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: Шубин Григорий Сергеевич
Группа: 80-208
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

Репозиторий Github: https://github.com/Garigoriy/oop_exercise_03

1.Постановка задачи. Вариант 10.

Разработать классы фигур: квадрат, прямоугольник, трапеция. 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.

2. Описание программы

- Класс Point “Точка на плоскости”:
  - - конструктор по умолчанию (нулевые координаты);
  - - конструктор с параметрами;
  - - деконструктор;
  - - поля класса x,y типа double, отвечающие за соответственно координаты точки по оси X и Y;
  - - перегруженный оператор вывода.
- Класс Figure “Фигура на плоскости”:
  - - виртуальные методы вычисления площади фигуры,

геометрического центра и печать фигуры.

- Классы Square, Rectangle, Trapezoid соответственно “Квадрат”, “Прямоугольник”, “Трапеция” — наследники класса Figure :
    - - конструкторы, задающие соответствующее количество вершин для данной фигуры и выводящие сообщение о том, что фигура успешно создана;
    - - деконструкторы, сообщающие об удалении фигуры;
    - - Переопределенные функции для вычисления площади фигур.
- Функция main() - главная функция программа.
    - У пользователя запрашивается количество фигур. Затем в цикле обрабатывается ввод фигур: пользователь вводит тип фигуры (“s”, “r” или “t”) и координаты всех вершин по или против часовой стрелки. Ввод координат не проверяется. Фигуры хранятся в векторе figures типа данных Figure *(указатели на введённые фигуры).
    - После ввода в консоль будет выведен список всех введенных фигур, их координат, площадей и геометрических центров. Также будет рассчитана суммарная площадь всех введенных фигур.
    - Пользователь может также удалить любое 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.

3. Набор тестов и результаты выполнения

Тестовые данные состоят из следующего набор аргументов: первое число отвечает за количество фигур в векторе figures, далее идет тип фигуры (тип “r” для прямоугольника, “t” для трапеции, “s” для квадрата), после идут сами координаты вершин выбранной фигуры. После ввода фигур вводится количество фигур для удаления и их индексы.

Таблица 1. Тестовые данные.

Тест 1	Тест 2
3 r 0 0 0 2 4 2 4 0 t 0 0 1 2 3 2 4 0 s 0 0 0 3 3 3 3 0 1 1	4 s 0 0 0 1 1 1 1 0 s 4 0 0 0 0 4 4 4 r 0 0 0 3 5 3 5 0 t 0 0 2 2 5 2 7 0 4 0 0 1 0

Результаты выполнения тестов:

-Тест 1

```
Enter the amount of figures you want to enter:
3
Enter the type of figure (r - rectangle, t - trapezoid, s - square)
r
Creating figure...
Rectangle is created!
Enter vertices of this figure
```

```
0 0
0 2
4 2
4 0
Enter the type of figure (r - rectangle, t - trapezoid, s - square)
t
Creating figure...
Trapezoid is created!
Enter vertices of this figure
0 0
1 2
3 2
4 0
Enter the type of figure (r - rectangle, t - trapezoid, s - square)
s
Creating figure...
Square is created!
Enter vertices of this figure
0 0
0 3
3 3
3 0

List of figures:
Rectangle{(0;0); (0;2); (4;2); (4;0); }
Square: 8
Center: (2;1);

Trapezoid{(0;0); (1;2); (3;2); (4;0); }
Square: 6
Center: (2;1);

Square{(0;0); (0;3); (3;3); (3;0); }
Square: 9
Center: (1.5;1.5);

Total square of figures is 23

Enter the amount of figures you want to delete:
1

Enter the id of figure you want to delete (from 0 to 2):
1
Deleting figure...

List of remaining figures:
Rectangle{(0;0); (0;2); (4;2); (4;0); }
Square: 8
Center: (2;1);

Square{(0;0); (0;3); (3;3); (3;0); }
Square: 9
Center: (1.5;1.5);


Deleting remained figures:
Deleting figure...
Deleting figure...
```

