Министерство Образования и Науки Российской Федерации  
Новосибирский Государственный Технический Университет  
Кафедра Теоретической Прикладной Информатики

**Лабораторная работа № 4**

**по дисциплине «Объектно-ориентированное программирование»**

Факультет: ФПМИ

Группа: ПМ-53

Студент: Тябин Егор Алексеевич

Преподаватель: Еланцева Ирина Леонидовна

### Новосибирск 2016

# Условие задачи

Путем модификации программ, разработанных в лабораторных работах № 1, 2, разработать программу такую, чтобы в ней были определены несколько классов, реализующих понятие геометрической фигуры в графической системе: абстрактный класс «Фигура», содержащий чисто виртуальные функции; класс «Закрашенный», позволяющий задать кисть, ее параметры и, возможно, осуществить закраску; класс «Фигура-контур» – потомок класса «Фигура»; класс «Закрашенная фигура» – потомок класса «Фигура-контур», класс «Закрашенный» при этом использовать либо как второго родите-ля (множественное наследование), либо как часть класса «Закрашенная фигура» (агрегация); класс «Комбинированная фигура», реализующий две вложенные фигуры с закраской между ними.

Реализацию классов поместить в отдельный файл. Разработать функцию, демонстрирующую поведение разработанных классов, **включая демонстрацию механизма позднего связывания**.

# 2. Анализ задачи

*Дано:*

Ромб – четырехугольник, противоположные стороны которого параллельны, и все стороны равны.  
Последовательность целых чисел, которые являются координатами вершин ромба, всего 8 чисел.  
Далее следуют графические характеристики.   
Формат: <тип пера[0-6]> <толщина пера> (<цвет границы(пера) в формате RGB>) <тип кисти(заливки)> (<цвет заливки в формате RGB>)  
Пример задания фигуры в файле: 200 150 250 100 300 150 250 200 0 5 200 200 0  
 -1 100 100 200  
В файле может находиться ромб-полость внутри первого ромба, третьей строкой вписываются координаты.

*Результат:*   
Нарисованный ромб/ромбы в графическом интерфейсе visual studio или сообщение об ошибке.

## Метод решения

Для работы с нашей задачей выделим несколько основных подзадач.

## Выделение основных подзадач

1. Установка новых значений вершин ромба
2. Проверка, является ли фигура ромбом
3. Проверка, не выходит ли фигура за границы окна
4. Рисование ромба
5. Загрузка из файла
6. Сохранение в файл
7. Перемещение ромба

1) Атрибутам класса, отвечающие за значения координат вершин, присваивается новое значение

2) Ромб – четырехугольник, в котором все стороны равны и противоположные стороны параллельны. Соответственно проверяется равны ли стороны, и параллельны ли противоположные стороны.

3) Каждая вершина проверяется на выход за границы консольного окна

4) Рисуется многоугольник по 4 вершинам ромба (функция Polygon)

5) Данные считываются из файла (формат представления данных можно посмотреть в *Дано*), после чего проверяется, является ли фигура ромбом, и не выходит ли она за границы окна

6) Данные сохраняются в указанный файл (формат смотреть в *Дано*)

7) Каждая вершина ромба перемещается на указанное количество пикселей-координат по осям Ох и Оу. Если ромб с полостью, то для полости проделывается тоже самое.

# 3. Структуры данных, используемые для представления исходных данных и результатов задачи

## Внешнее представление входных данных

Последовательность целых чисел и символов.

## Внешнее представление выходных данных

Надпись или графическое изображение. А так-же последовательность целых чисел и символов в файле.

# Внутреннее представление входных данных

*Class shape:*класс “shape” , реализующий понятие геометрической фигуры – ромба, в графической системе.

*Class romb:*класс “romb” , реализующий понятие геометрической фигуры – контура ромба.

POINT t[5] - массив типа POINT, в котором хранятся координаты 4 вершин ромба. Пятая координата равна начальной для достижения замкнутости.

color col– элементы типа color, которые являются параметрическими характеристиками цвета пера, формат RGB.

int type,– целочисленный элемент, в котором записывается тип пера.

int width – целочисленный элемент, являющийся толщиной пера; не может быть меньше 0.

*Class RbFill:*класс “RbFill” , который является характеристиками заливки ромба.

color col– элементы типа color, которые являются параметрическими характеристиками цвета кисти, формат RGB.

int type, – целочисленный элемент, в котором записывается тип кисти.

*Class SolidRomb:*класс “SolidRomb” , реализующий понятие геометрической фигуры – закрашенного ромба.

Является наследником RbFill и romb, образуя закрашенный ромб.

*Class HollowRomb:*класс “HollowRomb” , реализующий понятие геометрической фигуры – закрашенного ромба с полостью в виде ромба внутри него.

