Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа программной инженерии

КУРСОВАЯ РАБОТА

по дисциплине «Системный анализ и принятие решений»

Выполнил студент:

гр. № 3540904/10101 Томилин И. С.

Руководитель:

к.т.н. Амосов В. В.

Санкт-Петербург

2022 г.

ОГЛАВЛЕНИЕ

[1. Введение 3](#__RefHeading___Toc3440_507806098)

[2. Исходные данные для проверки 3](#__RefHeading___Toc3442_507806098)

[Варианты 4](#__RefHeading___Toc3444_507806098)

[Предпочтения 4](#__RefHeading___Toc3446_507806098)

[Бинарные отношения 5](#__RefHeading___Toc3448_507806098)

[3. Результаты работы программы 6](#__RefHeading___Toc3450_507806098)

[4. Вывод 8](#__RefHeading___Toc3452_507806098)

[5. Листинг программы 10](#__RefHeading___Toc3454_507806098)

# Введение

Качественный подход к принятию решения подразумевает формализацию ситуации с использованием аппарата бинарных отношений или предпочтений, задание для каждого бинарного отношения весового коэффициента, проведение выбора решений с помощью аппарата функций выбора, поиск оптимальных вариантов решений для каждого бинарного отношения и сведение полученных результатов в обобщающую таблицу для наглядности и автоматизации выбора лицом, принимающим решение.

В системе применяются для оценки предпочтений бинарные отношения «больше» и «меньше»

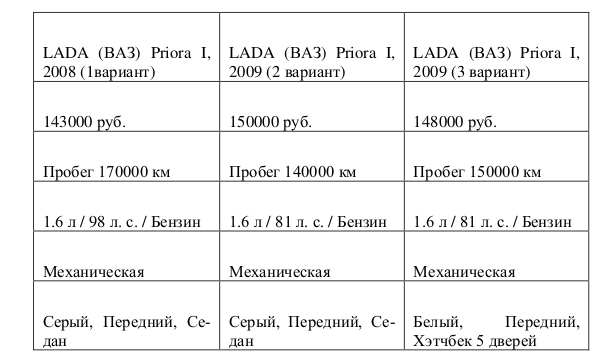
Аппарат функций выбора задаётся для каждого бинарного отношения механизмами доминирования, блокировки, турнирным механизмом и механизмом определения К - максимальных вариантов. Для каждого бинарного отношения задаются весовые коэффициенты. Полученные по каждому механизму результаты ранжируются с учётом весовых коэффициентов бинарных отношений.

# **Исходные данные для проверки**

Для тестирования системы используются данные, загружаемые из файлов, как стандартные тестовые данные шаблона системы поддержки принятия решения на основе качественного подхода, так и приведенные ниже данные.

В качестве тестирования корректности расчета и вывода результатов был взят шаблон выбора наилучшей модели автомобиля LADA (ВАЗ). Цель тестирования, убедиться в корректности работы приложения.

## Варианты

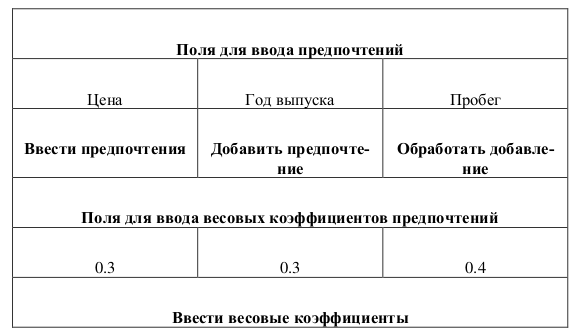
Сравниваются три автомобиля:

Названия вариантов вводим в файл следующим образом:

Каждый вариант с новой строки через нижнее подчеркивание.

## Предпочтения

Выбраны следующие данные о предпочтениях:

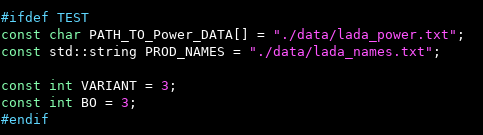


Файл для ввода предпочтений нужно ввести в файл следующим образом:



Названия предпочтений вводим через нижнее подчеркивание, затем через пробел вводим коэффициент.