-Тест 2

```
Enter the amount of figures you want to enter:
4
Enter the type of figure (r - rectangle, t - trapezoid, s - square)
s
Creating figure...
Square is created!
Enter vertices of this figure
0 0
0 1
1 1
1 0
Enter the type of figure (r - rectangle, t - trapezoid, s - square)
s
Creating figure...
Square is created!
Enter vertices of this figure
4 0
0 0
0 4
4 4
Enter the type of figure (r - rectangle, t - trapezoid, s - square)
r
Creating figure...
Rectangle is created!
Enter vertices of this figure
0 0
0 3
5 3
5 0
Enter the type of figure (r - rectangle, t - trapezoid, s - square)
t
Creating figure...
Trapezoid is created!
Enter vertices of this figure
0 0
2 2
5 2
7 0

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

Square{(4;0); (0;0); (0;4); (4;4); }
Square: 16
Center: (2;2);

Rectangle{(0;0); (0;3); (5;3); (5;0); }
Square: 15
Center: (2.5;1.5);

Trapezoid{(0;0); (2;2); (5;2); (7;0); }
```

```
Square: 10
Center: (3.5;1);

Total square of figures is 42

Enter the amount of figures you want to delete:
4

Enter the id of figure you want to delete (from 0 to 3):
0
Deleting figure...
Enter the id of figure you want to delete (from 0 to 2):
0
Deleting figure...
Enter the id of figure you want to delete (from 0 to 1):
1
Deleting figure...
Enter the id of figure you want to delete (from 0 to 0):
0
Deleting figure...

List of remaining figures:
```