Является наследником SolidRomb.

POINT tin[5] – массив типа POINT, в котором хранятся координаты 4-х вершин ромба-полости. Пятая координата равна начальной для достижения замкнутости.

## Внутреннее представление данных

struct color { int r, g, b; } – структура color, которая является совокупностью 3-х целочисленных элементов являющимися параметрами цвета в формате RGB

int i – целочисленный элемент используемый для перебора

float koef – элемент вещественного типа используемый для перебора

float dl – элемент вещественного типа обозначающий длину стороны

bool exit – элемент типа bool, который отвечает за выход из программы, когда того пожелает пользователь.

unsigned \_\_int8 MenuFlags – элемент типа unsigned\_\_int8, который отвечает за раскраску операций, которые на данный момент доступны.

# 4. Укрупненный алгоритм решения задачи

check\_figure(POINT \*ppt) { Проверка, является ли фигура ромбом

Начиная с i=0, Повторять:  
 koef[i] = slope(ppt[i], ppt[i + 1]);

dl[i] = sqrt((ppt[i].x - ppt[i + 1].x)\*(ppt[i].x - ppt[i + 1].x) + (ppt[i].y - ppt[i + 1].y)\*(ppt[i].y - ppt[i + 1].y));  
i=i+1;

Пока i<4;

Начиная с i=0, Повторять:

Если dl[i] ≠ dl[i + 1]: возврат 2;  
 i=i+1;  
 Пока i<3;

Начиная с i=0, Повторять:

Если koef[i] ≠ koef[i + 2]: возврат 3;  
i=i+1;

Пока i<2;  
  
Возврат 0;  
}

check\_borders(POINT \*ppt, long rr, long rb) { Проверка выхода вершин ромба за границы окна

Начиная с i=0, Повторять:

Если ppt[i].x >= rr || ppt[i].y >= rb: возврат 1;  
i=i+1;

Пока i<4;  
  
Возврат 0;  
}

check\_FinF(POINT m1, POINT m2, POINT \*n) **{ Проверка на вложенность отрезка в первый ромб.**

T=0;  
 Начиная с i=0, Повторять:   
 z = (n[i].x - m1.x)\*(m2.y - m1.y) - (n[i].y - m1.y)\*(m2.x - m1.x);  
 Если z<0 : t=t+1;  
 Иначе: возврат 0;  
 i=i+1;  
 Пока: i<4  
 Возврат t;  
 **}**

SetPos(int new\_x, int new\_y) { Перемещение фигуры по указанному вектору

Начиная с i=0, Повторять:  
 Если координата вершины + длина перемещения выходит за границы окна, то   
 ошибка  
 Пока i<4  
Начиная с i=0, Повторять:  
 t[i].x=new\_x+t[i].x  
 t[i].y=new\_y+t[i].y  
 Пока i<4

} //Для фигуры с полостью в SetPos добавляется приращение для точек tin[i].

main() {

Функция-переключатель:

Если ‘1’: Задать новую фигуру.  
Если ‘2’: Загрузить фигуру из файла  
Если ‘3’: Изменить фигуру.  
 Функция переключатель:  
 Если ‘1’: изменить контур  
 Если ‘2’: изменить заливку  
 Если ‘3’: изменить полость  
 Если ‘4’: переместить фигуру  
Если ‘4’: Нарисовать фигуру-контур.  
Если ‘5’: Нарисовать фигуру с заливкой  
Если ‘6’: Нарисовать фигуру с полостью  
Если ‘7’: Сохранить фигуру в файл  
Если ‘m’: Ввод характеристик ромба()  
Если Esc, то завершение программы

}

# Структура программы

## Взаимосвязь функций

Обычным шрифтом обозначены функции класса «romb», *курсивом – «Stack»*

Void main()

## Составные части программы

Наименование функции:   
SetPos  
Прототип функции:  
virtual void SetPos(int new\_x, int new\_y)  
Данная функция перемещает каждую вершину на указанный вектор xy. В классе HollowRomb перемещает две фигуры.

Наименование функции:  
draw  
Прототип функции:  
virtual void draw()  
Данная функция рисует фигуру в зависимости от выбора пользователя.

Наименование функции:  
save  
Прототип функции:  
virtual void save(char \*fname)  
Данная функция записывает в файл данные о последней нарисованной фигуре.

Наименование функции:  
validate  
Прототип функции:  
virtual void validate()  
Данная функция проверяет фигуру на принадлежность классу фигур: «ромб» или «romb», а так же проверяет, чтобы фигура находилась в пределах консольного окна. Включает в себя функции check\_figure и check\_borders, в классе HollowRomb включает еще check\_FinF.

