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

**Лабораторная работа № 5**

**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 № 3, 4, разработать шаблон контейнера для хранения объектов классов, реализующих геометрические фигуры.   
2.1. Преобразовать класс-контейнер, разработанный в лабораторной работе № 3, в шаблон, так чтобы элементами контейнеров могли быть различные классы, разработанные в лабораторной работе № 4 (при различном инстанцировании шаблона).   
2.2. Разработать функцию, демонстрирующую поведение разработанного шаблона: провести инстанцирование шаблона для каждого из классов-фигур, продемонстрировать их функционирование.

# 2. Анализ задачи

*Дано:*

Ромб – четырехугольник, противоположные стороны которого параллельны, и все стороны равны.  
Слово, указывающее какой контейнер создавать, возможны три варианта: hollow, romb, solid  
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>)  
Если это файл hollow-фигуры, то второй строкой пишется координаты внутренней фигуры.  
Пример задания фигуры в файле: 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

Пример выше описывает задание hollow ромба, в файле задания solid ромба, нету координат внутренней фигуры, а в файле romb нету графических характеристик заливки.

В файле могут находится множество фигур. Каждой строке соответствует своя фигура, кроме hollow-romb файла, в котором каждой фигуре соответствуют 2 строки.  
При загрузке/сохранении в файл – последовательность символов(название файла).

*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. Конструктор, деструктор
8. Загрузка данных из файла
9. Сохранение данных в файл
10. Добавление ромба в стек
11. Удаление ромба из стека
12. Поиск ромба с указанными характеристиками

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*)

1. Конструктор – создаётся пустой динамический стек. Деструктор – удаляется каждый элемент стека.

2. Данные считываются пока не конец файла и заносятся в стек. (Многократное повторение пункта 5) )

3. Данные записываются в файл, указанный пользователем (формат представления данных можно посмотреть в *Дано*).

4. Данные считываются непосредственно. После классовой проверки (п. 2) п. 3) ), данные о фигуре вносятся в начало стека.

5. Данные о ромбе в вершине стека (в начале) запоминаются, но сам элемент(ромб) удаляется из стека. Вершиной становится следующий элемент.

6. Данные считываются непосредственно. После классовой проверки (п.2) п.3) ) ищется элемент с такими-же характеристиками методом перебора.

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

## Внешнее представление входных данных

Последовательность целых чисел и символов. Характеристики для поиска – не полные, т.е. для правильной работы функции необходимо ввести только координаты.

## Внешнее представление выходных данных

Надпись или графическое изображение. А также последовательность целых чисел и символов в файле.

# Внутреннее представление входных данных

struct RbStack { Type sq; RbStack \*next; } \*top;

struct RbStack – структура, «открытый стек» содержащая поля:  
 Type sq – элемент шаблона типа фигуры, который устанавливается в соответствии с данными, считанными из файла  
 RbStack \*next – указатель на следующий элемент  
 top – указатель на начало списка(стека).

*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.

RbStack \*top – указатель на элемент стека. Используется для перемещения по стеку.

RbStack \*puch – указатель на новый элемент стека, используется для вставки элемента в начало.

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’: Считывание данных из указанного файла, и их занос в стек.  
Если ‘2’: Сохранение данных о фигурах в стеке, в файл.  
Если ‘3’: Ввод характеристик ромба и последующее добавление в стек.  
Если ‘4’: Удаление первого элемента(ромба) из стека.  
Если ‘5’: Рисование ромба, который находится в памяти(последний удалённый ромб)  
Если ‘6’: Ввод характеристик ромба, проверка на принадлежность классу «ромб», затем поиск соответствующего элемента в контейнере, при нахождении, спрашивается, выводить фигуру или нет.  
Если ‘7’: Рисование графического изображения каждого элемента контейнера(ромбов)  
Если ‘m’: Ввод характеристик ромба()  
Если Esc, то завершение программы

}

# Структура программы

## Взаимосвязь функций

Обычным шрифтом обозначены функции класса «Type», *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.

Наименование функции:  
GetFrameCoord  
Прототип функции:  
inline POINT\* GetFrameCoord()  
Данная функция возвращает координаты вершин ромба

Наименование функции:  
GetFrameType  
Прототип функции:  
inline int GetFrameType()  
Данная функция возвращает тип пера.

Наименование функции:  
GetFrameWidth  
Прототип функции:  
inline int GetFrameWidth()  
Данная функция возвращает толщину пера.

