МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №4**

по курсу “Объектно-ориентированное программирование» 1 семестр, 2021/22 уч. год

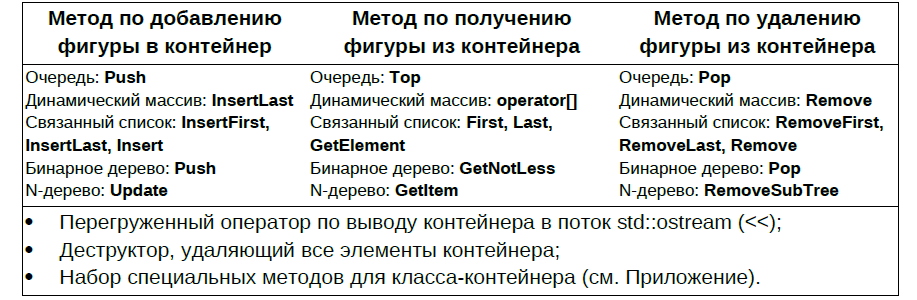
Студентка: *Волошинская Евгения Владимировна, группа М8О-207Б-20*

Преподаватель: *Дорохов Евгений Павлович*

**Задание**

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру (колонка фигура 1), согласно вариантам задания. Классы должны удовлетворять следующим правилам:

* Требования к классу фигуры аналогичны требованиям из лабораторной работы №1;
* Иметь общий родительский класс Figure;
* Классы фигур должны содержать набор следующих методов:
  + Перегруженный оператор ввода координат вершин фигуры из потока std::istream (>>). Он должен заменить конструктор, принимающий координаты вершин из стандартного потока;
  + Перегруженный оператор вывода в поток std::ostream (<<), заменяющий метод Print из лабораторной работы 1;
  + Оператор копирования (=);
  + Оператор сравнения с такими же фигурами (==).
* Класс-контейнер должен содержать объекты фигур “по значению” (не по ссылке);
* Класс-контейнер должен содержать набор следующих методов:



Полное описание всех методов можно найти в приложении к лабораторной.

Нельзя использовать:

* Стандартные контейнеры std;
* Шаблоны (template);
* Различные варианты умных указателей (unique\_ptr, shared\_ptr, weak\_ptr,...).

Программа должна позволять:

* Вводить произвольное количество фигур и добавлять их в контейнер;
* Распечатывать содержимое контейнера;
* Удалять фигуры из контейнера.

***Вариант 9:***

|  |  |  |  |
| --- | --- | --- | --- |
| **Фигура №1** | **Имя класса** | **Контейнер 1-го уровня** | **Имя класса** |
| Треугольник | Triangle | Связанный список | TLinkedList |

**Вариант: Связный список:**

class TLinkedList {

public:

// Конструктор по умолчанию

TLinkedList();

// Конструктор копирования

TLinkedList(const TLinkedList& other);

// Метод, возвращающий первую фигуру списка

const Polygon& First();

// Метод, возвращающий последнюю фигуру списка

const Polygon& Last();

// Метод, добавляющий элемент в начало списка

void InsertFirst(const Polygon& polygon);

// Метод, добавляющий фигуру в конец списка

void InsertLast(const Polygon& polygon);

// Метод, добавляющий фигуру в произвольное место списка

void Insert(const Polygon& polygon, size\_t position);

// Метод, удаляющий первый элемент списка

void RemoveFirst();

// Метод, удаляющий последний элемент списка

void RemoveLast();

// Метод, удаляющий произвольный элемент списка

void Remove(size\_t position);

// Метод получения фигуры списка по индексу.

const Polygon& GetItem(size\_t idx);

// Метод, проверяющий пустоту списка

bool Empty();

// Метод, возвращающий длину массива

size\_t Length();

// Оператор вывода для массива в формате:

// "S1 -> S2 -> ... -> Sn", где Si - площадь фигуры

friend std::ostream& operator<<(std::ostream& os, const TLinkedList& list);

// Метод, удаляющий все элементы контейнера,

// но позволяющий пользоваться им.

void Clear();

// Деструктор

virtual ~TLinkedList();

};

**Описание программы**

Исходный код лежит в 10 файлах:

1. main.cpp: часть программы, отвечающая за взаимодействие с пользователем через консоль. В ней происходит инициализация объектов и вызов функций работы с ними, заполнение стандартного контейнера вектор введенными объектами и печать его содержимого;
2. point.h: описание класса Point точек A(a1, a2);
3. point.cpp: реализация класса Point;
4. figure.h: описание абстрактного класса-родителя Figure;
5. figure.cpp: реализация класса Figure;
6. triangle.h: описание класса Triangle треугольников, заданных по трем точкам, наследника Figure;
7. triangle.cpp: реализация класса Triangle;
8. item.h: описание класса Item, объектами которого являются элементы связанного списка;
9. item.cpp: реализация класса Item;
10. tlinkedlist.h: описание класса TLinkedList, объекты которого – связанные списки элементов типа Item;
11. tlinkedlist.cpp: реализация класса TLinkedList.

