Московский авиационный институт
(Национальный исследовательский университет)

Институт: 8 «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 3

Тема: Механизмы наследования в C++

Студент: Моисеенков Илья Павлович
Группа: 80-208
Преподаватель: Чернышов Л.Н.
Дата: 26.10.2020
Оценка: 13/15

Москва, 2020

1. Постановка задачи

a. Ознакомиться с теоретическим материалом по механизмам наследования и полиморфизма в языке C++.
b. Разработать классы “Rhombus”, “Pentagon”, “Hexagon” для работы с ромбами, пятиугольниками и шестиугольниками соответственно. Классы должны наследоваться от базового класса “Figure”. Все фигуры равносторонние и являются фигурами вращения. Все классы должны поддерживать набор общих методов: вычисление геометрического центра фигуры, вывод в стандартный поток вывода координат вершин фигуры, вычисление площади фигуры.
c. Создать программу, которая позволяет вводить фигуры из стандартного потока ввода, сохранять введенные фигуры в динамическом массиве, выводить для всего массива общие функции, вычислять общую площадь фигур, удалять из массива фигуру по индексу.
d. Настроить CMake файл для сборки программы.
e. Подготовить наборы тестовых данных.
f. Загрузить файлы лабораторной работы в репозиторий GitHub.
g. Подготовить отчёт по лабораторной работе.

2. Описание программы

Программа имеет многофайловую структуру. Описание всех классов выделены в отдельные файлы.

Point.h / Point.cpp

Point - класс “Точка на плоскости”. Включает в себя описание и реализацию следующих методов и атрибутов:

- Координата по оси X и по оси Y. Выражается типом double.
- Конструктор. Принимает два аргумента - координаты x и y. В случае отсутствия аргументов используются аргументы по умолчанию - нулевые координаты.
- Метод length. Вычисляет длину отрезка между двумя точками. Аргумент – вторая точка отрезка. Вычисляется как корень из суммы квадратов соответствующих координат.

Для класса Point переопределены операторы ввода и вывода. Ввод происходит путём считывания двух координат. Выводится точка в виде “(x;y)”.

Figure.h / Figure.cpp

Figure - базовый абстрактный класс для создания остальных фигур. Включает в себя описание и реализацию следующих методов и атрибутов:

- Конструктор и деструктор, сообщающие о создании и удалении фигуры соответственно. Деструктор объявлен виртуальным.
- Чисто виртуальный метод square для вычисления площади.
- Метод center, возвращающий геометрический центр геометрической фигуры (тип Point). Вычисляется как среднее арифметическое всех координат фигуры.
- Атрибут name, хранящий название фигуры.
- Вектор вершин vertices. Изначально вектор пустой.

- Защищённая функция check_vertices, которая проверяет массив vertices и определяет, является ли введённая фигура равносторонней. Также определяет момент, когда пользователь ввёл координаты одной и той же вершины.
- Дружественные функции - переопределение операторов ввода и вывода. Вывод производится в формате “Фигура { набор вершин }”. Для ввода функции нужно ввести координаты всех вершин по/против часовой стрелке.

Rhombus.h/.cpp, Hexagon.h/.cpp, Pentagon.h/.cpp

Rhombus - класс “Ромб”, Hexagon - класс “Шестиугольник”, Pentagon - класс “пятиугольник”. Все эти классы являются наследниками класса Figure. В этих классах реализованы следующие методы:

- Конструкторы, записывающие название фигуры и количество ее вершин в соответствующие атрибуты.
- Деструкторы, сообщающие об удалении фигуры.
- Переопределяются функции для вычисления площадей фигур.

main.cpp

В функции main создаётся вектор указателей на Figure (Figure - абстрактный класс, поэтому можно хранить только указатели на него). У пользователя запрашивается количество фигур, которые он хочет ввести. Затем в цикле обрабатывается ввод фигур: пользователь должен ввести тип фигуры (“r”, “p” или “h”) и координаты всех вершин по или против часовой стрелки. Если пользователь ввёл координаты не равносторонней фигуры, то программа попросит его повторить ввод.

После ввода в консоль будет выведен список всех введенных фигур, их координат, площадей и геометрическими центрами. Также будет рассчитана суммарная площадь всех введенных фигур.