Наименование функции:  
GetFrameColor  
Прототип функции:  
inline color GetFrameColor()  
Данная функция возвращает цвет пера

Наименование функции:  
SetFrameStyle  
Прототип функции:  
inline void SetFrameStyle(int new\_type, int new\_width, color new\_color)  
Данная функция устанавливает новые параметры-атрибуты пера в классе.

Наименование функции:  
GetFillType  
Прототип функции:  
inline int GetFillType() { return type; };  
Данная функция устанавливает возвращает тип кисти(заливки).

Наименование функции:  
GetFillColor  
Прототип функции:  
inline color GetFillColor() { return col; };  
Данная функция устанавливает возвращает цвет кисти(заливки)

Наименование функции:  
SetFillStyle  
Прототип функции:  
void SetFillStyle(int new\_type, color new\_col);  
Данная функция устанавливает новые параметры-атрибуты кисти в классе.

Наименование функции:  
compare  
Прототип функции:  
bool Stack::compare(Type rb, Type rb2)  
Данная функция сравнивает два элемента класса «Type». Для этого сравниваются графические характеристики двух элементов и вызывается функция для сравнения пространственных характеристик (смотреть ниже). При сравнении двух элементов класса «HollowRomb» функция переопределяется, чтобы сравнить полости двух фигур.

Наименование функции:  
comparePOINTs  
Прототип функции:  
bool Stack::comparePOINTs(POINT k1[4], POINT k2[4])  
Данная функция сравнивает два массива типа POINT, в которых лежат координаты двух фигур.

Наименование функции:  
Деструктор  
Прототип функции:  
Stack::~Stack()  
Данная функция удаляет все элементы из стека.

Наименование функции:  
Stack  
Прототип функции:  
Stack::Stack(FILE \*fptr)  
Данная функция считывает данные из файла, применяя функцию загрузки из класса-шаблона, таким образом данная функция работает одинаково для всех классов, но содержит виртуальную функцию загрузки одной фигуры из класса фигуры. После проверки на принадлежность классу «romb» (п. 2) п. 3) ) одной фигуры, заносит её в стек. Если считывается hollow romb, то проверяется ромб, а затем полость, после этого данные заносятся в стек. Выход за границы окна у полости не проверяется. Для

Наименование функции:  
Push  
Прототип функции:  
void Stack::push(Type rb)  
Данная функция создаёт новый элемент типа RbStack, делает его вершиной стека и вставляет элемент класса в соответствующее поле. Для каждого класса эта функция работает одинаково.

Наименование функции:  
pop  
Прототип функции:  
Type Stack::pop()  
Данная функция возвращает элемент класса вершину стека и удаляет элемент из стека. Вершиной становится следующий элемент. Данная функция работает для каждого класса одинаково.

Наименование функции:  
search  
Прототип функции:  
bool Stack::search(Type rb)  
Данная функция ищет элемент равный данному методом перебора, если его нету или он есть, возвращает false и true соответственно.

Наименование функции:  
DrawSt  
Прототип функции:  
void Stack::DrawSt()  
Данная функция выводит в консольное окно графическое изображение каждой фигуры в стеке, по порядку начиная с вершины. Фигуры могут накладываться друг на друга. Для каждого класса фигуры внутри функции вызывается виртуальная функция рисования фигуры, поэтому для каждого класса DrawSt работает одинаково.

Наименование функции:  
save  
Прототип функции:  
void Stack::save(char \*fname)  
Данная функция вызывает виртуальную функцию save из класса фигуры для каждого элемента, т.е. выводит в файл характеристики каждого элемента стека.

# 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(FILE \*fptr)

{

fprintf(fptr, "%d %d %d %d %d %d %d %d; %d %d (%d, %d, %d)\n", 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);

}

void romb::load(FILE \*fptr)

{

GetClientRect(hwnd, &rt);

POINT tk[5];

int type, twidth;

color tcol;

if (!fptr) throw IO\_ERR;

if (!fscanf(fptr, "%d %d %d %d %d %d %d %d; ", &tk[0].x, &tk[0].y, &tk[1].x, &tk[1].y, &tk[2].x, &tk[2].y, &tk[3].x, &tk[3].y)) throw IO\_ERR;

tk[4] = tk[0];

if (!fscanf(fptr, "%d %d (%d, %d, %d)\n", &type, &twidth, &tcol.r, &tcol.g, &tcol.b)) throw IO\_ERR;

validate(tk, type, twidth, tcol);

