Московский авиационный институт
(Национальный исследовательский университет)

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

Лабораторная работа № 3

Тема: Наследование, полиморфизм C++

Студент: Пермяков Никита Александрович

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

Постановка задачи

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.
Необходимо настроить сборку лабораторной работы с помощью CMake.

- 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 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;

Вариант: 32

- Треугольник
- Квадрат
- Восьмиугольник

Цель:

- Изучение механизмов работы с наследованием в C++

Описание программы

1. Программа выполняет определённые действия по введённым командам:
    a. 0 — выход из программы;
    b. 1,2,3 — создание фигуры (Квадрат, Прямоугольник, Восьмиугольник соответственно), получение вершин через ввод, проверка, вывод данных вершин, вычисление центра и площади;
    c. 4 — создание кортежа как производного четырёхугольника, получение вершин через ввод, проверка, вывод данных вершин, вычисление центра и площади;
2. Шаблонная функция print() печатает координаты всех точек данной фигуры или кортежа. Она определена для моих фигур и tuple. Во втором случае все дело вычисляется рекурсивно.
3. Функция center() возвращает точку с x –деление суммы иксов всех точек данной фигуры на их количество, y – аналогично x. Она определена для моих фигур и tuple. Во втором случае все дело вычисляется рекурсивно;
4. Функция area() вычисляет площадь данной фигуры или совокупности точек в кортеже в зависимости от типа фигуры по методу Гаусса (формула землемера, метод шунтирования) и возвращает это значение.

Набор тестов

Пояснение:

1) На ввод подается число либо строка выбора действия из текстового интерфейса
2) В случае ввода строки – принимается еще одна строка, указывающая на фигуру, с которой производить действие
3) При необходимости указывается ID фигуры
4) Выводится информация о состоянии вектора фигур, происходит проверка тестов

Test 1

1

0 0 0 4 4 4 4 0

4 0

5

6

7

2

0 0 2 3 6 0

4 1

4 0

7

6

5

Test 2

3

0 -2 -2 0 -2 2 0 4 2 4 4 2 4 0 2 -2

4 0

5

6

7

2

0 1 2 3 6 -2

4 1

4 0

7

6

5

Результаты выполнения тестов

1. 'add square' --> Create square
2. 'add triangle' --> Create triangle
3. 'add octagon' --> Create octagon
4. 'show <ID>' --> Output figure
5. 'print total' --> Output total area
6. 'print area' --> Output area
7. 'print center' --> Output center
8. 'delete <ID>' --> Delete figure
9. 'help' --> Get help
10. 'exit' --> Exit

1
input <A.x> <A.y> >> <B.x> <B.y> >> <C.x> <C.y> >> <D.x> <D.y> 0 0 0 4 4 4 4 0
done
4 0
Square: A (0, 0), B (0, 4), C (4, 4), D (4, 0)
done
5
Total area: 16
done
6
Display area: 16
done
7
Display center: (2, 2)
done
2
input <A.x> <A.y> >> <B.x> <B.y> >> <C.x> <C.y> 0 0 2 3 6 0
done
4 1

Triangle: A (0, 0), B (2, 3), C (6, 0)
done
4 0
Square: A (0, 0), B (0, 4), C (4, 4), D (4, 0)
done
7
Display center: (2, 2)
(2.66667, 1)
done
6
Display area: 16
9
done
5
Total area: 25
done
3
input <A.x> <A.y> >> <B.x> <B.y> >> <C.x> <C.y> >> <D.x> <D.y> >> <E.x> <E.y> >> <F.x> <F.y> >> <G.x> <G.y> >> <H.x> <H.y> 0 -2 -2 0 -2 2 0 4 2 4 4 2 4 0 2 -2
done
4 0
Square: A (0, 0), B (0, 4), C (4, 4), D (4, 0)
done
5
Total area: 63.624
done
6
Display area: 16
9
38.624
done
7
Display center: (2, 2)
(2.66667, 1)
(1, 1)
done

2
input <A.x> <A.y> >> <B.x> <B.y> >> <C.x> <C.y> 0 1 2 3 6 -2
done
4 1
Triangle: A (0, 0), B (2, 3), C (6, 0)
done
4 0
Square: A (0, 0), B (0, 4), C (4, 4), D (4, 0)
done
7
Display center: (2, 2)
(2.66667, 1)
(1, 1)
(2.66667, 0.666667)
done
6
Display area: 16
9
38.624
9
done
5
Total area: 72.624
done

