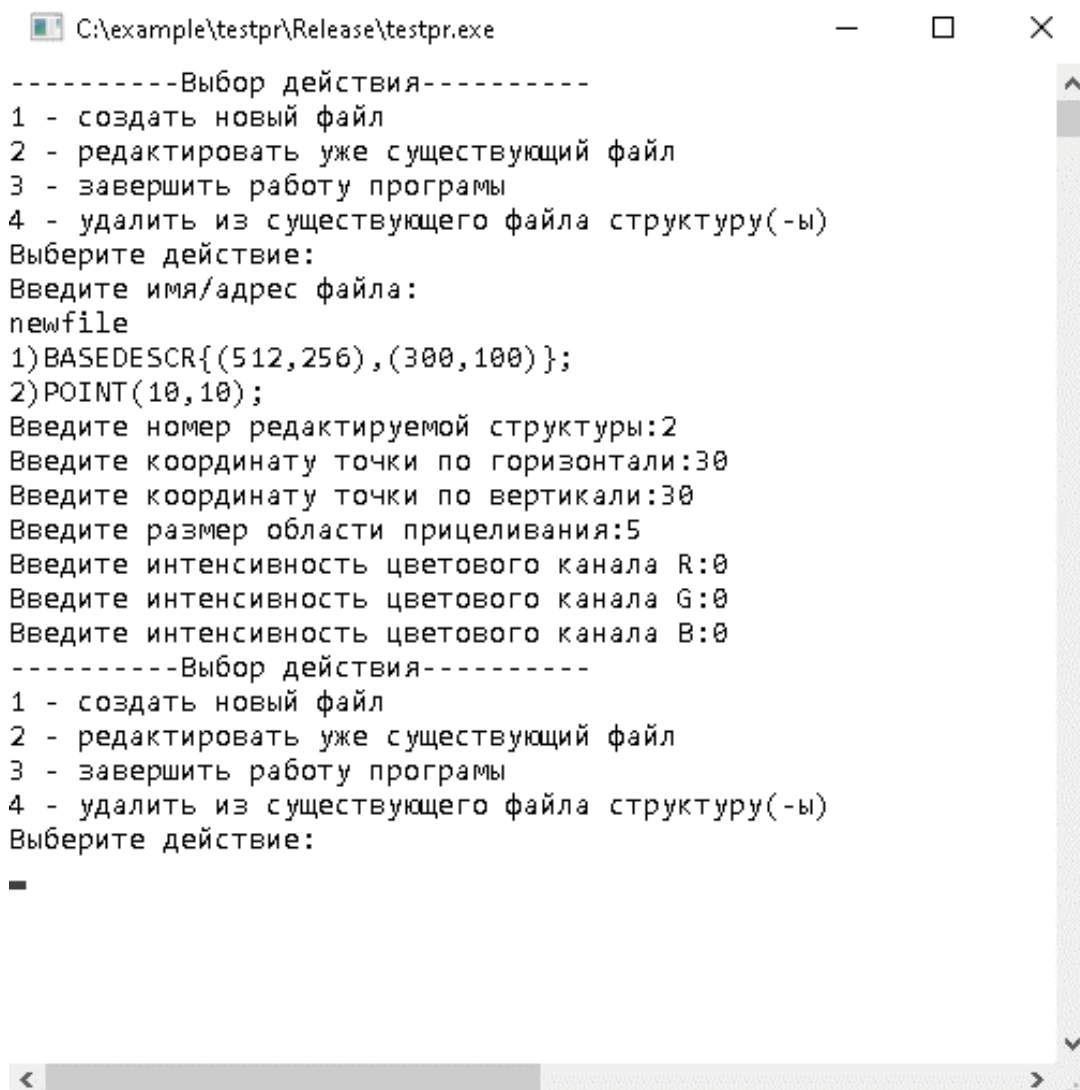


Рис. 13 Работа программы при нормальных входных данных и выборе опции создания файла



```
C:\example\testpr\Release\testpr.exe

-----Выбор действия-----
1 - создать новый файл
2 - редактировать уже существующий файл
3 - завершить работу программы
4 - удалить из существующего файла структуру(-ы)
Выберите действие:
Введите имя/адрес файла:
newfile
1)BASEDESCR{(512,256),(300,100)};
2)POINT(10,10);
Введите номер редактируемой структуры:2
Введите координату точки по горизонтали:30
Введите координату точки по вертикали:30
Введите размер области прицеливания:5
Введите интенсивность цветового канала R:0
Введите интенсивность цветового канала G:0
Введите интенсивность цветового канала B:0
-----Выбор действия-----
1 - создать новый файл
2 - редактировать уже существующий файл
3 - завершить работу программы
4 - удалить из существующего файла структуру(-ы)
Выберите действие:
■
```

Рис. 14 Работа программы при нормальных входных данных и выборе опции редактирования примитива файла

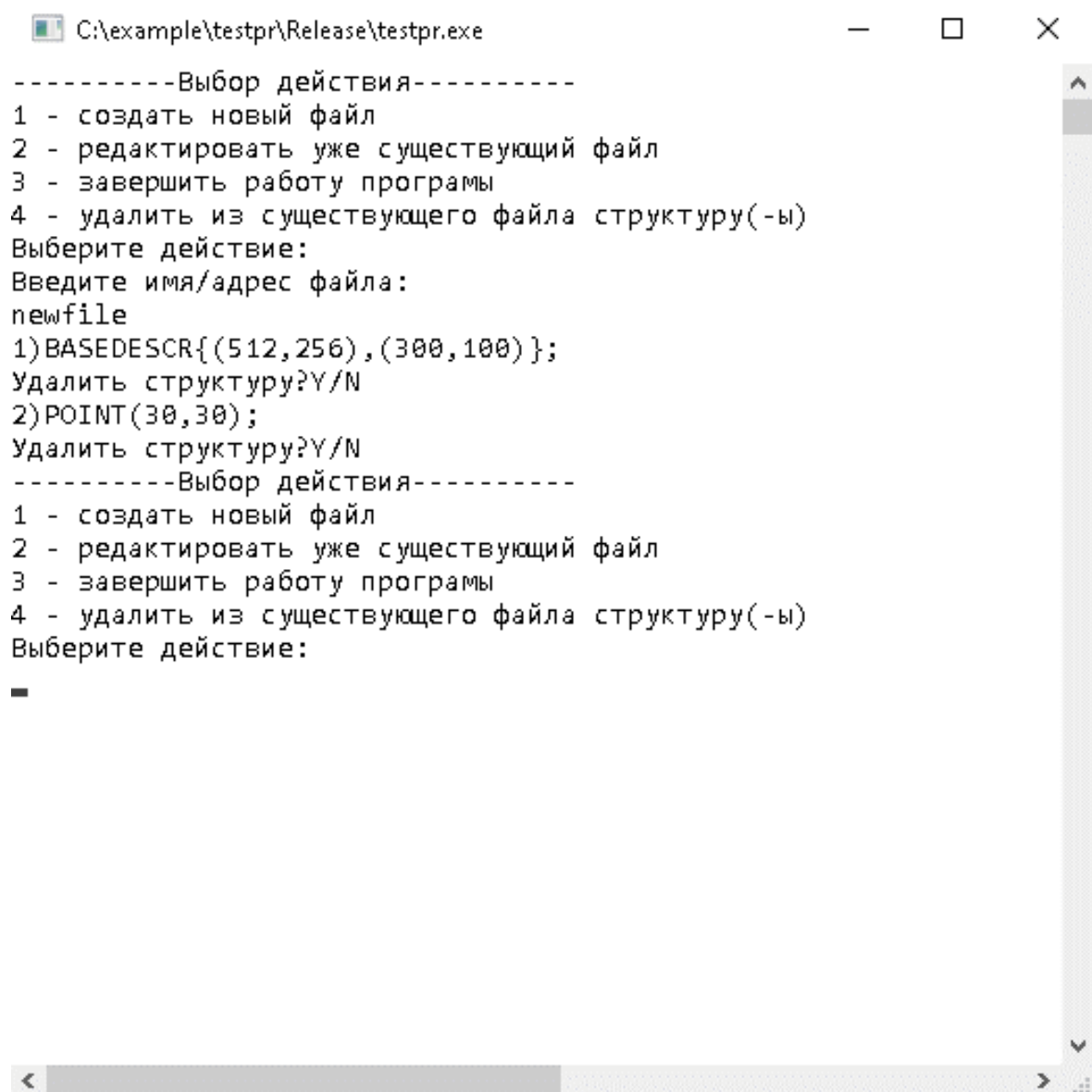


Рис. 15 Работа программы при нормальных входных данных и выборе опции удаления примитива файла

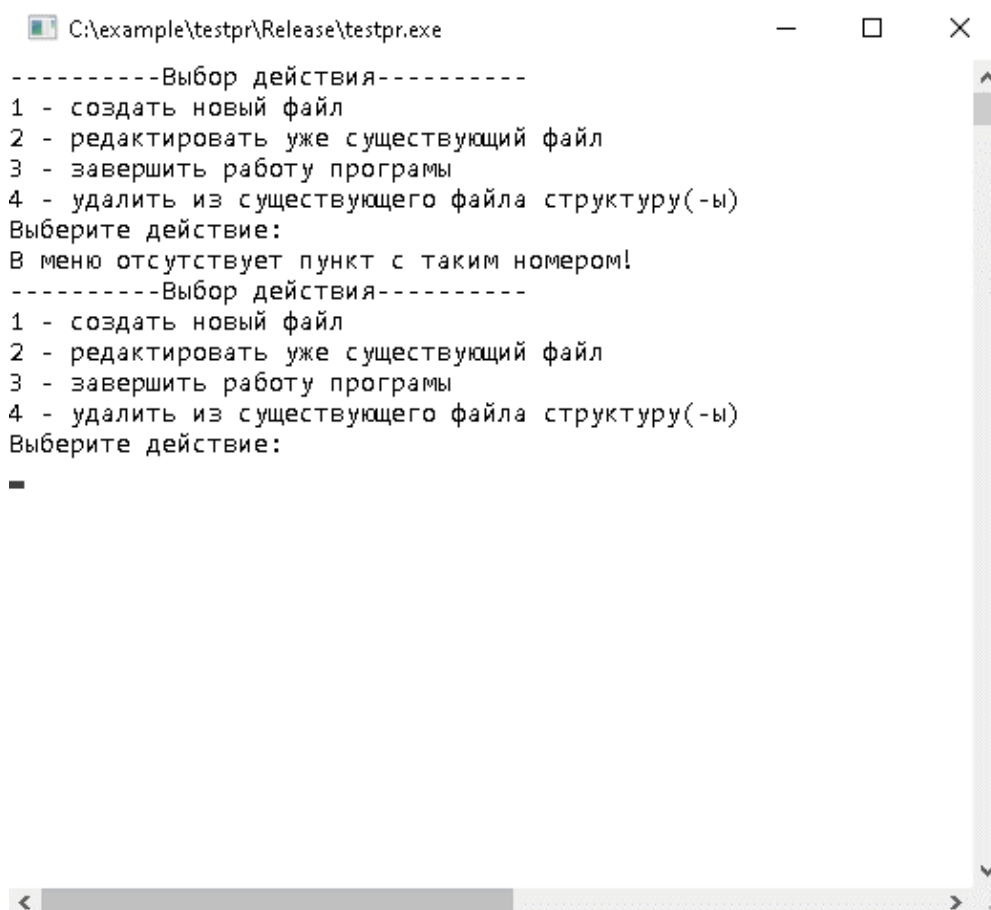


Рис. 16 Работа программы при выборе пункта, отсутствующего в интерактивном меню

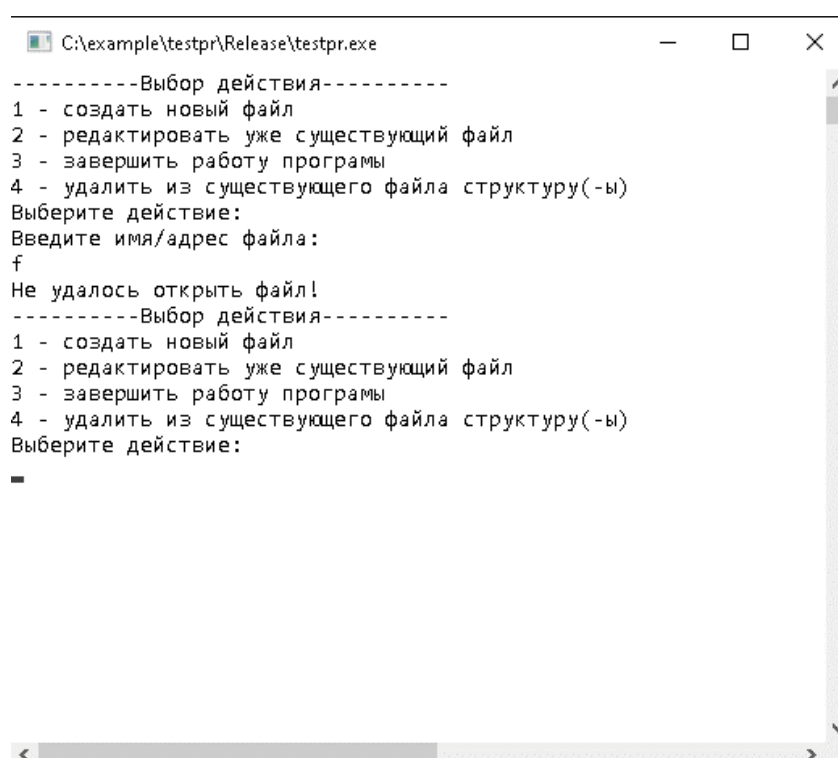
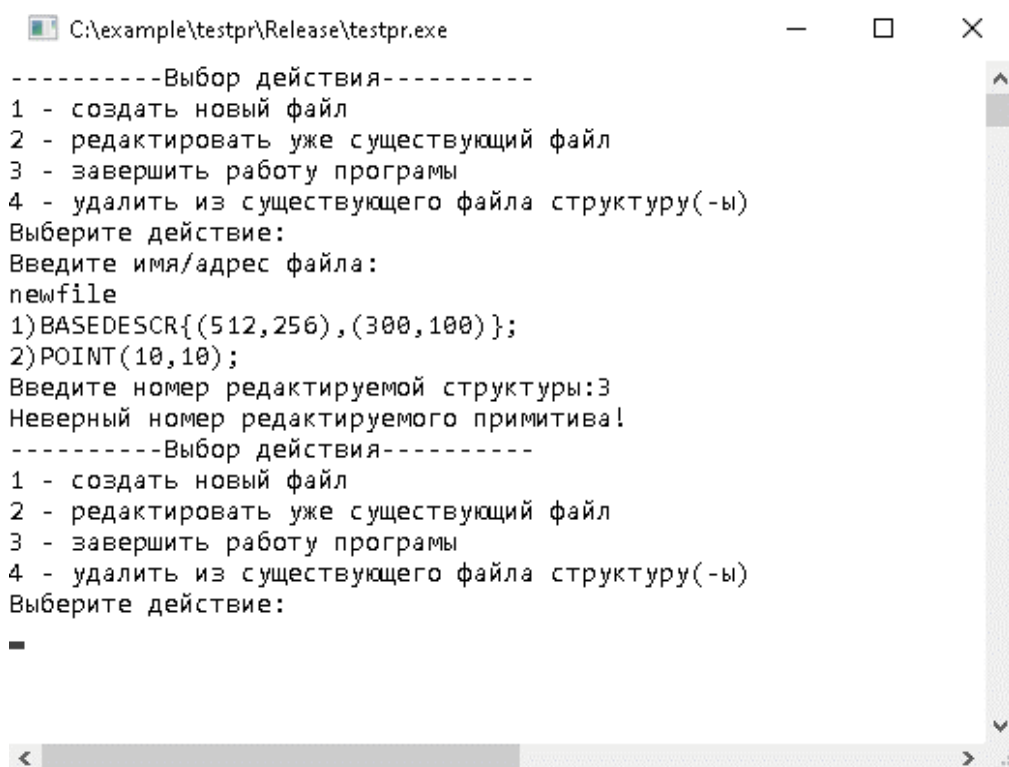


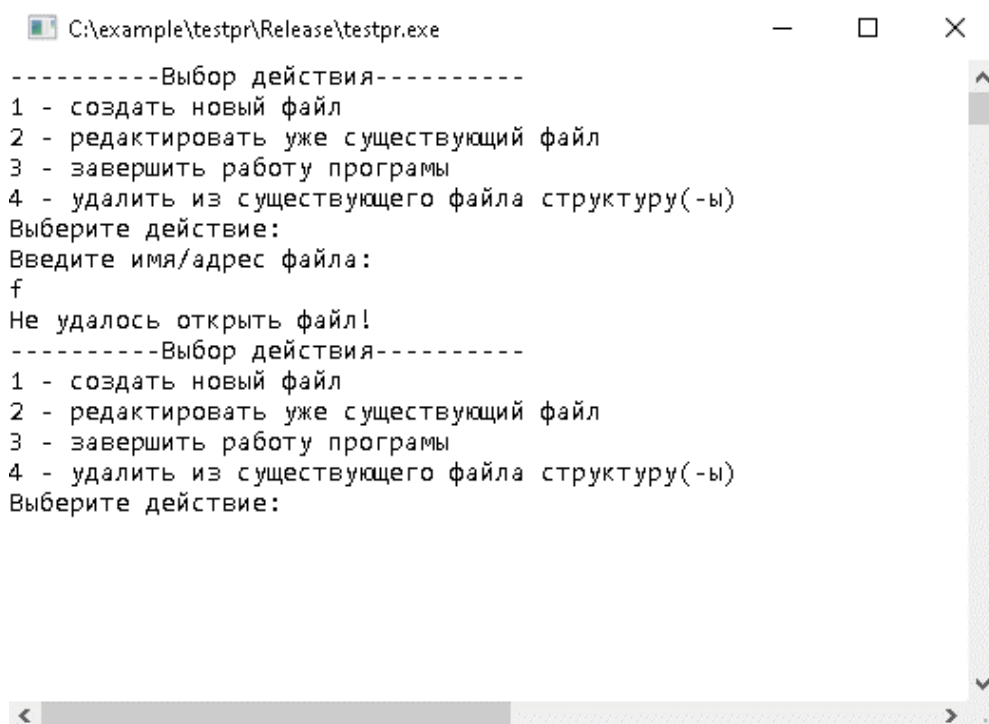
Рис. 17 Работа программы при попытке отредактировать примитив несуществующего файла



```
C:\example\testpr\Release\testpr.exe

-----Выбор действия-----
1 - создать новый файл
2 - редактировать уже существующий файл
3 - завершить работу программы
4 - удалить из существующего файла структуру(-ы)
Выберите действие:
Введите имя/адрес файла:
newfile
1)BASEDESCR{(512,256),(300,100)};
2)POINT(10,10);
Введите номер редактируемой структуры:3
Неверный номер редактируемого примитива!
-----Выбор действия-----
1 - создать новый файл
2 - редактировать уже существующий файл
3 - завершить работу программы
4 - удалить из существующего файла структуру(-ы)
Выберите действие:
newfile
```