Наименование функции:  
erase  
Прототип функции:  
virtual void erase()  
Данная функция удаляет фигуру из консольного окна, т.е. очищает его.

Наименование функции:  
check\_FinF  
Прототип функции:  
int check\_FinF(POINT m1, POINT m2, POINT \*n)  
Данная функция проверяет нахождение стороны ромба внутри другого ромба. Таким образом, эту функцию надо вызывать 4 раза.

Наименование функции:  
check\_borders  
Прототип функции:  
int romb::check\_borders(POINT \*ppt)  
Данная функция проверяет нахождение вершин ромба внутри консольного окна. Если одна из вершин за границами окна, возвращается 1.

Наименование функции:  
check\_figure  
Прототип функции:  
void romb::check\_figure(POINT \*ppt)  
Данная функция вычисляет длины всех сторон и коэффициенты. Сравнивает длины, если они не равны, то ошибка. Сравнивает коэффициенты противоположных сторон, если они не равны, то ошибка.

Наименование функции:  
slope  
Прототип функции:  
float slope(POINT a, POINT b)  
Данная функция вычисляет угловой коэффициент прямой. Если угловые коэффициенты двух прямых равны, то они параллельны.

Наименование функции:  
SetFigure  
Прототип функции:  
void romb::SetFigure(POINT\* new\_t)  
Данная функция присваивает атрибутам класса(координаты вершин) новые значения.

Наименование функции:  
SetInnerFigure  
Прототип функции:  
SetInnerFigure(POINT \*new\_t)  
Данная функция присваивает атрибутам класса HollowRomb(координаты вершин полости ромба) новые значения.

# Текст программы

**/\* Файл romb.h \*/**

#ifndef ROMB

#define ROMB

#include <windows.h>

#include <stdio.h>

#include <iostream>

#include <fstream>

#include <stdio.h>

//коды ошибок

#define OUTSIDE\_BORDERS "Фигура за границами окна\n"

#define NEG\_VALUE "Неверное значение толщины пера\n"

#define WRONG\_PARALLEL "Противоположные стороны не параллельны\n"

#define IO\_ERR "Файл не существует или содержит неверный формат данных\n"

#define TYPE\_ERR "Неверно указан тип границы или заливки\n"

#define RGB\_ERR "Неверно задан цвет границы или заливки\n"

#define WRONG\_DLINA "Длины сторон не равны\n"

#define OVERLAP "Внутренняя фигура больше или вне основной\n"

#define EMPTY\_SLOT "Не задана фигура, с которой вы хотели бы работать. Для продолжения нарисуйте интересующую вас фигуру.\n"

struct color { int r, g, b; };

class shape

{

public:

shape() {};

virtual void validate() = 0;

virtual void draw() = 0;

virtual void erase() = 0;

virtual void SetPos(int new\_x, int new\_y) = 0;

virtual void save(char \*fname) = 0;

};

class romb : public shape

{

protected:

POINT t[5];

int type, width;

color col;

public:

romb();

romb(POINT\* new\_t);

void SetFigure(POINT \*new\_t);

inline void SetFrameStyle(int new\_type, int new\_width, color new\_color) { type = new\_type; width = new\_width; col = new\_color; };

virtual void SetPos(int new\_x, int new\_y);

inline POINT\* GetFrameCoord() { return t; };

inline int GetFrameType() { return type; };

inline int GetFrameWidth() { return width; };

inline color GetFrameColor() { return col; };

float slope(POINT a, POINT b);

void check\_figure(POINT \*ppt);

void check\_borders(POINT \*ppt);

virtual void validate();

virtual void draw();

virtual void erase();

virtual void save(char \*fname);

};

class RbFill

{

protected:

int type;

color col;

public:

RbFill() { type = -1; col.r = 0; col.g = 0; col.b = 0; };

RbFill(int new\_type, color new\_col);

void SetFillStyle(int new\_type, color new\_col);

inline int GetFillType() { return type; };

inline color GetFillColor() { return col; };

};

class SolidRomb : public romb, public RbFill

{

public:

void operator =(romb sq);

void operator =(RbFill F);

virtual void SetPos(int new\_x, int new\_y);

virtual void validate();

virtual void draw();

virtual void erase();

virtual void save(char \*fname);

};

class HollowRomb : public SolidRomb

{

protected:

POINT tin[5];

public:

HollowRomb();

int check\_FinF(POINT m1, POINT m2, POINT \*n);

void operator =(RbFill F);

void operator =(romb sq);

void operator =(SolidRomb sq);

void SetInnerFigure(POINT \*new\_t);

inline POINT\* GetInnerCoord() { return tin; };

virtual void SetPos(int new\_x, int new\_y);

virtual void validate();

virtual void draw();

virtual void erase();

virtual void save(char \*fname);

