МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**ЛАБОРАТОРНАЯ РАБОТА № 1**

по дисциплине

**«Объектно-ориентированное программирование»**

на тему:

**«Наследование»**

*Вариант № 10*

Выполнил:

Студент группы

КТбо1-8

Макаров С. В.

Проверил:

Тарасов С. А.

Оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Таганрог 2020

# **1 ФОРМУЛИРОВКА ЗАДАНИЯ**

Согласно варианту задания, описать класс Element (элемент логической схемы) с двумя входами и одним выходом и полем, хранящим название элемента. Определить функцию, которая преобразует входные двоичные значения в выходное. На его основе реализовать классы AND и OR - двоичные вентили, которые реализуют логическое умножение и сложение соответственно. В дополнительном классе Sсheme создать массив элементов (до 10) и обеспечить подачу двоичных сигналов на их входы с выводом выходных значений. Входные сигналы хранятся в файле.

Варианты предполагают создание иерархии классов с виртуальными функциями. Если есть необходимость, то в производном классе можно объявлять дополнительные компоненты. Проектировать иерархию надо так, чтобы гипотетическое добавление нового производного класса (типа) происходило бы без существенного изменения уже существующих классов, т.е. процент повторного использования кода должно быть большим. Безусловно, виртуальные функции должны вызываться правильным образом. По-прежнему запрещается использование контейнеров STL.

# **2 СПЕЦИФИКАЦИЯ КЛАССОВ**

# **3 ИСПОЛЬЗУЕМЫЕ МАТЕМАТИЧЕСКИЕ ЗАВИСИМОСТИ И АЛГОРИТМЫ**

В данной работе использовались логические формулы: AND, OR, XOR.

# **4 ДИАГРАММА КЛАССОВ**

Uml диаграмма классов изображена на рисунке 1.