Рис. 18 Работа программы при попытке отредактировать несуществующий примитив файла



```
C:\example\testpr\Release\testpr.exe

-----Выбор действия-----
1 - создать новый файл
2 - редактировать уже существующий файл
3 - завершить работу программы
4 - удалить из существующего файла структуру(-ы)
Выберите действие:
Введите имя/адрес файла:
f
Не удалось открыть файл!
-----Выбор действия-----
1 - создать новый файл
2 - редактировать уже существующий файл
3 - завершить работу программы
4 - удалить из существующего файла структуру(-ы)
Выберите действие:
f
```

Рис. 19 Работа программы при попытке удаления примитивов несуществующего файла

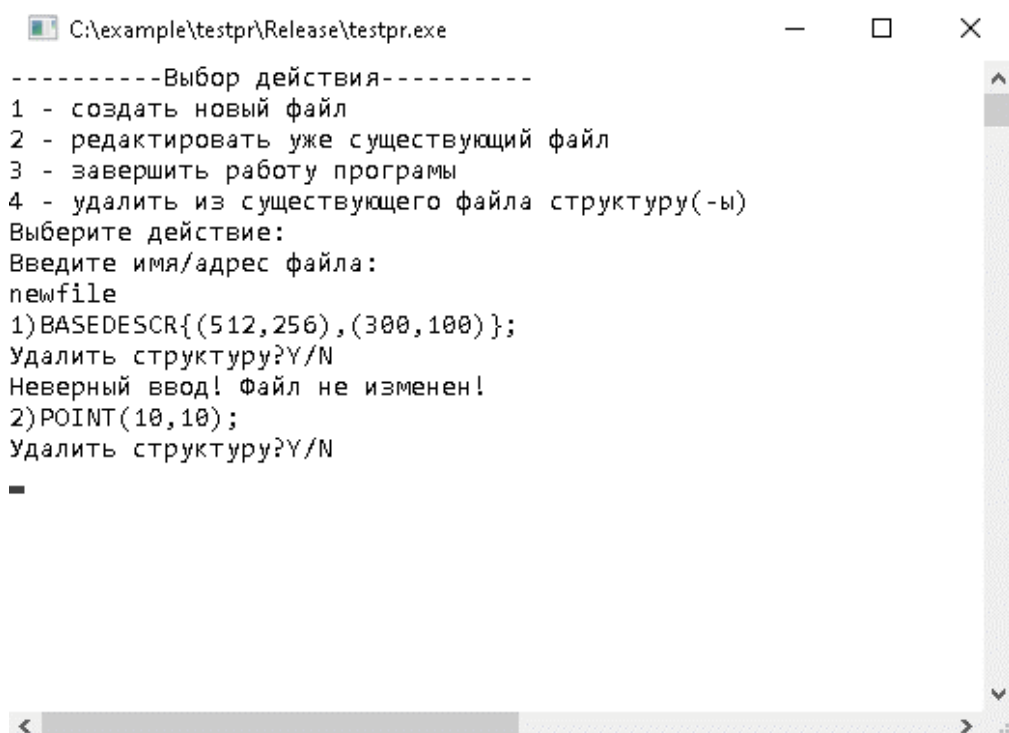


Рис. 20 Работа программы при удалении примитивов из файла и неверном выборе операции

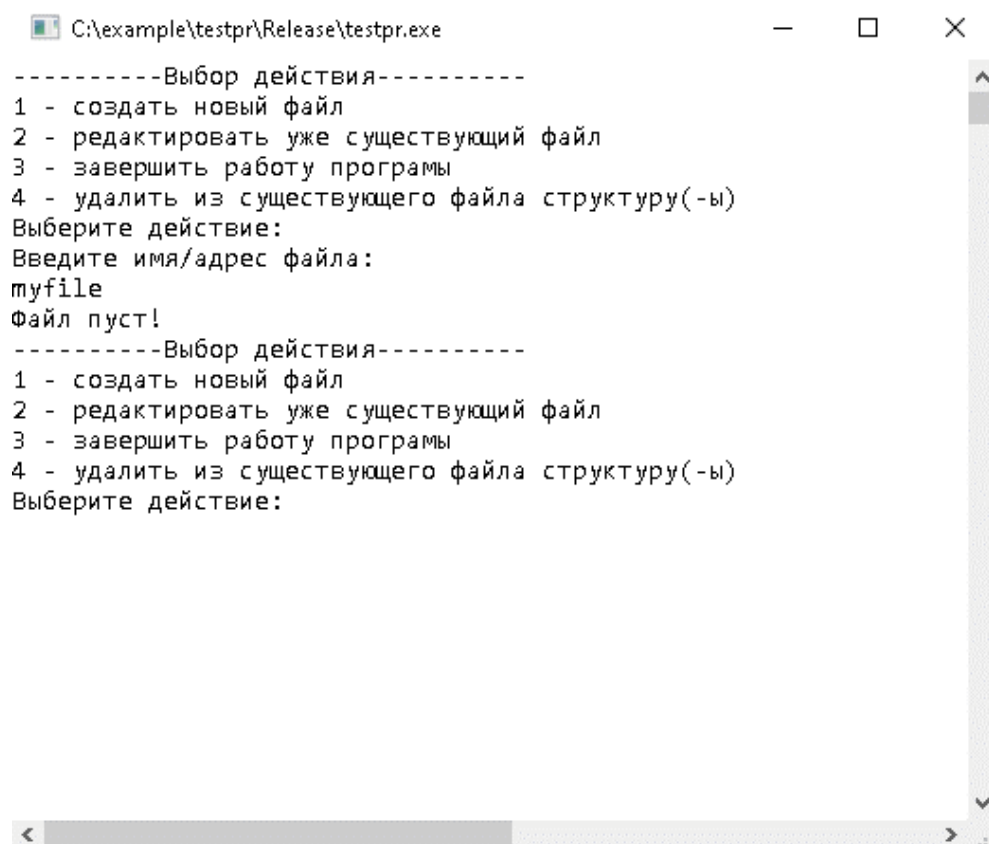


Рис. 21 Работа программы при редактировании примитива пустого файла

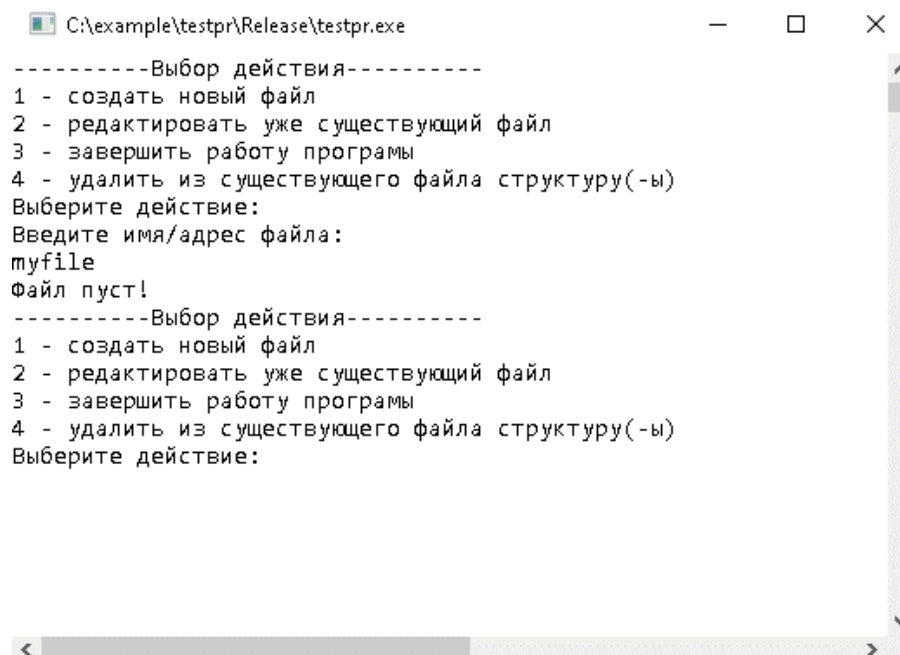


Рис. 22 Работа программы при удалении примитивов пустого файла

3.4. Тестирование демонстрационной программы

Тестирование проводится в соответствии с ГОСТ 19.301-79 «Программа и методика испытаний».

Объект испытания: файл example1.exe.

Цель испытания: проверка работоспособности при различных входных данных.

Средства проверки: персональный компьютер с процессором AMD A6-7310 APU, графической картой AMD Radeon R4 Graphics, 4Гб ОЗУ, ОС Windows 10 Home в стандартной комплектации ПО, 200 Гб свободной памяти.

Порядок испытаний: последовательная проверка работоспособности различных команд в составе приложения.

Методы испытаний: испытания проводятся методом функционального тестирования в нормальных и исключительных случаях.

Результаты тестирования программы-редактора представлены в табл. 3.

Таблица 3

Результаты тестирования программы-редактора

Условие	Объект тестирования	Входные данные	Выходные данные
Нормальная работа	Отображение векторного слоя графики поверх растровой подложки	example1 - myfile.bmp -myfile	Нормальный вывод при успешном открытии файлов: отображение векторных данных на растровой подложке (Рис. 23)
	Сглаживание отображаемых векторных данных	Запуск программы: example1 - myfile.bmp –myfile Нажатие клавиши «А» (включение сглаживания)	Нормальный вывод при успешном включении сглаживания: отображение сглаженных векторных данных на растровой подложке (Рис. 24)
	Прицеливание для векторного слоя графики	Запуск программы: example1 - myfile.bmp –myfile Клик мыши в области прицеливания одного из примитивов (прямой линии)	Нормальный вывод при успешном попадании координат клика мыши в область прицеливания примитива (Рис. 25)

Продолжение Таблицы 3

Условие	Объект тестирования	Входные данные	Выходные данные
Нормальная работа	Изменение масштаба отображения исходных данных (масштаб – 200%)	Запуск программы: example1 - myfile.bmp –myfile Нажатие клавиши «М» (увеличение масштаба отображения)	Нормальный вывод при успешном увеличении масштаба отображения до 200% (Рис. 26)
	Изменение масштаба отображения исходных данных (масштаб – 300%)	Запуск программы: example1 - myfile.bmp –myfile Двукратное нажатие клавиши «М» (увеличение масштаба отображения)	Нормальный вывод при успешном увеличении масштаба отображения до 300% (Рис. 27)
	Изменение масштаба отображения исходных данных (масштаб – 50%)	Запуск программы: example1 - myfile.bmp –myfile Нажатие клавиши «L» (уменьшение масштаба отображения)	Нормальный вывод при успешном уменьшении масштаба отображения до 50% (Рис. 28)
	Изменение масштаба отображения исходных данных (масштаб – 33%)	Запуск программы: example1 - myfile.bmp –myfile Двукратное нажатие клавиши «L» (уменьшение масштаба отображения)	Нормальный вывод при успешном уменьшении масштаба отображения до 33% (Рис. 29)

Продолжение Таблицы 3

Условие	Объект тестирования	Входные данные	Выходные данные
	Работа ползунков при масштабе отображения 200%	Запуск программы: example1 - myfile.bmp –myfile Нажатие клавиши «М» (увеличение масштаба отображения) Перемещение обоих ползунков в случайную позицию	Нормальный вывод при успешном изменении позиции ползунков в масштабе 200% (Рис. 30)
	Работа ползунков при масштабе отображения 300%	Запуск программы: example1 - myfile.bmp –myfile Двукратное нажатие клавиши «М» (увеличение масштаба отображения) Перемещение обоих ползунков в случайную позицию	Нормальный вывод при успешном изменении позиции ползунков в масштабе 300% (Рис. 31)
Вызов программы без параметров	Открытие файлов разработанного формата	Запуск программы: example1 (без аргументов)	Вывод сообщения об ошибке: «Command line arguments are not set!» (Рис. 32)

Продолжение Таблицы 3

Условие	Объект тестирования	Входные данные	Выходные данные
Вызов программы только с одним из параметров	Открытие файлов разработанного формата	Запуск программы: example1 – myfile.bmp (один из аргументов)	Вывод сообщения об ошибке: «Command line arguments are not set!» (Рис. 33)
Файл растровой графики по переданному адресу отсутствует	Открытие файлов разработанного формата	Запуск программы: example1 - myfile.bmp –myfile (myfile.bmp отсутствует)	Вывод сообщения об ошибке: «Unable to open bitmap file!» (Рис. 34)
Файл векторной графики по переданному адресу отсутствует	Открытие файлов разработанного формата	Запуск программы: example1 - myfile.bmp –myfile (myfile отсутствует)	Вывод сообщения об ошибке: «Unable to open vector file!» (Рис. 35)

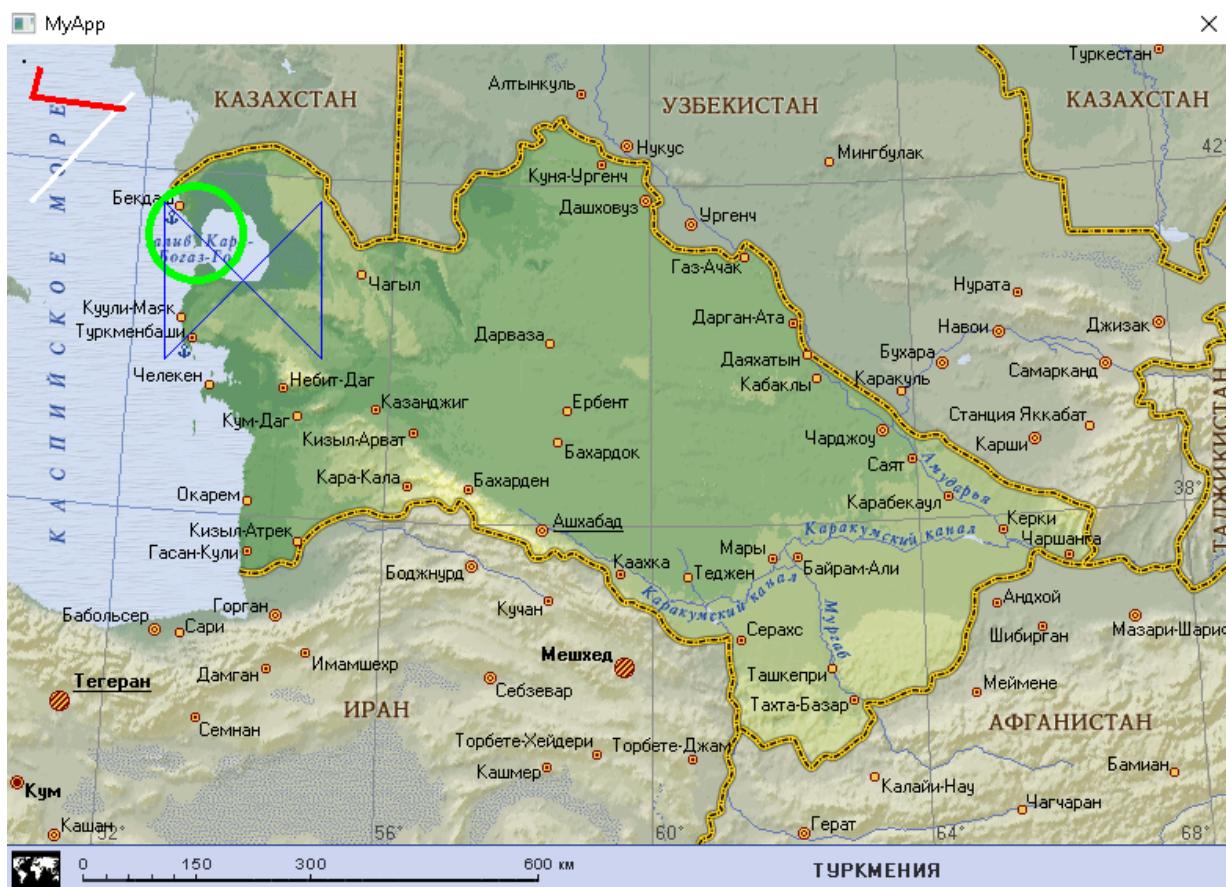


Рис. 23 Работа программы при нормальных входных данных и тестировании функции отображения векторных данных поверх растровой подложки



Рис. 24 Работа программы при нормальных входных данных и тестировании функции сглаживания отображаемых векторных данных