};

#endif

**/\* Файл romb.cpp \*/**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <windows.h>

#include <windowsx.h>

#include "romb.h"

#include <iostream>

#include <fstream>

#include <stdio.h>

HWND hwnd = GetConsoleWindow();

RECT rt;

//SHAPE-------------------------------------------------------------

//ROMB------------------------------------------------------------

romb::romb()

{

t[0].x = 0; t[1].x = 10; t[2].x = 10; t[3].x = 0; t[4].x = t[0].x;

t[0].y = 0; t[1].y = 0; t[2].y = 10; t[3].y = 10; t[4].y = t[0].y;

type = 0; width = 1; col.r = 0; col.g = 0; col.b = 0;

};

romb::romb(POINT \*new\_t)

{

int i;

for (i = 0; i < 4; i++) {

t[i].x = new\_t[i].x; t[i].y = new\_t[i].y;

}

t[4].x = t[0].x; t[4].y = t[0].y;

type = 0; width = 1; col.r = 128; col.g = 128; col.b = 128;

}

//функция проверки на пренадлежность классу "romb"

void romb::validate()

{

GetClientRect(hwnd, &rt);

check\_borders(t);

check\_figure(t);

if (type < 0 || type > 6) throw TYPE\_ERR;

if (width < 0 || width > 6) throw NEG\_VALUE;

if (col.r < 0 || col.r > 255) throw RGB\_ERR;

if (col.g < 0 || col.g > 255) throw RGB\_ERR;

if (col.b < 0 || col.b > 255) throw RGB\_ERR;

}

//функция вычисляющая коэффициент наклона прямой

float romb::slope(POINT a, POINT b)

{

if ((b.x - a.x) != 0)

return (b.y - a.y) / (b.x - a.x);

else return 3000;

}

//функция проверки фигуры(является ли ромбом)

void romb::check\_figure(POINT \*ppt)

{

float koef[4];

float dl[4];

int i;

for (i = 0; i < 4; i++)

{

koef[i] = slope(ppt[i], ppt[i + 1]);

dl[i] = sqrt((ppt[i].x - ppt[i + 1].x)\*(ppt[i].x - ppt[i + 1].x) + (ppt[i].y - ppt[i + 1].y)\*(ppt[i].y - ppt[i + 1].y));

}

for (i = 0; i < 3; i++)

{

if (dl[i] != dl[i + 1] || dl[i] == 0) throw WRONG\_DLINA;

}

for (i = 0; i < 2; i++)

{

if (koef[i] != koef[i + 2]) throw WRONG\_PARALLEL;

}

}

//функция проверки выхода фигуры за границы окна

void romb::check\_borders(POINT \*ppt)

{

GetClientRect(hwnd, &rt);

int i;

for (i = 0; i < 4; i++)

{

if (ppt[i].x >= rt.right || ppt[i].y >= rt.bottom || ppt[i].x<0 || ppt[i].y<0) throw OUTSIDE\_BORDERS;

}

}

void romb::SetFigure(POINT \*new\_t)

{

for (int i = 0; i < 4; i++) {

t[i].x = new\_t[i].x; t[i].y = new\_t[i].y;

}

t[4].x = t[0].x; t[4].y = t[0].y;

}

//функция перемещения фигуры

void romb::SetPos(int new\_x, int new\_y)

{

int i;

GetClientRect(hwnd, &rt);

for (i = 0; i < 4; i++)

{

if (t[i].x + new\_x < 0 || t[i].x + new\_x > rt.right || t[i].y + new\_y < 0 || t[i].y + new\_y > rt.bottom) throw OUTSIDE\_BORDERS;

}

for (i = 0; i < 4; i++)

{

t[i].x += new\_x;

t[i].y += new\_y;

}

}

//рисование ромба

void romb::draw()

{

HDC hdc = GetDC(hwnd);

COLORREF BG = RGB(0, 0, 0);

HPEN pen = CreatePen(type, width, RGB(col.r, col.g, col.b));

HBRUSH brush = GetStockBrush(NULL\_BRUSH);

SetBkColor(hdc, BG);

SelectPen(hdc, pen);

SelectBrush(hdc, brush);

Polygon(hdc, t, 4);

DeletePen(pen);

ReleaseDC(hwnd, hdc);

}

void romb::erase()

{

HDC hdc = GetDC(hwnd);

COLORREF BG = RGB(0, 0, 0);

HPEN pen = CreatePen(PS\_SOLID, width, BG);

HBRUSH brush = GetStockBrush(NULL\_BRUSH);

SelectPen(hdc, pen); SelectBrush(hdc, brush);

Polygon(hdc, t, 4);

