**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 С++

Студент: Шавандрин Федор Михайлович

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи**

a. Ознакомиться с теоретическим материалом по механизмам наследования и полиморфизма в языке С++.

b. Разработать классы “Rectangle”, “Trapezoid”, “Rhombus”. 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.

1. **Описание программы**
2. Класс Point “Точка на плоскости”:

- конструктор по умолчанию (нулевые координаты);

- конструктор с параметрами;

- метод length, который считает длину отрезка, заданного двумя; точками, как квадратный корень и суммы квадратов соответствующих координат.

- поля класса x,y типа double, отвечающие за соответственно координаты точки по оси X и Y;

- перегруженные операторы ввода и вывода.

1. Класс Figure “Фигура на плоскости”:

- конструктор по умолчанию, который выводит сообщение о том, что фигура создана;

- деконструктор, который выводит сообщение о том, что фигура удалена;

- вектор вершин verticles, который изначально пуст;

- строка name, отвечающая за название фигуры;

- метод center, который вычисляет геометрический центр фигуры, находя среднее арифметическое соответствующих координат;

- булевая функция check\_verticles, которая проверяет, является ли фигура равносторонней, а также выводит сообщение о том, если пользователь ввёл несколько раз одни и те же координаты;

- перегруженные операторы ввода и вывода. Ввод вершин фигуры осуществляется по/против часовой стрелке. Вывод вершин производится в формате “Фигура {набор вершин}”.

1. Классы Rectangle, Trapezoid, Rhombus соответственно “Прямоугольник”, “Трапеция”, “Ромб” - 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.

1. Функция main() - главная функция программа.

У пользователя запрашивается количество фигур. Затем в цикле обрабатывается ввод фигур: пользователь вводит тип фигуры (“r”, “t” или “h”) и координаты всех вершин по или против часовой стрелки. Если пользователь ввёл координаты не равносторонней фигуры или координаты одной и той же вершины, то программа попросит его повторить ввод. 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 (т.к. они создавались динамически в куче).

1. **Набор тестов**

Тестовые данные состоят из следующего набор аргументов: первое число отвечает за количество фигур в векторе figures, далее идет тип фигуры в зависимости от того, для какой фигуры пользователь хочет ввести координаты вершин (тип “r” для прямоугольника, “t” для трапеции, “h” для ромба) и дальше идут сами координаты вершин выбранной фигуры.

Таблица 1. Тестовые данные.

|  |  |
| --- | --- |
| Тест 1 | Тест 2 |
| 2  r  0 0  0 2  2 2  2 0  h  0 1  1 2  2 1  1 0 | 3  t  0 0  1 1  4 1  5 0  t  0 1  2 2  4 2  5 1  r  0 0  0 1  1 1  10 |

1. **Результаты выполнения тестов**

Enter the amount of figures you want to enter: 2 Enter the type of figure (r - rectangle, t - trapezoid, h - rhombus) r Creating figure… Rectangle is created! Enter vertices of this figure 0 0 0 2 2 2 2 0 Enter the type of figure (r - rectangle, t - trapezoid, h - rhombus) h Creating figure… Rhombus is created! Enter vertices of this figure 0 1 1 2 2 1 1 0 List of figures: Rectangle{(0;0); (0;2); (2;2); (2;0); } Square: 4 Center: (1;1); Rhombus{(0;1); (1;2); (2;1); (1;0); } Square: 2 Center: (1;1); Total square of figures is 6 Enter the amount of figures you want to delete: 1 Enter the id of figure you want to delete (from 0 to 1): 0 Deleting figure… Rectangle is successfully deleted! List of remaining figures: Rhombus{(0;1); (1;2); (2;1); (1;0); } Square: 2 Center: (1;1); Deleting remained figures: Deleting figure… Rhombus is successfully deleted! ------------------------------------------------------------------------------------------ Enter the amount of figures you want to enter: 3 Enter the type of figure (r - rectangle, t - trapezoid, h - rhombus) t Creating figure… Trapezoid is created! Enter vertices of this figure 0 0 1 1 4 1 5 0 Enter the type of figure (r - rectangle, t - trapezoid, h - rhombus) t Creating figure… Trapezoid is created! Enter vertices of this figure 0 1 2 2 4 2 5 1 Enter the type of figure (r - rectangle, t - trapezoid, h - rhombus) r Creating figure... Rectangle is created! Enter vertices of this figure 0 0 0 1 1 1 1 0 List of figures: Trapezoid{(0;0); (1;1); (4;1); (5;0); } Square: 4 Center: (2.5;0.5); Trapezoid{(0;1); (2;2); (4;2); (5;1); } Square: 5.80409 Center: (2.75;1.5); Rectangle{(0;0); (0;1); (1;1); (1;0); } Square: 1 Center: (0.5;0.5); Total square of figures is 10.8041

Enter the amount of figures you want to delete: 2 Enter the id of figure you want to delete (from 0 to 2): 1 Deleting figure… Trapezoid is successfully deleted! Enter the id of figure you want to delete (from 0 to 1): 0 Deleting figure… Trapezoid is successfully deleted! List of remaining figures: Rectangle{(0;0); (0;1); (1;1); (1;0); } Square: 1 Center: (0.5;0.5); Deleting remained figures: Deleting figure… Rectangle is successfully deleted!

1. **Листинг программы**

/\*

Created by Шавандрин Фёдор

Ссылка на репозиторий https://github.com/Adi6e/oop\_exercise\_02

Реализовать класс Complex для работы с комплексными числами. Реализовать арифметические операции и операции сравнения по действительной части.