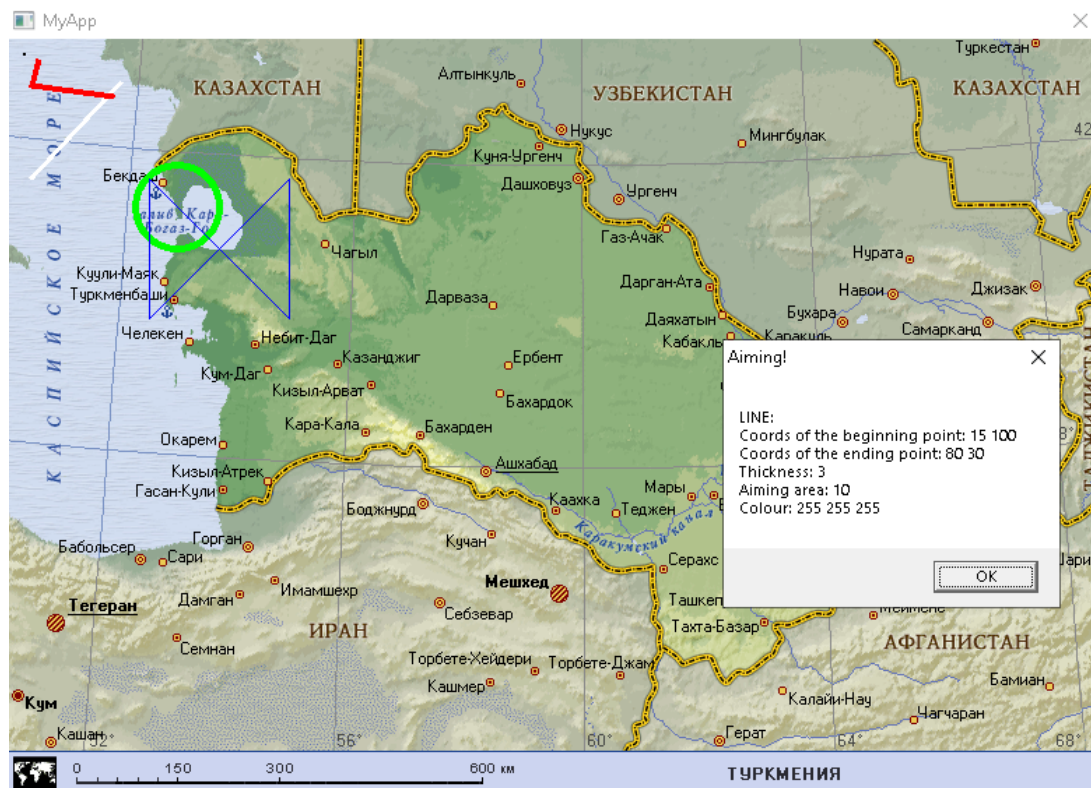


Рис. 25 Работа программы при нормальных входных данных и тестировании функции прицеливания отображаемых векторных данных

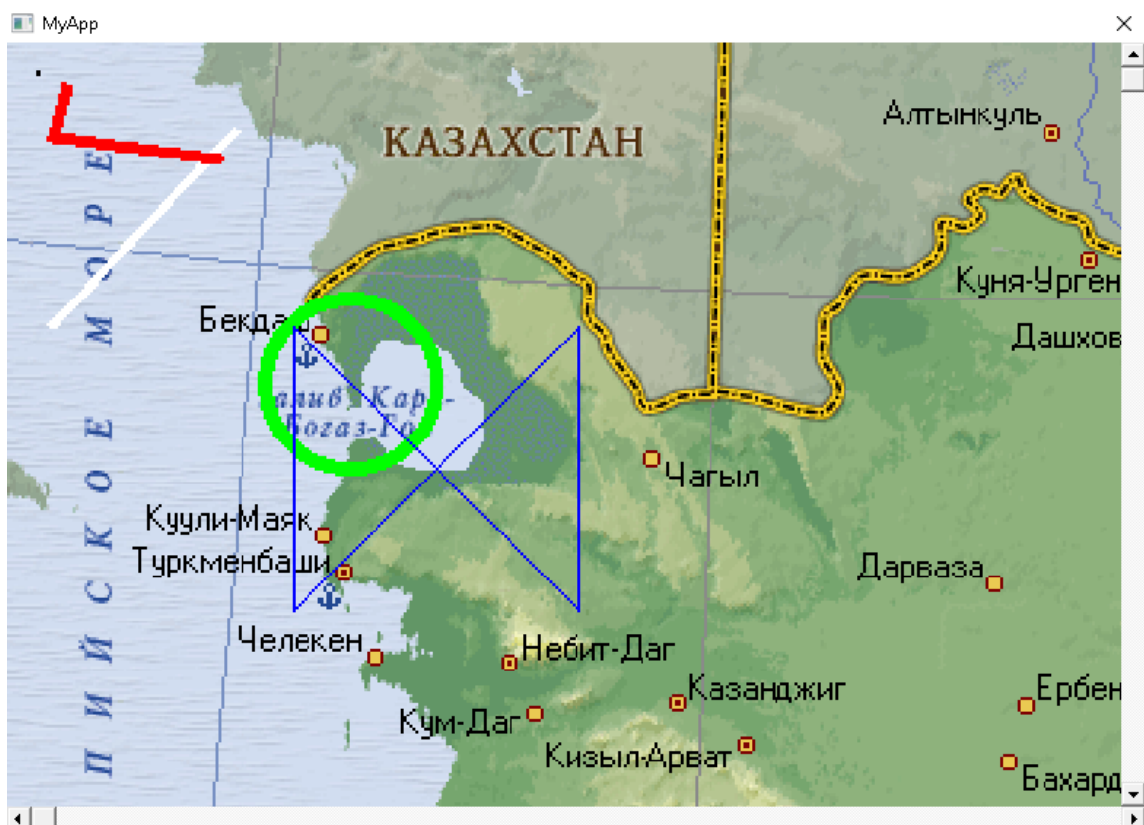


Рис. 26 Работа программы при нормальных входных данных и тестировании функции изменения масштаба отображения (масштаб – 200%)

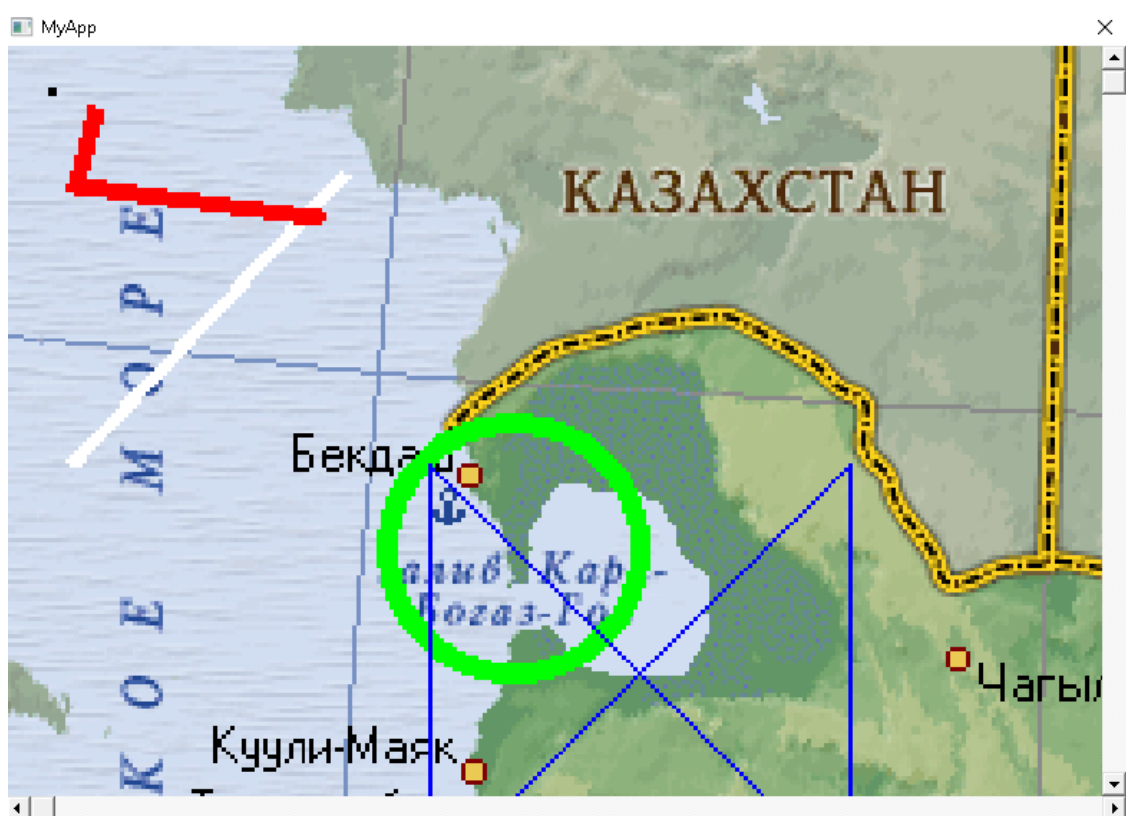


Рис. 27 Работа программы при нормальных входных данных и тестировании функции изменения масштаба отображения (масштаб – 300%)

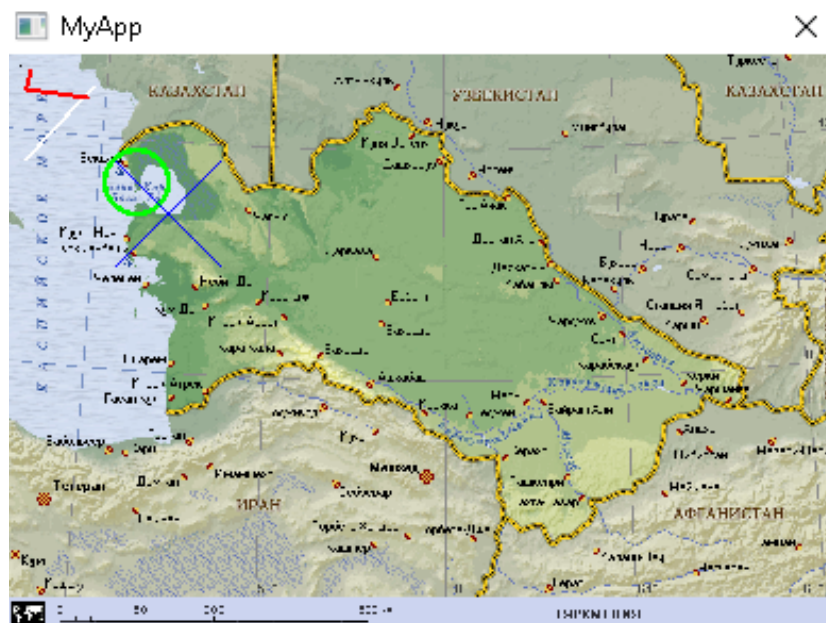


Рис. 28 Работа программы при нормальных входных данных и тестировании функции изменения масштаба отображения (масштаб – 50%)

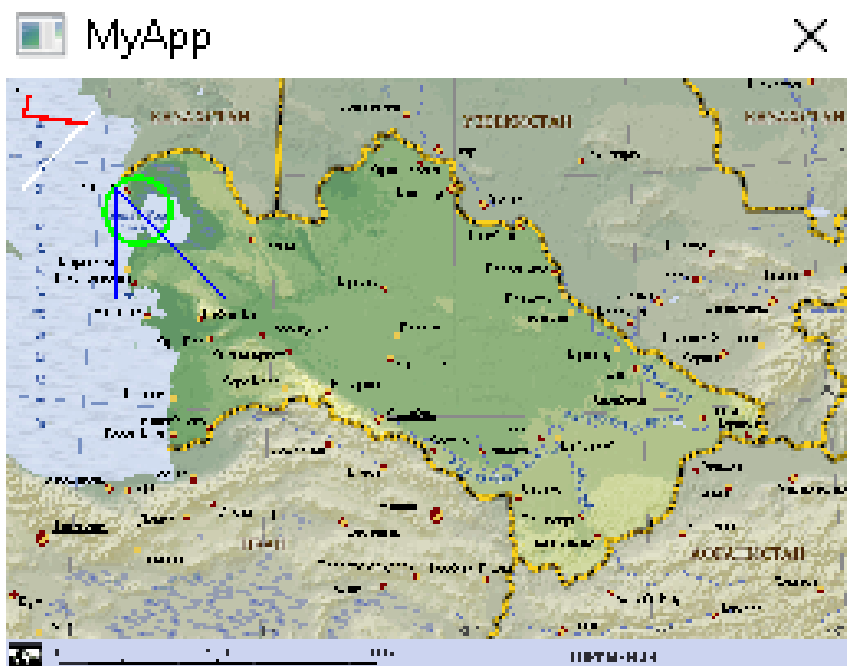


Рис. 29 Работа программы при нормальных входных данных и тестировании функции изменения масштаба отображения (масштаб – 33%)



Рис. 30 Нормальный вывод при успешном изменении позиции ползунков (масштаб – 200%)



Рис. 31 Нормальный вывод при успешном изменении позиции ползунков (масштаб – 300%)

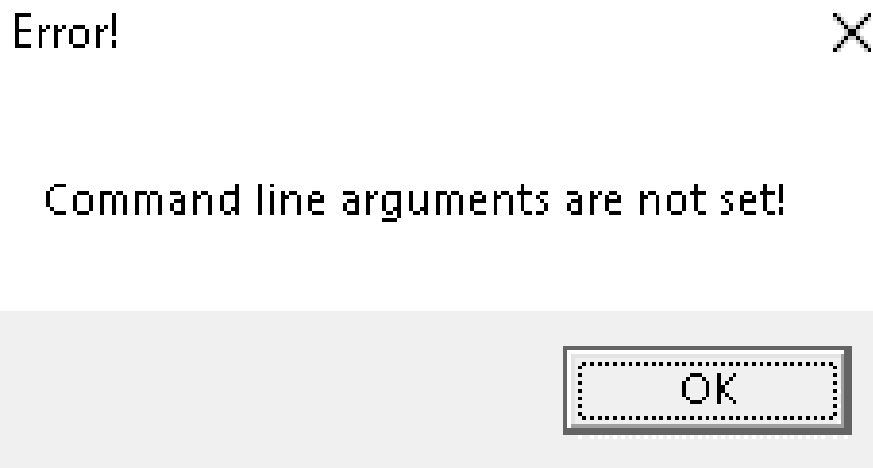


Рис. 32 Работа программы при вызове без аргументов

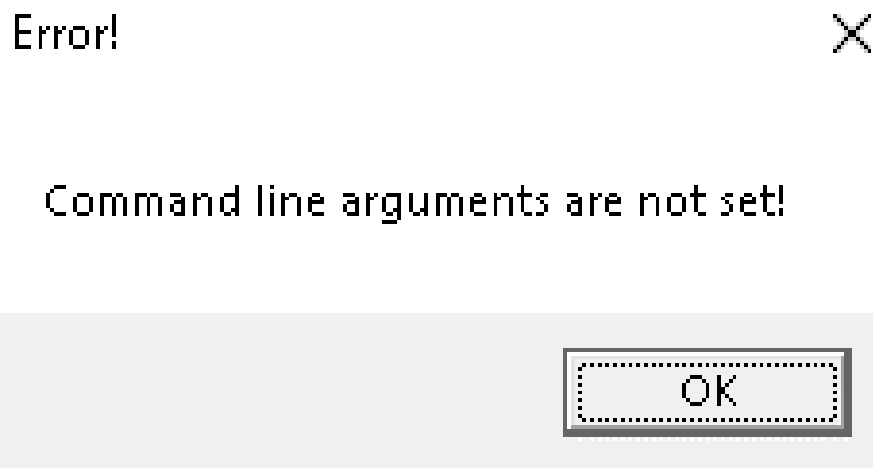


Рис. 33 Работа программы при вызове с одним из аргументов

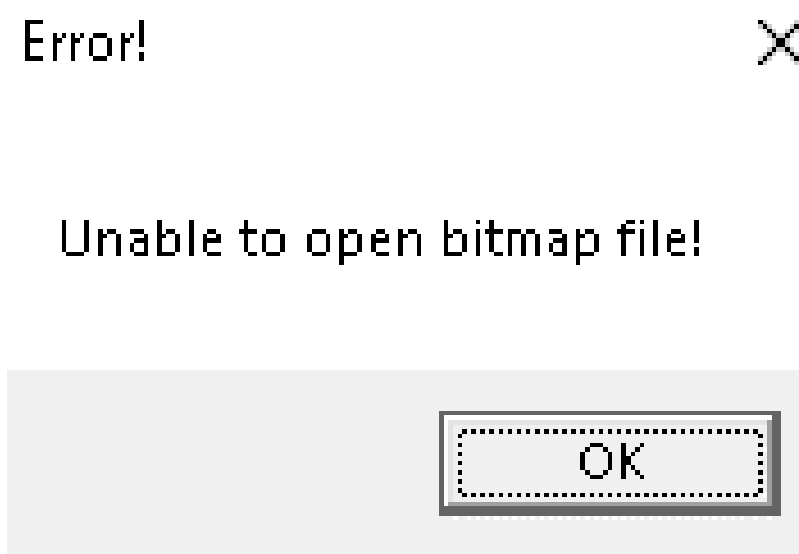


Рис. 34 Работа программы при отсутствии файла растровой графики

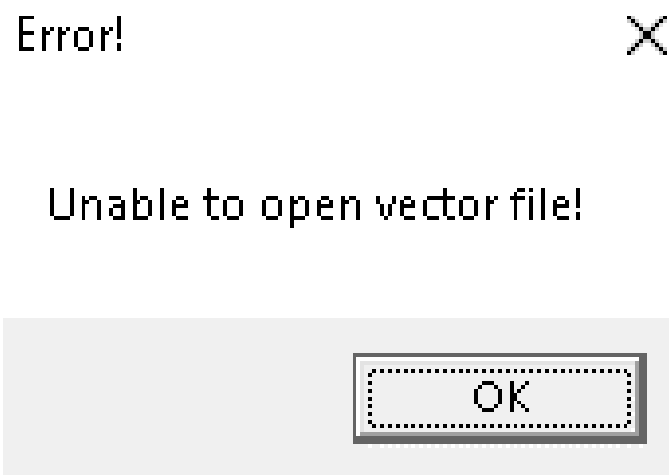


Рис. 35 Работа программы при отсутствии файла векторной графики

ЗАКЛЮЧЕНИЕ

Целью данной работы являлась разработка программного инструментария для работы с расположением объектов на плоскости.