ReleaseDC(hwnd, hdc);

}

void romb::save(char \*fname)

{

using namespace std;

ofstream fin; fin.open(fname);

fin<<t[0].x<<t[0].y<<t[1].x<<t[1].y<<t[2].x<<t[2].y<<t[3].x<<t[3].y<<type<<width<<col.r<<col.g<<col.b<<endl;

fin.close();

}

//FILL-------------------------------------------------------------

RbFill::RbFill(int new\_type, color new\_col)

{

type = new\_type; col = new\_col;

}

void RbFill::SetFillStyle(int new\_type, color new\_col)

{

type = new\_type; col = new\_col;

}

//SOLIDSQUARE-------------------------------------------------------

void SolidRomb::validate()

{

romb::validate();

if (RbFill::type < -1 || RbFill::type > 5) throw TYPE\_ERR;

if (RbFill::col.r < 0 || RbFill::col.r > 255) throw RGB\_ERR;

if (RbFill::col.g < 0 || RbFill::col.g > 255) throw RGB\_ERR;

if (RbFill::col.b < 0 || RbFill::col.b > 255) throw RGB\_ERR;

}

void SolidRomb::operator =(romb sq)

{

POINT\*k=sq.GetFrameCoord();

for (int i = 0; i < 5; i++)

{

t[i] = k[i];

}

SetFrameStyle(sq.GetFrameType(), sq.GetFrameWidth(), sq.GetFrameColor());

}

void SolidRomb::operator =(RbFill F)

{

SetFillStyle(F.GetFillType(), F.GetFillColor());

}

//функция перемещения фигуры

void SolidRomb::SetPos(int new\_x, int new\_y)

{

int i;

GetClientRect(hwnd, &rt);

for (i = 0; i < 4; i++)

{

if (t[i].x + new\_x < 0 || t[i].x + new\_x > rt.right || t[i].y + new\_y < 0 || t[i].y + new\_y > rt.bottom) throw OUTSIDE\_BORDERS;

}

for (i = 0; i < 4; i++)

{

t[i].x += new\_x;

t[i].y += new\_y;

}

}

void SolidRomb::draw()

{

HDC hdc = GetDC(hwnd);

COLORREF BG = RGB(0, 0, 0);

HPEN pen = CreatePen(romb::type, width, RGB(romb::col.r, romb::col.g, romb::col.b));

HBRUSH brush;

if (RbFill::type == -1)

brush = CreateSolidBrush(RGB(RbFill::col.r, RbFill::col.g, RbFill::col.b));

else

brush = CreateHatchBrush(RbFill::type, RGB(RbFill::col.r, RbFill::col.g, RbFill::col.b));

SetBkColor(hdc, BG);

SelectPen(hdc, pen);

SelectBrush(hdc, brush);

Polygon(hdc, t, 4);

DeletePen(pen);

ReleaseDC(hwnd, hdc);

}

void SolidRomb::erase()

{

HDC hdc = GetDC(hwnd);

COLORREF BG = RGB(0, 0, 0);

HPEN pen = CreatePen(PS\_SOLID, width, BG);

HBRUSH brush = GetStockBrush(BLACK\_BRUSH);

SelectPen(hdc, pen); SelectBrush(hdc, brush);

Polygon(hdc, t, 4);

ReleaseDC(hwnd, hdc);

}

void SolidRomb::save(char \*fname)

{

using namespace std;

ofstream fin; fin.open(fname);

int wr = GetFrameWidth();

int tr = GetFrameType();

color colr = GetFrameColor();

int tf = GetFillType();

color colf = GetFillColor();

fin << t[0].x << t[0].y << t[1].x << t[1].y << t[2].x << t[2].y << t[3].x << t[3].y << tr << wr << colr.r << colr.g << colr.b << endl;

fin << tf << colf.r << colr.g << colr.b;

fin.close();

}

//HOLLOWSQUARE-----------------------------------------------------------

HollowRomb::HollowRomb()

{

for (int i = 0; i < 4; i++)

{

tin[i] = t[i];

}

tin[4] = tin[0];

};

int HollowRomb::check\_FinF(POINT m1, POINT m2, POINT \*n)

{

int k = 0, z;

for (int i = 0; i < 4; i++)

{

z = (n[i].x - m1.x)\*(m2.y - m1.y) - (n[i].y - m1.y)\*(m2.x - m1.x);

if (z < 0) k++;

else return 0;

}

return k;

}

void HollowRomb::validate()

{

SolidRomb::validate();

for (int i = 0; i < 4; i++)

{

if (check\_FinF(tin[i], tin[i + 1], t) != 0) throw OVERLAP;

}

}

void HollowRomb::operator =(RbFill F)