Операции сложения, вычитания, умножения, деления, сравнения на равенство должны быть выполнены в виде перегрузки операторов.

Необходимо реализовать пользовательский литерал для работы с константами типа Complex.

\*/

#include <iostream>

#include <vector>

#include <string>

#include <cmath>

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

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

double x;

double y;

};

std::istream &operator>>(std::istream &in, Point &p){

in >> p.x;

in >> p.y;

return in;

}

std::ostream &operator<<(std::ostream &out, Point p){

out << "(" << p.x << ";" << p.y << ");";

return out;

}

class Figure{

public:

Figure(){

std::cout << "Creating figure..." << std::endl;

name = "Unknown..";

}

~Figure(){

std::cout << "Deleting figure..." << std::endl;

}

std::vector<Point> verticles;

std::string name;

Point center(){

double x\_mid = 0, y\_mid = 0;

for (Point &p : verticles){

x\_mid += p.x;

y\_mid += p.y;

}

return Point(x\_mid / verticles.size(),y\_mid / verticles.size());

}

bool check\_verticles(){

double cur\_len = verticles[0].length(verticles[verticles.size() - 1]);

for (int i = 0; i < verticles.size() - 1; ++i){

double fig\_len = verticles[i].length(verticles[i + 1]);

if (abs(fig\_len - cur\_len) >= 2e+1){

std::cout << "Figure must have equal sides. Try again!" << std::endl;

return false;

}

if (fig\_len == 0){

std::cout << "Points should be different. Try again!" << std::endl;

return false;

}

}

return true;

}

virtual double square() = 0;

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

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

};

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

do{

for (auto &vertex : figure.verticles){

in >> vertex;

}

}while (!figure.check\_verticles());

return in;

}

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

out << figure.name << "{";

for (Point &p : figure.verticles){

out << p << " ";

}

out << "}";

return out;

}

class Rectangle : public Figure{

public:

Rectangle(){

verticles.resize(4);

name = "Rectangle";

std::cout << "Rectangle is created!" << std::endl;

}

~Rectangle(){

std::cout << "Deleting Rectangle..." << std::endl << "Rectangle is successfully deleted!" << std::endl;

}

double square(){

return pow(verticles[0].length(verticles[1]),2);

}

};

class Trapezoid : public Figure{

public:

Trapezoid(){

verticles.resize(4);

name = "Trapezoid";

std::cout << "Trapezoid is created!" << std::endl;

}

~Trapezoid(){

std::cout << "Deleting Trapezoid..." << std::endl << "Trapezoid is successfully deleted!" << std::endl;

}

double square(){

double a = verticles[1].length(verticles[2]);

double b = verticles[0].length(verticles[3]);

double c = (verticles[0].length(verticles[1]));

double p = (a + b + 2 \* c) / 2;

return sqrt((p - a) \* (p - b) \* pow((p - c),2));

}

};

class Rhombus : public Figure{

public:

Rhombus(){

verticles.resize(4);

name = "Rhombus";

std::cout << "Rhombus is created!" << std::endl;

}

~Rhombus(){

std::cout << "Deleting Rhombus..." << std::endl << "Rhombus is successfully deleted!" << std::endl;

}

double square(){

return (verticles[0].length(verticles[2]) \* verticles[1].length(verticles[3])) / 2;

}

};

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 - rectangle, t - trapezoid, h - rhombus)" << std::endl;

std::cin >> type;

if (type != 'r' && type != 'R' && type != 't' && type != 'T' && type != 'h' && type != 'H') {

std::cout << "Incorrect type. Try again." << std::endl;

}

} while (type != 'r' && type != 'R' && type != 't' && type != 'T' && type != 'h' && type != 'H');

if (type == 'R' || type == 'r') {

auto \*R = new Rectangle;

std::cout << "Enter vertices of this figure" << std::endl;

std::cin >> \*R;

figures.push\_back(R);

} else if (type == 'T' || type == 't') {

auto \*T = new Trapezoid;

std::cout << "Enter vertices of this figure" << std::endl;

std::cin >> \*T;

figures.push\_back(T);

} else if (type == 'H' || type == 'h') {

auto \*RH = new Rhombus;

std::cout << "Enter vertices of this figure" << std::endl;

std::cin >> \*RH;

figures.push\_back(RH);

}

}

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;

if (!figures.empty()) {

std::cout << "Deleting remained figures:" << std::endl;

for (auto &figure : figures) {

delete figure;

}

}

return 0;

}

1. **Выводы**

Данная лабораторная работа направлена на изучение принципов наследования в С++. 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.

1. **Список литературы**
2. Механизм наследования в С++ [Электронный ресурс]. URL: <https://ravesli.com/urok-154-bazovoe-nasledovanie-v-c/> (дата обращения 20.10.2020).
3. Переопределение функций в С++ [Электронный ресурс]. URL: <https://ravesli.com/urok-159-vyzov-i-pereopredelenie-metodov-roditelskogo-klassa/> (дата обращения 19.10.2020).
4. Классы в С++ [Электронный ресурс]. URL: <http://cppstudio.com/post/439/> (дата обращения 20.10.2020).
5. Конструкторы класса С++ [Электронный ресурс]. URL: <https://metanit.com/cpp/tutorial/5.2.php> (дата обращения 20.10.2020).
6. Вектор в С++ [Электронный ресурс]. URL: <https://code-live.ru/post/cpp-vector/> (дата обращения 20.10.2020).