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

**Лабораторная работа № 3**

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

Факультет: ФПМИ

Группа: ПМ-53

Студент: Тябин Егор Алексеевич

Преподаватель: Еланцева Ирина Леонидовна

### Новосибирск 2016

# Условие задачи

На основе списка разработать класс, реализующий понятие: 5а) «открытого» стека. По предложенному преподавателем варианту разработать программу на языке С++, в которой был бы определен класс-контейнер, т. е. класс объектов, служащих для хранения объектов класса, разработанного в предыдущей лабораторной работе. Контейнер должен быть реализован как динамическая структура данных. Разработать следующие функции-члены класса: конструктор, деструктор, функции для помещения объектов-фигур в контейнер, их возвращения (удаления), поиска в контейнере, «распечатки содержимого» контейнера – вывода информации о содержащихся в объекте-контейнере объектах и/или их графических образов, а также функции, обеспечивающие сохраняемость контейнера с использованием файла.

# 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 могут находится множество фигур. Каждой строке соответствует своя фигура.  
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 – полные, т.е. для правильной работы функции необходимо ввести координаты и графические характеристики.

## Внешнее представление выходных данных

Надпись или графическое изображение  
При «распечатке содержимого» контейнера(стека), выводятся графические изображения фигур в том порядке, в котором они расположены в контейнере. (Ромбы могут накладываться друг на друга)

# Внутреннее представление входных данных

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

POINT t[4] (tk[4])– массив типа POINT, в котором хранятся координаты 4 вершин ромба.

color pcolor, bcolor – элементы типа color, которые являются параметрическими характеристиками цветов пера и кисти соответственно; формат RGB

int ptype, btype – целочисленный элемент, в котором записывается тип пера и кисти

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

*Class Stack:*класс “Stack”, реализующий понятие класса-контейнера, т. е. класс объектов, служащих для хранения объектов класса “romb”.

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

struct RbStack – структура, «открытый стек» содержащая поля:  
 romb sq – элемент класса «romb»  
 RbStack \*next – указатель на следующий элемент  
 top – указатель на начало списка(стека).

## 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

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

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

# 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;  
}

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

}

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 «romb», *курсивом – «Stack»*

Void main()

## Составные части программы

Наименование функции:   
SetPos  
Прототип функции:  
void romb::SetPos(int new\_x, int new\_y)  
Данная функция перемещает каждую вершину на указанный вектор xy.

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

Наименование функции:  
GetCoord  
Прототип функции:  
POINT\* romb::GetCoord()  
Данная функция возвращает координаты вершин фигуры

Наименование функции:  
GetPchar  
Прототип функции:  
int\* romb::GetPchar()  
Данная функция возвращает координаты параметры пера (тип, ширина, цвет(RGB))

Наименование функции:  
GetBchar  
Прототип функции:  
int\* romb::GetBchar()  
Данная функция возвращает параметры кисти(тип, цвет).

Наименование функции:  
SetPStyle  
Прототип функции:  
void romb::SetPStyle(int new\_ptype, int new\_width, color new\_pcolor)  
Данная функция устанавливает новые параметры-атрибуты пера в классе.

Наименование функции:  
SetBStyle  
Прототип функции:  
void romb::SetBStyle(int new\_btype, color new\_bcolor)  
Данная функция устанавливает новые параметры-атрибуты кисти в классе.

Наименование функции:  
draw  
Прототип функции:  
void romb::draw()  
Данная функция рисует фигуру с применением всех атрибутов класса.

Наименование функции:  
load  
Прототип функции:  
void romb::load(char \*fname)  
Данная функция загружает данные о фигуре из файла, и проверяет является ли фигура ромбом, и не выходит ли она за границы окна, если какое-либо из этих условий не выполняется, то ошибка.

Наименование функции:  
save  
Прототип функции:  
void romb::save(char \*fname)  
Данная функция загружает данные в файл.

Наименование функции:  
check\_borders  
Прототип функции:  
int romb::check\_borders(POINT \*ppt, long rr, long rb)  
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:  
compare  
Прототип функции:  
bool Stack::compare(romb rb, romb rb2)  
Данная функция сравнивает два элемента класса «romb». Для этого сравниваются графические характеристики двух элементов и вызывается функция для сравнения пространственных характеристик(смотреть ниже).

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(char \*fname)  
Данная функция считывает данные из файла(каждая строка соответствует своей фигуре). После проверки на принадлежность классу «romb» (п. 2) п. 3) ) одной фигуры, заносит её в стек.

Наименование функции:  
Push  
Прототип функции:  
void Stack::push(romb rb)  
Данная функция создаёт новый элемент типа RbStack, делает его вершиной стека и вставляет элемент(ромб) в соответствующее поле.

Наименование функции:  
pop  
Прототип функции:  
romb Stack::pop()  
Данная функция возвращает вершину стека и удаляет элемент из стека. Вершиной становится следующий элемент.

Наименование функции:  
search  
Прототип функции:  
bool Stack::search(romb 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.