{

SetFillStyle(F.GetFillType(), F.GetFillColor());

}

void HollowRomb::operator =(romb sq)

{

POINT\*k = sq.GetFrameCoord();

for (int i = 0; i < 5; i++)

{

t[i] = k[i];

}

SetFrameStyle(sq.GetFrameType(), sq.GetFrameWidth(), sq.GetFrameColor());

}

void HollowRomb::operator =(SolidRomb sq)

{

POINT\*k = sq.GetFrameCoord();

for (int i = 0; i < 5; i++)

{

t[i] = k[i];

}

SetFrameStyle(sq.GetFrameType(), sq.GetFrameWidth(), sq.GetFrameColor());

SetFillStyle(sq.GetFillType(), sq.GetFillColor());

}

void HollowRomb::SetPos(int new\_x, int new\_y)

{

int i;

GetClientRect(hwnd, &rt);

for (i = 0; i < 4; i++)

{

if (t[i].x + new\_x < 0 || t[i].x + new\_x > rt.right || t[i].y + new\_y < 0 || t[i].y + new\_y > rt.bottom) throw OUTSIDE\_BORDERS;

}

for (i = 0; i < 4; i++)

{

tin[i].x += new\_x;

tin[i].y += new\_y;

}

for (i = 0; i < 4; i++)

{

t[i].x += new\_x;

t[i].y += new\_y;

}

}

void HollowRomb::SetInnerFigure(POINT \*new\_t)

{

for (int i = 0; i < 4; i++) {

tin[i].x = new\_t[i].x; tin[i].y = new\_t[i].y;

}

tin[4].x = tin[0].x; tin[4].y = tin[0].y;

}

void HollowRomb::draw()

{

HDC hdc = GetDC(hwnd);

COLORREF BG = RGB(0, 0, 0);

HPEN pen = CreatePen(romb::type, width, RGB(romb::col.r, romb::col.g, romb::col.b));

HBRUSH brush, wbrush = GetStockBrush(BLACK\_BRUSH);

if (RbFill::type == -1)

brush = CreateSolidBrush(RGB(RbFill::col.r, RbFill::col.g, RbFill::col.b));

else

brush = CreateHatchBrush(RbFill::type, RGB(RbFill::col.r, RbFill::col.g, RbFill::col.b));

SetBkColor(hdc, BG);

SelectPen(hdc, pen); SelectBrush(hdc, brush);

Polygon(hdc, t, 4);

SelectBrush(hdc, wbrush);

Polygon(hdc, tin, 4);

DeletePen(pen); ReleaseDC(hwnd, hdc);

}

void HollowRomb::erase()

{

HDC hdc = GetDC(hwnd);

COLORREF BG = RGB(0, 0, 0);

HPEN pen = CreatePen(PS\_SOLID, width, BG);

HBRUSH brush = GetStockBrush(BLACK\_BRUSH);

SelectPen(hdc, pen); SelectBrush(hdc, brush);

Polygon(hdc, t, 4);

ReleaseDC(hwnd, hdc);

}

void HollowRomb::save(char \*fname)

{

using namespace std;

ofstream fin; fin.open(fname);

int wr = GetFrameWidth();

int tr = GetFrameType();

color colr = GetFrameColor();

int tf = GetFillType();

color colf = GetFillColor();

fin << t[0].x << t[0].y << t[1].x << t[1].y << t[2].x << t[2].y << t[3].x << t[3].y << tr << wr << colr.r << colr.g << colr.b << endl;

fin << tf << colf.r << colr.g << colr.b;

fin << tin[0].x << tin[0].y << tin[1].x << tin[1].y << tin[2].x << tin[2].y << tin[3].x << tin[3].y;

fin.close();

}

**/\* Файл colors.h \*/**

#include <windows.h>

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

void header()

{

SetConsoleTextAttribute(hStdOut,

BACKGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

}

void grayed()

{

SetConsoleTextAttribute(hStdOut,

FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_RED);

}

void result()

{

SetConsoleTextAttribute(hStdOut,

FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

}

void plain()

{

SetConsoleTextAttribute(hStdOut,

FOREGROUND\_GREEN | FOREGROUND\_RED | FOREGROUND\_BLUE |

FOREGROUND\_INTENSITY);

}

void items()

{

SetConsoleTextAttribute(hStdOut, FOREGROUND\_RED | FOREGROUND\_GREEN |

FOREGROUND\_INTENSITY);

}

void red()

{

SetConsoleTextAttribute(hStdOut, FOREGROUND\_RED |

FOREGROUND\_INTENSITY);

}

**/\* Файл main.cpp \*/**

#include <conio.h>

#include <locale.h>

#include <iostream>

#include <fstream>

#include "user.h"

void main()

