**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 2**

Тема: Наследование, полиморфизм

Студент: Ефимов Александр

Группа: 80-201

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

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 геометрического 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;

1. **Описание программы**

Программа разбита на три исходных файла: объявление класса и функций в *Shape.h*, их определения в *Shape.cpp*, а также созданное элементарное меню в *main.cpp*.

* *Shape.h*

Содержит объявление класса *Shape*, его наследников и дружественную перегрузку оператора вывода <<, который вызывает метод вывода подклассов. *Shape* имеет интерфейс, показанный с помощью чистых приватных виртуальных функций *Centre, Area, print,* которые обозначают минимальный набор функций для унаследованных классов (т.е. нельзя создать класс без перегрузки виртуальных методов), и публичную структуру *Point*, определяющую пару переменных *x* и *y* для точек,а также имеющую перегрузку оператора выходящего потока <<. Также в нем объявлены унаследованные классы *Triangle, Rectangle, Square*.

* *Shape.cpp*

Имплементация объявленных методов. Для помощи в проверке входных данных (т.е. если они образуют нужную фигуру) используются вспомогательные функции *IsRight* (проверка на прямой угол), IsRectangle (проверка прямоугольности), distance (вывод квадрата размера вектора),

*IsSquare* (проверка если фигура - квадрат).

* *main.cpp*

В *main.cpp* содержится простое меню для работы с библиотечными методами, а также функция *printList*, выдающая список этих фигур.

1. **Набор testcases**

При запуске программы создается строка с значением 101101 (ведущие нули опущены) для демонстрации работы пользовательского литерала. Следующие списки чисел последовательно вводятся в меню. Они позволяют вызвать каждую (или большинство) функций для проверки работоспособности.

|  |  |  |
| --- | --- | --- |
| № | **test\_01.txt** | **test\_02.txt** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | 1  1  4 9  -2 4  -1 -2  1  2  0 -2  5 3  -7 15  -12 10  1  3  0 4  4 0  0 -4  -4 0  2  5  2  6  2  0 | 1  3  -5 1  -1 5  5 -1  1 -5  1  2  -5 1  -1 5  5 -1  1 -5  1  1  0 0  0 0  3 5  2  0 |

1. Результаты выполнения тестов.

Изначально тесты выдавали неверные результаты и принимали неверные фигуры. Результат совпал с ожидаемым после перепроверки и исправления неверного кода.

1. Листинг программы

* **Shape.h**

#ifndef SHAPE\_BASE\_H

#define SHAPE\_BASE\_H

#include <iostream>

class Shape {

public:

friend std::ostream& operator << (std::ostream&, const Shape&);

struct Point {

int x {0};

int y {0};

bool operator == (const Point& rhs)

{ return x == rhs.x && y == rhs.y; }

};

Point GetCentre() const

{ return centre; }

double GetArea() const

{ return area; }

protected:

Point centre;

double area{0};

private:

virtual Point Centre() const = 0;

virtual double Area() const = 0;

virtual std::ostream& print(std::ostream&) const = 0;

};

class Triangle : public Shape {

public:

Triangle () { }

Triangle (std::istream&);

Point Centre() const override;

double Area() const override;

std::ostream& print(std::ostream&) const override;

private:

Point p[3];

};

class Rectangle : public Shape {

public:

Rectangle () { }

Rectangle (std::istream&);

Point Centre() const override;

double Area() const override;

std::ostream& print(std::ostream&) const override;

private:

Point p[4];

};

class Square : public Shape {

public:

Square () { }

Square (std::istream&);

Point Centre() const override;

double Area() const override;

std::ostream& print(std::ostream&) const override;

private:

Point p[4];

};

#endif

* **Shape.cpp**

#include <cmath>

#include <exception>

#include "Shape.h"

typedef Shape::Point Point;

//Auxiliary fuctions

bool IsRight(Point a, Point b, Point c)

{

Point vec1 = {b.x - a.x, b.y - a.y}, vec2 = {c.x - a.x, c.y - a.y};

if ( (vec1.x \* vec2.x + vec1.y \* vec2.y) == 0 ) return true;

return false;

}

bool IsRectangle(Point p[4])

{

if (

IsRight(p[2], p[1], p[3]) &&

IsRight(p[3], p[2], p[0]) &&

IsRight(p[1], p[0], p[2]) &&

IsRight(p[0], p[3], p[1])