Также используется файл CMakeLists.txt с конфигурацией CMake для автоматизации сборки программы.

**Дневник отладки**

**Вывод**

В данной лабораторной работе я продолжила знакомиться с основами ООП в языке C++. Было изучено понятие абстрактного класса и виртуальной функции. Кроме принципов абстракции и инкапсуляции, были применены на практике принципы наследования и полиморфизма. Базовым классом в наследовании стал класс Figure, от которого дочерние Triangle, Rectangle и Square унаследовали общие имена методов VertexesNumber, Area, Print со своей реализацией для каждого класса, что возможно в C++ благодаря полиморфизму. Таким образом, в данной работе я получила важные теоретические навыки для работы с объектно-ориентированными языками, а также расширила свои знания о программировании в C++.

**Исходный код**

main.cpp:

#include <iostream>

#include <vector>

#include "triangle.h"

#include "rectangle.h"

#include "square.h"

int main() {

std::vector<Figure\*> figures;

std::cout << "Enter triangle vertices' coordinates: "; // -4 2 2 0 0 -2

Triangle t1(std::cin);

std::cout << "Number of vertices: " << t1.VertexesNumber() << std::endl;

t1.Print(std::cout);

std::cout << "Area:" << t1.Area() << std::endl;

figures.push\_back(&t1);

std::cout << "Enter rectangle vertices' coordinates: "; // 0 0 5 0 5 -2 0 -2

Rectangle r1(std::cin);

std::cout << "Number of vertices: " << r1.VertexesNumber() << std::endl;

r1.Print(std::cout);

std::cout << "Area:" << r1.Area() << std::endl;

figures.push\_back(&r1);

std::cout << "Enter square vertices' coordinates: "; // -1 1 1 1 1 -1 -1 -1

Square s1(std::cin);

std::cout << "Number of vertices: " << s1.VertexesNumber() << std::endl;

s1.Print(std::cout);

std::cout << "Area:" << s1.Area() << std::endl;

figures.push\_back(&s1);

std::cout << "Enter rectangle vertices' coordinates: "; // -2.5 3 2.5 3 2.5 -3 0 -3

Rectangle r2(std::cin);

std::cout << "Number of vertices: " << r2.VertexesNumber() << std::endl;

r2.Print(std::cout);

std::cout << "Area:" << r2.Area() << std::endl;

figures.push\_back(&r2);

std::cout << "\nFigures in container:" << std::endl;

for (int i = 0; i < figures.size(); i++)

{

figures[i]->Print(std::cout);

}

return 0;

}

point.h:

#ifndef POINT\_H

#define POINT\_H

#include <iostream>

class Point {

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

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

public:

Point();

Point(double x, double y);

Point(std::istream &is);

double dist(Point& other);

private:

double x\_;

double y\_;

};

#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);

}

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 "point.h"

class Figure {

public:

virtual size\_t VertexesNumber() = 0;

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

virtual double Area() = 0;

virtual ~Figure() {};

};

#endif // FIGURE\_H

triangle.h:

#ifndef TRIANGLE\_H

#define TRIANGLE\_H

#include <iostream>

#include "figure.h"

class Triangle : public Figure {

public:

Triangle();

Triangle(Point a, Point b, Point c);

Triangle(std::istream &is);

Triangle(const Triangle& other);

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

virtual ~Triangle();

private:

Point p1;

Point p2;

Point p3;

};

#endif // TRIANGLE\_H

triangle.cpp:

#include "triangle.h"

#include <iostream>

#include <cmath>

Triangle::Triangle()

: p1(0.0, 0.0), p2(0.0, 0.0), p3(0.0, 0.0) { // можно, но длиннее p1(Point(0.0, 0.0))

std::cout << "Default triangle created" << std::endl;

}

Triangle::Triangle(Point a, Point b, Point c)