{

using namespace std;

setlocale(LC\_CTYPE, "russian");

bool exit = false, showMenu=true;

char input; unsigned \_\_int8 MenuFlags = 0;

shape \*S = NULL;

romb \*sq = NULL;

SolidRomb \*ssq = NULL;

HollowRomb \*hsq = NULL;

while (!exit)

{

if (showMenu) { PrintMenu(MenuFlags); showMenu = false; }

input = \_getch();

switch (input)

{

case '1': //Задание новой фигуры

{

if (S) { S->erase(); S = NULL; } //if (F) delete F;

if (hsq) { delete hsq; hsq = NULL; }

if (ssq) { delete ssq; ssq = NULL; }

if (sq) { delete sq; sq = NULL; }

MenuFlags = 0;

cout << "Введите координаты вершин ромба: ";

POINT s[4];

cin >> s[0].x >> s[0].y >> s[1].x >> s[1].y >> s[2].x >> s[2].y >> s[3].x >> s[3].y;

sq = new romb(s);

cout << "Задайте стиль границы (тип, толщина, цвет RGB): ";

int t, w; color c;

cin >> t >> w >> c.r >> c.g >> c.b;

sq->SetFrameStyle(t, w, c);

try

{

sq->validate();

MenuFlags |= 0x03;

cout << "Задать заливку? [Y/N] ";

input = \_getch(); cout << endl;

if (input == 'y')

{

cout << "Задайте стиль заливки (тип, цвет RGB): ";

cin >> t >> c.r >> c.g >> c.b;

ssq = new SolidRomb; \*ssq = \*sq;

ssq->SetFillStyle(t, c);

try

{

ssq->validate();

MenuFlags |= 0x04;

cout << "Задать полую фигуру? [Y/N] ";

input = \_getch(); cout << endl;

if (input == 'y')

{

cout << "Задайте координаты вершин полости: ";

cin >> s[0].x >> s[0].y >> s[1].x >> s[1].y >> s[2].x >> s[2].y >> s[3].x >> s[3].y;

hsq = new HollowRomb; \*hsq = \*ssq;

hsq->SetInnerFigure(s);

try

{

hsq->validate();

MenuFlags |= 0x08;

}

catch (char \*err) { red(); cout << err << endl; plain(); }

}

}

catch (char \*err) { red(); cout << err << endl; plain(); }

}

}

catch (char \*err) { red(); cout << err << endl; plain(); }

} break;

case '2': //Загрузка новой фигуры

{

if (S) { S->erase(); S = NULL; }

if (hsq) { delete hsq; hsq = NULL; }

if (ssq) { delete ssq; ssq = NULL; }

if (sq) { delete sq; sq = NULL; }

MenuFlags = 0;

char fname[30];

cout << "Введите имя файла: "; cin >> fname;

ifstream fin; fin.open(fname);

POINT s[4];

int t, w; color c;

fin >> s[0].x >> s[0].y >> s[1].x >> s[1].y >> s[2].x >> s[2].y >> s[3].x >> s[3].y >> t >> w >> c.r >> c.g >> c.b;

sq = new romb(s);

sq->SetFrameStyle(t, w, c);

try

{

sq->validate();

MenuFlags |= 0x03;

if (!fin.eof())

{

fin >> t >> c.r >> c.g >> c.b;

ssq = new SolidRomb; \*ssq = \*sq;

ssq->SetFillStyle(t, c);

try

{

ssq->validate();

MenuFlags |= 0x04;

if (!fin.eof())

{

fin >> s[0].x >> s[0].y >> s[1].x >> s[1].y >> s[2].x >> s[2].y >> s[3].x >> s[3].y;

hsq = new HollowRomb; \*hsq = \*ssq;

hsq->SetInnerFigure(s);

try

{

hsq->validate();

MenuFlags |= 0x08;

}

catch (char \*err) { red(); cout << err << endl; plain(); }

}

}

catch (char \*err) { red(); cout << err << endl; plain(); }

}

}

catch (char \*err) { red(); cout << err << endl; plain(); }

fin.close();

} break;

case '3': //Изменить фигуру

{

if (!(MenuFlags & 0x01))

{

red(); cout << "Сперва задайте или загрузите фигуру" << endl;

plain();

}

else

{

EditMenu(MenuFlags);

input = \_getch();

switch (input)

{

case '1':

{

romb oldsq = \*sq;

cout << "Введите координаты фигуры: ";

POINT s[4];

cin >> s[0].x >> s[0].y >> s[1].x >> s[1].y >> s[2].x >> s[2].y >> s[3].x >> s[3].y;

sq->SetFigure(s);

cout << "Задайте стиль границы (тип, толщина, цвет RGB): ";

int t, w; color c;

cin >> t >> w >> c.r >> c.g >> c.b;

sq->SetFrameStyle(t, w, c);

try

{

sq->validate();

if (ssq) \*ssq = \*sq;

if (hsq) \*hsq = \*ssq;