Листинг программы

figure.cpp

```
#include "figure.h"


std::ostream& operator<< (std::ostream& out, const Figure& fig) {
	fig.Print(out);
	return out;
}

std::istream& operator>> (std::istream& in, Figure& fig) {
	fig.Scan(in);
	return in;
```

```
}
```

myvector.cpp

```
#include "myvector.h"


Vector::Vector(double x_cord, double y_cord) : x{ x_cord },
y{ y_cord } {}

Vector::Vector(Point& p1, Point& p2) : x{ p2.X() - p1.X() },
y{ p2.Y() - p1.Y() } {}

double Vector::operator*(const Vector& a) const {
	return (this->x * a.x) + (this->y * a.y);
}

Vector& Vector::operator=(const Vector& a) {
	this->x = a.x;
	this->y = a.y;
	return *this;
}

double Length(const Point& a, const Point& b) {
	return sqrt(pow((b.X() - a.X()), 2) + pow((b.Y() - a.Y()),
2));
}

double Length(const Vector& a) {
	return sqrt(pow(a.x, 2) + pow(a.y, 2));
}

bool isParallel(const Vector& a, const Vector& b) {
	return (a.x * b.y) - (a.y * b.x) == 0;
}
```

point.cpp

```
#include "point.h"


Point::Point() : x{ 0.0 }, y{ 0.0 } {}
Point::Point(double a, double b) : x{ a }, y{ b } {}
Point::Point(const Point& other) : x{ other.x }, y{ other.y } {}

double Point::X() const { return x; }
double Point::Y() const { return y; }
```

```
Point Point::operator+ (const Point& a) const {
	return { this->x + a.x, this->y + a.y };
}

Point Point::operator- (const Point& a) const {
	return { this->x - a.x, this->y - a.y };
}

Point Point::operator* (double a) const {
	return { this->x * a, this->y * a };
}

Point Point::operator/ (double a) const {
	return { this->x / a, this->y / a };
}

std::ostream& operator<< (std::ostream& out, const Point& a) {
	out << "( " << a.x << ", " << a.y << " )";
	return out;
}

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

square.cpp

```
#include "square.h"


Square::Square() : A{ Point{} }, B{ Point{} }, C{ Point{} },
D{ Point{} } {}

Square::Square(Point a, Point b, Point c, Point d) :
	A{ a }, B{ b }, C{ c }, D{ d } {

	Vector AB{ A, B }, BC{ B, C }, CD{ C, D }, DA{ D, A };

	if (!isParallel(DA, BC)) {
		std::swap(A, B);
		AB = { A, B };
		BC = { B, C };
		DA = { D, A };
```

```
	}
	if (!isParallel(AB, CD)) {
		std::swap(B, C);
		AB = { A, B };
		BC = { B, C };
		DA = { D, A };
	}
	if (AB * BC || BC * CD || CD * DA || DA * AB) {
		throw std::logic_error("The sides of the square should be
perpendicular");
	}
	if (Length(AB) != Length(BC) || Length(BC) != Length(CD) ||
Length(CD) != Length(DA) || Length(DA) != Length(AB)) {
		throw std::logic_error("The sides of the square should be
equal");
	}
	if (!Length(AB) || !Length(BC) || !Length(CD) || !Length(DA))
{
		throw std::logic_error("The sides of the square must be
greater than zero");
	}
}

Point Square::Center() const {
	return Point{ (B + D) / 2 };
}

double Square::Area() const {
	return Length(A, B) * Length(A, B);
}

std::ostream& Square::Print(std::ostream& out) const {
	out << "Square: A " << A << ", B " << B << ", C " << C << ", D
" << D << std::endl;
	return out;
}

std::istream& Square::Scan(std::istream& in) {
	in >> A >> B >> C >> D;
	(*this) = Square(A, B, C, D);
	return in;
}
```

triangle.cpp

```
#include "triangle.h"


Triangle::Triangle() : A{ Point() }, B{ Point() }, C{ Point() } {}

Triangle::Triangle(Point a, Point b, Point c) : A{ a }, B{ b },
C{ c } {
	double AB = Length(A, B);
	double BC = Length(B, C);
	double AC = Length(A, C);
	if (AB >= BC + AC || BC >= AB + AC || AC >= AB + BC) {
		throw std::logic_error("Points must not be on the same
line.");
	}
}

