**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 3**

Тема: Механизмы наследования в С++

Студент: Волков

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи**

Разработать классы согласно варианту задания, классы должны наследоваться от базового класса Figure. Фигуры являются фигурами вращения. Все классы должны поддерживать набор общих методов:

1. Вычисление центра фигуры;

2. Вывод в стандартный поток вывода std::cout координат вершин фигуры;

3. Вычисление площади фигуры;

Создать программу, которая позволяет:

· Вводить из стандартного ввода std::cin фигуры, согласно варианту задания.

· Сохранять созданные фигуры в динамический массив std::vector<Figure\*>

· Вызывать для всего массива общие функции (1-3 см. выше).Т.е. распечатывать для каждой фигуры в массиве геометрический центр, координаты вершин и площадь.

· Необходимо уметь вычислять общую площадь фигур в массиве.

· Удалять из массива фигуру по индексу;

Вариант: прямоугольник, трапеция, ромб

1. **Описание программы**
2. **Набор и результаты тестов**

**тест 1:**

Input rectangle. The First one is point. The second one are a and b

x, y is a left bottom corner coords

a and b are width and height

0 0

-1 -1

Invalid rectangle parameters!

Process finished with exit code 1

**тест 2:**

5 1

Rectangle vertices: [(0, 0), (0, 2), (4, 2), (4, 0)]

Center of figure is (2, 1)

Square of figure is 8

Trapeze vertices: [(0, 0), (0.5, 2.95804), (1.5, 2.95804), (2, 0)]

Center of figure is (1, 1.47902)

Square of figure is 4.43706

Rhombus vertices: [ (0, 0), (2.5, 0.5), (5, 0), (-2.5, -0.5) ]

Center of figure is (2.5, 0)

Square of figure is 2.5

Total square: 14.9371

Input index to remove figure

2

Rectangle vertices: [(0, 0), (0, 2), (4, 2), (4, 0)]

Center of figure is (2, 1)

Square of figure is 8

Rhombus vertices: [ (0, 0), (2.5, 0.5), (5, 0), (-2.5, -0.5) ]

Center of figure is (2.5, 0)

Square of figure is 2.5

Total square after erase: 10.5

Process finished with exit code 0

**тест 3:**

Input rectangle. The First one is point. The second one are a and b

x, y is a left bottom corner coords

a and b are width and height

f

Invalid rectangle parameters!