Реализованный программный комплекс решает задачи хранения векторной графики, ее редактирования, а также отображения слоя векторной графики на растровой подложке.

Редактор файлов разработанного формата реализует функции создания файлов разработанного формата, добавления в них примитивов, редактирования и удаления примитивов из файла разработанного формата, вывод данных о примитивах на экран в текстовом виде.

Демонстрационное приложение реализует функции отображения слоя векторной графики поверх растровой подложки, переключения режима сглаживания отображаемых векторных данных, масштабирования отображаемых данных, а также вывода данных о векторных примитивах при попадании координат клика мыши в область прицеливания одной из фигур.

Отметим, что структура базиса позволяет реализовывать гибкую настройку слоя векторной графики, координаты примитивов которого подстраиваются под различные размеры используемых растровых подложек, а использование базисных координат позволяет реализовать сдвиг всей графики векторного слоя в любом направлении без редактирования данных каждого примитива. Также разработанный формат косвенным образом позволяет реализовать систему приоритетов разных примитивов при прицеливании за счет изменения расположения самих примитивов в файле векторной графики.

Возможными направлениями доработки данного программного комплекса является расширение набора векторных примитивов, которые

может хранить разработанный формат, создание более гибкого интерфейса прицеливания (к примеру, на основе определения расстояния от точки клика мыши до самого примитива), а также добавление новых функций в редактор файлов разработанного формата (к примеру, дописывание новых примитивов в конец открытого файла).

Таким образом, разработанный программный комплекс решает все поставленные перед ним задачи и удовлетворяет предъявленным к нему техническим требованиям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Why C++ is the perfect choice for modern app development. — Текст : электронный // betanews : [сайт]. — URL: <https://betanews.com/2014/07/22/why-c-is-the-perfect-choice-for-modern-app-development/> (дата обращения: 07.06.2023).
2. Порядок байтов. — Текст : электронный // Википедия : [сайт]. — URL: https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%B1%D0%B0%D0%B9%D1%82%D0%BE%D0%B2 (дата обращения: 07.06.2023).
3. Принц, П. Язык С. Справочник. Полное описание языка / П. Принц, Т. Кроуфорд. — 2-е изд. — М. : Вильямс, 2017. — 880 с. — Текст : непосредственный.
4. Страуструп, Б. Язык программирования C++. Специальное издание / Б. Страуструп. — 3-е изд. — М. : Бином, 2006. — 1104 с. — Текст : непосредственный.
5. Поиск утечки GDI объектов: Как загнать мастодонта. — Текст : электронный // Хабр : [сайт]. — URL: <https://habr.com/ru/articles/318948/> (дата обращения: 07.06.2023).
6. Region::IsVisible(constPoint&,constGraphics*) method (gdiplusheaders.h). — Текст : электронный // MSDN : [сайт]. — URL: [https://learn.microsoft.com/en-us/windows/win32/api/gdiplusheaders/nf-gdiplusheaders-region-isvisible\(constpoint__constgraphics\)](https://learn.microsoft.com/en-us/windows/win32/api/gdiplusheaders/nf-gdiplusheaders-region-isvisible(constpoint__constgraphics)) (дата обращения: 07.06.2023).
7. Петзолд, Ч. Программирование для Windows 95 в двух томах / Ч. Петзолд. — СПб. : BHV, 1997. — Т. 1. — 495 с. — Текст : непосредственный.

8. Функция WinMain. — Текст : электронный // MSDN - Windows API Персональный сайт Владимира Соковикова : [сайт]. — URL: http://vsokovikov.narod.ru/New_MSDN_API/Window/fn_winmain.htm (дата обращения: 07.06.2023).
9. Керниган, Б. Язык программирования Си / Б. Керниган, Д. Ритчи. — 3-е изд., испр. — СПб. : Невский диалект, 2003. — 352 с. — Текст : непосредственный.
10. CreateEllipticRgn function (wingdi.h). — Текст : электронный // MSDN : [сайт]. — URL: <https://learn.microsoft.com/en-us/windows/win32/api/wingdi/nf-wingdi-createellipticrgn> (дата обращения: 07.06.2023).

ПРИЛОЖЕНИЕ А

Задание на выпускную квалификационную работу



МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное
учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт ИВТИ
Кафедра ВМСС

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ (бакалаврскую работу)

Направление 09.03.01 Информатика и вычислительная техника
(код и наименование)

Направленность (профиль) Вычислительные машины, комплексы,
системы и сети

Форма обучения очная
(очная/очно-заочная/заочная)

Тема: Разработка программных инструментов для работы с
расположением объектов на плоскости

Студент А-12-19 Винокуров Р.Н.
группа подпись фамилия и инициалы

Научный к.т.н. доцент Гольцов А.Г.
руководитель
уч. степень должность подпись фамилия и инициалы

Зав. кафедрой к.т.н. доцент Вишняков С.В.
уч. степень звание подпись фамилия и инициалы

Место выполнения работы кафедра ВМСС «НИУ МЭИ»

СОДЕРЖАНИЕ РАЗДЕЛОВ ЗАДАНИЯ И ИСХОДНЫЕ ДАННЫЕ

Для решения задачи ввода, хранения, и отображения объектов на двумерном изображении (например, отображения границ земельных участков на карте или расположения точек и зон на медицинском изображении) необходимо разработать редактор для подготовки и изменения

таких данных, формат хранения и библиотеку доступа для использования в программах, где требуется отображение таких данных, а также тестовое приложение, демонстрирующее возможности разработанного формата.

Для хранения и отображения объектов на двумерном изображении необходимо разработать формат, который сможет хранить в себе векторные фигуры и их положение на двумерном растровом полотне. Данный формат должен поддерживать представление таких графических примитивов как точка, линия, ломаная и областей (многоугольник, окружность), а также слои растровой графики, на которые накладываются векторные фигуры. В файлах данного формата должны храниться данные о положении фигур (к примеру, координаты начала и конца линии), их идентификатор(уникальный для каждой фигуры набор символов), а также важные для их отрисовки данные – толщина и цвет. Дополнительно в файле должны храниться данные о размере области прицеливания для фигур.

Для упрощения процесса создания приложения для работы с данным форматом необходимо разработать DLL-библиотеку для отображения и редактирования данных, хранящихся в файлах разработанного формата. Функции данной библиотеки должны работать с векторными графическими примитивами и областями, описанными в виде структур и слоев растровой графики. Также предусмотреть в данных функциях отображение объектов с учетом масштабирования и «прицеливание» (проверку попадания координат клика мыши в область «захвата» фигуры), использующее данные об области «прицеливания» из файла разработанного формата.

Для подготовки данных (создания файлов данного формата) необходимо разработать на языке Си редактор, осуществляющий работу с созданным графическим форматом. Также на языке Си средствами WinAPI необходимо разработать тестовое приложение для демонстрации возможностей разработанного формата. Данное приложение должно отображать на экране содержимое созданного с помощью редактора файла, поддерживать масштабирование отображаемых данных, а также прицеливание (например, при «захвате» какой-либо фигуры приложение должно отображать подробные данные об этой фигуре). Для доступа к хранящимся внутри файлов данным приложение должно использовать разработанную ранее DLL-библиотеку. Для вывода графических данных на экран использовать библиотеку GDI+ и предоставить пользователю возможность включить/выключить режим сглаживания из данной библиотеки при отображении данных.

ПЕРЕЧЕНЬ ГРАФИЧЕСКОГО МАТЕРИАЛА

- 1.Интерфейс редактора файлов разработанного формата
- 2.Структура файла разработанного графического формата
- 3.Схема алгоритмов прицеливания для разных фигур
- 4.Схема алгоритма работы редактора файлов разработанного формата

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Петзолд Ч. Программирование для Windows 95 в 2 томах / Ч. Петзолд. — СПб : BHV, 1997. — 1080 с.
2. Побегайло, А. П. Системное программирование в Windows / А. П. Побегайло. — СПб : BHV, 2006. — 1056 с.
3. КАК рисовать в Win32 API?. — Текст : электронный // radiofront.narod.ru : [сайт]. — URL: <http://radiofront.narod.ru/htm/prog/htm/winda/api/paint.html> (дата обращения: 19.12.2022).
4. Знакомство с SVG-графикой. — Текст : электронный // Хабр : [сайт]. — URL: <https://habr.com/ru/post/157087/> (дата обращения: 19.12.2022).

Примечания:

1. Задание брошюруется вместе с выпускной работой после титульного листа (страницы задания имеют номера 2, 3).
2. Отзыв руководителя, рецензия(и), отчет о проверке на объем заимствований и согласие студента на размещение работы в открытом доступе вкладываются в конверт (файловую папку) под обложкой работы.

Приложение Б

Листинг программ

Модуль ndllproj.h

```
// Приведенный ниже блок ifdef - это стандартный метод создания макросов, упрощающий
// процедуру
// экспорта из библиотек DLL. Все файлы данной DLL скомпилированы с использованием
// символа NDLLPROJ_EXPORTS,
// в командной строке. Этот символ не должен быть определен в каком-либо проекте
// использующем данную DLL. Благодаря этому любой другой проект, чьи исходные файлы
// включают данный файл, видит
// функции NDLLPROJ_API как импортированные из DLL, тогда как данная DLL видит символы,
// определяемые данным макросом, как экспортированные.
#ifdef NDLLPROJ_EXPORTS
#define NDLLPROJ_API __declspec(dllexport)
#else
#define NDLLPROJ_API __declspec(dllimport)
#endif

using namespace ATL;
using namespace std;

//подключаем заголовочные файлы для разработки приложений в ОС Windows
#include <stdint.h>
#include <stdio.h>
#include <windows.h>
#include <algorithm>
#include <atlbase.h>
#include <atlconv.h>
#include <atlstr.h>
#include <objidl.h>

//подключаем к приложению GDI+
#include <gdiplus.h>
using namespace Gdiplus;
#pragma comment (lib,"Gdiplus.lib")

//Перестановка байтов в 32-битном беззнаковом целом числе
#define SWAP_UINT32(x) (((x) >> 24) | (((x) & 0x00FF0000) >> 8) | (((x) & 0x0000FF00) <<
8) | ((x) << 24))

//Объявление структур примитивов
#pragma pack(push, 1)
typedef struct basisdescr
{
    uint32_t refwidth;
    uint32_t refheight;
    int32_t basex;
    int32_t basey;
} basisd_;
typedef struct point
{
    uint32_t x;
    uint32_t y;
    unsigned char aim_area;
    unsigned char red;
```

```

        unsigned char green;
        unsigned char blue;
    } point_;
    typedef struct line
    {
        uint32_t x;
        uint32_t y;
        uint32_t x1;
        uint32_t y1;
        unsigned char thickness;
        unsigned char aim_area;
        unsigned char red;
        unsigned char green;
        unsigned char blue;
    } line_;
    typedef struct brokenline
    {
        unsigned char thickness;
        unsigned char red;
        unsigned char green;
        unsigned char blue;
        unsigned char count;
        uint32_t arr[20];
    } brokenline_;
    typedef struct circle
    {
        uint32_t xcenter;
        uint32_t ycenter;
        uint32_t radius;
        unsigned char thickness;
        unsigned char aim_area;
        unsigned char red;
        unsigned char green;
        unsigned char blue;
    } circle_;
    typedef struct polyline
    {
        unsigned char thickness;
        unsigned char red;
        unsigned char green;
        unsigned char blue;
        unsigned char count;
        uint32_t arr[20];
    } polyline_;
    #pragma pack(pop)
    typedef enum {
        PTYPE_NULL=0,
        PTYPE_BASE,
        PTYPE_POINT,
        PTYPE_LINE,
        PTYPE_BROKENLINE,
        PTYPE_CIRCLE,
        PTYPE_POLYLINE,
    } ptype_t;
    //элемент направленного списка
    typedef struct datatype {
        void* data; //данные примитива
        ptype_t type; //тип примитива
        Gdiplus::Region* r; //регион прицеливания
        //указатели на следующий/предыдущий элемент направленного списка
        struct datatype* next;
        struct datatype* prev;
    } datatype_t;
    //экспортируемые из DLL-функции

```

```

extern "C" NDLLPROJ_API void addbasisdescr(FILE* fp,basisd_* basd);
extern "C" NDLLPROJ_API void addpoint(FILE* fp,point_* p);
extern "C" NDLLPROJ_API void addline(FILE* fp,line_* l);
extern "C" NDLLPROJ_API void addbrokenline(FILE* fp,brokenline_* brl);
extern "C" NDLLPROJ_API void addcircle(FILE* fp,circle_* cr);
extern "C" NDLLPROJ_API void addpolyline(FILE* fp,polyline_* pl);
extern "C" NDLLPROJ_API ptype_t streadd(FILE* fp,void** data);
extern "C" NDLLPROJ_API datatype_t* IsInside(uint32_t x,uint32_t y,datatype_t* head);
extern "C" NDLLPROJ_API void ppoint(Graphics* graphics,uint32_t x,uint32_t y,char red,
char green, char blue);
extern "C" NDLLPROJ_API void pline(Graphics* graphics,uint32_t x,uint32_t y,uint32_t
x1,uint32_t y1,char thickness,char red,char green,char blue);
extern "C" NDLLPROJ_API void pbrokenline(Graphics* graphics,uint32_t* arrx,uint32_t*
arry,int count,char thickness,char red,char green,char blue);
extern "C" NDLLPROJ_API void pcircle(Graphics* graphics,uint32_t xcenter,uint32_t
ycenter,uint32_t radius,char thickness,char red,char green,char blue);
extern "C" NDLLPROJ_API void ppolyline(Graphics* graphics,uint32_t* arrx,uint32_t*
arry,int count,char thickness,char red,char green,char blue);
extern "C" NDLLPROJ_API void PaintBitmapinHDC(HWND hWnd,HWND hWndScroll,HWND
hWndScroll1,HDC hDC,HDC memBit,int state,int x_,int y_,int width,int height);

```

Модуль ndllproj.cpp

```

// ndllproj.cpp: определяет экспортированные функции для приложения DLL.
//
//подключение дополнительных заголовочных файлов
#include "stdafx.h"
#include "ndllproj.h"
#include "list_helper.h"

using namespace ATL;
using namespace std;

//подключаем заголовочные файлы для разработки приложений в ОС Windows
#include <algorithm>
#include <atlbase.h>
#include <atlconv.h>
#include <atlstr.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
#include <windows.h>
#include <objidl.h>

//подключаем к приложению GDI и GDI+
#include <gdiplus.h>
using namespace Gdiplus;
#pragma comment (lib,"Gdiplus.lib")

//Добавление примитива "базис" в файл
//fp - указатель на открытый файловый поток
//basd - указатель на структуру типа basisd_
extern "C" NDLLPROJ_API void addbasisdescr(FILE* fp,basisd_* basd)
{
    fprintf(fp,"%c%c%c%c",0xFE,0xDE,0xAE,0xBE);
    fwrite(basd,sizeof(basisd_),1,fp);
}
//Добавление примитива "точка" в файл
//fp - указатель на открытый файловый поток

```

```

//p - указатель на структуру типа point_
extern "C" NDLLPROJ_API void addpoint(FILE* fp, point_* p)
{
    fprintf(fp, "%c%c%c%c", 0x50, 0x4F, 0x49, 0x4E);
    fwrite(p, sizeof(point_), 1, fp);
}
//Добавление примитива "линия" в файл
//fp - указатель на открытый файловый поток
//line - указатель на структуру типа line_
extern "C" NDLLPROJ_API void addline(FILE* fp, line_* l)
{
    fprintf(fp, "%c%c%c%c", 0x4C, 0x49, 0x4E, 0x45);
    fwrite(l, sizeof(line_), 1, fp);
}
//Добавление примитива "ломаная" в файл
//fp - указатель на открытый файловый поток
//brl - указатель на структуру типа brokenline_
extern "C" NDLLPROJ_API void addbrokenline(FILE* fp, brokenline_* brl)
{
    fprintf(fp, "%c%c%c%c", 0x42, 0x52, 0x4C, 0x49);
    fwrite(brl, sizeof(brokenline_), 1, fp);
}
//Добавление примитива "окружность" в файл
//fp - указатель на открытый файловый поток
//cr - указатель на структуру типа circle_
extern "C" NDLLPROJ_API void addcircle(FILE* fp, circle_* cr)
{
    fprintf(fp, "%c%c%c%c", 0x43, 0x49, 0x52, 0x43);
    fwrite(cr, sizeof(circle_), 1, fp);
}
//Добавление примитива "многоугольник" в файл
//fp - указатель на открытый файловый поток
//pl - указатель на структуру типа polyline_
extern "C" NDLLPROJ_API void addpolyline(FILE* fp, polyline_* pl)
{
    fprintf(fp, "%c%c%c%c", 0x50, 0x4F, 0x4C, 0x59);
    fwrite(pl, sizeof(polyline_), 1, fp);
}
// Функция fread считывает данные из файла и возвращает тип данных прочитанной
структуры
// Функция принимает указатель на файл fp и указатель на данные data
extern "C" NDLLPROJ_API ptype_t fread(FILE* fp, void** data)
{
    basisd_* bd;
    point_* p;
    line_* l;
    brokenline_* brl;
    circle_* cr;
    polyline_* pl;
    uint32_t id;
    // Считываем первые 4 байта из файла и переставляем байты в порядке little-end
    fread(&id, 4, 1, fp);
    id = SWAP_UINT32(id);

    // Используем оператор switch для определения типа данных, которые нужно считать из
    файла
    switch(id)
    {
        // Если id соответствует типу PTYPE_BASE, выделяем память для структуры basisd_ и
        считываем ее из файла
        case 0xFEDEAEBE:
            bd = (basisd_*)malloc(sizeof(basisd_));
            fread(bd, sizeof(basisd_), 1, fp);
            *data = (void*)bd;
    }
}

