**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

Лабораторная работа № 4

по ОБЪЕКТНО-ОРИЕНТИРОВАННОМУ ПРОГРАММИРОВАНИЮ

«Параметризованные классы»

Студент \_\_\_\_\_\_\_\_\_\_\_ Целищев А.Е.

Группа ПМ-21-2

Руководитель

Доцент \_\_\_\_\_\_\_\_\_\_\_ Кургасов В.В.

Липецк 2022г.

Оглавление

Цель работы 2

Задание кафедры 2

Код программы 3

Пример работы программы 11

Вывод 13

Контрольные вопросы...........................................................................................14

**Цель работы:**

Изучить механизм параметрического полиморфизма на основе создания и использования параметризованных классов.

**Задание кафедры:**

Реализовать на языке С++ параметризованный класс «Матрица», типы элементов которого могут быть заданы в соответствии с вариантом:

Классы и возможные действия над классами

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Тип 1 | Тип 2 | Возможности класса |
| 11 | Вектор в декартовой системе N координат | Вектор в полярных координатах | Разность двух матриц и присвоение результата третьей переменной-матрице осуществляется одной строкой С = А - В |

***Код программы***

#include <iostream>

#include <random> // rand(), srand()

using namespace std;

// Класс вектор в декартовой системе N координат.

// По умолчанию имеет размерность dim = 3.

class DecartVector

{

private:

int dim; // Размерность вектора и его точек

double \*vector; // Вектор размерности dim, вычисляемый как разность координат конечной и начальной точек.

// Функция генерации случайного целого числа от a до b включительно. Используется только внутри класса

static int getrand(int a, int b) { return a + rand()%(b - a + 1);}

void fillrand() // Заполнение вектора случайными числами

{

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

{

double start\_point = getrand(-50, 50);

double end\_point = getrand(-50, 50);

vector[i] = end\_point - start\_point;

}

}

public:

DecartVector() // Конструктор по умолчанию

{

dim = 3;

vector = new double[dim];

fillrand();

}

DecartVector(int user\_dim) // Конструктор с одним принимаемым параметром - пользовательской размерностью вектора

{

dim = user\_dim;

vector = new double[dim];

fillrand();

}

DecartVector(const DecartVector& D) // Конструктор копирования

{

dim = D.dim;

vector = D.vector;

}

~DecartVector() // Деструктор. Так как размеры массивов точек и вектора динамические, надо освобождать память через деструктор

{

delete [] vector;

}

friend ostream& operator <<(ostream& out, DecartVector& D) // Перегрузка оператора потокового вывода

{

out << '(';

for (int i = 0; i < D.dim - 1; i++) out << D.vector[i] << ", ";

out << D.vector[D.dim-1] << ")";

return out;

}

DecartVector operator -(const DecartVector& other) // Перегрузка оператора вычитания между двумя векторами

{

DecartVector res = DecartVector(dim);

for (int i = 0; i < res.dim; i++) res.vector[i] = vector[i] - other.vector[i];

return res;

}

DecartVector operator +(const DecartVector& other) // Перегрузка оператора сложения между двумя векторами

{

DecartVector res = DecartVector(dim);

for (int i = 0; i < res.dim; i++) res.vector[i] = vector[i] + other.vector[i];

return res;

}

};

// Класс вектор в полярных координатах.

// Имеет возможность заполнения случайными значениями

// полярного радиуса и полярного угла.

class PolarVector

{

private:

double radius; // Значение полярного радиуса вектора

double angle; // Значение полярного угла вектора

// Функция генерации случайного числа от a до b включительно. Используется только внутри определения класса.

static int getrand(int from, int to) { return from + rand()%(to - from + 1);}

public:

PolarVector() // Конструктор по умолчанию

{

radius = getrand(0, 25);

angle = getrand(-180, 180);

}

PolarVector(const PolarVector& P) // Конструктор копирования

{

radius = P.radius;

angle = P.angle;

}

friend ostream& operator <<(ostream& out, PolarVector& P) // Перегрузка оператора потокового вывода

{

out << '(' << P.radius << ", " << P.angle << ')';

return out;

}

PolarVector operator -(const PolarVector& other) // Перегрузка оператора вычитания между двумя векторами

