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 C++

Студент: Суханов Егор Алексеевич
Группа: 80-206
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

Вариант задания: 21. Разработать классы для ромба, 5-угольника и 6-угольника, 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;
- 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.

Задачу можно разбить на следующие подзадачи:

1. Класс Figure:
   a. Виртуальные методы.
2. Классы потомков:
   a. Ввод фигуры;
   b. Вывод геометрического центра фигуры;
   c. Вывод координат вершин фигуры;
   d. Вычисление площади фигуры.
3. Работа с вектором фигур:
   a. Ввод фигур;
   b. Вызов для массива функции фигуры;
   c. Вычисление общей площади фигур в массиве;
   d. Удалять из массива фигуру по индексу.
4. Тестирование программы;
5. Коммит в github репозиторий.

2. Описание программы

GITHUB репозиторий: https://github.com/Reterer/oop_exercise_03

Программа состоит из следующих компонент:

- Интерактивная обработка команд. Включает в себя работу с вектором figure и вводом-выводом. Ввод всех фигур стандартизирован: <название фигуры> <координата центра фигуры> <координата первой вершины>. При необходимости, остальные вершины могут быть выведены из этой информации.
- Абстрактный класс Figure для работы с различными типами фигур. Определяет метод для вывода центра фигуры. А еще объявляет 2 виртуальные функции для вычисления площади и вывода координат вершин. Содержит два защищенных поля: центр фигуры и одну из ее вершин.
- Дочерние классы Figure: ромб, пятиугольник и шестиугольник. Переопределить виртуальные методы. Так как алгоритм работы этих методов одинаковый для правильных n-угольных фигур, то я решил использовать шаблонные функции, чтобы не дублировать код. Однако, поскольку нужно продимостирорать наследование классов, шаблонные функции обернуты в методы соответствующих классов.
- Класс Point для работы с точками и векторами. Были перегружены операции ввода вывода, сложения и вычитания, умножение на скаляр и скалярное умножение векторов.

Работа итоговой программы начинается с функции main, где инициализируется вектор фигур и запускается цикл для обработки команд, пока поток ввода не закрыт.

Обработка команд представляет из себя простой ввод имени команды и вызов соответствующей функции.

В самой функции осуществляется ввод и проверка дополнительных аргументов и сама обработка команды.

Список команд:
help -- выводит справку о программе

exit -- завершает работу программы
print -- выводит вектор фигур (выполняет их методы)
calc -- вычисляет сумму площадей
remove <id> -- удаляет элемент с индексом id
add <type> <vertex> -- добавляет указанную фигуру

3. Набор тестов

test_01.txt -- проверяет работоспособность ввода и команд

```
add r 0 0 1 1
add p 0 0 1 1
add h 0 0 1 1
print
calc
remove 0
print
remove 1
remove 0
exit
help
```

Программа должна вывести такой же текст:

```
0 - ромб
Координаты центра: (0,0)
Координаты: (1,1) (-1,1) (-1,-1) (1,-1)
Площадь: 4

1 - пятиугольник
Координаты центра: (0,0)
Координаты: (1,1) (-0.64204,1.26007) (-1.3968,-0.221232) (-0.221232,-1.3968) (1.26007,-0.64204)
Площадь: 4.75528

2 - шестиугольник
Координаты центра: (0,0)
Координаты: (1,1) (-0.366025,1.36603) (-1.36603,0.366025) (-1,-1) (0.366025,-1.36603) (1.36603,-0.366025)
Площадь: 5.19615
```

Общая площадь фигур: 13.9514

0 - пятиугольник
Координаты центра: (0,0)
Координаты: (1,1) (-0.64204,1.26007) (-1.3968,-0.221232) (-0.221232,-1.3968) (1.26007,-0.64204)
Площадь: 4.75528

1 - шестиугольник
Координаты центра: (0,0)
Координаты: (1,1) (-0.366025,1.36603) (-1.36603,0.366025) (-1,-1) (0.366025,-1.36603) (1.36603,-0.366025)
Площадь: 5.19615

test_02.txt проверка корректности работы функции вывода вершин и площади фигуры

add r 0 0 1 1
add r 0 0 1 0
add r 1 1 2 3
print

Программа должна вывести следующее сообщение:

0 - ромб
Координаты центра: (0,0)
Координаты: (1,1) (-1,1) (-1,-1) (1,-1)
Площадь: 4