Пользователю будет предложено удалить некоторое количество фигур из вектора. Для этого он должен ввести количество фигур для удаления и их индексы в векторе. Предусмотрена проверка на корректность введенных индексов. Удаление происходит при помощи стандартного метода класса vector. После удаления всех фигур будет выведен список оставшихся фигур.

В завершение работы программы будут удалены остальные фигуры (т.к. они создавались динамически в куче).

3. Набор тестов

Первое число - количество фигур. Для каждой фигуры указывается её тип (“r” для ромба, “p” – для пятиугольника, “h” – для шестиугольника) и координаты вершин.

После ввода фигур вводится количество тех фигур, которые нужно удалить, и их индексы.

Тест 1

```
5 (количество фигур)
h (шестиугольник)
0.86 -0.5
0.86 0.5
0 1
-0.86 0.5
-0.86 -0.5
0 -1
r (ромб)
0 0
0 1
1 1
1 0
r (ромб)
2 2
4 7
6 2
4 -3
p (пятиугольник)
28.09 0.62
30 2.01
31.91 0.62
31.18 -1.63
28.82 -1.63
r (ромб)
0 0
-3 2
-6 0
-3 -2
2 (количество фигур для удаления)
4 (id фигур для удаления)
0
```

Тест 2

```
3 (количество фигур)
r (ромб)
0 6
1 2
0 -2
-1 2
h (шестиугольник)
1.73 0
1.73 1
0.86 1.5
0 1
0 0
0.86 -0.5
r (ромб)
2 2
2 7
-1 3
-1 -2
3 (количество фигур для удаления)
0 (id фигур для удаления)
0
0
```

4. Результаты выполнения тестов

Тест 1

```
Enter the amount of figures you want to enter:
5
Enter the type of figure (r - rhombus, p - pentagon, h - hexagon)
h
Creating figure...
Created Hexagon!
Enter vertices of this figure
0.86 -0.5
0.86 0.5
0 1
-0.86 0.5
-0.86 -0.5
0 -1
Enter the type of figure (r - rhombus, p - pentagon, h - hexagon)
r
Creating figure...
Created Rhombus!
Enter vertices of this figure
0 0
0 1
1 1
1 0
Enter the type of figure (r - rhombus, p - pentagon, h - hexagon)
r
Creating figure...
Created Rhombus!
Enter vertices of this figure
2 2
4 7
6 2
4 -3
Enter the type of figure (r - rhombus, p - pentagon, h - hexagon)
p
Creating figure...
Created Pentagon!
Enter vertices of this figure
28.09 0.62
30 2.01
31.91 0.62
31.18 -1.63
28.82 -1.63
Enter the type of figure (r - rhombus, p - pentagon, h - hexagon)
r
Creating figure...
Created Rhombus!
Enter vertices of this figure
0 0
-3 2
-6 0
-3 -2

List of figures:
Hexagon { (0.86;-0.5) (0.86;0.5) (0;1) (-0.86;0.5) (-0.86;-0.5) (0;-1) }
Square: 2.6
Center: (0;0)

Rhombus { (0;0) (0;1) (1;1) (1;0) }
Square: 1
```

```
Center: (0.5;0.5)

Rhombus { (2;2) (4;7) (6;2) (4;-3) }
Square: 20
Center: (4;2)

Pentagon { (28.1;0.62) (30;2.01) (31.9;0.62) (31.2;-1.63) (28.8;-1.63) }
Square: 9.6
Center: (30;-0.002)

Rhombus { (0;0) (-3;2) (-6;0) (-3;-2) }
Square: 12
Center: (-3;0)

Total square of figures is 45.2

Enter the amount of figures you want to delete:
2

Enter the id of figure you want to delete (from 0 to 4):
4
Deleting Rhombus...
Figure successfully deleted!
Enter the id of figure you want to delete (from 0 to 3):
0
Deleting Hexagon...
Figure successfully deleted!

List of remaining figures:
Rhombus { (0;0) (0;1) (1;1) (1;0) }
Square: 1
Center: (0.5;0.5)

Rhombus { (2;2) (4;7) (6;2) (4;-3) }
Square: 20
Center: (4;2)

Pentagon { (28.1;0.62) (30;2.01) (31.9;0.62) (31.2;-1.63) (28.8;-1.63) }
Square: 9.6
Center: (30;-0.002)


Deleting remained figures:
Deleting Rhombus...
Figure successfully deleted!
Deleting Rhombus...
Figure successfully deleted!
Deleting Pentagon...
Figure successfully deleted!
```