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

# Текст программы

**/\* Файл romb.h \*/**

#ifndef ROMB

#define ROMB

#include <windows.h>

#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"

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

class romb

{

private:

POINT t[4];

color pcolor, bcolor;

int ptype, btype;

int width;

public:

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

void SetFigure(POINT\* new\_t);

void SetPStyle(int new\_ptype, int new\_width, color new\_pcolor);

void SetBStyle(int new\_btype, color new\_bcolor);

POINT\* GetCoord();

int\* GetPchar();

int\* GetBchar();

float slope(POINT a, POINT b);

void check\_figure(POINT \*ppt);

void check\_borders(POINT \*ppt);

void draw();

void load(FILE \*fptr);

void save(FILE \*fptr);

};

#endif

**/\* Файл romb.cpp \*/**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <windows.h>

#include <windowsx.h>

#include "romb.h"

#include <iostream>

#include <stdio.h>

HWND hwnd = GetConsoleWindow();

RECT rt;

//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;

}

}

//функция присваивания значения новых координат вершин

void romb::SetFigure(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;

}

}

//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;

}

}

//Получение координат фигуры

POINT\* romb::GetCoord()

{

return t;

}

//характеристик пера

int\* romb::GetPchar()

{

int a[5];

a[2] = pcolor.r;

a[3] = pcolor.g;

a[4] = pcolor.b;

a[0] = ptype;

a[1] = width;

return a;

}

//характеристик кисти

int\* romb::GetBchar()

{

int a[4];

a[0] = bcolor.r;

a[1] = bcolor.g;

a[2] = bcolor.b;

a[3] = btype;

return a;

}

//установка стиля пера

void romb::SetPStyle(int new\_ptype, int new\_width, color new\_pcolor)

{

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

if (new\_width < 0) throw NEG\_VALUE;

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

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

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

ptype = new\_ptype;

width = new\_width;

pcolor = new\_pcolor;

}

//установка стиля кисти

void romb::SetBStyle(int new\_btype, color new\_bcolor)

{

if (new\_btype < -2 || new\_btype > 5) throw TYPE\_ERR;

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

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

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

btype = new\_btype;

bcolor = new\_bcolor;

}

//рисование ромба

void romb::draw()

{

check\_borders(t);

HDC hdc = GetDC(hwnd);

COLORREF BG = GetPixel(hdc, 10, 10);

HPEN pen = CreatePen(ptype, width, RGB(pcolor.r, pcolor.g, pcolor.b));

HBRUSH brush;

if (btype == -2) brush = GetStockBrush(NULL\_BRUSH);

else

if (btype == -1) brush = CreateSolidBrush(RGB(bcolor.r, bcolor.g, bcolor.b));

else brush = CreateHatchBrush(btype, RGB(bcolor.r, bcolor.g, bcolor.b));

SetBkColor(hdc, BG);

SelectPen(hdc, pen);

SelectBrush(hdc, brush);

Polygon(hdc, t, 4);

ReleaseDC(hwnd, hdc);

}

//загрузка фигуры из файла

void romb::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];

check\_figure(tk);

check\_borders(tk);

SetFigure(tk);

int type, twidth;

color tcol;

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

SetPStyle(type, twidth, tcol);

if (!feof(fptr))

{

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

SetBStyle(type, tcol);

}

else SetBStyle(-2, tcol);

}

//сохранение фигуры в файл

void romb::save(FILE \*fptr)

{

POINT\*k=GetCoord();

printf("%d %d %d %d %d %d %d %d; ", k[0].x, k[0].y, k[1].x, k[1].y, k[2].x, k[2].y, k[3].x, k[3].y);

int\*p = GetPchar();

printf("%d %d (%d, %d, %d) ", p[0], p[1], p[2], p[3], p[4]);

int\*b = GetBchar();

printf("%d (%d, %d, %d) ", b[0], b[1], b[2], b[3]);

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

fprintf(fptr, "%d %d (%d, %d, %d) ", ptype, width, pcolor.r, pcolor.g, pcolor.b);

if (btype != -2) fprintf(fptr, "%d (%d, %d, %d)\n", btype, bcolor.r, bcolor.g, bcolor.b);

}

**/\* Файл rombStack.h \*/**

#include "romb.h"

#include <iostream>

#ifndef ROMB\_CONTAINER

#define ROMB\_CONTAINER

#define EMPTY\_STACK "Стек пуст \n"

//Открытый стек

class Stack

{

private:

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

bool compare(romb rb, romb rb2);

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

public:

Stack();

~Stack();

Stack(char \*fname);

void push(romb rb);

romb pop();

bool search(romb rb);

void DrawSt();

void save(char \*fname);

};

#endif

**/\* Файл rombStack.cpp \*/**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "rombStack.h"

#include <stdio.h>

Stack::Stack() { top = NULL; };

Stack::~Stack()

{

if (top != NULL)

{

RbStack \*St;

while (top != NULL)

{

St = top;

top = top->next;

delete St;

St = top;

}

}

};

Stack::Stack(char \*fname)

{

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

if (!fptr) throw IO\_ERR;

romb squ;

squ.load(fptr);

top = new RbStack;

top->sq = squ;

top->next = NULL;

while (!feof(fptr))