) return true;

return false;

}

inline int distance(Point a, Point b)

{

return (b.x - a.x) \* (b.x - a.x) + (b.y - a.y) \* (b.y - a.y);

}

bool IsSquare(Point p[4])

{

if (

distance(p[0], p[1]) == distance(p[1], p[2]) &&

distance(p[1], p[2]) == distance(p[2], p[3]) &&

distance(p[2], p[3]) == distance(p[3], p[0]) &&

distance(p[3], p[0]) == distance(p[0], p[1])

) return true;

return false;

}

// Declaring the printing operator for all

std::ostream& operator << (std::ostream& os, const Shape& shape)

{

shape.print(os);

return os;

}

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

{

os << '(' << p.x << " , " << p.y << ')';

return os;

}

//Defining the triangle

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

{

std::cout << "Input point coordinates\nA: ";

is >> p[0].x >> p[0].y;

std::cout <<"B: ";

is >> p[1].x >> p[1].y;

std::cout <<"C: ";

is >> p[2].x >> p[2].y;

if (

p[0] == p[1] || p[0] == p[2] || p[1] == p[2]

) throw std::runtime\_error("Coordinates do not create triangle");

centre = Centre();

area = Area();

}

Point Triangle::Centre() const

{

Point o;

o.x = (p[0].x + p[1].x + p[2].x) / 3;

o.y = (p[0].y + p[1].y + p[2].y) / 3;

return o;

}

double Triangle::Area() const

{

double area = (double) (p[0].x \* (p[1].y - p[2].y) + p[1].x \* (p[2].y - p[0].y) + p[2].x \* (p[0].y - p[1].y)) / 2;

return area < 0 ? -area : area;

}

std::ostream& Triangle::print(std::ostream& os) const

{

os << "Triangle: " << p[0] << "; " << p[1] << "; " << p[2] << "; Centre = " << centre << "; Area = " << area;

return os;

}

//Defining the rectangle

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

{

std::cout << "Input point coordinates clockwise or counter clockwise\nA: ";

is >> p[0].x >> p[0].y;

std::cout <<"B: ";

is >> p[1].x >> p[1].y;

std::cout <<"C: ";

is >> p[2].x >> p[2].y;

std::cout <<"D: ";

is >> p[3].x >> p[3].y;

centre = Centre();

area = Area();

if (!IsRectangle(p)) throw std::runtime\_error("Coordinates do not create rectangle");

}

Point Rectangle::Centre() const

{

Point o;

o.x = (p[0].x + p[1].x + p[2].x + p[3].x) / 4;

o.y = (p[0].y + p[1].y + p[2].y + p[3].y) / 4;

return o;

}

double Rectangle::Area() const

{

return sqrt( distance(p[0], p[1]) \* distance(p[1], p[2]) );

}

std::ostream& Rectangle::print(std::ostream& os) const

{

os << "Rectangle: " << p[0] << "; " << p[1] << "; " << p[2] << "; " << p[3] << "; Centre = " << centre << "; Area = " << area;

return os;

}

//Defining square

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

{

std::cout << "Input point coordinates clockwise or counter clockwise\nA: ";

is >> p[0].x >> p[0].y;

std::cout <<"B: ";

is >> p[1].x >> p[1].y;

std::cout <<"C: ";

is >> p[2].x >> p[2].y;

std::cout <<"D: ";

is >> p[3].x >> p[3].y;

centre = Centre();

area = Area();

if (!IsRectangle(p) || !IsSquare(p) ) throw std::runtime\_error("Coordinates do not create square");

}

Point Square::Centre() const

{

Point o;

o.x = (p[0].x + p[1].x + p[2].x + p[3].x) / 4;

o.y = (p[0].y + p[1].y + p[2].y + p[3].y) / 4;

return o;

}

double Square::Area() const

{

return sqrt( distance(p[0], p[1]) \* distance(p[1], p[2]) );

}

std::ostream& Square::print(std::ostream& os) const

{

os << "Square: " << p[0] << "; " << p[1] << "; " << p[2] << "; " << p[3] << "; Centre = " << centre << "; Area = " << area;

return os;

}

* **main.cpp**

/\*

Разработать классы согласно варианту задания, классы должны наследоваться от базового класса Figure. Фигуры являются фигурами вращения.