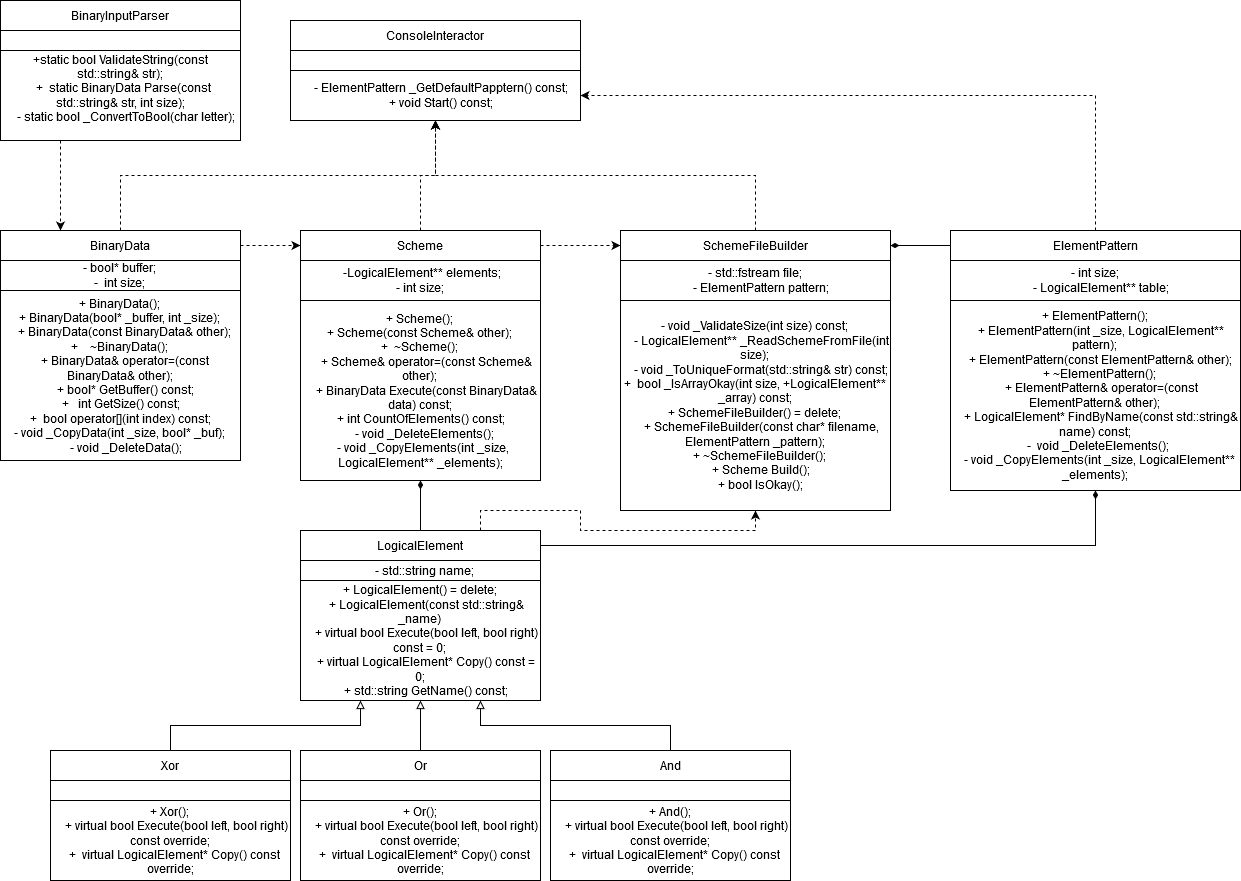


Рисунок 1 – Диаграмма классов

**ЛИСТИНГ ПРОГРАММЫ**

class ConsoleInteractor

{

public:

void Run();

private:

void \_UpdateConsole(const Triangle& t1, const Triangle& t2);

Point \_ReadPointFromConsole();

};

struct Point

{

float x;

float y;

Point(float \_x, float \_y) : x(\_x), y(\_y)

{}

void Add(const Point& other);

friend Point operator+(const Point& left, const Point& right);

friend Point operator-(const Point& left, const Point& right);

static float FindMagnitude(const Point& from, const Point& to);

};

class Triangle

{

public:

Triangle() = delete;

Triangle(const Point& p1, const Point& p2, const Point& p3);

Triangle(float x1, float y1, float x2, float y2, float x3, float y3);

Point FindCenter() const;

float FindArea() const;

float FindPerimeter() const;

void Move(Point offset);

Point GetFirstPoint() const;

Point GetSecondPoint() const;

Point GetThirdPoint() const;

void SetFirstPoint(const Point& other);

void SetSecondPoint(const Point& other);

void SetThirdPoint(const Point& other);

bool IsAreasEqual(const Triangle& other) const;

bool IsExisting() const;

friend bool operator>(const Triangle& left, const Triangle& right);

friend bool operator<(const Triangle& left, const Triangle& right);

friend bool operator>=(const Triangle& left, const Triangle& right);

friend bool operator<=(const Triangle& left, const Triangle& right);

private:

Point points[3];

void \_CalculateMagnitudes(float& a, float& b, float& c) const;

int \_CompareArea(const Triangle& other) const;

};

class TriangleExistingException : public std::exception

{

public:

TriangleExistingException(const Triangle \_triangle);

const char\* what() const noexcept;

private:

std::string what\_string;

Triangle triangle;

};

Triangle::Triangle(const Point& p1, const Point& p2, const Point& p3)

: points{ p1, p2, p3 }

{}

Triangle::Triangle(float x1, float y1, float x2, float y2, float x3, float y3)

: Triangle({ x1, y1 }, { x2, y2 }, { x3, y3 })

{}

Point Triangle::FindCenter() const

{

if (!IsExisting())

{

throw TriangleExistingException(\*this);

}

float centerX = (points[0].x + points[1].x + points[2].x) / 3;

float centerY = (points[0].y + points[1].y + points[2].y) / 3;

return Point(centerX, centerY);

}

float Triangle::FindArea() const

{

if (!IsExisting())

{

throw TriangleExistingException(\*this);

}

float a, b, c;

\_CalculateMagnitudes(a, b, c);

float p = (a + b + c) / 2;

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

}

float Triangle::FindPerimeter() const

{

if (!IsExisting())

{

throw TriangleExistingException(\*this);

}

float a, b, c;

\_CalculateMagnitudes(a, b, c);

return a + b + c;

}

void Triangle::Move(Point offset)

{

points[0].Add(offset);

points[1].Add(offset);

points[2].Add(offset);

}

Point Triangle::GetFirstPoint() const

{

return points[0];

}

Point Triangle::GetSecondPoint() const

{

return points[1];

}

Point Triangle::GetThirdPoint() const

{

return points[2];

}

void Triangle::SetFirstPoint(const Point& other)

{

points[0] = other;

}

void Triangle::SetSecondPoint(const Point& other)

{

points[1] = other;

}

void Triangle::SetThirdPoint(const Point& other)

{

points[2] = other;

}

bool Triangle::IsAreasEqual(const Triangle& other) const

{

if (this->\_CompareArea(other) == 0)

{

return true;

}

return false;

}

bool Triangle::IsExisting() const

{

float a, b, c;