Чтение коэффициентов и вариантов указывается в коде, количество бинарных отношений и вариантов нужно также задать:

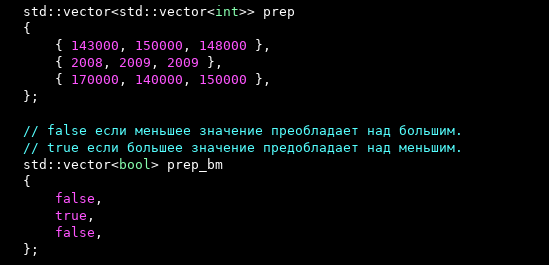


## Бинарные отношения

Матрицы бинарных отношений генерируются автоматически, нужно только ввести два вектора:

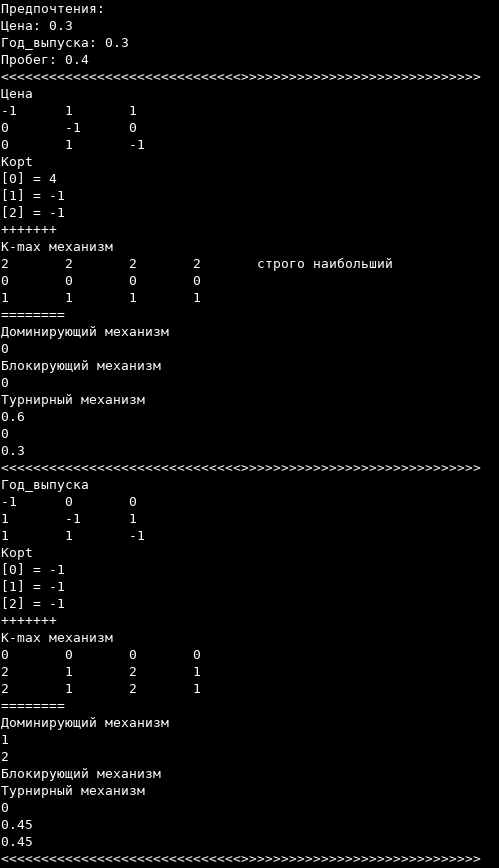
1) варианты предпочтения;

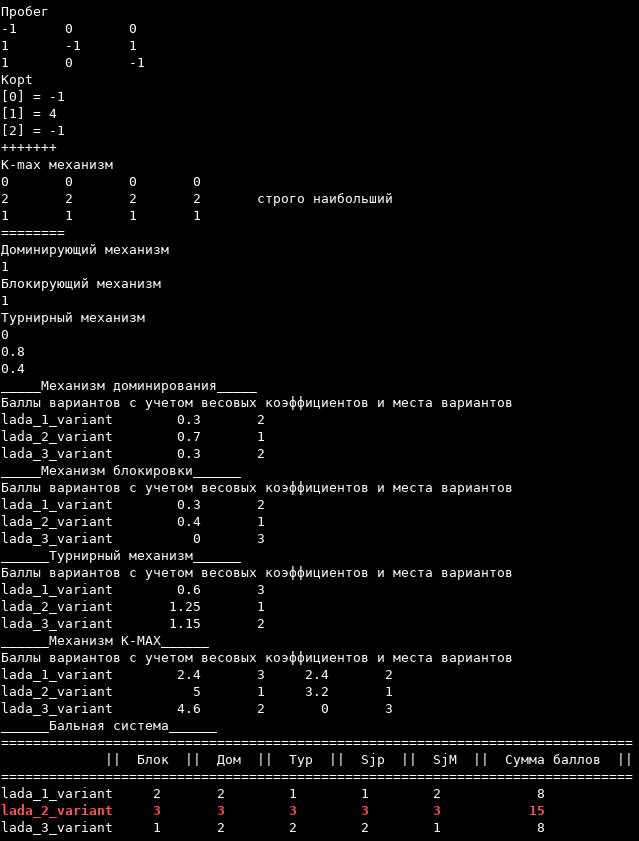
2) вектор определяющий преобладание предпочтения (например чем новее год выпуска автомобиля тем лучше, 2009 предпочтительнее чем 2008 или наоборот чем меньше пробег тем лучше, 170000км менее предпочтительней чем 140000км)





# Результаты работы программы





# Вывод

Произведена модификация исходного кода студента гр. в3530904/00030 О.С. Клабукова, исправлена часть недочетов, введены новые возможности. Работа программы протестирована на данных шаблона СППР, в результате работы приложения, были получены данные как в шаблоне, что может сказать о правильности работы программы.