SetFigure(tk);

SetFrameStyle(type, twidth, tcol);

}

//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(FILE \*fptr)

{

fprintf(fptr, "%d %d %d %d %d %d %d %d; %d %d (%d, %d, %d) %d (%d, %d, %d) \n", t[0].x, t[0].y, t[1].x, t[1].y, t[2].x, t[2].y, t[3].x, t[3].y, romb::type, width, romb::col.r, romb::col.g, romb::col.b, RbFill::type, RbFill::col.r, RbFill::col.g, RbFill::col.b);

}

void SolidRomb::load(FILE \*fptr)

{

GetClientRect(hwnd, &rt);

POINT tk[5];

if (!fptr) throw IO\_ERR;

if (!fscanf(fptr, "%d %d %d %d %d %d %d %d; ", &tk[0].x, &tk[0].y, &tk[1].x, &tk[1].y, &tk[2].x, &tk[2].y, &tk[3].x, &tk[3].y)) throw IO\_ERR;

tk[4] = tk[0];

int type, ftype, width;

color col, fcol;

if (!fscanf(fptr, "%d %d (%d, %d, %d) ", &type, &width, &col.r, &col.g, &col.b)) throw IO\_ERR;

if (!fscanf(fptr, "%d (%d, %d, %d)\n", &ftype, &fcol.r, &fcol.g, &fcol.b)) throw IO\_ERR;

validate(tk, type, width, col, ftype, fcol);

SetFigure(tk);

SetFrameStyle(type, width, col);

SetFillStyle(ftype, fcol);

}

//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(FILE \*fptr)

{

fprintf(fptr, "%d %d %d %d %d %d %d %d; %d %d (%d, %d, %d) %d (%d, %d, %d) \n", t[0].x, t[0].y, t[1].x, t[1].y, t[2].x, t[2].y, t[3].x, t[3].y, romb::type, width, romb::col.r, romb::col.g, romb::col.b, RbFill::type, RbFill::col.r, RbFill::col.g, RbFill::col.b);

fprintf(fptr, "%d %d %d %d %d %d %d %d\n", tin[0].x, tin[0].y, tin[1].x, tin[1].y, tin[2].x, t[2].y, t[3].x, t[3].y);

}

void HollowRomb::load(FILE \*fptr)

{

GetClientRect(hwnd, &rt);

POINT tk[5], tki[5];

if (!fptr) throw IO\_ERR;

if (!fscanf(fptr, "%d %d %d %d %d %d %d %d; ", &tk[0].x, &tk[0].y, &tk[1].x, &tk[1].y, &tk[2].x, &tk[2].y, &tk[3].x, &tk[3].y)) throw IO\_ERR;

tk[4] = tk[0];

int type,ftype, width;

color col, fcol;

if (!fscanf(fptr, "%d %d (%d, %d, %d) ", &type, &width, &col.r, &col.g, &col.b)) throw IO\_ERR;

if (!fscanf(fptr, "%d (%d, %d, %d)", &ftype, &fcol.r, &fcol.g, &fcol.b)) throw IO\_ERR;

if (!fscanf(fptr, "%d %d %d %d %d %d %d %d\n", &tki[0].x, &tki[0].y, &tki[1].x, &tki[1].y, &tki[2].x, &tki[2].y, &tki[3].x, &tki[3].y)) throw IO\_ERR;

tki[4] = tk[0];

validate(tk, type, width, col, ftype, fcol, tki);

SetFigure(tk);

SetFrameStyle(type, width, col);

SetFillStyle(ftype, fcol);

SetInnerFigure(tki);

}

**/\* 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 user.h \*/**

#include <stdio.h>

#include "romb.h"

#include "rombStack.h"

#include "colors.h"

#include <fstream>

#include <iostream>

void printMenu(bool extended)

{

header();

printf\_s("Выберите действие:\n");

items();

printf\_s("[1] - загрузить данные из файла\n");

printf\_s("[2] - ручной ввод ромба и его последующее добавление в стек\n");

if (!extended) grayed();

printf\_s("[3] - сохранить данные в файл\n");

printf\_s("[4] - удалить ромб из стека\n");

printf\_s("[5] - отобразить ромб из памяти\n");

printf\_s("[6] - поиск фигуры\n");

printf\_s("[7] - отобразить стек\n");

items();

printf\_s("[m] - показать меню\n");

printf\_s("[esc] - выход из программы\n");

plain();