```

```

        return PTYPE_BASE;
    break;

    // Если id соответствует типу PTYPE_POINT, выделяем память для структуры point_ и
    считываем ее из файла
    case 0x504F494E:
        p = (point_*)malloc(sizeof(point_));
        fread(p, sizeof(point_), 1, fp);
        *data = (void*)p;
        return PTYPE_POINT;
    break;

    // Если id соответствует типу PTYPE_LINE, выделяем память для структуры line_ и
    считываем ее из файла
    case 0x4C494E45:
        l = (line_*)malloc(sizeof(line_));
        fread(l, sizeof(line_), 1, fp);
        *data = (void*)l;
        return PTYPE_LINE;
    break;

    // Если id соответствует типу PTYPE_BROKENLINE, выделяем память для структуры
    brokenline_ и считываем ее из файла
    case 0x42524C49:
        brl = (brokenline_*)malloc(sizeof(brokenline_));
        fread(brl, sizeof(brokenline_), 1, fp);
        *data = (void*)brl;
        return PTYPE_BROKENLINE;
    break;

    // Если id соответствует типу PTYPE_CIRCLE, выделяем память для структуры circle_
    и считываем ее из файла
    case 0x43495243:
        cr = (circle_*)malloc(sizeof(circle_));
        fread(cr, sizeof(circle_), 1, fp);
        *data = (void*)cr;
        return PTYPE_CIRCLE;
    break;

    // Если id соответствует типу PTYPE_POLYLINE, выделяем память для структуры
    polyline_ и считываем ее из файла
    case 0x504F4C59:
        pl = (polyline_*)malloc(sizeof(polyline_));
        fread(pl, sizeof(polyline_), 1, fp);
        *data = (void*)pl;
        return PTYPE_POLYLINE;
    break;

    // Если id не соответствует ни одному из известных типов данных, возвращаем
    PTYPE_NULL
    default:
        return PTYPE_NULL;
    break;
}

}

// Функция IsInside проверяет, находится ли точка с координатами (x, y) внутри какой-либо
// фигуры из списка head.
// Функция принимает три аргумента: координаты точки (x, y), указатель на первый элемент
// списка head и возвращает указатель на элемент списка, внутри которого находится точка.
// Если точка не находится внутри ни одной фигуры, функция возвращает NULL.
extern "C" NDLLPROJ_API datatype_t* IsInside(uint32_t x, uint32_t y, datatype_t* head)
{
    datatype_t* currelem = head;

```



```

datatype_t* ret = NULL;

// Создаем объект Point с координатами (x, y)
Point p;
p.X = x;
p.Y = y;

// Проходим по всем элементам списка
while (currelem)
{
    // Если тип элемента не является типом базиса или пустым типом
    if(currelem->type!=PTYPE_BASE && currelem->type!=PTYPE_NULL)
    {
        // Если точка находится внутри текущей фигуры
        if(currelem->r->IsVisible(p,NULL))
        {
            // Запоминаем указатель на текущий элемент и выходим из цикла
            ret = currelem;
            break;
        }
    }
    currelem = currelem->next;           // Переходим к следующему элементу
списка
}
return ret;                             // Возвращаем
указатель на элемент списка, внутри которого находится точка
}

// Функция ppoint рисует точку на графическом контексте graphics с координатами (x, y) и
цветом, заданным значениями red, green и blue.
// Функция принимает 6 аргументов: указатель на графический контекст graphics, координаты
точки (x, y) и значения цвета red, green и blue.
extern "C" NDLLPROJ_API void ppoint(Graphics* graphics,uint32_t x,uint32_t y,char red,
char green, char blue)
{
    Pen* pen = new Pen(Color(255, red, green, blue),1);    // Создаем объект Pen с
цветом, заданным значениями red, green и blue и толщиной, равной 1
    graphics->DrawRectangle(pen,Rect(x,y,1,1));           // Рисуем прямоугольник на
графическом контексте graphics с координатами (x, y) и размером 1x1, используя объект Pen
    delete pen;
    // Освобождаем память, занятую объектом Pen
}

// Функция pline рисует линию на графическом контексте graphics, соединяющую точки (x, y)
и (x1, y1) с толщиной и цветом, заданными значениями thickness, red, green и blue.
// Функция принимает девять аргументов: указатель на графический контекст graphics,
координаты начальной точки (x, y), координаты конечной точки (x1, y1), толщину линии
thickness и значения цвета red, green и blue.
extern "C" NDLLPROJ_API void pline(Graphics* graphics,uint32_t x,uint32_t y,uint32_t
x1,uint32_t y1,char thickness,char red,char green,char blue)
{
    Pen* pen = new Pen(Color(255, red, green, blue),thickness);    // Создаем объект Pen
с цветом, заданным значениями red, green и blue, и толщиной, заданной значением thickness
    graphics->DrawLine(pen, (int)x, (int)y, (int)x1, (int)y1);    // Рисуем линию на
графическом контексте graphics, соединяющую точки (x, y) и (x1, y1), используя объект Pen
    delete pen;
    // Освобождаем память, занятую объектом Pen
}

// Функция pbrokenline рисует ломаную линию на графическом контексте graphics, проходящую
через точки, заданные массивами arrx и arry, с толщиной и цветом, заданными значениями
thickness, red, green и blue.
// Функция принимает восемь аргументов: указатель на графический контекст graphics,
массивы координат точек arrx и arry, количество точек count, толщину линии thickness и
значения цвета red, green и blue.
extern "C" NDLLPROJ_API void pbrokenline(Graphics* graphics,uint32_t* arrx,uint32_t*
arry,int count,char thickness,char red,char green,char blue)

```

```

{
    Pen* pen = new Pen(Color(255, red, green, blue),thickness);    // Создаем объект Pen
    с цветом, заданным значениями red, green и blue, и толщиной, заданной значением thickness

    // Добавляем линии между точками, заданными массивами arrx и arry, в объект
    GraphicsPath
    GraphicsPath grpath;
    for(int i=0;i<count-1;i++)
    {
        grpath.AddLine(Point(arrx[i],arry[i]),Point(arrx[i+1],arry[i+1]));
    }

    graphics->DrawPath(pen,&grpath);    // Рисуем ломаную линию на графическом
    контексте, используя объект Pen и объект GraphicsPath
    delete pen;    // Освобождаем память,
    занятую объектом Pen
}
// Функция pcircle рисует окружность на графическом контексте graphics с центром в точке
(xcenter, ycenter) и радиусом radius, с толщиной и цветом заданными значениями thickness,
red, green и blue.
// Функция принимает восемь аргументов: указатель на графический контекст graphics,
координаты центра окружности xcenter и ycenter радиус окружности radius, толщину линии
thickness и значения цвета red, green и blue.
extern "C" NDLLPROJ_API void pcircle(Graphics* graphics,uint32_t xcenter,uint32_t
ycenter,uint32_t radius,char thickness,char red,char green,char blue)
{
    Pen* pen = new Pen(Color(255, red, green, blue),thickness);
    // Создаем объект Pen с цветом, заданным значениями red, green и blue, и толщиной,
    заданной значением thickness
    RectF ellipseRect(xcenter-radius, ycenter-radius, 2*radius, 2*radius); // Создаем
    объект RectF, задающий квадрат, в который вписана окружность
    graphics->DrawEllipse(pen,ellipseRect);    // Рисуем окружность на
    графическом контексте graphics, используя объект Pen и объект RectF
    delete pen;    //
    Освобождаем память, занятую объектом Pen
}
// Функция ppolyline рисует многоугольник на графическом контексте graphics, проходящую
через точки, заданные массивами arrx и arry, с толщиной и цветом, заданными thickness,
red, green и blue.
// Функция принимает восемь аргументов: указатель на графический контекст graphics,
массивы координат точек arrx и arry, количество точек count, толщину линии thickness и
значения цвета red, green и blue.
extern "C" NDLLPROJ_API void ppolyline(Graphics* graphics,uint32_t* arrx,uint32_t*
arry,int count,char thickness,char red,char green,char blue)
{
    POINT arrp[10];    // Создаем массив POINT arrp для хранения точек полилинии
    HRGN region;
    Pen* pen = new Pen(Color(255, red, green, blue),thickness);    // Создаем
    объект Pen с цветом, заданным значениями red, green и blue, и толщиной, заданной
    значением thickness
    GraphicsPath grpath;

    // Добавляем линии между точками, заданными массивами arrx и arry, в объект
    GraphicsPath
    for(int i=0;i<count-1;i++)
    {
        grpath.AddLine(Point(arrx[i],arry[i]),Point(arrx[i+1],arry[i+1]));
    }

    grpath.AddLine(Point(arrx[0],arry[0]),Point(arrx[count-1],arry[count-1]));    //
    Добавляем линию, соединяющую последнюю точку с первой, в объект GraphicsPath
    graphics->DrawPath(pen,&grpath);    // Рисуем многоугольник на графическом
    контексте, используя объект Pen и объект GraphicsPath
}

```

```

        delete pen; //Освобождаем память,
занятую объектом Pen
    }
    // Функция getWindowHeadSize возвращает высоту заголовка окна, заданного дескриптором
hWnd.
    // Функция принимает один аргумент: дескриптор окна hWnd.
    int getWindowHeadSize(HWND hWnd)
    {
        RECT Rect;
        GetWindowRect(hWnd, &Rect);
        POINT point = { 0, 0 };
        ClientToScreen(hWnd, &point);
        return point.y - Rect.top + GetSystemMetrics(SM_CYSIZEFRAME) -
GetSystemMetrics(SM_CYEDGE) * 2;
    }
    // Функция getWindowHeadSizew возвращает ширину заголовка окна, заданного дескриптором
hWnd.
    // Функция принимает один аргумент: дескриптор окна hWnd.
    int getWindowHeadSizew(HWND hWnd)
    {
        RECT Rect;
        GetWindowRect(hWnd, &Rect);
        POINT point = { 0, 0 };
        ClientToScreen(hWnd, &point);
        return point.x - Rect.left + GetSystemMetrics(SM_CXSIZEFRAME) -
GetSystemMetrics(SM_CXEDGE) * 2;
    }
    // Функция PaintBitmapinHDC отображает изображение в контексте устройства (HDC) окна
(HWND).
    // Функция принимает следующие аргументы:
    // hWnd - дескриптор окна, в котором будет отображаться изображение;
    // hWndScroll - дескриптор окна, содержащего горизонтальный ползунок прокрутки;
    // hWndScroll1 - дескриптор окна, содержащего вертикальный ползунок прокрутки;
    // hDC - контекст устройства окна, в котором будет отображаться изображение;
    // memBit - контекст устройства, содержащий изображение;
    // state - состояние отображения изображения (0 - без масштабирования, 1 -
масштабирование в 2 раза, 2 - масштабирование в 3 раза,
    // -1 - уменьшение в 2 раза, -2 - уменьшение в 3 раза);
    // x_ - координата x верхнего левого угла области изображения, которая будет отображаться
в окне;
    // y_ - координата y верхнего левого угла области изображения, которая будет отображаться
в окне;
    // width - ширина области изображения, которая будет отображаться в окне;
    // height - высота области изображения, которая будет отображаться в окне.
    void PaintBitmapinHDC(HWND hWnd,HWND hWndScroll,HWND hWndScroll1,HDC hDC,HDC memBit,int
state,int x_,int y_,int width,int height)
    {
        SetStretchBltMode(hDC,COLORONCOLOR);

        // Выполняем действия в зависимости от состояния отображения изображения
        switch(state)
        {

            // Если состояние равно 0, то устанавливаем размер окна, отображаем
изображение без масштабирования, скрываем ползунки прокрутки
            case 0:
                SetWindowPos(hWnd,0,0,0,width+getWindowHeadSizew(hWnd)-
1,height+getWindowHeadSize(hWnd)-1,SWP_NOMOVE);
                StretchBlt(hDC,0,0,width,height,memBit,0,0,width,height,SRCCOPY);
                ShowWindow(hWndScroll,SW_HIDE);
                ShowWindow(hWndScroll1,SW_HIDE);
                break;

```

```

        // Если состояние равно 1, то масштабируем изображение в 2 раза, отображаем
        ползунки прокрутки
        case 1:
StretchBlt(hDC,0,0,2*width,2*height,memBit,x_,y_,width+17,height+17,SRCCOPY);
        SetScrollRange(hWndScroll, SB_CTL, 0, width/2, TRUE);
        SetScrollRange(hWndScroll1, SB_CTL, 0, height/2, TRUE);
        ShowWindow(hWndScroll,SW_SHOW);
        ShowWindow(hWndScroll1,SW_SHOW);
        break;

        // Если состояние равно 2, то масштабируем изображение в 3 раза, отображаем
        ползунки прокрутки
        case 2:
        SetScrollRange(hWndScroll, SB_CTL, 0, width*2/3, TRUE);
        SetScrollRange(hWndScroll1, SB_CTL, 0, height*2/3, TRUE);

        StretchBlt(hDC,0,0,3*width,3*height,memBit,x_,y_,width+17,height+17,SRCCOPY
        );
        break;

        // Если состояние равно -1, то уменьшаем изображение в 2 раза, устанавливаем
        размер окна
        case -1:
        SetWindowPos(hWnd,0,0,0,width/2+getWindowHeadSize(hWnd)-
1,height/2+getWindowHeadSize(hWnd)-1,SWP_NOMOVE);
        StretchBlt(hDC,0,0,width,height,memBit,0,0,2*width,2*height,SRCCOPY);
        break;

        // Если состояние равно -2, то уменьшаем изображение в 3 раза, устанавливаем
        размер окна
        case -2:
        SetWindowPos(hWnd,0,0,0,width/3+getWindowHeadSize(hWnd)-
1,height/3+getWindowHeadSize(hWnd)-1,SWP_NOMOVE);
        StretchBlt(hDC,0,0,width,height,memBit,0,0,3*width,3*height,SRCCOPY);
        break;
    }
}

```

Модуль list_helper.h

```

#ifndef LIST_HELPER_H
#define LIST_HELPER_H

// Макрос для добавления элемента (add) в двусвязный список (head)
#define LIST_ADD(head, add) \
do { \
    if (head) { \
        (add)->prev = (head)->prev; \
        (head)->prev->next = (add); \
        (head)->prev = (add); \
        (add)->next = NULL; \
    } else { \
        (head)=(add); \
        (head)->prev = (head); \
        (head)->next = NULL; \
    } \
} while (0)

// Макрос для удаления элемента (del) из двусвязного списка (head)
#define LIST_DEL(head, del) \
do { \

```

```

    if ((del)->prev == (del)) { \
        (head)=NULL; \
    } else if ((del)==(head)) { \
        (del)->next->prev = (del)->prev; \
        (head) = (del)->next; \
    } else { \
        (del)->prev->next = (del)->next; \
        if ((del)->next) { \
            (del)->next->prev = (del)->prev; \
        } else { \
            (head)->prev = (del)->prev; \
        } \
    } \
} while (0)

#endif /* LIST_HELPER_H */

```

Модуль stdafx.h

```

// stdafx.h: включаемый файл для стандартных системных включаемых файлов
// или включаемых файлов для конкретного проекта, которые часто используются, но
// не часто изменяются
//

#pragma once

#include "targetver.h"

#define WIN32_LEAN_AND_MEAN           // Исключите редко используемые компоненты из
заголовков Windows
// Файлы заголовков Windows:
#include <windows.h>

// Файлы заголовков C RunTime
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include <tchar.h>

// TODO: Установите здесь ссылки на дополнительные заголовки, требующиеся для программы

```

Модуль targetver.h

```

#pragma once

// Включение SDKDDKVer.h обеспечивает определение самой последней доступной платформы
Windows.

// Если требуется выполнить построение приложения для предыдущей версии Windows, включите
WinSDKVer.h и
// задайте для макроса _WIN32_WINNT значение поддерживаемой платформы перед включением
SDKDDKVer.h.

#include <SDKDDKVer.h>

```

Модуль stdafx.cpp

```
// stdafx.cpp: исходный файл, содержащий только стандартные включаемые модули
// ndllproj.pch будет предкомпилированным заголовком
// stdafx.obj будет содержать предварительно откомпилированные сведения о типе

#include "stdafx.h"

// TODO: Установите ссылки на любые требуемые дополнительные заголовки в файле STDAFX.H
// , а не в данном файле
```

Модуль dllmain.cpp

```
// dllmain.cpp: определяет точку входа для приложения DLL.
#include "stdafx.h"

BOOL APIENTRY DllMain( HMODULE hModule,
                      DWORD  ul_reason_for_call,
                      LPVOID lpReserved
                      )
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
            break;
    }
    return TRUE;
}
```