Тест 2

```
Enter the amount of figures you want to enter:
3
Enter the type of figure (r - rhombus, p - pentagon, h - hexagon)
r
Creating figure...
Created Rhombus!
Enter vertices of this figure
0 6
```

```
1 2
0 -2
-1 2
Enter the type of figure (r - rhombus, p - pentagon, h - hexagon)
h
Creating figure...
Created Hexagon!
Enter vertices of this figure
1.73 0
1.73 1
0.86 1.5
0 1
0 0
0.86 -0.5
Enter the type of figure (r - rhombus, p - pentagon, h - hexagon)
r
Creating figure...
Created Rhombus!
Enter vertices of this figure
2 2
2 7
-1 3
-1 -2

List of figures:
Rhombus { (0;6) (1;2) (0;-2) (-1;2) }
Square: 8
Center: (0;2)

Hexagon { (1.73;0) (1.73;1) (0.866;1.5) (0;1) (0;0) (0.866;-0.5) }
Square: 2.6
Center: (0.86;0.5)

Rhombus { (2;2) (2;7) (-1;3) (-1;-2) }
Square: 15
Center: (0.5;2.5)

Total square of figures is 25.6

Enter the amount of figures you want to delete:
3

Enter the id of figure you want to delete (from 0 to 2):
0
Deleting Rhombus...
Figure successfully deleted!
Enter the id of figure you want to delete (from 0 to 1):
0
Deleting Hexagon...
Figure successfully deleted!
Enter the id of figure you want to delete (from 0 to 0):
0
Deleting Rhombus...
Figure successfully deleted!

List of remaining figures:
```

5. Листинг программы

Point.h

```
#ifndef OOP_LAB3_POINT_H
```

```
#define OOP_LAB3_POINT_H

#include <iostream>
#include <cmath>

class Point {
public:
    Point(double x1 = 0.0, double y1 = 0.0);

    double length(Point &p) const;

    double x;
    double y;
};

std::ostream &operator<<(std::ostream &out, Point point);

std::istream &operator>>(std::istream &in, Point &point);


#endif //OOP_LAB3_POINT_H
```

Point.cpp

```
#include "Point.h"


Point::Point(double x1, double y1) : x(x1), y(y1) {}

double Point::length(Point &p) const {
    return sqrt(pow(p.x - x, 2) + pow(p.y - y, 2));
}

std::ostream &operator<<(std::ostream &out, Point point) {
    std::cout.precision(3);
    out << "(" << point.x << ";" << point.y << ")";
    return out;
}

std::istream &operator>>(std::istream &in, Point &point) {
    in >> point.x >> point.y;
    return in;
}
```

Figure.h

```
#ifndef OOP_LAB3_FIGURE_H
#define OOP_LAB3_FIGURE_H

#include <iostream>
#include <vector>
#include "Point.h"

class Figure {
public:
    Figure();

    virtual ~Figure();

    Point center();

    virtual double square() = 0;

    friend std::ostream &operator<<(std::ostream &out, Figure &figure);

    friend std::istream &operator>>(std::istream &in, Figure &figure);

protected:
    std::vector<Point> vertices;
    std::string name;
```

```
    bool check_vertices();
};


#endif //OOP_LAB3_FIGURE_H
```

