МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №3

по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент Абдуль-Хади Филипп, группа М8О-207Б-21

Преподаватель Дорохов Евгений Павлович

Условие

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий **все три** фигуры класса фигуры, согласно вариантам задания. Классы должны удовлетворять следующим правилам:

1. Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
2. Требования к классу контейнера аналогичны требованиям из лабораторной работы 2.
3. Класс-контейнер должен соджержать объекты используя std:shared\_ptr<…>.
4. Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

1. Стандартные контейнеры std.
2. Шаблоны (template).
3. Объекты «по-значению»

Программа должна позволять:

1. Вводить произвольное количество фигур и добавлять их в контейнер.
2. Распечатывать содержимое контейнера.
3. Удалять фигуры из контейнера.

Описание программы

Исходный код лежит в 15 файлах:

1. main.cpp: основная программа, взаимодействие с пользователем посредством ввода команд
2. figure.h: класс фигуры
3. figure.cpp: реализация класса фигуры
4. point.h: класс фигуры
5. point.cpp: реализация класса фигуры
6. square.h: класс квадрата
7. square.cpp: реализация класса квадрата
8. rectangle.h: класс квадрата
9. rectangle.cpp: реализация класса квадрата
10. trapezoid.h: класс квадрата
11. trapezoid.cpp: реализация класса квадрата
12. tvector\_item.h: класс контейнер для класса квадрата
13. tvector\_item.cpp: реализация класса контейнера для класса квадрата
14. tvector.h: класс вектора
15. tvector.cpp: реализация вектора

Дневник отладки

|  |  |
| --- | --- |
| **Проблема** | **Исправление** |
| Неудобно собирать без make файла | Создал make файл |

Недочёты

Отсутствуют

Выводы

Умные указатели могут очень сильно упростить работу с памятью предоставляя свои возможности по автоматическому удалению объектов.

Исходный код ниже:

*//MAIN.CPP*

**#include <iostream>**

**#include "square.h"**

**#include "rectangle.h"**

**#include "trapezoid.h"**

**#include "point.h"**

**#include "tvector.h"**

**using** **namespace** std;

**int** **main**()

{

cout << "Comands:" << endl;

cout << "a - add new figure (a r|s|t [input])" << endl;

cout << "d - erase square by index (d [idx])" << endl;

cout << "p - print all containing squares (p)" << endl;

cout << "q - quit (q)" << endl;

**char** running = 1;

TVector \*vect = **new** TVector();

**char** cmd;

**while**(running)

{

cout << "> ";

cin >> cmd;

**switch**(cmd)

{

**case** 'a':

{

**char** fig;

cin >> fig;

**switch**(fig)

{

**case** 's':

vect->push\_back(shared\_ptr<Figure>(**new** Square(cin)));

**break**;

**case** 'r':

vect->push\_back(shared\_ptr<Figure>(**new** Rectangle(cin)));

**break**;

**case** 't':

vect->push\_back(shared\_ptr<Figure>(**new** Trapezoid(cin)));

**break**;

}

**break**;

}

**case** 'd':

{

**int** di;

cin >> di;

vect->erase(di);

**break**;

}

**case** 'p':

{

cout << \*vect << endl;

**break**;

}

**case** 'q':

{

running = 0;

**break**;

}

**default**:

cout << "wrong input" << endl;

}

}

**delete** vect;

}

*//POINT.H*

**#ifndef POINT\_H**

**#define POINT\_H**

**#include <iostream>**

**class** **Point** {

**public**:

Point();

Point(std::istream &is);

Point(**double** x, **double** y);

**double** x\_;

**double** y\_;

**double** **dist**(Point& other);

Point& **operator**=(**const** Point& sq);

**bool** **operator**==(**const** Point& sq);

**friend** std::istream& **operator**>>(std::istream& is, Point& p);

**friend** std::ostream& **operator**<<(std::ostream& os, Point& p);

};

**#endif *// POINT\_H***

*//POINT.CPP*

**#include "point.h"**

**#include <cmath>**

Point::Point() : x\_(0.0), y\_(0.0) {}

Point::Point(**double** x, **double** y) : x\_(x), y\_(y) {}

Point::Point(std::istream &is) {

is >> x\_ >> y\_;

}

**double** **Point::dist**(Point& other) {

**double** dx = (other.x\_ - x\_);

**double** dy = (other.y\_ - y\_);

**return** std::sqrt(dx\*dx + dy\*dy);

}

Point& Point::**operator**=(**const** Point& sq)