\_CalculateMagnitudes(a, b, c);

if ((a + b > c) && (b + c > a) && (a + c > b))

{

return true;

}

return false;

}

void Triangle::\_CalculateMagnitudes(float& a, float& b, float& c) const

{

a = Point::FindMagnitude(points[0], points[1]);

b = Point::FindMagnitude(points[1], points[2]);

c = Point::FindMagnitude(points[2], points[0]);

}

int Triangle::\_CompareArea(const Triangle& other) const

{

float S1 = this->FindArea();

float S2 = other.FindArea();

if (S1 > S2)

{

return 1;

}

else if (S1 == S2)

{

return 0;

}

else

{

return -1;

}

}

bool operator>(const Triangle& left, const Triangle& right)

{

if (left.\_CompareArea(right) > 0)

{

return true;

}

return false;

}

bool operator<(const Triangle& left, const Triangle& right)

{

if (left.\_CompareArea(right) < 0)

{

return true;

}

return false;

}

bool operator>=(const Triangle& left, const Triangle& right)

{

if (left.\_CompareArea(right) >= 0)

{

return true;

}

return false;

}

bool operator<=(const Triangle& left, const Triangle& right)

{

if (left.\_CompareArea(right) <= 0)

{

return true;

}

return false;

}

void Point::Add(const Point& other)

{

x += other.x;

y += other.y;

}

float Point::FindMagnitude(const Point& from, const Point& to)

{

float subX = to.x - from.x;

float subY = to.y - from.y;

return sqrt(subX \* subX + subY \* subY);

}

Point operator+(const Point& left, const Point& right)

{

return Point(left.x + right.x, left.y + right.y);

}

Point operator-(const Point& left, const Point& right)

{

return Point(left.x - right.x, left.y - right.y);

}

TriangleExistingException::TriangleExistingException(const Triangle \_triangle)

: triangle(\_triangle)

{

std::stringstream sstream;

Point p1 = triangle.GetFirstPoint();

Point p2 = triangle.GetSecondPoint();

Point p3 = triangle.GetThirdPoint();

sstream << "Triangle doesn't exist with points: \n"

<< "x1: " << p1.x << " y1: " << p1.y << "\n"

<< "x2: " << p2.x << " y2: " << p2.y << "\n"

<< "x3: " << p3.x << " y3: " << p3.y;

what\_string = sstream.str();

}

const char\* TriangleExistingException::what() const noexcept

{

return what\_string.c\_str();

}

void ConsoleInteractor::Run()

{

float x1, y1, x2, y2, x3, y3;

cout << "Enter a coords of first triangle vertexes in format: \nx1 y1\nx2 y2\nx3 y3\n\n";

cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;

Triangle t1(x1, y1, x2, y2, x3, y3);

cout << "Enter a coords of second triangle vertexes in format: \nx1 y1\nx2 y2\nx3 y3\n\n";

cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;

Triangle t2(x1, y1, x2, y2, x3, y3);

\_UpdateConsole(t1, t2);

int switch\_on;

Point C1(0.f, 0.f);

Point C2(0.f, 0.f);

while (true)

{

cout << ">>> ";

cin >> switch\_on;

try

{

switch (switch\_on)

{

case 0:

return;

break;

case 1:

cout << "-------------------------------------\n";

cout << "Perimeter of first triangle = " << t1.FindPerimeter() << "\n";

cout << "Perimeter of second triangle = " << t2.FindPerimeter() << "\n";

cout << "-------------------------------------\n\n";

break;

case 2:

cout << "-------------------------------------\n";

cout << "Area of first Triangle = " << t1.FindArea() << "\n";

cout << "Area of second Triangle = " << t2.FindArea() << "\n";

cout << "-------------------------------------\n\n";

break;

case 3:

cout << "-------------------------------------\n";

C1 = t1.FindCenter();

C2 = t2.FindCenter();

cout << "Coords of first triangle's center = x: " << C1.x << " y: " << C1.y << "\n";

cout << "Coords of second triangle's center = x: " << C2.x << " y: " << C2.y << "\n";

cout << "-------------------------------------\n\n";

break;

case 4:

cout << "-------------------------------------\n";

t1.Move(\_ReadPointFromConsole());

cout << "-------------------------------------\n\n";

\_UpdateConsole(t1, t2);

break;

case 5:

cout << "-------------------------------------\n";

t2.Move(\_ReadPointFromConsole());

cout << "-------------------------------------\n\n";