}

catch (char \*err) { red(); cout << err << endl; plain(); \*sq = oldsq; }

} break;

case '2':

{

int t; color c;

cout << "Задайте стиль заливки (тип, цвет RGB): ";

cin >> t >> c.r >> c.g >> c.b;

if (!ssq) { ssq = new SolidRomb; \*ssq = \*sq; };

ssq->SetFillStyle(t, c);

try

{

ssq->validate();

if (!(MenuFlags & 0x04)) MenuFlags |= 0x04;

if (hsq) \*hsq = \*ssq;

}

catch (char \*err)

{

red(); cout << err << endl; plain();

delete ssq; ssq = NULL; MenuFlags &= 0xFB;

if (hsq) { delete hsq; hsq = NULL; MenuFlags &= 0xF7; }

}

} break;

case '3':

{

if (!(MenuFlags & 0x04))

{

red(); cout << "Сперва задайте заливку" << endl;

plain();

}

else

{

POINT s[4];

cout << "Задайте координаты полости: ";

cin >> s[0].x >> s[0].y >> s[1].x >> s[1].y >> s[2].x >> s[2].y >> s[3].x >> s[3].y;

if (!hsq) { hsq = new HollowRomb; \*hsq = \*ssq; }

hsq->SetInnerFigure(s);

try

{

hsq->validate();

if (!(MenuFlags & 0x08)) MenuFlags |= 0x08;

}

catch (char \*err) { red(); cout << err << endl; plain(); delete hsq; hsq = NULL; MenuFlags &= 0xF7; }

}

} break;

case '4': //перемещение фигуры

{

try{

if (S == NULL) throw EMPTY\_SLOT;

int new\_x, new\_y;

cout << "Введите координаты перемещения(по Ох и по Оу):\n";

cin >> new\_x >> new\_y;

S->erase();

S->SetPos(new\_x, new\_y);

S->draw();

}

catch (char \*err) { red(); cout << err << endl; plain(); }

} break;

}

}

} break;

case '4': //Нарисовать контур

{

if (!(MenuFlags & 0x02))

{

red(); cout << "Сперва задайте или загрузите фигуру" << endl;

plain();

}

else

{

if (!S) { S = sq; S->draw(); }

else { S->erase(); S = sq; S->draw(); }

}

} break;

case '5': //Нарисовать сплошную фигуру

{

if (!(MenuFlags & 0x04))

{

red(); cout << "Сперва задайте заливку" << endl;

plain();

}

else

{

if (!S) { S = ssq; S->draw(); }

else { S->erase(); S = ssq; S->draw(); }

}

} break;

case '6': //Нарисовать полую фигуру

{

if (!(MenuFlags & 0x08))

{

red(); cout << "Сперва задайте полость фигуры" << endl;

plain();

}

else

{

if (!S) { S = hsq; S->draw(); }

else { S->erase(); S = hsq; S->draw(); }

}

} break;

case '7': //Сохранение фигуры

{

try

{

if (S==NULL) throw EMPTY\_SLOT;

char fname[30];

cout << "Введите имя файла: "; cin >> fname;

S->save(fname);

}

catch (char \*err) { red(); cout << err; plain(); }

} break;

case 'm': { showMenu = true; } break;

case 27: { exit = true; } break; //Выход

}

}

}

# 7.Тесты

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Результат | Примечания |
| 1 | 300 100 400 100 400 200 300 200 3 3 183 110 60  1 255 255 0  310 150 350 190 390 150 350 110 |  | Рисование всех трех типов фигур. Загрузка из файла |
| 2 | 310 150 350 190 390 150 350 110 3 3 183 110 60 3 125 125 125 |  | Рисование ромба Непосредственное задание. |
| 4 | 310 150 350 190 390 150 350 100 3 3 183 110 60 3 125 125 125 | Длины сторон не равны | Длины сторон фигуры не равны, значит фигура не является ромбом. |
| 4 | 3 4 5 6 7 | Не задана фигура, с которой вы хотели бы работать. Для продолжения нарисуйте интересующую вас фигуру | При отсутствии задания фигуры, реакция на команды. |
| 5 | -400 0 |  | Перемещение предыдущей фигуры влево на 400 пиксилей  Фигура из первого теста. |
| 6 | 300 100 400 100 400 200 300 200  -5 1 30 30 30 |  | Задание фигуры не являющейся ромбом |

# 8. Результат работы программы

-Программа выдала верное решение на всех тестах и, следовательно, правильно работает.