Модуль testpr.cpp

```
//исходный код программы-редактора
//консольное приложение ОС Windows

//подключаем заголовочные файлы для разработки приложений в ОС Windows
#include <stdio.h>
#include "stdafx.h"
#include <stdlib.h>
#include <stdint.h>
#include <conio.h>
#include <malloc.h>
#include <windows.h>
#include "ndllproj.h" //заголовочный файл DLL-библиотеки ndllproj

int main()
{
    char a,b; //для хранения выбора пунктов меню
    void* v; //указатель приведения
    char name[30]; //имя открываемого файла
    int
arr[6]={sizeof(basisd_)+4,sizeof(point_)+4,sizeof(line_)+4,sizeof(brokenline_)+4,sizeof(c
ircle_)+4,sizeof(polyline_)+4}; //размеры примитивов файла с учетом
идентификатора

    //промежуточные данные обработки
    char str[256], res[256];
```

```

int n,k,strnum;
ptype_t t; //хранит тип прочитанного через streadd примитива
uint32_t x,y;
FILE* fp,*fp2; //указатели потоков для работы с файлами
bool flag = true;
bool flag1;
basisd_bd;
point_p;
line_l;
brokenline_brl;
circle_cr;
polyline_pl;
int si=0;
char* nametmp;

//включения консольного вывода на русском языке
SetConsoleCP(1251);
SetConsoleOutputCP(1251);

while(flag)
{

    flag1 = true; //флаг завершения работы программы
    //вывод пунктов интерактивного меню
    printf("-----Выбор действия-----\n");
    printf("1 - создать новый файл\n");
    printf("2 - редактировать уже существующий файл\n");
    printf("3 - завершить работу програмы\n");
    printf("4 - удалить из существующего файла структуру(-ы)\n");

    printf("Выберите действие:\n");
    a = (char)getch();
    switch(a)
    {
        case '1':
            printf("Введите название создаваемого файла:\n");
            scanf("%s",name);
            if ((fp = fopen(name, "wb")) == NULL) //создаем файл для записи
                printf("Не удалось создать файл!\n");
            else
            {
                while(flag1)
                {

                    //интерактивное меню
                    printf("-----Добавление примитивов-----\n");
                    printf("Клавиши и их соответствие примитивам\n");
                    printf("A - установить параметры базиса\n");
                    printf("B - точка\n");
                    printf("C - линия\n");
                    printf("D - ломаная\n");
                    printf("E - окружность\n");
                    printf("F - многоугольник\n");
                    printf("G - в главное меню\n");

                    //Выбор пункта меню пользователем
                    printf("Выберите опцию:\n");
                    a = (char)getch();

                    //Инициализация структур и их добавление в файл
                    switch(toupper(a)) //выбор пунктов с помощью символов обеих

```

```

{
    case 'A':
        // Инициализация структуры basisd_ нулями
        memset(&bd,0,sizeof(basisd_));

        // Запрос у пользователя значений для заполнения
        printf("Введите refwidth:");
        scanf("%d",&(bd.refwidth));
        printf("Введите refheight:");
        scanf("%d",&(bd.refheight));
        printf("Введите basex:");
        scanf("%d",&(bd.basex));
        printf("Введите basey:");
        scanf("%d",&(bd.basey));

        addbasisdescr(fp,&bd); // Добавление структуры в файл
        break;

    case 'B':
        // Инициализация структуры point_ нулями
        memset(&p,0,sizeof(p));

        // Запрос у пользователя значений для заполнения
        printf("Введите координату точки по горизонтали:");
        scanf("%d",&(p.x));
        printf("Введите координату точки по вертикали:");
        scanf("%d",&(p.y));
        printf("Введите размер области прицеливания:");
        scanf("%d",&si);
        p.aim_area = (char)(si & 0xff);
        printf("Введите интенсивность цветового канала R:");
        scanf("%d",&si);
        p.red = (char)(si & 0xff);
        printf("Введите интенсивность цветового канала G:");
        scanf("%d",&si);
        p.green = (char)(si & 0xff);
        printf("Введите интенсивность цветового канала B:");
        scanf("%d",&si);
        p.blue = (char)(si & 0xff);

        // Проверка входных значений и добавление структуры в
        if(p.aim_area==0)
        {
            printf("Неверный набор входных значений!");
            continue;
        }
        else
            addpoint(fp,&p);
        break;

    case 'C':
        // Инициализация структуры line_ нулями
        memset(&l,0,sizeof(line_));

        // Запрос у пользователя значений для заполнения
        printf("Введите координату начала линии по
        горизонтали:");
        scanf("%d",&(l.x));

```



```

вертикали:");

scanf("%d",&(l.y));
printf("Введите координату конца линии по

горизонтали:");

scanf("%d",&(l.x1));
printf("Введите координату конца линии по

вертикали:");

scanf("%d",&(l.y1));
printf("Введите толщину линии:");
scanf("%d",&(l.thickness));
printf("Введите размер области прицеливания:");
scanf("%d",&si);
l.aim_area = (char)(si & 0xff);
printf("Введите интенсивность цветового канала R:");
scanf("%d",&si);
l.red = (char)(si & 0xff);
printf("Введите интенсивность цветового канала G:");
scanf("%d",&si);
l.green = (char)(si & 0xff);
printf("Введите интенсивность цветового канала B:");
scanf("%d",&si);
l.blue = (char)(si & 0xff);

// Проверка входных значений и добавление структуры в
файл
if(l.aim_area==0 || l.thickness==0 || l.aim_area==0)
{
    printf("Неверный набор входных значений!

    continue;
}
else
    addline(fp,&l);
break;

case 'D':
    // Инициализация структуры brl нулями
    memset(&brl,0,sizeof(brokenline_));

    // Запрос у пользователя значений для заполнения
структуры
printf("Введите толщину линии:");
scanf("%d",&(brl.thickness));
printf("Введите интенсивность цветового канала R:");
scanf("%d",&si);
brl.red = (char)(si & 0xff);
printf("Введите интенсивность цветового канала G:");
scanf("%d",&si);
brl.green = (char)(si & 0xff);
printf("Введите интенсивность цветового канала B:");
scanf("%d",&si);
brl.blue = (char)(si & 0xff);
printf("Введите количество точек ломаной (до 10

штук):");

scanf("%d",&n);
brl.count = n;

// Заполнение массива точек ломаной
for(int i=0;i<n;i++)
{
    printf("Введите координату точки по

горизонтали:");

```

```

scanf("%d",&x);
printf("Введите координату точки по вертикали:");
scanf("%d",&y);
br1.arr[2*i]=x;
br1.arr[2*i+1]=y;
}

// Проверка входных значений и добавление структуры в
файл
if(br1.count==0 || br1.thickness==0)
{
    printf("Неверный набор входных значений!");
    continue;
}
else
    addbrokenline(fp,&br1);
break;

Повторите ввод!\n");

case 'E':
    // Инициализация структуры cr нулями
    memset(&cr,0,sizeof(circle_));

    // Запрос у пользователя значений для заполнения
структуры
printf("Введите координату точки по горизонтали:");
scanf("%d",&(cr.xcenter));
printf("Введите координату точки по вертикали:");
scanf("%d",&(cr.ycenter));
printf("Введите радиус окружности:");
scanf("%d",&(cr.radius));
printf("Введите толщину линии окружности:");
scanf("%d",&(cr.thickness));
printf("Введите размер области прицеливания:");
scanf("%d",&si);
cr.aim_area = (char)(si & 0xff);
printf("Введите интенсивность цветового канала R:");
scanf("%d",&si);
cr.red = (char)(si & 0xff);
printf("Введите интенсивность цветового канала G:");
scanf("%d",&si);
cr.green = (char)(si & 0xff);
printf("Введите интенсивность цветового канала B:");
scanf("%d",&si);
cr.blue = (char)(si & 0xff);

// Проверка входных значений и добавление структуры в
файл
if(cr.radius==0 || cr.thickness==0 || cr.aim_area==0)
{
    printf("Неверный набор входных значений!");
    continue;
}
else
    addcircle(fp,&cr);
break;

Повторите ввод!\n");

case 'F':
    // Инициализация структуры pl нулями
    memset(&pl,0,sizeof(polyline_));

    // Запрос у пользователя значений для заполнения
структуры

```

```

printf("Введите толщину линии:");
scanf("%d",&(pl.thickness));
printf("Введите интенсивность цветового канала R:");
scanf("%d",&si);
pl.red = (char)(si & 0xff);
printf("Введите интенсивность цветового канала G:");
scanf("%d",&si);
pl.green = (char)(si & 0xff);
printf("Введите интенсивность цветового канала B:");
scanf("%d",&si);
pl.blue = (char)(si & 0xff);
printf("Введите количество точек многоугольника (до
10 штук):");

scanf("%d",&n);
pl.count = n;

// Заполнение массива точек многоугольника
for(int i=0;i<n;i++)
{
    printf("Введите координату точки по
горизонтали:");

    scanf("%d",&x);
    printf("Введите координату точки по вертикали:");
    scanf("%d",&y);
    pl.arr[2*i]=x;
    pl.arr[2*i+1]=y;
}

// Проверка входных значений и добавление структуры в
файл

if(pl.count==0 || pl.thickness==0)
{
    printf("Неверный набор входных значений!
Повторите ввод!\n");

    continue;
}
else
    addpolyline(fp,&pl);
break;

//при завершении работы программы
case 'G':
    flag1 = false;
    break;

//при нажатии клавиши, не ассоциированной с пунктом меню
default:
    printf("В меню отсутствует данный пункт!\n");
    break;
}
}
fclose(fp);
}
break;
case '2':
    // Запрос у пользователя имени/адреса файла
    printf("Введите имя/адрес файла:\n");
    scanf("%s",name);

    // Проверка наличия файла и открытие его для чтения и записи
    if ((fp = fopen(name, "r+b")) == NULL) //создаем файл для записи
        printf("Не удалось открыть файл!\n");
    else

```

```

{
    k = 0; // Инициализация переменной k для подсчета
структур в файле

    // Чтение структур из файла и вывод их на экран
    while((t=strread(fp,&v))!=PTYPE_NULL)
    {
        switch(t)
        {
            case PTYPE_BASE:
printf("%d)BASEDESCR{(%d,%d),(%d,%d)};\n",++k,((basisd_*)v)->refwidth,((basisd_*)v)->refheight,((basisd_*)v)->basex,((basisd_*)v)->basey);
                break;
            case PTYPE_POINT:
printf("%d)POINT(%d,%d);\n",++k,((point_*)v)->x,((point_*)v)->y);
                break;
            case PTYPE_LINE:
printf("%d)LINЕ{(%d,%d),(%d,%d)};\n",++k,((line_*)v)->x,((line_*)v)->y,((line_*)v)->x1,((line_*)v)->y1);
                break;
            case PTYPE_BROKENLINE:
printf("%d)BROKENLINE{(%d,%d),(%d,%d),(%d,%d),...};\n",++k,(((brokenline_*)v)->arr)[0],(((brokenline_*)v)->arr)[1],(((brokenline_*)v)->arr)[2],(((brokenline_*)v)->arr)[3],(((brokenline_*)v)->arr)[4],(((brokenline_*)v)->arr)[5]);
                break;
            case PTYPE_CIRCLE:
printf("%d)CIRCLE{(%d,%d),%d);\n",++k,((circle_*)v)->xcenter,((circle_*)v)->ycenter,((circle_*)v)->radius);
                break;
            case PTYPE_POLYLINE:
printf("%d)POLYLINE{(%d,%d),(%d,%d),(%d,%d),...};\n",++k,(((polyline_*)v)->arr)[0],(((polyline_*)v)->arr)[1],(((polyline_*)v)->arr)[2],(((polyline_*)v)->arr)[3],(((polyline_*)v)->arr)[4],(((polyline_*)v)->arr)[5]);
                break;
        }
    }

    // Проверка наличия структур в файле
    if(k==0)
        printf("Файл пуст!\n");
    else
    {
        // Запрос у пользователя номера редактируемой структуры
        printf("Введите номер редактируемой структуры:");
        scanf("%d",&strnum);

        // Перемещение указателя на начало выбранной структуры
        fseek(fp,0,SEEK_SET);
        for(int i=0;i<strnum;i++)
            t=strread(fp,&v);
        fseek(fp,-arr[t-1],SEEK_CUR);

        // Редактирование выбранной структуры в зависимости от ее
        типа
        switch(t)
        {
            case PTYPE_BASE:
                // Запрос у пользователя значений для заполнения
                структур
                printf("Введите refwidth:");

```

```

scanf("%d",&(bd.refwidth));
printf("Введите refheight:");
scanf("%d",&(bd.refheight));
printf("Введите basex:");
scanf("%d",&(bd.basex));
printf("Введите basey:");
scanf("%d",&(bd.basey));

addbasisdescr(fp,&bd);           // Добавление

структуры в файл

break;
case PTYPE_POINT:
    // Запрос у пользователя значений для заполнения

    printf("Введите координату точки по

    scanf("%d",&(p.x));
    printf("Введите координату точки по вертикали:");
    scanf("%d",&(p.y));
    printf("Введите размер области прицеливания:");
    scanf("%d",&si);
    p.aim_area = (char)(si & 0xff);
    printf("Введите интенсивность цветового канала

R:");

    scanf("%d",&si);
    p.red = (char)(si & 0xff);
    printf("Введите интенсивность цветового канала

G:");

    scanf("%d",&si);
    p.green = (char)(si & 0xff);
    printf("Введите интенсивность цветового канала

B:");

    scanf("%d",&si);
    p.blue = (char)(si & 0xff);

    // Проверка входных значений и добавление

структуры в файл

if(p.aim_area==0)
{
    printf("Неверный набор входных значений!

    continue;
}
else
    addpoint(fp,&p);
break;
case PTYPE_LINE:
    // Запрос у пользователя значений для заполнения

    printf("Введите координату начала линии по

    scanf("%d",&(l.x));
    printf("Введите координату начала линии по

    scanf("%d",&(l.y));
    printf("Введите координату конца линии по

    scanf("%d",&(l.x1));
    printf("Введите координату конца линии по

    scanf("%d",&(l.y1));
    printf("Введите толщину линии:");
    scanf("%d",&(l.thickness));
    printf("Введите размер области прицеливания:");

```

```

scanf("%d",&si);
l.aim_area = (char)(si & 0xff);
printf("Введите интенсивность цветового канала

R:");

scanf("%d",&si);
l.red = (char)(si & 0xff);
printf("Введите интенсивность цветового канала

G:");

scanf("%d",&si);
l.green = (char)(si & 0xff);
printf("Введите интенсивность цветового канала

B:");

scanf("%d",&si);
l.blue = (char)(si & 0xff);

// Проверка входных значений и добавление
структуры в файл
l.aim_area==0)

Повторите ввод!\n");

break;
case PTYPE_BROKENLINE:
// Запрос у пользователя значений для заполнения
структуры

printf("Введите толщину линии:");
scanf("%d",&(brl.thickness));
printf("Введите интенсивность цветового канала

R:");

scanf("%d",&si);
brl.red = (char)(si & 0xff);
printf("Введите интенсивность цветового канала

G:");

scanf("%d",&si);
brl.green = (char)(si & 0xff);
printf("Введите интенсивность цветового канала

B:");

scanf("%d",&si);
brl.blue = (char)(si & 0xff);
printf("Введите количество точек ломаной (до 10
штук):");

scanf("%d",&n);
brl.count = n;

// Заполнение массива точек ломаной
for(int i=0;i<n;i++)
{
    printf("Введите координату точки по
горизонтали:");

    scanf("%d",&x);
    printf("Введите координату точки по
вертикали:");

    scanf("%d",&y);
    brl.arr[2*i]=x;
    brl.arr[2*i+1]=y;
}

// Проверка входных значений и добавление
структуры в файл

```

```

        if(brl.count==0 || brl.thickness==0)
        {
            printf("Неверный набор входных значений!");

            continue;
        }
        else
            addbrokenline(fp,&brl);
break;
case PTYPE_CIRCLE:
    // Запрос у пользователя значений для заполнения
    структуры
    горизонтали:");

    printf("Введите координату точки по

    scanf("%d",&(cr.xcenter));
    printf("Введите координату точки по вертикали:");
    scanf("%d",&(cr.ycenter));
    printf("Введите радиус окружности:");
    scanf("%d",&(cr.radius));
    printf("Введите толщину линии окружности:");
    scanf("%d",&(cr.thickness));
    printf("Введите размер области прицеливания:");
    scanf("%d",&si);
    cr.aim_area = (char)(si & 0xff);
    printf("Введите интенсивность цветового канала

R:");

    scanf("%d",&si);
    cr.red = (char)(si & 0xff);
    printf("Введите интенсивность цветового канала

G:");

    scanf("%d",&si);
    cr.green = (char)(si & 0xff);
    printf("Введите интенсивность цветового канала

B:");

    scanf("%d",&si);
    cr.blue = (char)(si & 0xff);

    // Проверка входных значений и добавление
    структуры в файл
    cr.aim_area==0)

    Повторите ввод!\n");

    if(cr.radius==0 || cr.thickness==0 ||

    {
        printf("Неверный набор входных значений!");

        continue;
    }
    else
        addcircle(fp,&cr);
break;
case PTYPE_POLYLINE:
    // Запрос у пользователя значений для заполнения
    структуры

    printf("Введите толщину линии:");
    scanf("%d",&(pl.thickness));
    printf("Введите интенсивность цветового канала

R:");

    scanf("%d",&si);
    pl.red = (char)(si & 0xff);
    printf("Введите интенсивность цветового канала

G:");

    scanf("%d",&si);
    pl.green = (char)(si & 0xff);
    printf("Введите интенсивность цветового канала

B:");

```

```

scanf("%d",&si);
pl.blue = (char)(si & 0xff);
printf("Введите количество точек многоугольника
(до 10 штук):");

scanf("%d",&n);
pl.count = n;

// Заполнение массива точек многоугольника
for(int i=0;i<n;i++)
{
    printf("Введите координату точки по
горизонтали:");

    scanf("%d",&x);
    printf("Введите координату точки по
вертикали:");

    scanf("%d",&y);
    pl.arr[2*i]=x;
    pl.arr[2*i+1]=y;
}

// Проверка входных значений и добавление
структуры в файл
if(pl.count==0 || pl.thickness==0)
{
    printf("Неверный набор входных значений!
Повторите ввод!\n");
    continue;
}
else
    addpolyline(fp,&pl);
break;

//вывод сообщения об ошибке, если нажатая клавиша не
принадлежит ни одному из пунктов меню
default:
    printf("Неверный номер редактируемого примитива!\n");
    break;
}
}
fclose(fp); //закрытие файла
break;

//для завершения работы программы сбрасываем флаг работы
case '3':
    flag = false;
    break;
case '4':
    // Запрос у пользователя имени/адреса файла
    printf("Введите имя/адрес файла:\n");
    scanf("%s",name); // Считывание имени файла

    nametmp = (char*)calloc(strlen(name)+5,1); // Выделение памяти
для временного имени файла
    sprintf(nametmp,"%s.tmp",name); // Формирование
имени временного файла
    if ((fp = fopen(name, "r+b")) == NULL || (fp2 = fopen(nametmp,
"w+b")) == NULL) //создаем файл для временной записи данных
        printf("Не удалось открыть файл!\n");
    else
    {
        k = 0;
        //Чтение структур из файла и вывод информации о них
        for(int i=0;(t = streadd(fp,&v))!=PTYPE_NULL;i++)