1 - ромб
Координаты центра: (0,0)
Координаты: (1,0) (6.12323e-17,1) (-1,1.22465e-16) (-1.83697e-16,-1)
Площадь: 2

2 - ромб
Координаты центра: (1,1)
Координаты: (2,3) (-1,2) (-4.44089e-16,-1) (3,-6.66134e-16)
Площадь: 10

test_03.txt -- проверка обработки неверного ввода

kfekj
add ekfj 1 2 1 2
add r 0 0 0 0
add r 1ikewjfw2 4iefj3 4edifj 0
add r 1 1 0 0
remove -1
remove 1

программа должна вывести:

Введена неизвестная команда. Чтобы вывести справку, введите "help"
Ошибка в вводе координат фигуры.
Ошибка в вводе координат фигуры.
Ошибка в вводе координат фигуры.
индекс элемента не может быть меньше нуля или больше размера вектора фигур.
индекс элемента не может быть меньше нуля или больше размера вектора фигур.

4. Результаты выполнения тестов

>oop_exercise_03.exe < ..\..\..\tests\test_01.txt
>>>>0 - ромб
Координаты центра: (0,0)
Координаты: (1,1) (-1,1) (-1,-1) (1,-1)
Площадь: 4

1 - пятиугольник
Координаты центра: (0,0)
Координаты: (1,1) (-0.64204,1.26007) (-1.3968,-0.221232) (-0.221232,-1.3968) (1.26007,-0.64204)
Площадь: 4.75528

2 - шестиугольник
Координаты центра: (0,0)
Координаты: (1,1) (-0.366025,1.36603) (-1.36603,0.366025) (-1,-1) (0.366025,-1.36603) (1.36603,-0.366025)
Площадь: 5.19615

>Общая площадь фигур: 13.9514
>>0 - пятиугольник
Координаты центра: (0,0)
Координаты: (1,1) (-0.64204,1.26007) (-1.3968,-0.221232) (-0.221232,-1.3968) (1.26007,-0.64204)
Площадь: 4.75528

1 - шестиугольник
Координаты центра: (0,0)
Координаты: (1,1) (-0.366025,1.36603) (-1.36603,0.366025) (-1,-1) (0.366025,-1.36603) (1.36603,-0.366025)
Площадь: 5.19615

>>>
>oop_exercise_03.exe < ..\..\..\tests\test_02.txt
>>>>0 - ромб
Координаты центра: (0,0)
Координаты: (1,1) (-1,1) (-1,-1) (1,-1)
Площадь: 4

1 - ромб
Координаты центра: (0,0)
Координаты: (1,0) (6.12323e-17,1) (-1,1.22465e-16) (-1.83697e-16,-1)
Площадь: 2

2 - ромб
Координаты центра: (1,1)
Координаты: (2,3) (-1,2) (-4.44089e-16,-1) (3,-6.66134e-16)
Площадь: 10

>
>oop_exercise_03.exe < ..\..\..\tests\test_03.txt
>Введена неизвестная команда. Чтобы вывести справку, введите "help"
>Ошибка в вводе координат фигуры.
>Ошибка в вводе координат фигуры.
>Ошибка в вводе координат фигуры.
>>индекс элемента не может быть меньше нуля или больше размера

вектора фигур.
>индекс элемента не может быть меньше нуля или больше размера вектора фигур.
>