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 геометрического 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;

Ефимов Александр

М8О-201Б-18

\*/

#include <vector>

#include <string>

#include <exception>

#include "Shape.h"

using namespace std;

struct list\_empty : public std::exception

{

const char\* what() const throw ()

{

return "The list is currently empty";

}

};

void PrintVec(const vector<Shape\* >& l)

{

int num = 1;

for (auto &e : l)

{

cout << num << ". " << \*e << endl;

++num;

}

}

double TotalArea(const vector<Shape\* >& l)

{

double res = 0;

for (auto &e : l)

{

res += e->GetArea();

}

return res;

}

int main()

{

/\*

Вариант 1:

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 геометрического 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;

Ефимов Александр Владимирович

M8O-201Б

\*/

#include <vector>

#include <string>

#include <exception>

#include "Shape.h"

using namespace std;

struct list\_empty : public std::exception

{

const char\* what() const throw ()

{

return "The list is currently empty";

}

};

void PrintVec(const vector<Shape\* >& l)

{

int num = 1;

for (auto &e : l)

{

cout << num << ". " << \*e << endl;

++num;

}

}

double TotalArea(const vector<Shape\* >& l)

{

double res = 0;

for (auto &e : l)

{

res += e->GetArea();

}

return res;

}

int main()

{

vector<Shape\* > ShapeVec;

int k = 1, num;

cout

<< "1. Add shape\n"

<< "2. Print all shapes\n"

<< "3. Print total area of all shapes\n"

<< "4. Print this menu\n"

<< "5. Delete shape\n"

<< "6. Delete all shapes\n"

<< "0. Exit\n"

<< flush;

do

{

cout << "---------------\nMenu choice: ";

try

{

cin >> k;

if (!cin) throw runtime\_error("Failed to get input");

switch (k)

{

case 1 :

cout << "1. Triangle\n"

<< "2. Rectangle\n"

<< "3. Square\n"

<< "Any other number returns to menu\n"

<< "Pick the type of shape: ";

cin >> k;

if (k >= 1 && k <= 3)

{

switch (k)

{

case 1:

ShapeVec.push\_back(new Triangle(cin));

break;

case 2:

ShapeVec.push\_back(new Rectangle(cin));

break;

case 3:

ShapeVec.push\_back(new Square(cin));

break;

}

}

k = 1;

break;

case 2 :

if (ShapeVec.empty()) throw list\_empty();

PrintVec(ShapeVec);

break;

case 3 :

if (ShapeVec.empty()) throw list\_empty();

cout << "The total area of all shapes is " << TotalArea(ShapeVec) << endl;

break;

case 4 :

cout

<< "1. Add shape\n"

<< "2. Print all shapes\n"

<< "3. Print total area of all shapes\n"

<< "4. Print this menu\n"

<< "5. Delete shape\n"

<< "6. Delete all shapes\n"

<< "0. Exit\n"

<< endl;

break;

case 5 :

if (ShapeVec.empty()) throw list\_empty();

cout << "Number of the shape: ";

cin >> num;

if (num < 1 || num > ShapeVec.size()) throw runtime\_error("Incorrect index");

delete ShapeVec[--num];

ShapeVec.erase(ShapeVec.begin() + num);

break;

case 6 :

if (ShapeVec.empty()) throw list\_empty();

for (auto &e : ShapeVec)

{ delete e; }

ShapeVec.erase(ShapeVec.begin(), ShapeVec.end());

cout << "Vector is now empty" << endl;

break;

case 0 :

break;

}

}

catch (list\_empty& err)

{

cout << err.what() << endl;

}

catch (runtime\_error& err)

{

cerr << err.what() << endl;

}

catch (...)

{

cerr << "Exception thrown, but undefined. Please open an issue at GitHub with steps to reproduce ." << endl;

}

} while (k);

return 0;

}

1. Вывод

Полиморфизм значительно упрощает задачу за счет сохранения места и утилизации схожих функций между подклассами путем передачи и каста указателей на эти подклассы

1. Список литературы
2. Статьи по компьютерным наукам [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/> (дата обращения: 19.09.2019).
3. Статьи по компьютерным наукам [Электронный ресурс]. URL:   
   <https://www.tutorialspoint.com/cplusplus/index.htm> (дата обращения: 19.09.2019)