Point Triangle::Center() const {
	Point middle { (A + C) / 2.0 };
	return { (B + middle * 2) / 3 };
}

double Triangle::Area() const {
	double AB = Length(A, B);
	double BC = Length(B, C);
	double AC = Length(A, C);
	double accumulate = AB + BC + AC;
	double heron = sqrt((accumulate / 2) * (accumulate / 2 - AB) *
(accumulate / 2 - BC) * (accumulate / 2 - AC));
	return heron;
}

std::ostream& Triangle::Print(std::ostream& out) const {
	out << "Triangle: A " << A << ", B " << B << ", C " << C <<
std::endl;
	return out;
}

std::istream& Triangle::Scan(std::istream& in) {
	in >> A >> B >> C;
	(*this) = Triangle(A, B, C);
	return in;
}
```

octagon.cpp

```
#include "octagon.h"


Octagon::Octagon() :
    A{ Point{} }, B{ Point{} }, C{ Point{} }, D{ Point{} },
     E{ Point{} }, F{ Point{} }, G{ Point{} }, H{ Point{} } {}

Octagon::Octagon(Point a, Point b, Point c, Point d,
    Point e, Point f, Point g, Point h) :
    A{ a }, B{ b }, C{ c }, D{ d },
    E{ e }, F{ f }, G{ g }, H{ h } {

    Vector AB{ A, B }, BC{ B, C }, CD{ C, D }, DE{ D, E }, EF{ E,
F }, FG{ F, G }, GH{ G, H }, HA{ H, A };
}

Point Octagon::Center() const {
    return { (A + B + C + D + E + F + G + H) / 8};
}

double Octagon::Area() const {
    return 4.828 * Length(A, B) * Length(A, B);
}

std::ostream& Octagon::Print(std::ostream& out) const {
    out << "Octagon: A " << A << ", B " << B << ", C " << C << ", D
" << D << ", E " << E << ", F " << F << ", G " << G << ", H " << H
<< std::endl;
    return out;
}

std::istream& Octagon::Scan(std::istream& in) {
    in >> A >> B >> C >> D >> E >> F >> G >> H;
    (*this) = Octagon(A, B, C, D, E, F, G, H);
    return in;
}
```

main.cpp

```
#include <iostream>
#include <vector>

#include "square.h"
#include "triangle.h"
```

```
#include "octagon.h"


void print_help() {
    std::cout << "1.  'add square'            --> Create square" <<
std::endl;
    std::cout << "2.  'add triangle'          --> Create triangle"
<< std::endl;
    std::cout << "3.  'add octagon'           --> Create octagon" <<
std::endl;
    std::cout << "4.  'show <ID>'             --> Output figure" <<
std::endl;
    std::cout << "5.  'print total'           --> Output total area"
<< std::endl;
    std::cout << "6.  'print area'            --> Output area" <<
std::endl;
    std::cout << "7.  'print center'          --> Output center" <<
std::endl;
    std::cout << "8.  'delete <ID>'           --> Delete figure" <<
std::endl;
    std::cout << "9.  'help'                  --> Get help" <<
std::endl;
    std::cout << "10. 'exit'                  --> Exit" <<
std::endl;
}

void print_errors(int&& err) {
    switch (err) {
        case 1: {
            std::cout << "Incorrect command" << std::endl;
            break;
        }
        case 2: {
            std::cout << "Incorrect coordinates for a figure" <<
std::endl;
            break;
        }
        case 3: {
            std::cout << "There is no item with the given index" <<
std::endl;
            break;
        }
        default: {
```

```
            std::cout << "Undefined error" << std::endl;
        }
    }
    char c;
    do {
        c = getchar();
    } while (c != '\n' && c != EOF);
}

int main() {
    print_help();
    std::vector<Figure*> figs;

    char com1[40];
    char com2[40];
    int num_com;

    Figure* fig = nullptr;

    while (true) {
        std::cin >> com1;
        num_com = atoi(com1);

        if ((1 <= num_com && num_com <= 3) || strcmp(com1, "add") ==
0) {
            if (1 <= num_com && num_com <= 3)
                com2[0] = '\0';
            else
                std::cin >> com2;

            if (num_com == 1 || strcmp(com2, "square") == 0) {
                std::cout << "input <A.x> <A.y> >> <B.x> <B.y> >>
<C.x> <C.y> >> <D.x> <D.y>\t";
                fig = new Square;
            }
            else if (num_com == 2 || strcmp(com2, "triangle") == 0)
{
                std::cout << "input <A.x> <A.y> >> <B.x> <B.y> >>
<C.x> <C.y>\t";
                fig = new Triangle;
            }
            else if (num_com == 3 || strcmp(com2, "octagon") == 0) {
                std::cout << "input <A.x> <A.y> >> <B.x> <B.y> >>
```