```



```

        {
            if(t==PTYPE_BASE)

printf("%d)BASEDESCR{(%d,%d),(%d,%d)};\n",++k,((basisd_*)v)->refwidth,((basisd_*)v)->refheight,((basisd_*)v)->basex,((basisd_*)v)->basey);
            if(t==PTYPE_POINT)
                printf("%d)POINT(%d,%d);\n",++k,((point_*)v)->x,((point_*)v)->y);
            if(t==PTYPE_LINE)
                printf("%d)LINE{(%d,%d),(%d,%d)};\n",++k,((line_*)v)->x,((line_*)v)->y,((line_*)v)->x1,((line_*)v)->y1);
            if(t==PTYPE_BROKENLINE)

printf("%d)BROKENLINE{(%d,%d),(%d,%d),(%d,%d),...};\n",++k,(((brokenline_*)v)->arr)[0],(((brokenline_*)v)->arr)[1],(((brokenline_*)v)->arr)[2],(((brokenline_*)v)->arr)[3],(((brokenline_*)v)->arr)[4],(((brokenline_*)v)->arr)[5]);
            if(t==PTYPE_CIRCLE)
                printf("%d)CIRCLE{(%d,%d),%d};\n",++k,((circle_*)v)->xcenter,((circle_*)v)->ycenter,((circle_*)v)->radius);
            if(t==PTYPE_POLYLINE)

printf("%d)POLYLINE{(%d,%d),(%d,%d),(%d,%d),...};\n",++k,(((polyline_*)v)->arr)[0],(((polyline_*)v)->arr)[1],(((polyline_*)v)->arr)[2],(((polyline_*)v)->arr)[3],(((polyline_*)v)->arr)[4],(((polyline_*)v)->arr)[5]);

            // Запрос пользователя на удаление структуры
printf("Удалить структуру?Y/N\n");
            b = (char)getch();

            // Если пользователь согласен удалить структуру, то
            // переходим к следующей итерации цикла
            if(b=='Y' || b=='y')
                continue;
            else
            {
                // В зависимости от типа структуры записываем ее во
                // временный файл

switch(t)
{
    case PTYPE_BASE:
        bd = *((basisd_*)v);
        addbasisdescr(fp2,&bd);
        break;
    case PTYPE_POINT:
        p = *((point_*)v);
        addpoint(fp2,&p);
        break;
    case PTYPE_LINE:
        l = *((line_*)v);
        addline(fp2,&l);
        break;
    case PTYPE_BROKENLINE:
        brl = *((brokenline_*)v);
        addbrokenline(fp2,&brl);
        break;
    case PTYPE_CIRCLE:
        cr = *((circle_*)v);
        addcircle(fp2,&cr);
        break;
    case PTYPE_POLYLINE:
        pl = *((polyline_*)v);
        addpolyline(fp2,&pl);
        break;
}
            }
        }

```

```

        if(b!='N' && b!='n')
            printf("Неверный ввод! Файл не изменен!\n");
    }
}
//проверка наличия структур в файле и закрытие потоков
if(k==0)
    printf("Файл пуст!\n");
fclose(fp);
fclose(fp2);

//переименование временного файла в редактируемый и удаление
старой версии редактируемого файла
remove(name);
rename(nametmp,name);
free(nametmp);
}
break;
//при выборе пункта, который отсутствует в меню первого уровня
default:
    printf("В меню отсутствует пункт с таким номером!\n");
    break;
}
}
}

```

Модуль example1.h

```

#pragma once

#define MAX_LOADSTRING 100
#include "resource.h"

```

Модуль example1.cpp

```

// example1.cpp: определяет точку входа для приложения.
//

using namespace ATL;
using namespace std;

//подключение дополнительных заголовочных файлов
#include "stdafx.h"
#include "example1.h"
#include "list_helper.h"
#include <ndllproj.h>

//подключаем заголовочные файлы для разработки приложений в ОС Windows
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <iostream>
#include <algorithm>
#include <atlbase.h>
#include <atlconv.h>
#include <atlstr.h>
#include <windows.h>
#include <shellapi.h>

//подключаем к приложению GDI и GDI+
#include <objidl.h>
#include <gdiplus.h>
using namespace Gdiplus;
#pragma comment (lib,"Gdiplus.lib")

```

```

//объявления глобальных переменных для рисования
static HDC memBit1;           //промежуточный контекст рисования
static HBITMAP hBitmap;
static BITMAP bm;
static HBITMAP hBMP, hOldBMP;
static HDC hdc;               //контекст устройства для рисования в окне приложения
static PAINTSTRUCT ps;
HWND hwndScroll, hwndScroll1; //хэнды для полос прокрутки

//промежуточные переменные для работы программы
int argc;                     //количество переданных программе аргументов
командной строки
char** argv;                  //массив аргументов командной строки программы
uint32_t basex;
uint32_t basey;
float scalex;
float scaley;
int x_ = 0;                   //координата x=0 при скроллинге окна по
горизонтали
int y_ = 0;                   //координата y=0 при скроллинге окна по
вертикали
int cx_ = 0;                  //коэффициент масштабирования по оси x (зависит
от state)
int cy_ = 0;                  //коэффициент масштабирования по оси y (зависит
от state)
uint32_t x,y;
bool AliasFlag = false;      //флаг сглаживания
//состояние масштаба окна - исходный = 0
//200 % по каждой оси = 1
//300 % по каждой оси = 2
//50 % по каждой оси = -1
//33 % по каждой оси = -2
int state = 0;
datatype_t* head = NULL;

//пересчет координаты с учетом базиса
//coord - пересчитываемая координата
//scale - коэффициент масштабирования
//base - базовый коэффициент пересчета
//возвращает пересчитанную координату
int recount(int coord, float scale, int base)
{
    return (int)(coord*scale+base);
}

//обратный пересчет координаты с учетом базиса
//coord - пересчитываемая координата
//scale - коэффициент масштабирования
//base - базовый коэффициент пересчета
//возвращает обратно пересчитанную координату
//rev_recount(recount(coord,...),...) = coord - если при пересчете не было округлений
int rev_recount(int coord, float scale, int base)
{
    return (int)(coord-base)/scale;
}

// Функция для очистки списка
// head - вершина списка
void clearlist(datatype_t* head)
{
    datatype_t* ptr, *nptr;
    ptr = head;               // Установка указателя на вершину списка
    while(ptr)                // Пока не достигнут конец списка

```

```

    {
        nptr = ptr->next;    //Переход к следующему элементу списка
        LIST_DEL(head,ptr); //Удаление предыдущего элемента списка
        free(ptr->data);     //Очистка памяти, выделенной через calloc
        delete ptr->r;       //Очистка памяти региона
        free(ptr);
        ptr = nptr;
    }
}

//заполнение списка данными о примитивах из файла
//fname - строка с именем открываемого файла
//head - указатель на указатель на вершину списка
bool readfile(char* fname,datatype_t** head)
{
    //промежуточные переменные для заполнения списка и вычислений
    void* v;
    ptype_t type = PTYPE_NULL;
    datatype_t* ptr;
    FILE* fp;
    basisd* ptrb;
    point* ptrp;
    line* ptrl;
    brokenline* ptrbrl;
    circle* ptrc;
    polyline* ptrpl;
    POINT points[4];
    POINT arrp[10];
    uint32_t arrx[10];
    uint32_t array[10];
    int x,y,x1,y1,aim_area,count,xcenter,ycenter,radius;
    HRGN region;

    // Если список уже содержит элементы, возвращаем true
    if(*head)
        return true;
    // Открываем файл для чтения
    if ((fp = fopen(fname, "rb")) == NULL) //открываем файл для чтения данных
        return false;
    else
    {
        // Читаем данные из файла, пока не достигнем конца файла
        while((type=strread(fp,&v))!=PTYPE_NULL)
        {
            //Если первый элемент файла не базис - файл поврежден
            if(*head==NULL && type!=PTYPE_BASE)
                return false;

            // Создаем новый элемент списка и устанавливаем его тип и
даные
            ptr = (datatype_t*)calloc(1,sizeof(datatype_t));
            ptr->type = type;
            ptr->data = v;

            // Создаем объект региона, связанный с элементом, в
зависимости от типа данных
            switch(type)
            {
                //примитиву базиса регион не требуется
                case PTYPE_BASE:
                    ptr->r = NULL;
                    break;

```

```

//примитив точка - квадратный регион размерами
2*aim_area на 2*aim_area
case PTYPE_POINT:
    aim_area = ((point_*)(ptr->data))->aim_area;
    x = ((point_*)(ptr->data))->x;
    y = ((point_*)(ptr->data))->y;
    //создание объекта HRGN
    region = CreateRectRgn(x-aim_area,y-
aim_area,x+aim_area,y+aim_area);
    //создание объекта региона на основе объекта
    HRGN
    ptr->r = new Region(region);
    //удаление объекта HRGN
    DeleteObject(region);

break;
//примитив линия - квадратный регион размерами
2*aim_area на 2*aim_area
case PTYPE_LINE:
    aim_area = ((line_*)(ptr->data))->aim_area;
    x = ((line_*)(ptr->data))->x;
    y = ((line_*)(ptr->data))->y;
    x1 = ((line_*)(ptr->data))->x1;
    y1 = ((line_*)(ptr->data))->y1;

    //вычисление координат середины прямой
    xcenter = (x1 + x)/2;
    ycenter = (y1 + y)/2;

    //создание объекта HRGN
    region = CreateRectRgn(xcenter-aim_area,ycenter-
aim_area,xcenter+aim_area,ycenter+aim_area);
    //создание объекта региона на основе объекта
    HRGN
    ptr->r = new Region(region);
    //удаление объекта HRGN
    DeleteObject(region);

break;
case PTYPE_BROKENLINE:
    count = ((brokenline_*)(ptr->data))->count;

    //разбиение массива arr на массивы абсцисс и
ординат точек
    for(int i=0;i<10;i++)
    {
        arrx[i]=((brokenline_*)(ptr->data))-
>arr[2*i];
        arry[i]=((brokenline_*)(ptr->data))-
>arr[2*i+1];
    }

    //создание объектов point на основе arrx и arry
    for(int i=0;i<=count-1;i++)
    {
        arrp[i].x = arrx[i];
        arrp[i].y = arry[i];
    }
    //создание объекта HRGN
    region = CreatePolygonRgn(arrp,count,WINDING);
    //создание объекта региона на основе объекта
    HRGN
    ptr->r = new Region(region);
    //удаление объекта HRGN
    DeleteObject(region);

```

```

        break;
    case PTYPE_CIRCLE:
        xcenter = ((circle_*)(ptr->data))->xcenter;
        ycenter = ((circle_*)(ptr->data))->ycenter;
        radius = ((circle_*)(ptr->data))->radius;
        aim_area = ((circle_*)(ptr->data))->aim_area;

        //создание объекта HRGN
        region = CreateEllipticRgn(xcenter-radius-
aim_area,ycenter+radius+aim_area,xcenter+radius+aim_area,ycenter-radius-aim_area);
        //создание объекта региона на основе объекта
        HRGN

        ptr->r = new Region(region);
        //удаление объекта HRGN
        DeleteObject(region);
    break;
    case PTYPE_POLYLINE:
        count = ((brokenline_*)(ptr->data))->count;

        //разбиение массива arr на массивы абсцисс и
ординат точек

        for(int i=0;i<10;i++)
        {
            arrx[i]=((brokenline_*)(ptr->data))-
>arr[2*i];
            array[i]=((brokenline_*)(ptr->data))-
>arr[2*i+1];
        }

        //создание объектов point на основе arrx и array
        for(int i=0;i<=count-1;i++)
        {
            arrp[i].x = arrx[i];
            arrp[i].y = array[i];
        }
        //создание объекта HRGN
        region = CreatePolygonRgn(arrp,count,WINDING);
        //создание объекта региона на основе объекта
        HRGN

        ptr->r = new Region(region);
        //удаление объекта HRGN
        DeleteObject(region);
    break;
}
LIST_ADD(*head,ptr); // Добавляем элемент в список
}
return true;
}
}

// Функция для отрисовки графических объектов из списка в контексте устройства
void vgraph(datatype_t* head_,HDC hdc,int reswidth,int resheight,bool isalias)
{
    ptype_t type;
    datatype_t* ptr = head_;
    basisd_* ptrb;
    point_* ptrp;
    line_* ptrl;
    brokenline_* ptrbrl;
    circle_* ptrc;
    polyline_* ptrpl;
    uint32_t arrx[10];
    uint32_t array[10];
    bool res = false;

```

```

Graphics* g = new Graphics(hdc);

// Если isalias равен true, устанавливаем режим сглаживания
if(isalias)
    g->SetSmoothingMode(SmoothingModeAntiAlias);

// Проходим по всем элементам списка и отрисовываем их в зависимости от типа
данных
while(ptr)
{
    switch(ptr->type)
    {
        case PTYPE_BASE:
            ptrb = (basisd_*)ptr->data;
            basex = ptrb->basex;
            basey = ptrb->basey;
            scalex = reswidth/(float)(ptrb->refwidth);
            scaley = resheight/(float)(ptrb->refheight);
            break;
        case PTYPE_POINT:
            ptrp = (point_*)ptr->data;
            ppoint(g, recount(ptrp->x, scalex, basex), recount(ptrp->y, scaley, basey), /*ptrp->aim_area,*/ ptrp->red, ptrp->green, ptrp->blue);
            break;
        case PTYPE_LINE:
            ptrl = (line_*)ptr->data;
            pline(g, recount(ptrl->x, scalex, basex), recount(ptrl->y, scaley, basey), recount(ptrl->x1, scalex, basex), recount(ptrl->y1, scaley, basey), ptrl->thickness, ptrl->red, ptrl->green, ptrl->blue);
            break;
        case PTYPE_BROKENLINE:
            ptrbrl = (brokenline_*)ptr->data;
            for(int i=0; i<10; i++)
            {
                arrx[i]=recount((ptrbrl->arr)[2*i], scalex, basex);
                array[i]=recount((ptrbrl->arr)[2*i+1], scaley, basey);
            }
            pbrokenline(g, arrx, array, ptrbrl->count, ptrbrl->thickness, ptrbrl->red, ptrbrl->green, ptrbrl->blue);
            break;
        case PTYPE_CIRCLE:
            ptrc = (circle_*)ptr->data;
            pcircle(g, recount(ptrc->xcenter, scalex, basex), recount(ptrc->ycenter, scaley, basey), ptrc->radius*(scalex <= scaley ? scalex : scaley), ptrc->thickness, ptrc->red, ptrc->green, ptrc->blue);
            break;
        case PTYPE_POLYLINE:
            ptrpl = (polyline_*)ptr->data;
            for(int i=0; i<10; i++)
            {
                arrx[i]=recount((ptrpl->arr)[2*i], scalex, basex);
                array[i]=recount((ptrpl->arr)[2*i+1], scaley, basey);
            }
            ppolyline(g, arrx, array, ptrpl->count, ptrpl->thickness, ptrpl->red, ptrpl->green, ptrpl->blue);
            break;
    }
    ptr = ptr->next;
}
delete g;

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

//точка входа в оконное приложение

```

```

//hInstance - дескриптор экземпляра приложения, который был передан операционной системой
при запуске приложения.
//lpCmdLine - указатель на строку, содержащую аргументы командной строки, переданные
приложению.
//nCmdShow - флаг, указывающий, как окно приложения должно быть показано при запуске.
INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE, PSTR, INT iCmdShow)
{
    HWND          hWnd;
    MSG           msg;
    WNDCLASS      wndClass;
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR     gdiplusToken;

    // Инициализация GDI+.
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);
    wndClass.style      = CS_HREDRAW | CS_VREDRAW;
    wndClass.lpfnWndProc = WndProc;
    wndClass.cbClsExtra  = 0;
    wndClass.cbWndExtra  = 0;
    wndClass.hInstance  = hInstance;
    wndClass.hIcon       = LoadIcon(NULL, IDI_APPLICATION);
    wndClass.hCursor     = LoadCursor(NULL, IDC_ARROW);
    wndClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndClass.lpszMenuName = NULL;
    wndClass.lpszClassName = TEXT("MyApp");

    RegisterClass(&wndClass);

    // Преобразование командной строки в массив аргументов в формате Unicode
    LPWSTR* lpArgv = CommandLineToArgvW( GetCommandLine(), &argc );
    argv = (char**)malloc(argc*sizeof(char*)); // Выделение памяти для массива аргументов
    в формате ASCII
    int size, i = 0; // Инициализация счетчика
    итераций
    // Цикл для преобразования каждого аргумента из формата Unicode в формат ASCII
    for( ; i < argc; ++i )
    {
        size = wcslen(lpArgv[i]) + 1; // Вычисление размера
        аргумента в символах, включая нулевой символ
        argv[i] = (char*)malloc(size); // Выделение памяти для
        аргумента в формате ASCII
        wcstombs(argv[i], lpArgv[i], size); // Преобразование аргумента
        из формата Unicode в формат ASCII
    }
    LocalFree( lpArgv ); // Освобождение памяти,
    выделенной для массива аргументов в формате Unicode

    //создание основного окна приложения
    hWnd = CreateWindow(
        TEXT("MyApp"), // название класса окна
        TEXT("MyApp"), // заголовок окна
        WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU, // стиль окна
        CW_USEDEFAULT, // начальная позиция x
        CW_USEDEFAULT, // начальная позиция y
        CW_USEDEFAULT, // начальный размер x
        CW_USEDEFAULT, // начальный размер y
        NULL,
        // дескриптор родительского окна
        NULL,
        // дескриптор меню окна
        hInstance, // дескриптор экземпляра программы
        NULL); // параметры создания

    RECT rcParent;