{

x\_ = sq.x\_;

y\_ = sq.y\_;

**return** \***this**;

}

**bool** Point::**operator**==(**const** Point& sq)

{

**return** x\_==sq.x\_ && y\_==sq.y\_;

}

std::istream& **operator**>>(std::istream& is, Point& p) {

is >> p.x\_ >> p.y\_;

**return** is;

}

std::ostream& **operator**<<(std::ostream& os, Point& p) {

os << "(" << p.x\_ << ", " << p.y\_ << ")";

**return** os;

}

*//FIGURE.H*

**#ifndef FIGURE\_H**

**#define FIGURE\_H**

**#include <iostream>**

**#include "point.h"**

**class** **Figure**

{

**public**:

**virtual** **size\_t** **VertexesNumber**() = 0;

**virtual** **double** **Area**() = 0;

**virtual** **void** **Print**(std::ostream& os) = 0;

**virtual** **void** **Read**(std::istream& is) = 0;

**double** **calcTriangleArea**(Point p1, Point p2, Point p3);

**virtual** ~Figure() {}

**friend** std::ostream& **operator**<<(std::ostream& os, Figure& p)

{

p.Print(os);

**return** os;

}

**friend** std::istream& **operator**>>(std::istream& is, Figure& p)

{

p.Read(is);

**return** is;

}

};

**#endif *// FIGURE\_H***

*//FIGURE.CPP*

**#include "figure.h"**

**#include <cmath>**

**double** **Figure::calcTriangleArea**(Point p1, Point p2, Point p3)

{

**return** abs((p1.x\_-p3.x\_)\*(p2.y\_-p3.y\_)-(p2.x\_-p3.x\_)\*(p1.y\_-p3.y\_))/2.0;

}

*//RECTANGLE.H*

**#ifndef RECTANGLE\_H**

**#define RECTANGLE\_H**

**#include <iostream>**

**#include "figure.h"**

**class** **Rectangle**: **public** Figure

{

**public**:

Rectangle(std::istream &is);

Rectangle(Point pnt1, Point pnt2, Point pnt3, Point pnt4);

**size\_t** **VertexesNumber**();

**double** **Area**();

**void** **Print**(std::ostream& os);

**void** **Read**(std::istream& is);

Rectangle& **operator**=(**const** Rectangle& sq);

**bool** **operator**==(**const** Rectangle& sq);

**friend** std::istream& **operator**>>(std::istream& is, Rectangle& p);

**friend** std::ostream& **operator**<<(std::ostream& os, Rectangle& p);

**private**:

Point p1;

Point p2;

Point p3;

Point p4;

};

**#endif**

*//RECTANGLE.CPP*

**#include "rectangle.h"**

**#include <cmath>**

Rectangle::Rectangle(std::istream &is)

{

is >> \***this**;

}

Rectangle::Rectangle(Point pnt1, Point pnt2, Point pnt3, Point pnt4)

{

p1 = pnt1;

p2 = pnt2;

p3 = pnt3;

p4 = pnt4;

}

**size\_t** **Rectangle::VertexesNumber**()

{

**return** 4;

}

**double** **Rectangle::Area**()

{

**return** calcTriangleArea(p1,p2,p3)+calcTriangleArea(p3,p4,p1);

}

**void** **Rectangle::Print**(std::ostream& os)

{

os << \***this**;

}

**void** **Rectangle::Read**(std::istream& is)

{

is >> \***this**;

}

Rectangle& Rectangle::**operator**=(**const** Rectangle& sq)

{

p1=sq.p1;

p2=sq.p2;

p3=sq.p3;

p4=sq.p4;

**return** \***this**;

}

**bool** Rectangle::**operator**==(**const** Rectangle& sq)

{

**return** p1==sq.p1 && p2==sq.p2 && p3==sq.p3 && p4==sq.p4;

}

std::istream& **operator**>>(std::istream& is, Rectangle& r) {

is >> r.p1 >> r.p2 >> r.p3 >> r.p4;

**return** is;

}

std::ostream& **operator**<<(std::ostream& os, Rectangle& r) {

os << "Rectangle: " << r.p1 << " " << r.p2 << " " << r.p3 << " " << r.p4;

**return** os;

}

*//SQUARE.H*

**#ifndef SQUARE\_H**

**#define SQUARE\_H**

**#include <iostream>**

**#include "figure.h"**

**class** **Square**: **public** Figure

{

**public**:

Square(std::istream &is);