```
<C.x> <C.y> >> <D.x> <D.y> >> <E.x> <E.y> >> <F.x> <F.y> >> <G.x>
<G.y> >> <H.x> <H.y>\t";
                fig = new Octagon;
            }
            else {
                print_errors(1);
            }
            fig->Scan(std::cin);
            figs.push_back(fig);
        }
        else if (num_com == 4 || strcmp(com1, "show") == 0) {
            int id;
            std::cin >> id;
            if (id >= figs.size()) {
                print_errors(3);
                continue;
            }
            figs[id]->Print(std::cout);
        }
        else if ((5 <= num_com && num_com <= 7) || strcmp(com1,
"print") == 0) {
            if (5 <= num_com && num_com <= 7)
                com2[0] = '\0';
            else
                std::cin >> com2;
            if (num_com == 5 || strcmp(com2, "total") == 0) {
                double total_area = 0;
                for (Figure* fig : figs) {
                    total_area += fig->Area();
                }
                std::cout << "Total area: " << total_area <<
std::endl;
            }
            else if (num_com == 6 || strcmp(com2, "area") == 0) {
                std::cout << "Display area: ";
                for (Figure* fig : figs) {
                    std::cout << fig->Area() << std::endl;
                }
            }
            else if (num_com == 7 || strcmp(com2, "center") == 0) {
                std::cout << "Display center: ";
                for (Figure* fig : figs) {
                    Point tmp = fig->Center();
```

```
                    std::cout << "( " << tmp.X() << ", " << tmp.Y()
<< " )" << std::endl;
                }
            }
            else {
                std::cout << "Incorrect command\n";
            }
        }
        else if (num_com == 8 || strcmp(com1, "delete") == 0) {
            int id;
            std::cin >> id;
            if (id >= figs.size()) {
                print_errors(3);
                continue;
            }
            delete figs[id];
            figs.erase(figs.begin() + id);
        }
        else if (num_com == 9 || strcmp(com1, "help") == 0) {
            print_help();
            continue;
        }
        else if (num_com == 10 || strcmp(com1, "exit") == 0) {
            break;
        }
        else {
            print_errors(1);
        }
        std::cout << "\tdone" << std::endl;
    }

    delete fig;

    for (int i = 0; i < figs.size(); ++i) {
        delete figs[i];
    }
    return 0;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)
```

```
project(lab3)

add_executable(lab3
  main.cpp
  figure.cpp
  myvector.cpp
  point.cpp
  octagon.cpp
  square.cpp
  triangle.cpp
)

set_property(TARGET lab3 PROPERTY CXX_STANDARD 11)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -g")
```

Ссылка на репозиторий

https://github.com/nikit34/oop_exercise_03

Объяснение результатов работы программы

1) Пользователю предоставляется 4 опции: задать фигуру (квадрат, треугольник или восьмиугольник), вывести информацию для каждой фигуры (координаты точек, площадь и геометрический центр), вывести общую площадь всех фигур и удалить фигуру по индексу.
2) Перед занесением фигур в вектор каждая фигура проверяется. У квадрата проверяется перпендикулярность и равенство сторон, у треугольника — сумма двух сторон не может быть больше третий стороны. После чего указатель на созданную фигуру заносится в вектор figures.
3) Вывод информации о всех фигурах производится с помощью цикла. Поочередно перебираются все элементы вектора figures, и с помощью метода Print() выводятся координаты, площадь и геометрический центр каждой из фигур.
4) Общая площадь фигур находится посредством суммирования результата работы метода Area() для всех фигур вектора.
5) Если пользователь вводит «0», то считывание завершается, а все фигуры удаляются из памяти с помощью delete.

Вывод

В ходе работы были приобретены навыки работы с шаблонами и кортежами в C++. Написана программа, производящая операции с помощью шаблонов и работающая с кортежами. Создал базовый класс и 3 производных от него класса, которые посредством override методов переопределяли виртуальные методы базового класса.

Список литературы

1. Перегрузка операторов C++ [Электронный ресурс]. URL: https://metanit.com/cpp/tutorial/5.14.php

(дата обращения: 29.09.2020).

2. Битовые операции C++ [Электронный ресурс]. URL: http://www.c-cpp.ru/books/bitovye-operatory

(дата обращения: 29.09.2020).