Figure.cpp

```
#include "Figure.h"


Figure::Figure() {
    std::cout << "Creating figure..." << std::endl;
    name = "Unknown";
}

Figure::~Figure() {
    std::cout << "Figure successfully deleted!" << std::endl;
}

Point Figure::center() {
    double x_mid = 0, y_mid = 0;
    for (Point &point : vertices) {
        x_mid += point.x;
        y_mid += point.y;
    }
    return Point(x_mid / vertices.size(), y_mid / vertices.size());
}

bool Figure::check_vertices() {
    double figure_length = vertices[0].length(vertices[vertices.size() - 1]);
    for (int i = 0; i < vertices.size() - 1; ++i) {
        double cur_length = vertices[i].length(vertices[i + 1]);
        if (std::abs(cur_length - figure_length) >= 1e-9) {
            std::cout << "Figure must have equal sides. Try again!" << std::endl;
            return false;
        }
        if (figure_length == 0) {
            std::cout << "Points should be different. Try again!" << std::endl;
            return false;
        }
    }
    return true;
}

std::ostream &operator<<(std::ostream &out, Figure &figure) {
    out << figure.name << " { ";
    for (Point &point : figure.vertices) {
        out << point << " ";
    }
    out << "}";
    return out;
}

std::istream &operator>>(std::istream &in, Figure &figure) {
    do {
        for (auto &vertex : figure.vertices) {
            in >> vertex;
        }
    } while (!figure.check_vertices());
    return in;
}
```

Rhombus.h

```
#ifndef OOP_LAB3_RHOMBUS_H
#define OOP_LAB3_RHOMBUS_H
```

```
#include <vector>
#include "Figure.h"

class Rhombus : public Figure {
public:
    Rhombus();

    ~Rhombus() override;

    double square() override;
};


#endif //OOP_LAB3_RHOMBUS_H
```

Rhombus.cpp

```
#include "Rhombus.h"

Rhombus::Rhombus() {
    vertices.resize(4);
    name = "Rhombus";
    std::cout << "Created Rhombus!" << std::endl;
}

Rhombus::~Rhombus() {
    std::cout << "Deleting Rhombus..." << std::endl;
}

double Rhombus::square() {
    return (vertices[0].length(vertices[2]) * vertices[1].length(vertices[3])) / 2;
}
```

Pentagon.h

```
#ifndef OOP_LAB3_PENTAGON_H
#define OOP_LAB3_PENTAGON_H


#include "Figure.h"

class Pentagon : public Figure {
public:
    Pentagon();

    ~Pentagon() override;

    double square() override;
};

#endif //OOP_LAB3_PENTAGON_H
```

Pentagon.cpp

```
#include "Pentagon.h"

Pentagon::Pentagon() {
    vertices.resize(5);
    name = "Pentagon";
    std::cout << "Created Pentagon!" << std::endl;
}

Pentagon::~Pentagon() {
    std::cout << "Deleting Pentagon..." << std::endl;
}

double Pentagon::square() {
    double side = vertices[0].length(vertices[1]);
    return sqrt(25 + 10 * sqrt(5)) * pow(side, 2) / 4;
```

```
}
```

Hexagon.h

```
#ifndef OOP_LAB3_HEXAGON_H
#define OOP_LAB3_HEXAGON_H


#include "Figure.h"

class Hexagon : public Figure {
public:
    Hexagon();

    ~Hexagon() override;

    double square() override;
};


#endif //OOP_LAB3_HEXAGON_H
```

Hexagon.cpp

```
#include "Hexagon.h"

Hexagon::Hexagon() {
    vertices.resize(6);
    name = "Hexagon";
    std::cout << "Created Hexagon!" << std::endl;
}

Hexagon::~Hexagon() {
    std::cout << "Deleting Hexagon..." << std::endl;
}

double Hexagon::square() {
    double side = vertices[0].length(vertices[1]);
    return pow(side, 2) * 3 * sqrt(3) / 2;
}
```