\_UpdateConsole(t1, t2);

break;

case 6:

cout << "-------------------------------------\n";

cout << "Result of 'First Triagnle > Second Triangle' is " << ((t1 > t2) ? "true\n" : "false\n");

cout << "-------------------------------------\n\n";

break;

case 7:

cout << "-------------------------------------\n";

cout << "Result of 'First Triagnle < Second Triangle' is " << ((t1 < t2) ? "true\n" : "false\n");

cout << "-------------------------------------\n\n";

break;

case 8:

cout << "-------------------------------------\n";

cout << "Result of 'First Triagnle >= Second Triangle' is " << ((t1 >= t2) ? "true\n" : "false\n");

cout << "-------------------------------------\n\n";

break;

case 9:

cout << "-------------------------------------\n";

cout << "Result of 'First Triagnle <= Second Triangle' is " << ((t1 <= t2) ? "true\n" : "false\n");

cout << "-------------------------------------\n\n";

break;

case 10:

cout << "-------------------------------------\n";

cout << "Result of 'Is areas Equal?' is " << (t1.IsAreasEqual(t2) ? "true\n" : "false\n");

cout << "-------------------------------------\n\n";

break;

case 11:

cout << "-------------------------------------\n";

t1.SetFirstPoint(\_ReadPointFromConsole());

cout << "-------------------------------------\n\n";

\_UpdateConsole(t1, t2);

break;

case 12:

cout << "-------------------------------------\n";

t1.SetSecondPoint(\_ReadPointFromConsole());

cout << "-------------------------------------\n\n";

\_UpdateConsole(t1, t2);

break;

case 13:

cout << "-------------------------------------\n";

t1.SetThirdPoint(\_ReadPointFromConsole());

cout << "-------------------------------------\n\n";

\_UpdateConsole(t1, t2);

break;

case 14:

cout << "-------------------------------------\n";

t2.SetFirstPoint(\_ReadPointFromConsole());

cout << "-------------------------------------\n\n";

\_UpdateConsole(t1, t2);

break;

case 15:

cout << "-------------------------------------\n";

t2.SetSecondPoint(\_ReadPointFromConsole());

cout << "-------------------------------------\n\n";

\_UpdateConsole(t1, t2);

break;

case 16:

cout << "-------------------------------------\n";

t2.SetThirdPoint(\_ReadPointFromConsole());

cout << "-------------------------------------\n\n";

\_UpdateConsole(t1, t2);

break;

case 17:

\_UpdateConsole(t1, t2);

break;

default:

cout << "Invalid command. Try again\n";

break;

}

}

catch (const std::exception & e)

{

cerr << e.what() << "\n";

}

}

}

void ConsoleInteractor::\_UpdateConsole(const Triangle& t1, const Triangle& t2)

{

system("cls");

cout << "COMMANDS: 1 - Find Perpimeter. 2 - Find Area. 3 - Find Center of Mass.\n";

cout << " 4 - Move first Triangle. 5 - Move second Triangle.\n";

cout << " 6 - Compare with operator >. 7 - Compare with operator <.\n";

cout << " 8 - Compare with operator >=. 9 - Compare with operator <=.\n";

cout << " 10 - Is areas equal? 0 - to Exit\n";

cout << " 11 - Set first point of first triangle. 12 - Set second point of first triangle\n";

cout << " 13 - Set third point of first triangle.\n";

cout << " 14 - Set first point of second triangle. 15 - Set second point of second triangle\n";

cout << " 16 - Set third point of second triangle.\n";

cout << " 17 - To refresh console.\n";

Point p1 = t1.GetFirstPoint();

Point p2 = t1.GetSecondPoint();

Point p3 = t1.GetThirdPoint();

cout << "\nCurrent Triangles: \n";

cout << setw(10) << left << "First: " << "|";

cout << "x1: " << p1.x << " y1: " << p1.y << "|";

cout << "x2: " << p2.x << " y2: " << p2.y << "|";

cout << "x3: " << p3.x << " y3: " << p3.y << " | \n\n";

p1 = t2.GetFirstPoint();

p2 = t2.GetSecondPoint();

p3 = t2.GetThirdPoint();

cout << setw(10) << left << "Second: " << "|";

cout << "x1: " << p1.x << " y1: " << p1.y << "|";

cout << "x2: " << p2.x << " y2: " << p2.y << "|";

cout << "x3: " << p3.x << " y3: " << p3.y << " | \n\n";

}

Point ConsoleInteractor::\_ReadPointFromConsole()

{

float x1, y1;

cout << "Enter a point if format: x y\n";

cin >> x1 >> y1;

return Point(x1, y1);

}

int main()

{

ConsoleInteractor program;

program.Run();

return 0;

}