}

**/\* Файл rombStack.h \*/**

#include "romb.h"

#include <iostream>

#ifndef ROMB\_CONTAINER

#define ROMB\_CONTAINER

#define EMPTY\_STACK "Стек пуст \n"

//Открытый стек

template <class Type> class Stack

{

private:

struct RbStack { Type sq; RbStack \*next; } \*top;

bool compare(Type rb, Type rb2);

bool comparePOINTs(POINT k1[4], POINT k2[4]);

public:

Stack() { top = NULL; };

~Stack()

{

if (top != NULL)

{

RbStack \*St;

while (top != NULL)

{

St = top;

top = top->next;

delete St;

St = top;

}

}

};

Stack(FILE \*fptr)

{

if (!fptr) throw IO\_ERR;

Type squ;

squ.load(fptr);

top = new RbStack;

top->sq = squ;

top->next = NULL;

while (!feof(fptr))

{

squ.load(fptr);

push(squ);

}

fclose(fptr);

};

void push(Type rb)

{

if (top == NULL)

{

top = new RbStack;

top->sq = rb;

top->next = NULL;

}

else

{

RbStack \*puch = new RbStack;

puch->next = top;

top = puch;

puch->sq = rb;

}

};

Type pop()

{

if (top == NULL) throw EMPTY\_STACK;

else if (top->next == NULL)

{

Type gt = top->sq;

top = NULL;

return gt;

}

else

{

Type gt = top->sq;

top = top->next;

return gt;

}

};

bool search(Type rb)

{

RbStack \*St = top;

if (top == NULL) throw EMPTY\_STACK;

while (St != NULL) {

if (compare(rb, St->sq)) return true;

St = St->next;

}

return false;

};

void DrawSt()

{

if (top == NULL) throw EMPTY\_STACK;

else {

RbStack \*St = top;

while (St != NULL)

{

St->sq.draw();

St = St->next;

}

}

};

void save(char \*fname)

{

RbStack \*St = top;

Type rb;

FILE \*fptr = fopen(fname, "r");

if (!fptr) throw IO\_ERR;

if (top == NULL) throw EMPTY\_STACK;

while (St != NULL) {

St->sq.save(fptr);

St = St->next;

}

fclose(fptr);

};

};

#endif

**/\* Файл rombStack.cpp \*/**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "rombStack.h"

#include <stdio.h>

#include <fstream>

template <> bool Stack <romb>::compare(romb rb, romb rb2)

{

POINT\*k1 = rb.GetFrameCoord();

POINT\*k2 = rb2.GetFrameCoord();

return (comparePOINTs(k1, k2));

}

template <> bool Stack <SolidRomb>::compare(SolidRomb rb, SolidRomb rb2)

{

POINT\*k1 = rb.GetFrameCoord();

POINT\*k2 = rb2.GetFrameCoord();

return (comparePOINTs(k1, k2));

}

template <> bool Stack <HollowRomb>::compare(HollowRomb rb, HollowRomb rb2)

{

POINT\*k1 = rb.GetFrameCoord();

POINT\*ki1 = rb.GetInnerCoord();

POINT\*k2 = rb2.GetFrameCoord();

POINT\*ki2 = rb2.GetInnerCoord();

return (comparePOINTs(k1, k2) && comparePOINTs(ki1, ki2));

}

template <class Type> bool Stack <Type>::comparePOINTs(POINT k1[4], POINT k2[4])

{

if (k1[0].x == k2[0].x && k1[0].y == k2[0].y &&k1[1].x == k2[1].x && k1[1].y == k2[1].y && k1[2].x == k2[2].x && k1[2].y == k2[2].y && k1[3].x == k2[3].x && k1[3].y == k2[3].y) return true;

else return false;

}

**/\* Файл main.cpp \*/**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <windows.h>

#include <windowsx.h>

#include <stdio.h>

#include <locale.h>

#include <conio.h>

#include <iostream>

#include <string.h>

#include "user.h"

/////

void main()

{

setlocale(LC\_CTYPE, "Russian");

bool exit = false, showMenu = true, ContExists = false;

char fin[30], fout[30], input;

POINT tk[5], tki[5];

int tptype, tbtype, twidth;

color tpcol, tbcol;

bool est = false, canf = true, canh = true;