5. Листинг программы

программа состоит из следующих файлов:
main.cpp:

```
/*
  Лабораторная работа: 3
  Вариант: 21
  Группа: М8О-206Б-19
  Автор: Суханов Егор Алексеевич

  Задание:
      Разработать классы для  ромба, 5-угольника и 6-угольника,
      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 вывода std::cout координат вершин
фигуры;
            Вычисление площади фигуры.
      Создать программу, которая позволяет:
            Вводить из стандартного ввода std::cin фигуры;
            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.

*/
#include <iostream>
#include <vector>
#include <string>
#include "figure.hpp"

void clear()
{
      std::cin.clear();
      std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

void help()
```

```
{
	std::cout <<
		"команды:\n"
		"    help        -- выводит этот текст\n"
		"    exit        -- завершает работу программы\n"
		"    print       -- выводит вектор фигур (выполняет их методы)\
n"
		"    calc        -- вычисляет сумму площадей\n"
		"    remove <id> -- удаляет элемент с индексом id\n"
		"    add <type> <center> <vertex> -- добавляет указанную
фигуру\n"
		<< std::endl;
}

std::string getNameFigure(Figure* figure)
{
	if (Rhombus* p = dynamic_cast<Rhombus*>(figure))
		return "ромб";
	else if (Pentagon* p = dynamic_cast<Pentagon*>(figure))
		return "пятиугольник";
	else if (Hexagon* p = dynamic_cast<Hexagon*>(figure))
		return "шестиугольник";
	else
		return "неизвестно";
}

void print(const std::vector<Figure*>& figures)
{
	for (int i = 0; i < figures.size(); ++i)
	{
		std::cout << i << "\t - " << getNameFigure(figures[i])<< '\n';
		std::cout << "Координаты центра: " << figures[i]->GetCenter()
<< '\n';
		figures[i]->PrintCords();
		std::cout << "Площадь: " << figures[i]->CalcArea() << "\n\n";
	}
}

void calc(const std::vector<Figure*>& figures)
{
	double sumOfAreas = 0;
	for (Figure* figure : figures)
	{
		sumOfAreas += figure->CalcArea();
	}
	std::cout << "Общая площадь фигур: " << sumOfAreas << std::endl;
}

void remove(std::vector<Figure*>& figures)
{
	int id;
	if (!(std::cin >> id))
	{
```

```
			std::cout << "Ошибка ввода." << std::endl;
			clear();
			return;
		}

		if (id < 0 || id >= figures.size())
		{
			std::cout << "индекс элемента не может быть меньше нуля или
больше размера вектора фигур." << std::endl;
			return;
		}
		delete figures[id];
		figures.erase(figures.begin() + id);
}

void add(std::vector<Figure*>& figures)
{
		std::string name;
		if (!(std::cin >> name))
		{
			std::cout << "Ошибка ввода имени." << std::endl;
			clear();
			return;
		}

		Point center, vertex;
		if (!(std::cin >> center >> vertex) || center == vertex)
		{
			std::cout << "Ошибка в вводе координат фигуры." << std::endl;
			clear();
			return;
		}

		Figure* figure;
		if (name == "r")
			figure = new Rhombus(center, vertex);
		else if (name == "p")
			figure = new Pentagon(center, vertex);
		else if (name == "h")
			figure = new Hexagon(center, vertex);
		else
		{
			std::cout << "Ошибка в названии фигуры." << std::endl;
			clear();
			return;
		}

		figures.push_back(figure);
}

int main()
{
		setlocale(LC_ALL, "russian");
```

```
	std::vector<Figure*> figures(0);

	std::string cmd;
	std::cout << '>';
	while (std::cin >> cmd)
	{
		if (cmd == "help")
			help();
		else if (cmd == "exit")
			break;
		else if (cmd == "print")
			print(figures);
		else if (cmd == "calc")
			calc(figures);
		else if (cmd == "remove")
			remove(figures);
		else if (cmd == "add")
			add(figures);
		else
		{
			std::cout << "Введена неизвестная команда. Чтобы вывести
справку, введите \"help\"" << std::endl;
			clear();
		}

		std::cout << '>';
	}

	return 0;
}
```

point.hpp:

```
#pragma once
#include <iostream>

struct Point {
	double x;
	double y;

	Point rotate(double radians);

	friend bool operator== (const Point a, const Point b);
	friend bool operator!= (const Point a, const Point b);

	friend Point operator+ (const Point a, const Point b);
	friend Point operator- (const Point a, const Point b);

	friend double operator* (const Point a, const Point b);
	friend Point operator* (const Point a, double b);

	friend std::ostream& operator<< (std::ostream& out, const Point& p);
	friend std::istream& operator>> (std::istream& in, Point& p);
```

```
};
```

point.cpp:

```
#include "point.hpp"

bool operator== (const Point a, const Point b)
{
	return a.x == b.x && a.y == b.y;
}
bool operator!= (const Point a, const Point b)
{
	return a.x != b.x || a.y != b.y;
}

Point operator+ (const Point a, const Point b)
{
	return { a.x + b.x, a.y + b.y };
}

Point operator- (const Point a, const Point b)
{
	return { a.x - b.x, a.y - b.y };
}

double operator*(const Point a, const Point b)
{
	return a.x * b.x + a.y * b.y;
}

Point operator*(const Point a, double b)
{
	return Point{ a.x * b, a.y * b };
}

std::istream& operator>>(std::istream& in, Point& p)
{
	double x, y;
	in >> x >> y;
	p = Point{ x, y };
	return in;
}

std::ostream& operator<< (std::ostream& out, const Point& p)
{
	out << '(' << p.x << ',' << p.y << ')';
	return out;
}

Point Point::rotate(double radians) {
	Point rotated;
	rotated.x = x * std::cos(radians) - y * std::sin(radians);
	rotated.y = x * std::sin(radians) + y * std::cos(radians);
```

```
	return rotated;
}
```