Process finished with exit code 1

1. **Листинг программы**

**main.cpp**

***/\* Волков Матвей Андреевич***

***\* Разработать классы согласно варианту задания,***

***\* классы должны наследоваться от базового класса Figure.***

***\* Фигуры являются фигурами вращения.***

***\* Все классы должны поддерживать набор общих методов:***

***\* - Вычисление геометрического центра фигуры***

***\* - Вывод в стандартный поток std::cout координат вершин фигуры***

***\* - Вычисление площади фигуры***

***\* Создать программу, которая позволяет:***

***\* - Вводить в стандартный поток std::cin фигуры***

***\* - Сохранять заданные фигуры в вектор std::vector<Figure\*>***

***\* - Вызывать для всего массива общие функции***

***\* - Необходимо уметь вычислять общую площадь фигур в массиве.***

***\* - Удалять из массива фигуру по индексу***

***\* Прямоугольник, трапеция, ромб.***

***\*/***

**#include "Figures.h"**

**#include "Rectangle.h"**

**#include "Rhombus.h"**

**#include "Trapeze.h"**

**#include <vector>**

***long double* TotalArea(std::vector<Figures\*> & figures) {**

***long double* res = 0.0;**

***for* (*auto* fig : figures) {**

**res = res + fig->Area();**

**}**

***return* res;**

**}**

***int* main() {**

***double* x, y, a, b, d1, d2, c;**

**std::vector<Figures \*> figure;**

***// rectangle***

**x = 0, y = 0, a = -1, b = -1;**

**std::cout << "Input rectangle. The First one is point. The second one are a and b" << std::endl**

**<< "x, y is a left bottom corner coords" << std::endl**

**<< "a and b are width and height" << std::endl;**

**std::cin >> x >> y >> a >> b;**

**Rectangle \*rectangle = *nullptr*;**

***try* {**

**rectangle = *new* Rectangle(x, y, a, b);**

**}**

***catch* (std::invalid\_argument &ex) {**

**std::cout << ex.what() << std::endl;**

***return* 1;**

**}**

**figure.push\_back(rectangle);**

***// Trapeze***

**x = 0.0, y = 0.0, a = -1.0, b = -1.0, c = -1.0;**

**std::cout << "Input trapeze. The First one is point. The second one are a, b and c." << std::endl**

**<< "x, y is a left bottom corner coords" << std::endl**

**<< "a, b and c are larger, smaller base and side" << std::endl;**

**std::cin >> x >> y >> a >> b >> c;**

**Trapeze \*trapeze = *nullptr*;**

***try* {**

**trapeze = *new* Trapeze(x, y, a, b, c);**

**}**

***catch* (std::invalid\_argument &ex) {**

**std::cout << ex.what() << std::endl;**

***return* 1;**

**}**

**figure.push\_back(trapeze);**

**x = 0.0, y = 0.0, d1 = -1, d2 = -1;**

**std::cout << "Input rhombus. The First one is point. The second one are d1 and d2." << std::endl**

**<< "x, y is a left corner coords" << std::endl**

**<< "d1 is a bigger diagonal. d2 is smaller diagonal" << std::endl;**

**std::cin >> x >> y >> d1 >> d2;**

**Rhombus \*rhombus = *nullptr*;**

***try* {**

**rhombus = *new* Rhombus(x, y, d1, d2);**

**}**

***catch* (std::invalid\_argument &ex) {**

**std::cout << ex.what() << std::endl;**

***return* 1;**

**}**

**figure.push\_back(rhombus);**

***// working with vector***

***for* (*auto* fig : figure) {**

**fig->Print();**

**std::cout << "Center of figure is " << fig->Center() << std::endl;**

**std::cout << "Square of figure is " << fig->Area() << std::endl;**

**}**

**std::cout << "Total square: " << TotalArea(figure) << std::endl;**

***int* i = 0;**

**std::cout << "Input index to remove figure" << std::endl;**

**std::cin >> i;**

***for* (*auto* fig = figure.begin(); fig != figure.end() *and* i > 0; ++fig) {**

**--i;**

***if* (i == 0) {**

**figure.erase(fig);**

**}**

**}**

***for* (*auto* fig : figure) {**

**fig->Print();**

**std::cout << "Center of figure is " << fig->Center() << std::endl;**

**std::cout << "Square of figure is " << fig->Area() << std::endl;**

**}**

**std::cout << "Total square after erase: " << TotalArea(figure) << std::endl;**

***return* 0;**

**}**

**Rectangle.h**

**#ifndef RECTANGLE\_HPP**

**#define RECTANGLE\_HPP**

**#include "Figures.h"**

***class* Rectangle : *public* Figures {**

***private*:**

***double* A, B;**

***public*:**

**Rectangle(*const double* &x, *const double* &y, *const double* &a, *const double* &b);**

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

**Cord Center() *override*;**

***void* Print() *override*;**

***friend* std::ostream & *operator* << (std::ostream & out, *const* Rectangle & rect);**

**};**

**#endif */\* RECTANGLE\_HPP \*/***

**Rectangle.cpp**

**#include "Rectangle.h"**

**Rectangle::Rectangle(*const double* &x, *const double* &y,**

***const double* &a, *const double* &b) : A(a), B(b) { *// x and y are left-bottom point of figure***

***this*->X = x;**

***this*->Y = y;**

***if* (A < 0.0 *or* B < 0.0) {**

***throw* std::invalid\_argument("Invalid rectangle parameters!");**

**}**

**}**

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

***return* A \* B;**

**}**

**Cord Rectangle::Center() {**

***return* Cord(X + A / 2.0, Y + B / 2.0);**

**}**

***void* Rectangle::Print() {**

**std::cout << \**this* << std::endl;**

**}**

**std::ostream &*operator*<<(std::ostream &out, *const* Rectangle &rectangle) {**

**out << "Rectangle vertices: [";**

**out << Cord(rectangle.X, rectangle.Y) << ", ";**

**out << Cord(rectangle.X, rectangle.Y + rectangle.B) << ", ";**

**out << Cord(rectangle.X + rectangle.A, rectangle.Y + rectangle.B) << ", ";**

**out << Cord(rectangle.X + rectangle.A, rectangle.Y);**

**out << "]";**

***return* out;**

**}**

**Trapeze.h**

**#ifndef OOP\_EXERSICE\_03\_TRAPEZE\_H**

**#define OOP\_EXERSICE\_03\_TRAPEZE\_H**

**#include "Figures.h"**

***class* Trapeze : *public* Figures{**

***private*:**

***double* A, B, C;**

***public*:**

**Trapeze(*const double* &x, *const double* &y, *const double* &a, *const double* &b, *const double* &c);**

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

**Cord Center() *override*;**

***void* Print() *override*;**

***friend* std::ostream & *operator* << (std::ostream & out, *const* Trapeze & trapeze);**

**};**

**#endif *//OOP\_EXERSICE\_03\_TRAPEZE\_H***

**Trapeze.cpp**

**#include "Trapeze.h"**

**Trapeze::Trapeze(*const double* &x, *const double* &y,**

***const double* &a, *const double* &b, *const double* &c) : A(a), B(b), C(c) { *// x and y are left-bottom point of figure***