4. Листинг программы

```
/*
Выполнил Шубин Григорий М8О-208Б-19
Разработать классы фигур: квадрат, прямоугольник, трапеция. 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.
*/
#include <iostream>
#include <vector>
#include <string>
#include <cmath>

using namespace std;

class Point{
public:
    Point(){
        x = 0.0;
        y = 0.0;
```

```
    }
    Point(double x1, double y1) : x(x1), y(y1) {}
    double length(Point &p){
        return sqrt(pow(p.x - x,2) + pow(p.y - y,2));
    }
    friend istream &operator>>(istream &in, Point &p);
    friend ostream &operator<<(ostream &out, Point p);
    double x;
    double y;
};

istream &operator>>(istream &in, Point &p){
    in >> p.x;
    in >> p.y;
    return in;
}

ostream &operator<<(ostream &out, Point p){
    out << "(" << p.x << ";" << p.y << ");";
    return out;
}

class Figure{
public:
    Figure(){
        cout << "Creating figure..." << endl;
        name = "Unknown..";
    }
    ~Figure(){
        cout << "Deleting figure..." << endl;
    }
    vector<Point> vertices;
    string name;
    Point center(){
        double x_mid = 0, y_mid = 0;
        for (Point &p : vertices){
            x_mid += p.x;
            y_mid += p.y;
        }
        return Point(x_mid / vertices.size(),y_mid / vertices.size());
    }
    bool check_vertices(){
        double cur_len = vertices[0].length(vertices[vertices.size() - 1]);
        for (int i = 0; i < vertices.size() - 1; ++i){
            double fig_len = vertices[i].length(vertices[i + 1]);
            if (abs(fig_len - cur_len) >= 2e+1){
                cout << "Figure must have equal sides. Try again!" << endl;
                return false;
            }
            if (fig_len == 0){
                cout << "Points should be different. Try again!" << endl;
                return false;
            }
        }
        return true;
    }
    virtual double square() = 0;
    friend istream& operator>>(istream &in, Figure &figure);
```

```
    friend ostream& operator<<(istream &out, Figure &figure);
};

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

    return in;
}

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

class Rectangle : public Figure{
public:
    Rectangle(){
        vertices.resize(4);
        name = "Rectangle";
        cout << "Rectangle is created!" << endl;
    }
    ~Rectangle(){
        cout << "Deleting Rectangle..." << endl << "Rectangle is successfully
deleted!" << endl;
    }
    double square(){
                                return    vertices[0].length(vertices[1])    *
vertices[1].length(vertices[2]);
    }
};

class Square : public Figure{
public:
    Square(){
        vertices.resize(4);
        name = "Square";
        cout << "Square is created!" << endl;
    }
    ~Square(){
           cout << "Deleting Square..." << endl << "Square is successfully
deleted!" << endl;
    }
    double square(){
        return pow(vertices[0].length(vertices[1]),2);
    }
};

class Trapezoid : public Figure{
public:
    Trapezoid(){
```

```
        vertices.resize(4);
        name = "Trapezoid";
        cout << "Trapezoid is created!" << endl;
    }
    ~Trapezoid(){
        cout << "Deleting Trapezoid..." << endl << "Trapezoid is successfully
deleted!" << endl;
    }
    double square(){
        double a = vertices[1].length(vertices[2]);
        double b = vertices[0].length(vertices[3]);
        double c = (vertices[0].length(vertices[1]));
        double p = (a + b + 2 * c) / 2;
        return sqrt((p - a) * (p - b) * pow((p - c),2));
    }
};

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

    vector<Figure *> figures;
    for (int i = 0; i < amount; ++i) {
        char type;
        do {
             cout << "Enter the type of figure (r - rectangle, t - trapezoid,
s - square)" << endl;
            cin >> type;
              if (type != 'r' && type != 'R' && type != 't' && type != 'T' &&
type != 's' && type != 'S') {
                  cout << "Incorrect type. Try again." << endl;
            }
         } while (type != 'r' && type != 'R' && type != 't' && type != 'T' &&
type != 's' && type != 'S');

        if (type == 'S' || type == 's') {
            auto *S = new Square;
            cout << "Enter vertices of this figure" << endl;
            cin >> *S;
            figures.push_back(S);
        } else if (type == 'T' || type == 't') {
            auto *T = new Trapezoid;
            cout << "Enter vertices of this figure" << endl;
            cin >> *T;
            figures.push_back(T);
        } else if (type == 'R' || type == 'r') {
            auto *R = new Rectangle;
            cout << "Enter vertices of this figure" << endl;
            cin >> *R;
            figures.push_back(R);
        }
    }
    cout << endl;

    double total_square = 0;
    cout << "List of figures:" << endl;
    for (auto &figure : figures) {
```

```
        cout << *figure << endl;
        double cur_square = figure->square();
        total_square += cur_square;
        cout << "Square: " << cur_square << endl;
        cout << "Center: " << figure->center() << endl;
        cout << endl;
    }
    cout << "Total square of figures is " << total_square << endl;
    cout << endl;

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

    for (int i = 0; i < amount_delete; ++i) {
        int id;
        do {
            cout << "Enter the id of figure you want to delete "
                        "(from " << 0 << " to " << figures.size() - 1 << "):
" << endl;
            cin >> id;
            if (id < 0 || id >= figures.size()) {
                cout << "Wrong id. Try again" << endl;
            }
        } while (id < 0 || id >= figures.size());
        delete figures[id];
        figures.erase(figures.begin() + id);
    }
    cout << endl;

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

    if (!figures.empty()) {
        cout << "Deleting remained figures:" << endl;
        for (auto &figure : figures) {
            delete figure;
        }
    }
    return 0;
}
```

5. Выводы

Данная лабораторная работа направлена на изучение принципов наследования в C++. Были приобретены навыки работы с дочерними классами, а также с переопределением функций. Механизм наследования позволяет дочерним классам наследовать, т.е. использовать все поля и методы родительского класса, а механизм полиморфизма позволяет изменять нужные возможности родителя под себя.

Список литературы

1. Механизм наследования в C++ [Электронный ресурс]. URL: https://ravesli.com/urok-154-bazovoe-nasledovanie-v-c/ (дата обращения 23.12.2020).
2. Переопределение функций в C++ [Электронный ресурс]. URL: https://ravesli.com/urok-159-vyzov-i-pereopredelenie-metodov-roditelskogo-klassa/ (дата обращения 23.12.2020).
3. Классы в C++ [Электронный ресурс]. URL: http://cppstudio.com/post/439/ (дата обращения 23.12.2020).
4. Конструкторы класса C++ [Электронный ресурс]. URL: https://metanit.com/cpp/tutorial/5.2.php (дата обращения 23.12.2020).
5. Вектор в C++ [Электронный ресурс]. URL: https://code-live.ru/post/cpp-vector/ (дата обращения 23.12.2020).