Square(Point pnt1, Point pnt2, Point pnt3, Point pnt4);

**size\_t** **VertexesNumber**();

**double** **Area**();

**void** **Print**(std::ostream& os);

**void** **Read**(std::istream& is);

Square& **operator**=(**const** Square& sq);

**bool** **operator**==(**const** Square& sq);

**friend** std::istream& **operator**>>(std::istream& is, Square& r);

**friend** std::ostream& **operator**<<(std::ostream& os, Square& r);

**private**:

Point p1;

Point p2;

Point p3;

Point p4;

};

**#endif**

*//SQUARE.CPP*

**#include "square.h"**

**#include <cmath>**

Square::Square(std::istream &is)

{

is >> \***this**;

}

Square::Square(Point pnt1, Point pnt2, Point pnt3, Point pnt4)

{

p1 = pnt1;

p2 = pnt2;

p3 = pnt3;

p4 = pnt4;

}

**size\_t** **Square::VertexesNumber**()

{

**return** 4;

}

**double** **Square::Area**()

{

**return** calcTriangleArea(p1,p2,p3)+calcTriangleArea(p3,p4,p1);

}

**void** **Square::Print**(std::ostream& os)

{

os << \***this**;

}

**void** **Square::Read**(std::istream& is)

{

is >> \***this**;

}

Square& Square::**operator**=(**const** Square& sq)

{

p1=sq.p1;

p2=sq.p2;

p3=sq.p3;

p4=sq.p4;

**return** \***this**;

}

**bool** Square::**operator**==(**const** Square& sq)

{

**return** p1==sq.p1 && p2==sq.p2 && p3==sq.p3 && p4==sq.p4;

}

std::istream& **operator**>>(std::istream& is, Square& r) {

is >> r.p1 >> r.p2 >> r.p3 >> r.p4;

**return** is;

}

std::ostream& **operator**<<(std::ostream& os, Square& r) {

os << "Square: " << r.p1 << " " << r.p2 << " " << r.p3 << " " << r.p4;

**return** os;

}

*//TRAPEZOID.H*

**#ifndef TRAPEZOID\_H**

**#define TRAPEZOID\_H**

**#include <iostream>**

**#include "figure.h"**

**class** **Trapezoid**: **public** Figure

{

**public**:

Trapezoid(std::istream &is);

Trapezoid(Point p1, Point p2, Point p3, Point p4);

**size\_t** **VertexesNumber**();

**double** **Area**();

**void** **Print**(std::ostream& os);

**void** **Read**(std::istream& is);

Trapezoid& **operator**=(**const** Trapezoid& sq);

**bool** **operator**==(**const** Trapezoid& sq);

**friend** std::istream& **operator**>>(std::istream& is, Trapezoid& r);

**friend** std::ostream& **operator**<<(std::ostream& os, Trapezoid& r);

**private**:

Point p1;

Point p2;

Point p3;

Point p4;

};

**#endif**

*//TRAPEZOID.CPP*

**#include "trapezoid.h"**

**#include <cmath>**

Trapezoid::Trapezoid(std::istream &is)

{

is >> \***this**;

}

Trapezoid::Trapezoid(Point p1, Point p2, Point p3, Point p4)

{

**this**->p1 = p1;

**this**->p2 = p2;

**this**->p3 = p3;

**this**->p4 = p4;

}

**size\_t** **Trapezoid::VertexesNumber**()

{

**return** 4;

}

**double** **Trapezoid::Area**()

{

**return** calcTriangleArea(p1,p2,p3)+calcTriangleArea(p3,p4,p1);

}

**void** **Trapezoid::Print**(std::ostream& os)

{

os << \***this**;

}

**void** **Trapezoid::Read**(std::istream& is)

{

is >> \***this**;

}

Trapezoid& Trapezoid::**operator**=(**const** Trapezoid& sq)

{

p1=sq.p1;

p2=sq.p2;

p3=sq.p3;

p4=sq.p4;

**return** \***this**;

}

**bool** Trapezoid::**operator**==(**const** Trapezoid& sq)

{

**return** p1==sq.p1 && p2==sq.p2 && p3==sq.p3 && p4==sq.p4;

}

std::istream& **operator**>>(std::istream& is, Trapezoid& r) {

is >> r.p1 >> r.p2 >> r.p3 >> r.p4;