: p1(a), p2(b), p3(c) {

std::cout << "Triangle created by parameters" << std::endl;

}

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

is >> p1 >> p2 >> p3;

}

Triangle::Triangle(const Triangle& other)

: Triangle(other.p1, other.p2, other.p3) {

std::cout << "Triangle copy created" << std::endl;

}

size\_t Triangle::VertexesNumber() {

return(size\_t)3;

}

double Triangle::Area() {

double p12 = p1.dist(p2);

double p13 = p1.dist(p3);

double p23 = p2.dist(p3);

double p = (p12 + p23 + p13) / 2.0;

return std::sqrt(p \* (p - p12) \* (p - p23) \* (p - p13));

}

void Triangle::Print(std::ostream& os) {

os << "Triangle: ";

os << p1 << ", ";

os << p2 << ", ";

os << p3 << std::endl;

}

Triangle::~Triangle() {

//std::cout << "Triangle deleted" << std::endl;

}

square.h:

#ifndef SQUARE\_H

#define SQUARE\_H

#include <iostream>

#include "figure.h"

class Square : public Figure {

public:

Square();

Square(Point a, Point b, Point c, Point d);

Square(std::istream &is);

Square(const Square& other);

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

virtual ~Square();

private:

Point p1;

Point p2;

Point p3;

Point p4;

};

#endif // SQUARE\_H

square.cpp:

#include "square.h"

#include <iostream>

#include <cmath>

Square::Square()

: p1(0.0, 0.0), p2(0.0, 0.0), p3(0.0, 0.0), p4(0.0, 0.0) {

std::cout << "Default square created" << std::endl;

}

Square::Square(Point a, Point b, Point c, Point d)

: p1(a), p2(b), p3(c), p4(d) {

std::cout << "Square created by parameters" << std::endl;

}

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

is >> p1 >> p2 >> p3 >> p4;

}

Square::Square(const Square& other)

: Square(other.p1, other.p2, other.p3, other.p4) {

std::cout << "Square copy created" << std::endl;

}

size\_t Square::VertexesNumber() {

return(size\_t)4;

}

double Square::Area() {

double p12 = p1.dist(p2);

double p = p12 \* p12;

return p;

}

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

os << "Square: ";

os << p1 << ", ";

os << p2 << ", ";

os << p3 << ", ";

os << p4 << std::endl;

}

Square::~Square() {

//std::cout << "Square deleted" << std::endl;

}

rectangle.h:

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <iostream>

#include "figure.h"

class Rectangle : public Figure {

public:

Rectangle();

Rectangle(Point a, Point b, Point c, Point d);

Rectangle(std::istream &is);

Rectangle(const Rectangle& other);

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

virtual ~Rectangle();

private:

Point p1;

Point p2;

Point p3;

Point p4;

};

#endif // RECTANGLE\_H

rectangle.cpp:

#include "rectangle.h"

#include <iostream>

#include <cmath>

Rectangle::Rectangle()

: p1(0.0, 0.0), p2(0.0, 0.0), p3(0.0, 0.0), p4(0.0, 0.0) {

std::cout << "Default rectangle created" << std::endl;

}

Rectangle::Rectangle(Point a, Point b, Point c, Point d)

: p1(a), p2(b), p3(c), p4(d) {

std::cout << "Rectangle created by parameters" << std::endl;

}

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

is >> p1 >> p2 >> p3 >> p4;

}

Rectangle::Rectangle(const Rectangle& other)

: Rectangle(other.p1, other.p2, other.p3, other.p4) {

std::cout << "Rectangle copy created" << std::endl;

}

size\_t Rectangle::VertexesNumber() {

return(size\_t)4;

}

double Rectangle::Area() {

double p12 = p1.dist(p2);

double p23 = p2.dist(p3);

double p = p12 \* p23;

return p;

}

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

os << "Rectangle: ";

os << p1 << ", ";

os << p2 << ", ";

os << p3 << ", ";

os << p4 << std::endl;

}

Rectangle::~Rectangle() {

//std::cout << "Rectangle deleted" << std::endl;

}

CMakeLists.txt:

cmake\_minimum\_required(VERSION 3.10)

project(lab1)

set(CMAKE\_CXX\_STANDARD 11)

add\_executable(lab1 point.h

point.cpp

main.cpp

figure.h

triangle.h triangle.cpp

rectangle.h rectangle.cpp square.h square.cpp)