{

squ.load(fptr);

push(squ);

}

fclose(fptr);

}

void Stack::push(romb rb)

{

if (top == NULL)

{

top = new RbStack;

top->sq = rb;

top->next = NULL;

}

else

{

RbStack \*puch = new RbStack;

puch->next = top->next;

top->next = puch;

puch->sq = rb;

}

}

romb Stack::pop()

{

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

else if (top->next==NULL)

{

romb gt = top->sq;

top = NULL;

return gt;

}

else

{

romb gt = top->sq;

top = top->next;

return gt;

}

}

bool Stack::search(romb 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;

}

bool Stack::compare(romb rb, romb rb2)

{

POINT\*k1 = rb.GetCoord();

int\*p1 = rb.GetPchar();

int\*b1 = rb.GetBchar();

POINT\*k2 = rb2.GetCoord();

int\*p2 = rb2.GetPchar();

int\*b2 = rb2.GetBchar();

if ((comparePOINTs(k1, k2)) && \*p1 == \*p2 && \*b1 == \*b2) return true;

else return false;

}

bool Stack::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;

}

void Stack::DrawSt()

{

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

else {

RbStack \*St = top;

while (St != NULL)

{

St->sq.draw();

St = St->next;

}

}

}

void Stack::save(char \*fname)

{

RbStack \*St = top;

romb rb;

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

if (!fptr) throw IO\_ERR;

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

while (St != NULL) {

St->sq.save(fptr);

St = St->next;

}

fclose(fptr);

}

**/\* 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;

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

romb sq, sw, se;

POINT tk[5];

int tptype, tbtype, twidth;

color tpcol, tbcol;

bool est = false;;

Stack \*rs=NULL;

RECT rt;

HWND hwnd = GetConsoleWindow();

while (!exit)

{

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

input = \_getch();

switch (input)

{

case '1': //загрузка

{

GetClientRect(hwnd, &rt);

printf\_s("Введите имя файла: ");

scanf("%s", &fin);

if (rs) rs->~Stack();

rs = NULL;

try

{

rs= new Stack(fin);

}

catch (char \*err) { printf\_s("%s", err); }

} break;

case '2': //сохранение

{

try

{

if (!rs) throw EMPTY\_STACK;

printf\_s("Введите имя файла: ");

scanf("%s", &fout);

rs->save(fout);

}

catch (char \*err) { printf\_s("%s", err); }

} break;

case '3': //ручной ввод ромба и его последующее добавление в стек

{

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");

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

printf\_s("Задайте стиль заливки (тип[-2 - без заливки; -1 - сплошная][-2 - 5], цвет RGB):\n");

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

tk[4] = tk[0];

try {

sq.SetFigure(tk);

sq.SetPStyle(tptype, twidth, tpcol);

sq.SetBStyle(tbtype, tbcol);

rs->push(sq);

}

catch (char \*err) { printf\_s("%s", err); }

}

case '4': //удаление ромба из стека

{

try

{

if (!rs) throw EMPTY\_STACK;

sw = rs->pop();

est = true;

}

catch (char \*err) { printf\_s("%s", err); }

} break;

case '5': //отобразить ромб из памяти

{

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

else {

try {

sw.draw();

}

catch (char \*err) { printf\_s("%s", err); }

}

} break;

case '6': //поиск фигуры

{

try

{

if (!rs) 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);

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

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

printf\_s("Задайте стиль заливки (тип[-2 - без заливки; -1 - сплошная][-2 - 5], цвет RGB):\n");

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

tk[4] = tk[0];

sq.SetFigure(tk);

sq.SetPStyle(tptype, twidth, tpcol);

sq.SetBStyle(tbtype, tbcol);

if (rs->search(sq))

{

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

printf("Отобразить найденную фигуру?[Y/N]\n");

input = \_getch();

switch (input) {

case 'y':

{

sq.draw();

}

case 'n': {}

}

}

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

}

catch (char \*err) { printf\_s("%s", err); }

break; }

case '7': //отобразить стек

{

try

{

if (!rs) throw EMPTY\_STACK;

rs->DrawSt();

}

catch (char \*err) { printf\_s("%s", err); }

} break;

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

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

}

}

}

**/\* Файл user.h \*/**

#include <stdio.h>

#include "rombStack.h"

void printMenu()

{

printf\_s("Выберите действие:\n");

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

printf\_s("[2] - сохранить данные в файл\n");

printf\_s("[3] - ручной ввод ромба и его последующее добавление в стек\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");

printf\_s("[m] - показать меню\n");

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

}

# 7.Тесты

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Результат | Примечания |
| 1 | 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) |  | Рисование стека Загрузка из файла |
| 2 | 200 100 250 100 250 150 200 150; 3 3 (183, 110, 60) 3 (255, 125, 0) |  | Рисование ромба Непосредственное задание. |
| 4 | 4 |  | Удаление из стека. |
| 4 |  |  | Загрузка из файла. |
| 5 | 4 5 6 7 |  | Проверки всех функций, на работу с пустым стеком |

# 8. Результат работы программы

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