**Лабораторная работа №2**

**Наследование в С++**

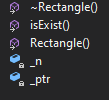
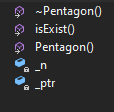
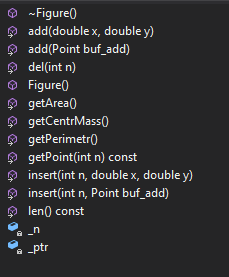
**Цель работы**

Ознакомление с отношением наследования и иерархией классов.

**Формулировка задачи**

Создать класс Figure, объект которого задается координатами точек, c вычислением площади, центра тяжести и периметра. На его основе реализовать классы Rectangle (прямоугольник) и Pentagon (пятиугольник).

**Диаграммы классов**



**Перечень сигнатур методов**

Метод add (): добавление вершины.

Метод insert (): вставка вершины.// пока закомментированна так как не стоит в задаче.

Метод getPoint (): получение координат точек и линий.

Метод getArea (): получение площади.

Метод getPerimetr (): получение периметра.

Метод getCentrMass (): получение центра масс.

Метод len (): количества вершин.

Метод del (): удаление точки.

Метод isExist (): проверка на соответствию типу.

**Листинг программы на языке С++**

**Lab2\_oop.cpp**

#include <iostream>

#include "Console.h"

int main()

{

Console View;

View.init();

return 0;

}

**Figure.h**

#include <stdexcept>

#include "Point.h"

#include <math.h>

class Figure {

public:

Figure() { \_n = 0; \_ptr = new Point[\_n]; };

~Figure() { delete [] \_ptr; };

void add(double x, double y);

void add(Point buf\_add);

/\*

void insert(int n, double x, double y);

void insert(int n, Point buf\_add);

\*/

double getPerimetr();

double getArea();

bool del(int n);

Point getCentrMass();

Point getPoint(int n)const;

int len()const;

protected:

int \_n = 0;

Point \*\_ptr;

};

**Figure.cpp**

#include "Figure.h"

void Figure::add(double x, double y)

{

Point\* buf = new Point[\_n + 1];

buf[\_n].X = x;

buf[\_n].Y = y;

for (int i = 0; i < \_n; i++)

buf[i] = \_ptr[i];

\_n++;

delete[] \_ptr;

\_ptr = buf;

};

void Figure::add(Point buf\_add)

{

Point\* buf = new Point[\_n + 1];

buf[\_n] = buf\_add;

for (int i = 0; i < \_n; i++)

buf[i] = \_ptr[i];

\_n++;

delete[] \_ptr;

\_ptr = buf;

};

/\*

void Figure::insert(int n, double x, double y)

{

if (n > \_n)

n = \_n-1;

if (n < 0)

n = 0;

Point\* buf = new Point[\_n + 1];

buf[n].X = x;

buf[n].Y = y;

for (int i = 0; i < n; i++)

buf[i] = \_ptr[i];

for (int i = n + 1; i < \_n + 1; i++)

buf[i] = \_ptr[i - 1];

\_n++;

delete[] \_ptr;

\_ptr = buf;

};

void Figure::insert(int n, Point buf\_add)

{

if (n > \_n)

n = \_n-1;

if (n < 0)

n = 0;

Point\* buf = new Point[\_n + 1];

buf[n] = buf\_add;

for (int i = 0; i < n; i++)

buf[i] = \_ptr[i];

for (int i = n + 1; i < \_n + 1; i++)

buf[i] = \_ptr[i - 1];

\_n++;

delete[] \_ptr;

\_ptr = buf;

};

\*/

double Figure::getPerimetr()

{

double sum = 0;

if(\_n>0)

{

if (\_n > 0)

{

for (int i = 0; i < \_n - 1; i++)

{

sum += sqrt((\_ptr[i + 1].X - \_ptr[i].X) \* (\_ptr[i + 1].X - \_ptr[i].X) + (\_ptr[i + 1].Y - \_ptr[i].Y) \* (\_ptr[i + 1].Y - \_ptr[i].Y));

}

sum += sqrt((\_ptr[0].X - \_ptr[\_n - 1].X) \* (\_ptr[0].X - \_ptr[\_n - 1].X) + (\_ptr[0].Y - \_ptr[\_n - 1].Y) \* (\_ptr[0].Y - \_ptr[\_n - 1].Y));

}

}

return sum;

};

double Figure::getArea()

{

double sum1 = 0, sum2 = 0;

if(\_n > 0)

{

sum1 += \_ptr[\_n - 1].X \* \_ptr[0].Y;

sum2 += \_ptr[\_n - 1].Y \* \_ptr[0].X;

for (int i = 0; i < \_n - 1; i++)

{

sum1 += \_ptr[i].X \* \_ptr[i + 1].Y;

sum2 += \_ptr[i].Y \* \_ptr[i + 1].X;

}

sum1 -= sum2;

if (sum1 < 0)

sum1 \*= -1;

}

return sum1/2;

};

Point Figure::getCentrMass()

{

double sum1 = 0, sum2 = 0;

Point f;

for (int i = 0; i < \_n; i++)

{

sum1 += \_ptr[i].X;

sum2 += \_ptr[i].Y;

}

sum1 /= \_n;

sum2 /= \_n;

f.X = sum1;

f.Y = sum2;

return f;

};

bool Figure::del(int n)

{

if (n < 0)

n += \_n;

if (n >= 0 && n < \_n)

{

int i = n;

for (i = n; i < \_n-1 ; i++)

{

\_ptr[i] = \_ptr[i + 1];

}

\_n--;

Point\* buf = new Point[\_n];

for (int i = 0; i < \_n; i++)

{

buf[i] = \_ptr[i];

}

delete[] \_ptr;

\_ptr = buf;

return true;

}

return false;