**return** is;

}

std::ostream& **operator**<<(std::ostream& os, Trapezoid& r) {

os << "Trapezoid: " << r.p1 << " " << r.p2 << " " << r.p3 << " " << r.p4;

**return** os;

}

*//TVECTOR\_ITEM.H*

**#ifndef TVECTORITEM\_H**

**#define TVECTORITEM\_H**

**#define et\_tvectoritem Figure**

**#include<iostream>**

**#include "square.h"**

**#include <memory>**

**class** **TVectorItem**

{

**public**:

TVectorItem(**const** std::shared\_ptr<et\_tvectoritem>& elem);

**const** std::shared\_ptr<et\_tvectoritem> **getElement**();

~TVectorItem();

**friend** std::ostream& **operator**<<(std::ostream& os, TVectorItem& obj);

**private**:

std::shared\_ptr<et\_tvectoritem> element;

};

**#endif**

*//TVECTOR\_ITEM.CPP*

**#include "tvector\_item.h"**

**#include <iostream>**

TVectorItem::TVectorItem(**const** std::shared\_ptr<et\_tvectoritem>& elem)

{

**this**->element = elem;

}

**const** std::shared\_ptr<et\_tvectoritem> **TVectorItem::getElement**()

{

**return** **this**->element;

}

std::ostream& **operator**<<(std::ostream& os, TVectorItem& obj)

{

os << \*(obj.element);

**return** os;

}

TVectorItem::~TVectorItem()

{

}

*//TVCETOR.H*

**#ifndef TVECTOR\_H**

**#define TVECTOR\_H**

**#define et\_tvector TVectorItem**

**#include<iostream>**

**#include "tvector\_item.h"**

**#include <memory>**

**class** **TVector**

{

**public**:

TVector();

~TVector();

**int** **size**();

**char** **empty**();

std::shared\_ptr<et\_tvector>& **operator**[](**int** idx);

**void** **resize**(**int** newsize);

**void** **push\_back**(**const** std::shared\_ptr<et\_tvectoritem>& sq);

std::shared\_ptr<et\_tvectoritem> **pop\_back**();

**void** **erase**(**int** pos);

**void** **clear**();

**friend** std::ostream& **operator**<<(std::ostream& os, TVector& obj);

**private**:

std::shared\_ptr<et\_tvector> \*vals;

**int** len;

**int** rLen;

};

**#endif**

*//TVCETOR.CPP*

**#include "tvector.h"**

**#include <iostream>**

**#include <cstring>**

TVector::TVector()

{

vals = NULL;

len = 0;

rLen = 0;

}

**int** **TVector::size**()

{

**return** len;

}

**char** **TVector::empty**()

{

**return** len == 0;

}

std::shared\_ptr<et\_tvector>& TVector::**operator**[](**int** idx)

{

**return** vals[idx];

}

**void** **TVector::resize**(**int** newsize)

{

vals = (std::shared\_ptr<et\_tvector>\*)realloc((**void**\*)vals, **sizeof**(std::shared\_ptr<et\_tvector>)\*newsize);

}

**void** **TVector::push\_back**(**const** std::shared\_ptr<et\_tvectoritem>& sq)

{

**if**(rLen)

{

**if**(len>=rLen)

{

rLen<<=1;

resize(rLen);

}

}

**else**

{

rLen=1;

resize(rLen);

}

vals[len]=std::shared\_ptr<TVectorItem>(**new** TVectorItem(sq));

len++;

}

std::shared\_ptr<et\_tvectoritem> **TVector::pop\_back**()

{

std::shared\_ptr<et\_tvector> ret = vals[len-1];

erase(len-1);

**return** ret->getElement();

}

**void** **TVector::erase**(**int** pos)

{

**for**(**int** i = pos; i < len - 1; i++)

{

vals[i] = vals[i + 1];

}

len--;

**if**(len==rLen>>1)

resize(len);

}

**void** **TVector::clear**()

{

**if**(!empty())

{

**for**(**int** i = 0; i < len; i++)

vals[i] = NULL;

**delete** vals;

len = 0;

rLen = 0;

}

}

std::ostream& **operator**<<(std::ostream& os, TVector& obj)

{

os << '[';

**for**(**int** i = 0; i < obj.len; i++)

{

os << \*(obj.vals[i]);

**if**(i != obj.len - 1)

os << ", ";

}

os << ']';

**return** os;

}

TVector::~TVector()

{

clear();

}