{

PolarVector res = PolarVector();

double new\_x = radius \* cos(angle) - other.radius \* cos(other.angle); // new\_x = r0cosA0 - r1cosA1

double new\_y = radius \* sin(angle) - other.radius \* sin(other.angle); // new\_y = r0sinA0 - r1sinA1

res.radius = sqrt(pow(new\_x, 2) + pow(new\_y, 2)); // res.radius = sqrt(new\_x^2 + new\_y^2)

res.angle = atan(new\_y / new\_x); // res.angle = arctg(new\_y / new\_x)

return res;

}

PolarVector operator +(const PolarVector& other) // Перегрузка оператора сложения между двумя векторами

{

PolarVector res = PolarVector();

double new\_x = radius \* cos(angle) + other.radius \* cos(other.angle); // new\_x = r0cosA0 + r1cosA1

double new\_y = radius \* sin(angle) + other.radius \* sin(other.angle); // new\_y = r0sinA0 + r1sinA1

res.radius = sqrt(pow(new\_x, 2) + pow(new\_y, 2)); // new\_radius = sqrt(new\_x^2 + new\_y^2)

res.angle = atan(new\_y / new\_x); // new\_angle = arctg(new\_y / new\_x)

return res;

}

};

// Параметризованный класс матрица, принимающий на входе любой класс,

// имеющий перегруженные операторы потокового вывода, сложения и вычитания между объектами класса,

// а также метод fillrand для автоматического заполнения случайными значениями

template<class T>

class Matrix

{

private:

T m[4][4]; // Матрица фиксированного размера 4х4, содержащая объекты класса-параметра Т

public:

// Заполнение матрицы объектами переданного класса-параметра Т.

// В данном случае это векторы в декартовой системе N координат или векторы в полярной системе координат

Matrix() = default;

Matrix(const Matrix& M) // Конструктор копирования для реализации операции присваивания

{

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

m[i][j] = M.m[i][j];

}

}

}

void print() // Вывод матрицы в консоль

{

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

cout << m[i][j] << ' ';

}

cout << endl;

}

cout << '\n';

}

Matrix operator -(const Matrix &M) // Перегрузка оператора вычитания

{

Matrix<T> res;

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

res.m[i][j] = m[i][j] - M.m[i][j];

}

}

return res;

}

Matrix operator +(const Matrix &M) // Перегрузка оператора сложения

{

Matrix<T> res;

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

res.m[i][j] = m[i][j] + M.m[i][j];

}

}

return res;

}

};

int main()

{

srand(time(nullptr)); // Для генерации случайных чисел

cout << ">>> Decart vectors matrices:\n";

cout << "m1:\n";

Matrix<DecartVector> m1;

m1.print();

cout << "m2:\n";

Matrix<DecartVector> m2;

m2.print();

cout << "\nm1 - m2:\n";

Matrix<DecartVector> m3 = m1 - m2;

m3.print();

cout << "\nm1 + m2:\n";

Matrix<DecartVector> m4 = m1 + m2;

m4.print();

cout << "\n>>> Polar vectors matrices:\n";

cout << "p1:\n";

Matrix<PolarVector> p1;

p1.print();

cout << "p2:\n";

Matrix<PolarVector> p2;

p2.print();

cout << "\np1 - p2:\n";

Matrix<PolarVector> p3 = p1 - p2;

p3.print();

cout << "\np1 + p2:\n";

Matrix<PolarVector> p4 = p1 + p2;

p4.print();

return 0;

