**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 С++

Студент: Савченко Илья Владимирович

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

Вариант 24

1. Постановка задачи  
   Разработать классы согласно варианту задания, 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.

1. Описание программы

* **Figure.h**
  + Структура **Point** из двух переменных типа double для хранения и обработки двумерных векторов, поддерживает необходимые операции арифм. действий, а также ввода/вывода, а также скалярные и векторные произведения
  + Класс **Figure**
    - Имеет публичную константу VERT, которая хранит кол-во вершин, перезаписывается наследуемыми классами при инициализации
    - Защищенный поинтер на массив вершин, создается новый при инициализации.
    - Виртуальные функции **center**, **printVertices**, **surface** по условию задачи
    - Защищенный инициатор, принимающий кол-во вершин, а также публичный виртуальный деструктор
  + Класс **Triangle** : Figure
  + Класс **Square** : Figure
  + Класс **Octagon** : Figure
    - Дополнительная приватная переменная, хранящая центр фигуры
* **main.cpp**
  + Функция **SurfaceComb**, которая возвращает общую площадь списка
  + Функция **quickList**, которая выводит короткий список фигур по названиям
  + **main**, консольный интерфейс для заполнения списка фигурами и его обработки

1. Набор тестов

|  |  |
| --- | --- |
| ==> test\_01.txt <==  t  0 0  0 4  s  0 0  5 5  o  0 0  1 3  x | Triangle:  A(0, 0)  B(0, 4)  C(3.4641, 2)  Center = (1.1547, 2)  S = 6.9282  Square:  A(0, 0)  B(0, 5)  C(5, 5)  D(5, 0)  Center = (2.5, 2.5)  S = 25  Octagon:  A(1, 3)  B(2.82843, 1.41421)  C(3, -1)  D(1.41421, -2.82843)  E(-1, -3)  F(-2.82843, -1.41421)  G(-3, 1)  H(-1.41421, 2.82843)  Center = (0, 0)  S = 28.2843  Surface combined  = 60.2125 |
| ==> test\_02.txt <==  t 3 2 7 -3  s 1 2 6 9  o 1 2 7 3  t 0 0 3 4  o 0 0 0 5  s 0 0 6 6  t 0 0 0 0  x | Triangle:  A(3, 2)  B(7, -3)  C(0.669873, -3.9641)  Center = (3.55662, -1.6547)  S = 17.7535  Square:  A(1, 2)  B(0, 8)  C(6, 9)  D(7, 3)  Center = (3.5, 5.5)  S = 37  Octagon:  A(7, 3)  B(5.94975, -1.53553)  C(2, -4)  D(-2.53553, -2.94975)  E(-5, 1)  F(-3.94975, 5.53553)  G(0, 8)  H(4.53553, 6.94975)  Center = (1, 2)  S = 104.652  Triangle:  A(0, 0)  B(3, 4)  C(4.9641, -0.598076)  Center = (2.6547, 1.13397)  S = 10.8253  Octagon:  A(0, 5)  B(3.53553, 3.53553)  C(5, 0)  D(3.53553, -3.53553)  E(0, -5)  F(-3.53553, -3.53553)  G(-5, 0)  H(-3.53553, 3.53553)  Center = (0, 0)  S = 70.7107  Square:  A(0, 0)  B(0, 6)  C(6, 6)  D(6, 0)  Center = (3, 3)  S = 36  Triangle:  A(0, 0)  B(0, 0)  C(0, 0)  Center = (0, 0)  S = 0  Surface combined  = 276.941 |

1. Результаты выполнения тестов  
   Пример удаления фигур из вектора:

# после ввода 7 фигур:  
 0: Triangle

1: Square

2: Octagon

3: Square

4: Triangle

5: Octagon

6: Square

^D to exit

enter <0 to list

Delete figure? 2

-1

0: Triangle

1: Square

2: Square

3: Triangle

4: Octagon

5: Square

5

-1

0: Triangle

1: Square

2: Square

3: Triangle

4: Octagon

8

Size is 5

0

-1

0: Square

1: Square

2: Triangle

3: Octagon

Прерывается процесс удаления командой ctrl-D (конец ввода) или вводом не числа

1. Листинг программы

**==> main.cpp <==**

/\* Савченко Илья Владимирович

\* М8О-208Б-19

\*

\* вариант 24

\* - Треугольник

\* - Квадрат

\* - Восьмиугольник

\*

\* Разработать классы согласно варианту задания,

\* 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<cstdio>

#include<vector>

#include"figure.h"

using namespace std;

double SurfaceComb(vector<Figure\*> &vec){

double toRet(0);

for(Figure \*f : vec)

if(f)

toRet += f->surface();

return toRet;