***this*->X = x;**

***this*->Y = y;**

***if* (A < 0.0 *or* B < 0.0 *or* C < 0.0) {**

***throw* std::invalid\_argument("Invalid trapeze parameters!");**

**}**

***if* (B > A) {**

**std::swap(A, B);**

**}**

**}**

***double* Trapeze::Area() {**

***double* diff = (A - B) / 2.0;**

***double* height = std::sqrt(C \* C - diff \* diff);**

***return* height \* (A + B) / 2.0;**

**}**

**Cord Trapeze::Center() {**

***double* diff = (A - B) / 2.0;**

***double* height = std::sqrt(C \* C - diff \* diff);**

***return* Cord(X + A / 2.0, Y + height / 2.0);**

**}**

***void* Trapeze::Print() {**

**std::cout << \**this* << std::endl;**

**}**

**std::ostream &*operator*<<(std::ostream &out, *const* Trapeze &trapeze) {**

***double* diff = (trapeze.A - trapeze.B) / 2.0;**

***double* height = std::sqrt(trapeze.C \* trapeze.C - diff \* diff);**

**out << "Trapeze vertices: [";**

**out << Cord(trapeze.X, trapeze.Y) << ", ";**

**out << Cord(trapeze.X + diff, trapeze.Y + height) << ", ";**

**out << Cord(trapeze.X + trapeze.A - diff, trapeze.Y + height) << ", ";**

**out << Cord(trapeze.X + trapeze.A, trapeze.Y);**

**out << "]";**

***return* out;**

**}**

**Rhombus.h**

**#ifndef OOP\_EXERSICE\_03\_RHOMBUS\_H**

**#define OOP\_EXERSICE\_03\_RHOMBUS\_H**

**#include "Figures.h"**

***class* Rhombus : *public* Figures {**

***private*:**

***double* D1, D2;**

***public*:**

**Rhombus(*const double* &x, *const double* &y, *const double* &d1, *const double* &d2);**

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

**Cord Center() *override*;**

***void* Print() *override*;**

***friend* std::ostream& *operator*<<(std::ostream& out, *const* Rhombus& rhombus);**

**};**

**#endif *//OOP\_EXERSICE\_03\_RHOMBUS\_H***

**Rhombus.cpp**

**#include "Rhombus.h"**

**Rhombus::Rhombus(*const double* &x, *const double* &y, *const double* &d1, *const double* &d2) : D1(d1), D2(d2) {**

***//point of left edge***

***this*->X = x;**

***this*->Y = y;**

***if* (D1 < 0 || D2 < 0) {**

***throw* std::invalid\_argument("Invalid Rhombus parameters!");**

**}**

**}**

***double* Rhombus::Area() {**

***return* D1 \* D2 / 2;**

**}**

**Cord Rhombus::Center() {**

***return* Cord(X + D1 / 2, Y);**

**}**

***void* Rhombus::Print() {**

**std::cout << \**this* << std::endl;**

**}**

**std::ostream &*operator*<<(std::ostream &out, *const* Rhombus &rhombus) {**

**out << "Rhombus vertices: [ " << Cord(rhombus.X, rhombus.Y) << ", "**

**<< Cord(rhombus.X + rhombus.D1 / 2, rhombus.Y + rhombus.D2 / 2) << ", "**

**<< Cord(rhombus.X + rhombus.D1, rhombus.Y) << ", "**

**<< Cord(rhombus.X - rhombus.D1 / 2, rhombus.Y - rhombus.D2 / 2) << " ]" << std::endl;**

***return* out;**

**}**

**Coord.h**

**#ifndef OOP\_EXERSICE\_03\_COORD\_H**

**#define OOP\_EXERSICE\_03\_COORD\_H**

**#include <iostream>**

**#include <cmath>**

***struct* Cord {**

***double* X, Y;**

**Cord() : X(NAN), Y(NAN) {};**

**Cord(*double* x, *double* y) : X(x), Y(y) {};**

**~Cord() = *default*;**

**};**

***inline* std::ostream& *operator*<< (std::ostream & out, *const* Cord & crd) {**

**out << "(" << crd.X << ", " << crd.Y << ")";**

***return* out;**

**}**

**#endif *//OOP\_EXERSICE\_03\_COORD\_H***

1. **Вывод**

В ходе проделанной работы я научился пользоваться наследованием класса, узнал что такое *inline* и *friend*. Также узнал чем отличаются уровни доступа *protected, public* и *private.*

1. **Список литературы**