}

***Пример работы программы***

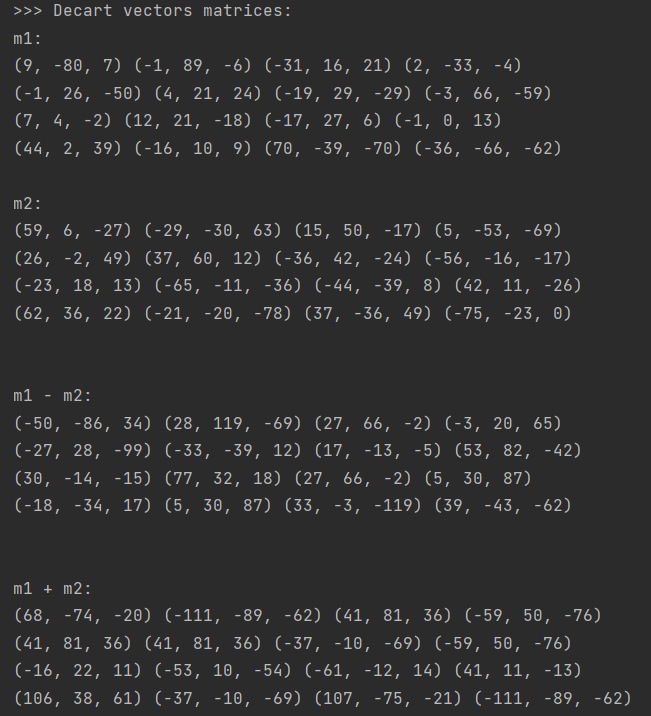


Рисунок 1. Матрицы векторов в декартовой системе 3 координат

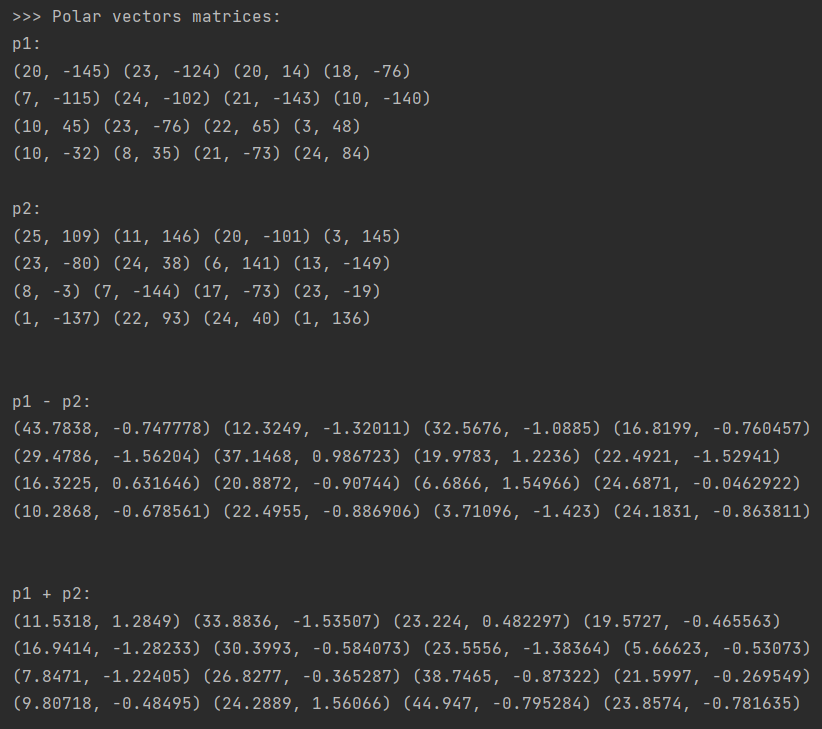


Рисунок 2. Матрицы векторов в полярной системе координат

***Вывод:***

Реализован параметризованный класс «Матрица», элементы которого могут быть либо векторами в декартовой системе N координат (по умолчанию 3), либо векторами в полярной системе координат. Разность (сумма) двух матриц и присвоение результата третьей матрице-переменной осуществляется одной строкой С = А ± В.

***Контрольные вопросы***

1. **Зачем нужны параметризованные классы?**

Параметризованные классы используются для создания семейства родственных функций и классов. Это позволяет автоматизировать создание функций и классов, способных обрабатывать данные различного типа. В отличие от перегрузки операторов, задание шаблона происходит один раз и это определение затем параметризуется.

1. **Какой из механизмов ООП реализуется с использованием параметризованных классов?**

Параметрический полиморфизм.

1. **Что такое шаблон класса?**

Средство создания параметризованных классов, обобщённое описание алгоритма без привязки к некоторым конкретным параметрам (типам данных, размерам буферов, значениям по умолчанию), которые не могут быть известны заранее.

1. **Во сколько раз сокращается исполняемый машинный код программы при использовании параметризованных классов?**

Исполняемый код не сокращается, так как для каждого набора параметров компилятор создаёт отдельный экземпляр класса.

1. **Какого вида могут быть параметры при задании шаблона класса?**

Параметром при задании шаблона класса может выступать как некоторый класс, обозначаемый ключевым словом class, так и некоторый неизвестный заранее тип данных, который описывается ключевым словом typename. Также в список параметров может входить параметр конкретного типа данных (например, размер буфера).

template <class T, typename T1, int size> class A {};