Stack <romb> \*st=NULL;

Stack <SolidRomb> \*sst = NULL;

Stack <HollowRomb> \*hst = NULL;

romb rb;

SolidRomb srb;

HollowRomb hrb;

RECT rt;

HWND hwnd = GetConsoleWindow();

while (!exit)

{

if (showMenu) { printMenu(ContExists); showMenu = false; }

input = \_getch();

switch (input)

{

case '1': //загрузка

{

ContExists = false;

GetClientRect(hwnd, &rt);

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

FILE \*fptr = fopen(fin, "r");

if (!fptr)

{

red(); cout << "Не удалось открыть файл"; plain();

}

else

{

if (st) { st->~Stack(); st = NULL; };

if (sst) { sst->~Stack(); sst = NULL; };

if (hst) { hst->~Stack(); hst = NULL; };

char type[20];

fscanf(fptr, "%s", type);

if (strcmp(type, "romb") == 0)

{

try { st = new Stack <romb>(fptr); ContExists = true; canf = false; canh = false; }

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

}

if (strcmp(type, "solid") == 0)

{

try { sst = new Stack <SolidRomb>(fptr); ContExists = true; canh = false; canf = true; }

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

}

if (strcmp(type, "hollow") == 0)

{

try { hst = new Stack <HollowRomb>(fptr); ContExists = true; canh = true; }

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

}

}

} break;

case '2': //ручной ввод ромба и его последующее добавление в стек

{

header(); printf\_s("Задайте параметры фигуры: координаты точек:\n");

scanf\_s("%d %d %d %d %d %d %d %d", &tk[0].x, &tk[0].y, &tk[1].x, &tk[1].y, &tk[2].x, &tk[2].y, &tk[3].x, &tk[3].y);

printf\_s("Задайте стиль границы (тип[0-6], толщина, цвет в формате RGB):\n");

tk[4] = tk[0];

scanf\_s("%d %d %d %d %d", &tptype, &twidth, &tpcol.r, &tpcol.g, &tpcol.b);

ContExists = true;

if (canf) {

printf\_s("Будет ли эта фигура закрашенной?[Y]/[N]");

input = \_getch();

switch (input)

{

case 'y':

{

printf\_s("Задайте стиль заливки (тип[-1 - сплошная][-1 - 5], цвет RGB):\n");

scanf\_s("%d %d %d %d", &tbtype, &tbcol.r, &tbcol.g, &tbcol.b);

if (canh) {

printf\_s("Будет ли эта фигура полой?[Y]/[N]");

switch (input)

{

case 'y':

{

printf\_s("Задайте координаты точек внутренней полости:\n");

scanf\_s("%d %d %d %d %d %d %d %d", &tki[0].x, &tki[0].y, &tki[1].x, &tki[1].y, &tki[2].x, &tki[2].y, &tki[3].x, &tki[3].y);

tki[4] = tki[0];

}

default:

{

canh = false;

}

}

}

}

default:

{

canf = false; canh = false;

}

}

}

try {

if (canh == true) {

hrb.validate(tk, tptype, twidth, tpcol, tbtype, tbcol, tki);

hrb.SetFigure(tk);

hrb.SetFrameStyle(tptype, twidth, tpcol);

hrb.SetFillStyle(tbtype, tbcol);

hrb.SetInnerFigure(tki);

if (!hst) hst = new Stack<HollowRomb>();

hst->push(hrb);

ContExists = true;

canf = false;

}

else if (canf == true) {

srb.validate(tk, tptype, twidth, tpcol, tbtype, tbcol);

srb.SetFigure(tk);

srb.SetFrameStyle(tptype, twidth, tpcol);

srb.SetFillStyle(tbtype, tbcol);

if (!sst) sst = new Stack<SolidRomb>();

sst->push(srb);

ContExists = true;

canh = false;

}

else {

rb.validate(tk, tptype, twidth, tpcol);

rb.SetFigure(tk);

rb.SetFrameStyle(tptype, twidth, tpcol);

if (!st) st = new Stack<romb>();

st->push(rb);

ContExists = true; canh = false; canf = false;

}

}

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

} break;

case '3': //сохранение

{

try

{

if (canh) {

if (!hst) throw EMPTY\_STACK;

}

else if (canf) {

if (!sst) throw EMPTY\_STACK;