}

void quickList(vector<Figure\*> &vec){

if(vec.empty()){

cout << "Empty\n";

return;

}

for(unsigned int i=0; i< vec.size(); ++i){

printf("%2u: ", i);

if(!vec[i]){

cout << "null\n";

continue;

}

switch(vec[i]->VERT){

case 3:

cout << "Triangle\n";

continue;

case 4:

cout << "Square\n";

continue;

case 8:

cout << "Octagon\n";

continue;

default:

cout << "?\n";

continue;

}

}

cout << '\n';

}

int main(){

vector<Figure\*> myVec;

while(true){

char t;

Figure \*f;

cout << "Next figure (t,s,o + 2 points, x to exit): ";

cin >> t;

switch(t){

case 't':

f = new Triangle;

cin >> \*static\_cast<Triangle\*>(f);

break;

case 's':

f = new Square;

cin >> \*static\_cast<Square\*>(f);

break;

case 'o':

f = new Octagon;

cin >> \*static\_cast<Octagon\*>(f);

break;

case 'x':

goto FigureInputEnd;

default:

cout << "No such option\n";

f = nullptr;

break;

}

if(f)

myVec.push\_back(f);

}

FigureInputEnd:

cout << '\n';

for(Figure \*f : myVec){

if(!f){

cout << "null\n\n";

continue;

}

f->printVertices();

cout << "Center = "

<< f->center()

<<"\n S = "

<< f->surface()

<< "\n\n";

}

cout << "Surface combined\n = "

<< SurfaceComb(myVec) << "\n\n";

quickList(myVec);

cout << "^D to exit\nenter <0 to list\n";

cout << "Delete figure? ";

int ind;

while(cin >> ind){

if(ind < 0){

quickList(myVec);

continue;

}

if(ind >= myVec.size()){

cout << "Size is "

<< myVec.size()

<< '\n';

continue;

}

delete myVec[ind];

myVec.erase(myVec.begin() + ind);

}

**==> figure.h <==**

#ifndef \_FIGURE\_

#define \_FIGURE\_

#include<iostream>

#include<cmath>

#include <istream>

struct Point{

double x, y;

Point(double x=0, double y=0)

: x(x),

y(y)

{}

static double dot(const Point &a, const Point &b)

{return (a.x\*b.x + a.y\*b.y);}

static double cross(const Point &a, const Point &b)

{return (a.x\*b.y - a.y\*b.x);}

Point& operator+=(const Point &p){

x += p.x;

y += p.y;

return \*this;

}

Point& operator-=(const Point &p){

x -= p.x;

y -= p.y;

return \*this;

}

Point& operator\*=(double v){

x \*= v;

y \*= v;

return \*this;

}

Point operator- () const

{return Point(-x, -y);}

};

inline Point operator\* (const Point &a, double b)

{return Point(a.x\*b, a.y\*b);}

inline Point operator\* (double b, const Point &a)

{return Point(a.x\*b, a.y\*b);}

inline Point operator/ (const Point &a, double b)

{return Point(a.x/b, a.y/b);}

inline Point operator+ (const Point &a, const Point &b)

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

inline Point operator- (const Point &a, const Point &b)

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

inline std::ostream& operator<<(std::ostream& outS, const Point& p){

outS<< '('

<< p.x

<< ", "

<< p.y

<< ')';

return outS;

}

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

inS >> p.x

>> p.y;

return inS;

}

// 24

// 8 triangle square

class Figure{

protected:

Figure(int v) : VERT(v)

{verts = new Point[VERT];}

Point \*verts;

public:

const int VERT;

virtual Point center() const{

Point toRet;

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

toRet += verts[i] / VERT;

return toRet;

}

virtual void printVertices() const{

for(unsigned char i=0; i<VERT; ++i){

std::cout

<< " "