main.cpp

```
/* Моисеенков Илья М8О-208Б-19
 *
 * github: mosikk
 *
 * Создать классы для геометрических фигур - Ромба, Пятиугольника и Шестиугольника.
Все классы должны быть
 * наследниками класса Figure. В классах реализованы методы определения
геометрического центра фигуры и
 * вычисления площадей. Переопределены функции для ввода и вывода фигур.
 *
 * В функции main создаётся вектор фигур. Пользователь может ввести любое
количество разных фигур. Затем
 * для каждой фигуры из вектора выводятся ее координаты, площадь и геометрический
центр. Вычисляется суммарная
 * площадь всех введенных фигур.
 *
 * Пользователь может удалить любое количество фигур из массива.
 *
 */

#include <iostream>
#include <vector>
#include "Rhombus.h"
#include "Pentagon.h"
#include "Hexagon.h"
```

```
int main() {
    unsigned int amount;
    std::cout << "Enter the amount of figures you want to enter: " << std::endl;
    std::cin >> amount;

    std::vector<Figure *> figures;
    for (int i = 0; i < amount; ++i) {
        char type;
        do {
            std::cout << "Enter the type of figure (r - rhombus, p - pentagon, h -
hexagon)" << std::endl;
            std::cin >> type;
            if (type != 'r' && type != 'R' && type != 'p' && type != 'P' && type !=
'h' && type != 'H') {
                std::cout << "Incorrect type. Try again." << std::endl;
            }
        } while (type != 'r' && type != 'R' && type != 'p' && type != 'P' && type
!= 'h' && type != 'H');

        if (type == 'R' || type == 'r') {
            auto *R = new Rhombus;
            std::cout << "Enter vertices of this figure" << std::endl;
            std::cin >> *R;
            figures.push_back(R);
        } else if (type == 'P' || type == 'p') {
            auto *P = new Pentagon;
            std::cout << "Enter vertices of this figure" << std::endl;
            std::cin >> *P;
            figures.push_back(P);
        } else if (type == 'H' || type == 'h') {
            auto *H = new Hexagon;
            std::cout << "Enter vertices of this figure" << std::endl;
            std::cin >> *H;
            figures.push_back(H);
        }
    }
    std::cout << std::endl;

    double total_square = 0;
    std::cout << "List of figures:" << std::endl;
    for (auto &figure : figures) {
        std::cout << *figure << std::endl;
        double cur_square = figure->square();
        total_square += cur_square;
        std::cout << "Square: " << cur_square << std::endl;
        std::cout << "Center: " << figure->center() << std::endl;
        std::cout << std::endl;
    }
    std::cout << "Total square of figures is " << total_square << std::endl;
    std::cout << std::endl;

    unsigned int amount_delete;
    do {
        std::cout << "Enter the amount of figures you want to delete: " <<
std::endl;
        std::cin >> amount_delete;
        if (amount_delete > figures.size()) {
            std::cout << "Size of vector is less than your number. Try again" <<
std::endl;
        }
    } while (amount_delete > figures.size());
    std::cout << std::endl;

    for (int i = 0; i < amount_delete; ++i) {
        int id;
        do {
            std::cout << "Enter the id of figure you want to delete "
                         "(from " << 0 << " to " << figures.size() - 1 << "): " <<
```

```
std::endl;
            std::cin >> id;
            if (id < 0 || id >= figures.size()) {
                std::cout << "Wrong id. Try again" << std::endl;
            }
        } while (id < 0 || id >= figures.size());
        delete figures[id];
        figures.erase(figures.begin() + id);
    }
    std::cout << std::endl;

    std::cout << "List of remaining figures:" << std::endl;
    for (auto &figure : figures) {
        std::cout << *figure << std::endl;
        double cur_square = figure->square();
        total_square += cur_square;
        std::cout << "Square: " << cur_square << std::endl;
        std::cout << "Center: " << figure->center() << std::endl;
        std::cout << std::endl;
    }
    std::cout << std::endl;

    std::cout << "Deleting remained figures:" << std::endl;
    for (auto &figure : figures) {
        delete figure;
    }

}
```

6. Выводы

В данной лабораторной работе были изучены механизмы наследования и полиморфизма в языке C++. Наследование – это мощный инструмент для построения иерархии классов, в которой дочерние классы наследуют все функции родительского. Но из-за механизма полиморфизма дочерние классы могут изменить некоторые возможности родителя под себя.

При выполнении этой работы можно легко убедиться в том, что в совокупности полиморфизм и наследование позволяют упрощать проектирование классов и работу с ними.

Список используемых источников

1. Руководство по языку C++ [Электронный ресурс]. URL: https://www.cplusplus.com/ (дата обращения 17.10.2020).
2. Построение классов C++ [Электронный ресурс]. URL: http://cppstudio.com/post/439/ (дата обращения 17.10.2020).
3. Полиморфизм в C++ [Электронный ресурс]. URL: https://ravesli.com/urok-163-virtualnye-funktsii-i-polimorfizm/ (дата обращения 18.10.2020).
4. Наследование классов C++ [Электронный ресурс]. URL: http://cppstudio.com/post/10103/ (дата обращения 18.10.2020).