}

else if (!st) throw EMPTY\_STACK;

printf\_s("Введите имя файла: ");

scanf("%s", &fout);

if (canh) {

hst->save(fout);

}

else if (canf) {

sst->save(fout);

}

else { st->save(fout); }

}

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

} break;

case '4': //удаление ромба из стека

{

try

{

if (canh) {

if (!hst) throw EMPTY\_STACK;

hrb=hst->pop(); est = true;

}

else if (canf) {

if (!sst) throw EMPTY\_STACK;

srb=sst->pop(); est = true;

}

else {

if (!st) throw EMPTY\_STACK;

rb=st->pop(); est = true;

}

}

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

} break;

case '5': //отобразить ромб из памяти

{

if (est == false) printf("В памяти ничего не лежит \n");

else {

try {

if (canh) {

hrb.draw();

}

else if (canf) {

srb.draw();

}

else { rb.draw(); }

}

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

}

} break;

case '6': //поиск фигуры

{

try

{

if (canh) {

if (!hst) throw EMPTY\_STACK;

}

else if (canf) {

if (!sst) throw EMPTY\_STACK;

}

else if (!st) throw EMPTY\_STACK;

printf\_s("Задайте параметры фигуры: координаты точек:\n");

scanf\_s("%d %d %d %d %d %d %d %d;", &tk[0].x, &tk[0].y, &tk[1].x, &tk[1].y, &tk[2].x, &tk[2].y, &tk[3].x, &tk[3].y);

tk[4] = tk[0];

if (canh) {

printf\_s("Задайте координаты вершин полости:\n");

scanf\_s("%d %d %d %d %d %d %d %d;", &tki[0].x, &tki[0].y, &tki[1].x, &tki[1].y, &tki[2].x, &tki[2].y, &tki[3].x, &tki[3].y);

tki[4] = tki[0];

hrb.SetFigure(tk);

hrb.SetInnerFigure(tki);

if (hst->search(hrb)) {

printf("Такая фигура есть в стеке \n");

}

else printf("Такой фигуры в стеке нету\n");

}

else if (canf)

{

srb.SetFigure(tk);

if (sst->search(srb))

{

printf("Такая фигура есть в стеке \n");

}

else printf("Такой фигуры в стеке нету\n");

}

else {

rb.SetFigure(tk);

if (st->search(rb))

{

printf("Такая фигура есть в стеке \n");

}

else printf("Такой фигуры в стеке нету\n");

}

}

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

break; }

case '7': //отобразить стек

{

try

{

if (canh)

{

if (!hst) throw EMPTY\_STACK;

hst->DrawSt();

}

else if (canf)

{

if (!sst) throw EMPTY\_STACK;

sst->DrawSt();

}

else

{

if (!st) throw EMPTY\_STACK;

st->DrawSt();

}

}

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

}

} break;

case 'm': { showMenu = true; } break; //показать меню

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

}

}}

# 7.Тесты

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Результат | Примечания |
| 1 | romb  200 100 250 100 250 150 200 150; 3 3 (183, 110, 60)  250 150 300 100 350 150 300 200; 0 5 (200, 200, 0)  200 150 250 100 300 150 250 200; 2 4 (255, 0, 0) |  | Загрузка стека из файла.  Рисование стека класса romb. |
| 2 | solid  200 100 250 100 250 150 200 150; 3 3 (183, 110, 60) 3 (255, 125, 0)  250 150 300 100 350 150 300 200; 0 5 (200, 200, 0) -1 (100, 100, 200)  200 150 250 100 300 150 250 200; 2 4 (255, 0, 0) 2 (255, 0, 123) |  | 4 раза удаление фигуры из вершины стека, затем загрузка фигуры из другого файла(solid). |
| 4 | hollow  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  350 100 450 100 450 200 350 200; 4 4 (76, 93, 14) 4 (183, 183, 8)  360 150 400 190 440 150 400 110 |  | Загрузка файла с классом hollow |
| 4 |  |  | Удаление вершины, и рисование фигуры из памяти |
| 5 |  |  | Проверка функций на работу с пустым стеком. Последовательное нажатие всех кнопок. |
| 6 |  |  | Ручное задание фигуры. Последнее действие (n) При следующей попытке ввода последней строки не будет, так как стек теперь содержит элементы только контуров(класс romb). |
|  | 320 100 400 100 400 200 300 200 |  | Попытка загрузки фигуры, из файла которая не является ромбом. |

# 8. Результат работы программы

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