<< (char)('A'+ i)

<< verts[i]

<< '\n';

}

}

virtual double surface() const = 0;

virtual ~Figure()

{delete[] verts;}

};

class Triangle : public Figure{

public:

Triangle() : Figure(3){}

friend std::istream& operator>>(std::istream &inS, Triangle &T){

/\* для треугольника

\* B ---- c

\* \ /

\* \ /

\* A

\* вводятся точки A и B

\* точка C расчитывается как поворот

\* вектора AB на 60 гр по часовой стрелке

\*/

inS >> T.verts[0]

>> T.verts[1];

Point vec = T.verts[1] - T.verts[0];

const double r3 = std::sqrt(3);

Point trVec; // умножение на матрицу поворота 60 гр.

trVec = (Point(1, -r3) \* vec.x) + (Point(r3, 1) \* vec.y);

T.verts[2] = T.verts[0] + (trVec \* 0.5);

return inS;

}

virtual void printVertices() const override{

std::cout << "Triangle:\n";

Figure::printVertices();

}

virtual double surface() const override{

double sur = Point::cross(

verts[1] - verts[0],

verts[2] - verts[0]

);

sur = 0.5 \* std::abs(sur);

return sur;

}

};

class Square : public Figure{

public:

Square() : Figure(4){}

friend std::istream& operator>>(std::istream &inS, Square &S){

/\* для квадрата

\* B ---- C

\* | |

\* | |

\* A ---- D

\* вводятся точки A и C

\* остальные точки расчитываются относительно центра O

\* через поворот вектора OC на 90 градусов

\*/

inS >> S.verts[0]

>> S.verts[2];

Point center = (S.verts[2] + S.verts[0]) \* 0.5;

Point vec = S.verts[2] - center;

vec.x \*= -1; // (x, y) --> (y, -x) поворот на 90

std::swap(vec.x, vec.y);

S.verts[3] = center + vec;

S.verts[1] = center - vec;

return inS;

}

virtual Point center() const override{

return 0.5 \*(verts[2] + verts[0]);

}

virtual void printVertices() const override{

std::cout << "Square:\n";

Figure::printVertices();

}

virtual double surface() const override{

Point side = verts[1] - verts[0];

return Point::dot(side, side);

}

};

class Octagon : public Figure{

private:

Point m\_center;

public:

Octagon() : Figure(8){}

friend std::istream& operator>>(std::istream &inS, Octagon &Oc){

/\* Вводятся две точки:

\* центр и одна вершина

\* Остальные точки достраиваются

\* до получения правильного восмиугольника

\*/

inS >> Oc.m\_center

>> Oc.verts[0];

Point vec = Oc.verts[0] - Oc.m\_center;

Oc.verts[4] = Oc.m\_center - vec;

vec.x \*= -1;

std::swap(vec.x, vec.y);

Oc.verts[2] = Oc.m\_center + vec;

Oc.verts[6] = Oc.m\_center - vec;

const double r2 = 1 / std::sqrt(2);

Point dia(vec.x + vec.y, vec.y - vec.x);

dia \*= r2;

Oc.verts[3] = Oc.m\_center + dia;

Oc.verts[7] = Oc.m\_center - dia;

dia.x \*= -1;

std::swap(dia.x, dia.y);

Oc.verts[5] = Oc.m\_center + dia;

Oc.verts[1] = Oc.m\_center - dia;

return inS;

}

virtual Point center() const override{

return m\_center;

}

virtual void printVertices() const override{

std::cout << "Octagon:\n";

Figure::printVertices();

}

virtual double surface() const override{

double sur = Point::cross(

verts[0] - m\_center,

verts[1] - m\_center);

sur = 4 \* std::abs(sur);

return sur;

}

};

Литература:

1. vector - C++ Reference | cplusplus.com [Электронный ресурс]  
   URL: <https://www.cplusplus.com/reference/vector/vector/>  
   (Дата обращения: 25.11.2020)
2. С++ Base to derived conversion | RIP tutorial [Электронный ресурс]  
   URL: <https://riptutorial.com/cplusplus/example/10518/base-to-derived-conversion>  
   (Дата обращения: 25.11.2020)