};

int Figure::len()const

{

return \_n;

}

Point Figure::getPoint(int n)const

{

if (n < 0)

{

n = \_n + n;

}

try {

if (n >= \_n || n < 0)

{

throw std::out\_of\_range("out of array");

}

}

catch (std::out\_of\_range & e)

{

while (n < 0)

n += \_n;

while (n >= \_n)

n -= \_n;

}

if (n < \_n)

return \_ptr[n];

};

**Rectangle.h**

class Rectangle:public Figure {

public:

Rectangle();

~Rectangle();

bool isExist();

private:

/\*int \_n = 0;

Point\* \_ptr;\*/

};

**Rectangle.cpp**

#include "Rectangle.h"

bool Rectangle::isExist() {

if (\_n == 4)

{

if (\_ptr[0].X == \_ptr[1].X && \_ptr[0].Y == \_ptr[3].Y && \_ptr[2].Y == \_ptr[1].Y && \_ptr[2].X == \_ptr[3].X)

return true;

}

return false;

};

Rectangle::Rectangle() {

\_n = 0;

\_ptr = new Point[\_n];

};

Rectangle::~Rectangle() {

Figure::~Figure();

//delete[] \_ptr;

};

**Pentagon.h**

#include "Figure.h"

class Pentagon :public Figure {

public:

Pentagon();

~Pentagon();

bool isExist();

private:

/\*int \_n = 0;

Point\* \_ptr;\*/

};

**Pentagon.cpp**

#include "Pentagon.h"

Pentagon::Pentagon() {

\_n = 0;

\_ptr = new Point[\_n];

};

Pentagon::~Pentagon() {

Figure::~Figure();

//delete[] \_ptr;

};

bool Pentagon::isExist() {

if (\_n == 5)

{

for (int i = 0; i < \_n; i++)

for (int j = i + 1; j < \_n; j++)

if (\_ptr[j].X == \_ptr[i].X && \_ptr[j].Y == \_ptr[i].Y)

return false;

return true;

}

return false;

};

**Console.h**

#pragma once

#include "Figure.h"

#include "Pentagon.h"

#include "Rectangle.h"

class Console {

public:

void init();

int viewPentagon();

int viewRectangle();

int viewFigure();

void pprint(Point buf);

private:

};

**Console.cpp**

#include "Console.h"

#include <iostream>

void Console::init() {

std::cout << "search: 1 - Figure 2 - Pentagon 3 - Rectangle\n";

int search;

std::cin >> search;

/\*switch (search)

{

case 1:viewFigure(); break;

case 2:viewPentagon(); break;

case 3:viewRectangle(); break;

default:break;

}\*/

if (search == 1)

viewFigure();

else if (search == 2)

viewPentagon();

else if (search == 3)

viewFigure();

// std::cout << "ok";

};

int Console::viewPentagon()

{

Pentagon Fig;

double x, y;

int n = 5;

for (int i = 0; i < n; i++)

{

std::cout << "enter vertex:";

std::cin >> x >> y;

Fig.add(x, y);

}

std::cout << " is Pentagon " << Fig.isExist() << " Area=" << Fig.getArea() << " Perimetr=" << Fig.getPerimetr() << " CentrMass=";

pprint(Fig.getCentrMass());

std::cout << "\nPrint figure\n";

for (int i = 0; i < Fig.len(); i++)

pprint(Fig.getPoint(i));

std::cout << "\nenter vertex for delete\n";

std::cin >> n;

Fig.del(n);

for (int i = 0; i < Fig.len(); i++)

pprint(Fig.getPoint(i));

Fig.~Pentagon();

return 0;

};

int Console::viewRectangle()

{

Rectangle Fig;

double x, y;

int n = 4;

for (int i = 0; i < n; i++)

{

std::cout << "enter vertex:";

std::cin >> x >> y;

Fig.add(x, y);

}

std::cout <<" is Rectangle "<< Fig.isExist() << " Area=" << Fig.getArea() << " Perimetr=" << Fig.getPerimetr() << " CentrMass=";

pprint(Fig.getCentrMass());

std::cout << "\nPrint figure\n";

for (int i = 0; i < Fig.len(); i++)

pprint(Fig.getPoint(i));

std::cout << "\nenter vertex for delete\n";

std::cin >> n;

Fig.del(n);

for (int i = 0; i < Fig.len(); i++)

pprint(Fig.getPoint(i));

Fig.~Rectangle();

return 0;

};

int Console::viewFigure()

{

Figure Fig;

double x, y;

int n;

std::cout << "enter the number of vertexes:";

std::cin >> n;

if (n > 0)

{

for (int i = 0; i < n; i++)

{

std::cout << "enter vertex:";

std::cin >> x >> y;

Fig.add(x, y);

}

std::cout << "Area=" << Fig.getArea() << " Perimetr=" << Fig.getPerimetr() << " CentrMass=";

pprint(Fig.getCentrMass());

std::cout << "\nPrint figure\n";

for(int i = 0; i < Fig.len();i++)

pprint(Fig.getPoint(i));

std::cout << "\nenter vertex for delete\n";

std::cin >> n;

Fig.del(n);

for (int i = 0; i < Fig.len(); i++)

pprint(Fig.getPoint(i));

Fig.~Figure();

}

else

std::cout << "there is no such figure";

return 0;

};

void Console::pprint(Point buf)

{

std::cout << buf.X << " " << buf.Y << "\n";

};

**Заключение**

В процессе выполнения лабораторной работы №2, мы ознакомились с отношением наследования и иерархией классов. Изучили, что такое виртуальные методы, как происходит наследование и какая разница между модификаторами доступа и наследования.

Автор Преподаватель: Тарасов

Шутенко Остап Русланович Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись \_\_\_\_\_\_\_ Дата: \_\_.\_\_.2020