```



```

GetClientRect(hWnd, &rcParent);
ShowWindow(hWnd, iCmdShow);
UpdateWindow(hWnd);

//цикл выборки и обработки сообщений
while(GetMessage(&msg, NULL, 0, 0)) // Цикл выполняется, пока не
получено сообщение WM_QUIT
{
    TranslateMessage(&msg); // Преобразование
    виртуальных кодов клавиш в символьные коды для сообщений WM_KEYDOWN и WM_KEYUP
    DispatchMessage(&msg); // Передача сообщения в
    оконную процедуру соответствующего окна
}

GdiplusShutdown(gdiplusToken);
return msg.wParam;
} // WinMain

LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam)
{
    // Обработчик сообщений окна
    switch(message)
    {
        // Обработка сообщения WM_CREATE
        case WM_CREATE:
            //Если программе передано недостаточное число аргументов
            if(argc<3)
            {
                // Вывод сообщения об ошибке и завершение работы приложения
                MessageBox(hWnd,TEXT("Command line arguments are not
set!"),TEXT("Error!"),MB_OK);
                PostQuitMessage(0);
                return 0;
            }
            // Загрузка изображения из файла
            hBMP = (HBITMAP)LoadImageA( NULL, (LPCSTR)argv[1], IMAGE_BITMAP, 0, 0,
LR_LOADFROMFILE);
            // Проверка успешности загрузки изображения
            if(hBMP == NULL)
            {
                // Вывод сообщения об ошибке и завершение работы приложения
                MessageBox(hWnd,TEXT("Unable to open bitmap
file!"),TEXT("Error!"),MB_OK);
                PostQuitMessage(0);
            }
            // Получение информации об изображении
            GetObject(hBMP, sizeof(bm), &bm);
            // Создание горизонтального скроллбара
            hWndScroll = CreateWindowEx(0, L"SCROLLBAR", NULL, WS_CHILD | WS_VISIBLE |
SBS_HORZ, 0, bm.bmHeight-17, bm.bmWidth, 17, hWnd, NULL,(HINSTANCE)GetWindowLongPtr(hWnd,
GWLP_HINSTANCE), NULL);
            // Создание вертикального скроллбара
            hWndScroll1 = CreateWindowEx(0, L"SCROLLBAR", NULL, WS_CHILD | WS_VISIBLE |
SBS_VERT, bm.bmWidth-17, 0, 17, bm.bmHeight-17, hWnd,
NULL,(HINSTANCE)GetWindowLongPtr(hWnd, GWLP_HINSTANCE), NULL);
            // Чтение векторного файла
            if(!readfile(argv[2],&head))
            {
                // Очистка списка и вывод сообщения об ошибке
                clearlist(head);
                MessageBox(hWnd,TEXT("Unable to open vector
file!"),TEXT("Error!"),MB_OK);
                PostQuitMessage(0);
            }

```

```

    }
    return 0;
// Обработка сообщения WM_LBUTTONDOWN
case WM_LBUTTONDOWN:
    char buff[300]; // Буфер для вывода информации о фигуре
    datatype_t* ptrd; // Указатель на данные фигуры

    // Получение координат клика мыши
    x=lParam & 0x0000FFFF;
    y=(lParam & 0xFFFF0000)>>16;

    // Преобразование координат в зависимости от масштаба
    switch(state)
    {
        case 1:
            x = x/2;
            y = y/2;
            break;
        case 2:
            x = x/3;
            y = y/3;
            break;
        case -1:
            x = x*2;
            y = y*2;
            break;
        case -2:
            x = x*3;
            y = y*3;
            break;
    }

    // Смещение координат на текущее положение изображения
    x += x_;
    y += y_;

    // Проверка, находится ли клик внутри какой-либо фигуры
    if((ptrd=IsInside(rev_recount(x, scalex, basex), rev_recount(y, scaley, basey), head))!=
NULL)
    {

        // Вывод информации о фигуре в буфер
        switch(ptrd->type)
        {
            case PTYPE_POINT:
                sprintf_s(buff, "POINT:\nCoords: %d %d\nAiming area:
%d\nColour: %d %d %d",((point_*)ptrd->data)->x,((point_*)ptrd->data)->y,((point_*)ptrd-
>data)->aim_area,((point_*)ptrd->data)->red,((point_*)ptrd->data)->green,((point_*)ptrd-
>data)->blue);
                break;
            case PTYPE_LINE:
                sprintf_s(buff, "LINE:\nCoords of the beginning point:
%d %d\nCoords of the ending point: %d %d\nThickness: %d\nAiming area: %d\nColour: %d %d
%d",((line_*)ptrd->data)->x,((line_*)ptrd->data)->y,((line_*)ptrd->data)-
>x1,((line_*)ptrd->data)->y1,((line_*)ptrd->data)->thickness,((line_*)ptrd->data)-
>aim_area,((line_*)ptrd->data)->red,((line_*)ptrd->data)->green,((line_*)ptrd->data)-
>blue);
                break;
            case PTYPE_BROKENLINE:
                sprintf_s(buff, "BROKENLINE:\nThickness: %d\nColour: %d
%d %d\nPoints:\n",((brokenline_*)ptrd->data)->thickness,((brokenline_*)ptrd->data)-
>red,((brokenline_*)ptrd->data)->green,((brokenline_*)ptrd->data)->blue);
                for(int i=0;i<((brokenline_*)ptrd->data)->count;i++)

```

```

        sprintf_s(buff+strlen(buff),sizeof(buff)-
strlen(buff),"%d)%d %d\n",i+1,((brokenline_*)ptrd->data)->arr[i*2],((brokenline_*)ptrd-
>data)->arr[i*2+1]);
        break;
        case PTYPE_CIRCLE:
            sprintf_s(buff, "CIRCLE:\nCoords of the center: %d
%d\nRadius: %d\nThickness: %d\nAiming area: %d\nColour: %d %d %d\n",((circle_*)ptrd-
>data)->xcenter,((circle_*)ptrd->data)->ycenter,((circle_*)ptrd->data)-
>radius,((circle_*)ptrd->data)->thickness,((circle_*)ptrd->data)-
>aim_area,((circle_*)ptrd->data)->red,((circle_*)ptrd->data)->green,((circle_*)ptrd-
>data)->blue);
            break;
        case PTYPE_POLYLINE:
            sprintf_s(buff, "POLYLINE:\nThickness: %d\nColour: %d
%d %d\nPoints:\n",((polyline_*)ptrd->data)->thickness,((polyline_*)ptrd->data)-
>red,((polyline_*)ptrd->data)->green,((polyline_*)ptrd->data)->blue);
            for(int i=0;i<((brokenline_*)ptrd->data)->count;i++)
                sprintf_s(buff+strlen(buff),sizeof(buff)-
strlen(buff),"%d)%d %d\n",i+1,((polyline_*)ptrd->data)->arr[i*2],((polyline_*)ptrd-
>data)->arr[i*2+1]);
            break;
    }

    // Вывод информации о фигуре в диалоговое окно
    MessageBox(hWnd,CA2W(buff),TEXT("Aiming!"),MB_OK);
}
return 0;
case WM_KEYDOWN:

    //включение/выключение сглаживания нажатием клавиши A
    if(wParam=='A')
    {
        AliasFlag = !AliasFlag;

        // Вызов функций перерисовки окна
        InvalidateRect(hWnd,NULL,TRUE);
        UpdateWindow(hWnd);
    }

    //увеличение масштаба окна нажатием клавиши M
    if(wParam=='M')
    {
        // Если текущий масштаб меньше 2
        if(state<2)
        {

            // Сброс смещения и положения скроллбаров
            x_ = 0;
            y_ = 0;
            SetScrollPos(hWndScroll,SB_CTL,0,true);
            SetScrollPos(hWndScroll1,SB_CTL,0,true);

            state++;        // Увеличение масштаба на 1

            // Вызов функций перерисовки окна
            InvalidateRect(hWnd,NULL,TRUE);
            UpdateWindow(hWnd);
        }
    }

    //уменьшение масштаба окна нажатием клавиши M
    if(wParam=='L')
    {

```

```

        // Если текущий масштаб больше -2
        if(state>-2)
        {
            // Сброс смещения и положения скроллбаров
            x_ = 0;
            y_ = 0;
            SetScrollPos(hwndScroll,SB_CTL,0,false);
            SetScrollPos(hwndScroll1,SB_CTL,0,false);

            state--;        // Уменьшение масштаба на 1

            // Вызов функций перерисовки окна
            InvalidateRect(hwnd,NULL,TRUE);
            UpdateWindow(hwnd);
        }
    }
    return 0;
// Обработчик сообщения WM_VSCROLL
case WM_VSCROLL:
    // Если перемещен ползунок скроллбара
    if (LOWORD(wParam) == SB_THUMBPOSITION)
    {
        int nPos = HIWORD(wParam);                // Получение новой
позиции ползунка
        SetScrollPos(hwndScroll1, SB_CTL, nPos, false);    // Установка новой
позиции скроллбара
        y_ = nPos;                // Установка нового
значения смещения по оси Y

        // Вызов функций перерисовки окна
        InvalidateRect(hwnd,NULL,TRUE);
        UpdateWindow(hwnd);
    }

    // Если нажата кнопка "Стрелка вверх"
    if (LOWORD(wParam) == SB_LINEUP)
    {
        int nPos = GetScrollPos(hwndScroll1, SB_CTL);    // Получение текущей
позиции скроллбара
        nPos -= 10;                // Уменьшение позиции
на 10
        nPos = (nPos < 0) ? 0 : nPos;                // Если позиция
меньше 0, установка в 0
        SetScrollPos(hwndScroll1, SB_CTL, nPos, true);    // Установка новой
позиции скроллбара
        y_ = nPos;                // Установка нового
значения смещения по оси Y

        // Вызов функций перерисовки окна
        InvalidateRect(hwnd,NULL,TRUE);
        UpdateWindow(hwnd);
    }

    // Если нажата кнопка "Стрелка вниз"
    if (LOWORD(wParam) == SB_LINEDOWN)
    {
        int nPos = GetScrollPos(hwndScroll1, SB_CTL);    // Получение текущей
позиции скроллбара
        int nMin,nMax;
        GetScrollRange(hwndScroll1, SB_CTL, &nMin, &nMax); // Получение
минимальной и максимальной позиции скроллбара
        nPos+=10;                // Увеличение позиции
на 10
    }
}

```

```

        nPos = (nPos > nMax) ? nMax : nPos;           // Если позиция
        больше максимальной, то установка в максимальную
        SetScrollPos(hwndScroll1, SB_CTL, nPos, true); // Установка новой
        позиции скроллбара
        y_ = nPos;                                     // Установка нового
        значения смещения по оси Y

        // Вызов функций перерисовки окна
        InvalidateRect(hwnd, NULL, TRUE);
        UpdateWindow(hwnd);
    }
    return 0;

// Обработчик сообщения WM_HSCROLL
case WM_HSCROLL:
    // Если перемещен ползунок скроллбара
    if (LOWORD(wParam) == SB_THUMBPOSITION)
    {
        int nPos = HIWORD(wParam);                    // Получение новой
        позиции ползунка
        SetScrollPos(hwndScroll, SB_CTL, nPos, false); // Установка новой
        позиции скроллбара
        x_ = nPos;
        // Установка нового значения смещения по оси X

        // Вызов функций перерисовки окна
        InvalidateRect(hwnd, NULL, TRUE);
        UpdateWindow(hwnd);
    }

    // Если нажата кнопка "Стрелка влево"
    if (LOWORD(wParam) == SB_LINEUP)
    {
        int nPos = GetScrollPos(hwndScroll, SB_CTL); // Получение текущей
        позиции скроллбара
        nPos -= 10;
        // Уменьшение позиции на 10
        nPos = (nPos < 0) ? 0 : nPos;                 // Если позиция
        меньше 0, то установка в 0
        SetScrollPos(hwndScroll, SB_CTL, nPos, false); // Установка новой
        позиции скроллбара
        x_ = nPos;
        // Установка нового значения смещения по оси X

        // Вызов функций перерисовки окна
        InvalidateRect(hwnd, NULL, TRUE);
        UpdateWindow(hwnd);
    }

    // Если нажата кнопка "Стрелка вправо"
    if (LOWORD(wParam) == SB_LINEDOWN)
    {
        int nPos = GetScrollPos(hwndScroll, SB_CTL); // Получение
        текущей позиции скроллбара
        int nMin, nMax;
        GetScrollRange(hwndScroll, SB_CTL, &nMin, &nMax); // Получение
        минимальной и максимальной позиции скроллбара
        nPos += 10;
        // Увеличение позиции на 10
        nPos = (nPos > nMax) ? nMax : nPos;
        // Если позиция больше максимальной, то установка в максимальную
        SetScrollPos(hwndScroll, SB_CTL, nPos, false); // Установка
        новой позиции скроллбара
    }
}

```

```

        x_ = nPos; // Установка
нового значения смещения по оси X

        // Вызов функций перерисовки окна
        InvalidateRect(hWnd, NULL, TRUE);
        UpdateWindow(hWnd);
    }
    return 0;
// Обработчик сообщения WM_PAINT
case WM_PAINT:
    //Если программе передано недостаточное число аргументов
    if(argc<3)
        return 0;
    hdc = BeginPaint(hWnd, &ps);
// Получение контекста устройства
для окна hWnd
    memBit1 = CreateCompatibleDC(hdc);
// Создание совместимого контекста
устройства
    //Загрузка изображения из файла в виде битовой карты
    hBMP = (HBITMAP)LoadImageA( NULL, (LPCSTR)argv[1], IMAGE_BITMAP, 0, 0,
LR_LOADFROMFILE);
    // Выбор загруженной битовой карты в контексте устройства memBit1
    hOldBMP = (HBITMAP)SelectObject(memBit1, hBMP);

    // Вызов функции vgraph для отрисовки графики на битовой карте
    vgraph(head, memBit1, bm.bmWidth, bm.bmHeight, AliasFlag);
    // Вызов функции PaintBitmapinHDC для отрисовки битовой карты в контексте
устройства hdc
    PaintBitmapinHDC(hWnd, hWndScroll, hWndScroll1, hdc, memBit1, state, x_, y_, bm.bmWidth, bm.bmHeight);
    SelectObject(memBit1, hOldBMP); // Возврат предыдущего объекта контекст
устройства memBit1
    DeleteObject(hBMP); // Удаление объекта HBITMAP
    DeleteDC(memBit1); // Удаление контекста устройства memBit1

    //Завершение рисования и освобождение контекста устройства hdc
    EndPaint(hWnd, &ps);
    ReleaseDC(hWnd, hdc);
    return 0;
case WM_DESTROY:
    //очистка связанного списка
    clearlist(head);
    //очистка массива для хранения переданных программе аргументов командной строки
    for(int i=0 ; i < argc; ++i )
        free( argv[i] );
    free( argv );
    //завершение работы программы
    PostQuitMessage(0);
    return 0;
default:
    //реакция на прочие сообщения окну - по умолчанию
    return DefWindowProc(hWnd, message, wParam, lParam);
}
} // WndProc
// Глобальные переменные:
HINSTANCE hInst; // текущий экземпляр
TCHAR szTitle[MAX_LOADSTRING]; // Текст строки заголовка
TCHAR szWindowClass[MAX_LOADSTRING]; // имя класса главного окна

// Отправить объявления функций, включенных в этот модуль кода:
ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);

```

```

INT_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY _tWinMain(HINSTANCE hInstance,
                      HINSTANCE hPrevInstance,
                      LPTSTR lpCmdLine,
                      int nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
    // TODO: разместите код здесь.
    WinMain(hInstance, hPrevInstance, reinterpret_cast<PSTR>(lpCmdLine), nCmdShow);
    MSG msg;
    HACCEL hAccelTable;

    // Инициализация глобальных строк
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadString(hInstance, IDC_EXAMPLE1, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Выполнить инициализацию приложения:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_EXAMPLE1));
    // Цикл основного сообщения:
    while (GetMessage(&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int) msg.wParam;
}

//
// ФУНКЦИЯ: MyRegisterClass()
//
// НАЗНАЧЕНИЕ: регистрирует класс окна.
//
// КОММЕНТАРИИ:
//
// Эта функция и ее использование необходимы только в случае, если нужно, чтобы данный
код
// был совместим с системами Win32, не имеющими функции RegisterClassEx'
// которая была добавлена в Windows 95. Вызов этой функции важен для того,
// чтобы приложение получило "качественные" мелкие значки и установило связь
// с ними.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;

```

```

    wcex.cbWndExtra          = 0;
    wcex.hInstance          = hInstance;
    wcex.hIcon              = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_EXAMPLE1));
    wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName       = MAKEINTRESOURCE(IDC_EXAMPLE1);
    wcex.lpszClassName      = szWindowClass;
    wcex.hIconSm            = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassEx(&wcex);
}

//
// ФУНКЦИЯ: InitInstance(HINSTANCE, int)
//
// НАЗНАЧЕНИЕ: сохраняет обработку экземпляра и создает главное окно.
//
// КОММЕНТАРИИ:
//
// В данной функции дескриптор экземпляра сохраняется в глобальной переменной, а
также
// создается и выводится на экран главное окно программы.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND hWnd;

    hInst = hInstance; // Сохранить дескриптор экземпляра в глобальной переменной
    hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

    if (!hWnd)
    {
        return FALSE;
    }

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}

// Обработчик сообщений для окна "О программе".
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
    return (INT_PTR)FALSE;
}

```