figure.hpp:

```
#pragma once
#include "point.hpp"

class Figure {
public:
	Figure(Point center, Point vertex);
	Point GetCenter();

	virtual void PrintCords() = 0;
	virtual double CalcArea() = 0;

protected:
	Point center;
	Point vertex;
};

class Rhombus : public Figure {
public:
	Rhombus(Point center, Point vertex);
	virtual void PrintCords() override;
	virtual double CalcArea() override;
};

class Pentagon : public Figure {
public:
	Pentagon(Point center, Point vertex);
	virtual void PrintCords() override;
	virtual double CalcArea() override;
};

class Hexagon : public Figure {
public:
	Hexagon(Point center, Point vertex);
	virtual void PrintCords() override;
	virtual double CalcArea() override;
};
```

figure.cpp:

```
#include <stdexcept>
#include <iostream>
#include <cmath>
#include "figure.hpp"

const double PI = 3.141592653589793;

Figure::Figure(Point center, Point vertex)
{
```

```
	if (center == vertex)
		throw std::invalid_argument("center can be eq vertex");

	this->center = center;
	this->vertex = vertex;
}

Point Figure::GetCenter()
{
	return this->center;
}

template <int VERTEX_COUNT>
void _printCords(const Point center, const Point vertex)
{
	const double ANGLE = 2 * PI / VERTEX_COUNT; // угол между двумя соседними вершинами и центром
	std::cout << "Координаты: " << vertex;

	Point vec = vertex - center;
	for (int i = 2; i <= VERTEX_COUNT; ++i)
	{
		vec = vec.rotate(ANGLE);
		std::cout << ' ' << center + vec;
	}
	std::cout << std::endl;
}

template <int VERTEX_COUNT>
double _calcArea(const Point center, const Point vertex)
{
	Point vecRadius = vertex - center; // Вектор-радиус описанной окружности
	double sqRadius = vecRadius * vecRadius;		// Квадрат радуиса описанной окружности
	return VERTEX_COUNT / 2. * sqRadius * std::sin(2 * PI / VERTEX_COUNT);
}

Rhombus::Rhombus(Point center, Point vertex)
	: Figure(center, vertex)
{}

void Rhombus::PrintCords()
{
	_printCords<4>(center, vertex);
}

double Rhombus::CalcArea()
{
	Point vecDiag = this->center - this->vertex;
	return 2 * (vecDiag * vecDiag);
}
```

```
Pentagon::Pentagon(Point center, Point vertex)
	: Figure(center, vertex)
{}

void Pentagon::PrintCords()
{
	_printCords<5>(center, vertex);
}

double Pentagon::CalcArea()
{
	return _calcArea<5>(center, vertex);
}


Hexagon::Hexagon(Point center, Point vertex)
	: Figure(center, vertex)
{}

void Hexagon::PrintCords()
{
	_printCords<6>(center, vertex);
}

double Hexagon::CalcArea()
{
	return _calcArea<6>(center, vertex);
}
```

6. Выводы

В данной ЛР я узнал о модификаторах доступа и вспомнил школьный курс геометрии. Научился использовать виртуальные функции. Но при этом стоит учитывать, что их использование немного увеличивает объем используемой памяти и немного замедляет вызов функции. Но, с другой стороны, это позволяет делать абстрактные классы и интерфейсы для классов, которые обладают похожими свойствами. Например можно сделать контейнер с элементами типа “указатель на абстрактный класс”, что позволит добавлять туда указатель на экземпляр любого дочернего класса, и обращаться к методам и атрибутам абстрактного класса.

7. Список литературы

1. Страуструп, Бьёрн. Язык программирования C++. Краткий курс, 2-е изд. : Пер. с англ. - СПб.: ООО "Диалектика", 2019. - 320 с.: ил. - Парал. тит. англ.

2. Упрощение кода с помощью if constexpr[Электронный ресурс]. URL: https://habr.com/ru/post/351970/ (дата обращения 22.10.2020).

3.