# Листинг программы

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <algorithm>

#include <iomanip>

#include <numeric>

#include <cmath>

#ifdef TEST

const char PATH\_TO\_Power\_DATA[] = "./data/lada\_power.txt";

const std::string PROD\_NAMES = "./data/lada\_names.txt";

const int VARIANT = 3;

const int BO = 3;

#endif

//выделение памяти дл¤ динамического двумерного массива

double\*\* createArr(int n, int m);

//подсчет весовые коэффициенты

void readPower(double\* arr, double n, std::ifstream& in,

std::vector<std::string> &pref\_names);

//посчитать массив размером NxM из вход¤щего потока

void readFile(double\*\* arr, double n, double m, std::ifstream& in);

//определение доминирующего варианта

std::vector<int> dominate(double\*\* arr, double n, double m);

//определение блокирующего варианта

std::vector<int> block(double\*\* arr, double n, double m);

//определение турнирного варианта

std::vector<double> turnir(double\*\* arr, const double power[BO], double n,

double m, int number);

//составление массива дл¤ варианта в случае механизма K-max

double\*\* createKarray(double\*\* arr, double n, double m);

//определение K-opt вариантов

void createKopt(double\*\* arr, double n, int\* kopt\_Array);

//вывести двумерный массив

void writeArr(double\*\* arr, double n, double m);

//вывести двумерный массив K-opt механизм

void writeArrKopt(double\*\* arr, double n, double m, int\* opt);

//уничтожить двумерный массив

void distractionArray(double\*\* arr, int n);

//расстановка мест

void placeRating(const double arr[VARIANT], int A[VARIANT]);

void read\_from\_file( std::vector<std::string> &vec, std::ifstream& in )

{

std::string data {};

while( in >> data )

{

vec.emplace\_back( data );

}

}

void build\_matrix( std::vector<int> vars, bool greater, double\*\* matrix )

{

for ( int i = 0; i < vars.size(); ++i )

{

for ( int j = 0; j < vars.size(); ++j )

{

if ( i == j )

{

matrix[i][j] = -1;

continue;

}

if ( !greater && vars.at(i) <= vars.at(j) )

{

matrix[i][j] = 1;

continue;

}

if ( greater && vars.at(i) >= vars.at(j) )

{

matrix[i][j] = 1;

}

}

}

}

int main()

{

setlocale(LC\_ALL, "RUS");

double rating[VARIANT] = { 0 };

double ratingBlock[VARIANT] = { 0 };

double ratingTurnir[VARIANT] = { 0 };

double kmax[VARIANT] = { 0 };

double kopt[VARIANT] = { 0 };

std::vector<std::vector<int>> prep

{

{ 143000, 150000, 148000 },

{ 2008, 2009, 2009 },

{ 170000, 140000, 150000 },

};

// false если меньшее значение преобладает над большим.

// true если большее значение предобладает над меньшим.

std::vector<bool> prep\_bm

{

false,

true,

false,

};

// Read names.

std::vector<std::string> names;

std::ifstream in\_names;

// in\_names.exceptions(std::ios::failbit | std::ios::badbit);

in\_names.open(PROD\_NAMES.c\_str());

read\_from\_file( names, in\_names );

//считывание весовых коэффициентов

std::ifstream in\_power;

in\_power.exceptions(std::ios::failbit | std::ios::badbit);

in\_power.open(PATH\_TO\_Power\_DATA);

double\* powerArr = new double[BO];

//powerArr - массив с весовыми коэффициентами

std::vector<std::string> pref\_names;

readPower(powerArr, BO, in\_power, pref\_names);

double sum { 0 };

sum = std::accumulate(powerArr, powerArr + BO, sum );

std::cout << "Sum of weight coeff: " << sum << std::endl;

std::cout << "Предпочтения:" << std::endl;

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

{

std::cout << pref\_names[i] << ": " << powerArr[i] << std::endl;

}

in\_power.close();

for (int k = 0; k < prep.size(); k++)

{

std::cout << "<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>" << std::endl;

std::cout << pref\_names[k] << std::endl;

double\*\* Dom\_data = createArr(VARIANT, VARIANT);

build\_matrix( prep[k], prep\_bm[k], Dom\_data );

//readFile(Dom\_data, VARIANT, VARIANT, in);

writeArr(Dom\_data, VARIANT, VARIANT);

//обработка данных

std::vector<int> dom\_Array;

std::vector<int> block\_Array;

std::vector<double> turnir\_Array;

//std::vector<int> kopt\_Array;

int\* kopt\_Array = new int[VARIANT];

std::fill\_n( kopt\_Array, VARIANT, -1 );

double\*\* Karray = createKarray(Dom\_data, VARIANT, VARIANT);

createKopt(Karray, VARIANT, kopt\_Array);

std::cout << "Kopt" << std::endl;

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

{

std::cout << '[' << i << "] = " << kopt\_Array[i] << std::endl;

}

std::cout << "+++++++" << std::endl;

std::cout << "K-max механизм " << std::endl;

writeArrKopt(Karray, VARIANT, 4, kopt\_Array);

std::cout << "======== " << std::endl;

dom\_Array = dominate(Dom\_data, VARIANT, VARIANT);//определение доминирующих вариантов

block\_Array = block(Dom\_data, VARIANT, VARIANT);//определение блокирующих вариантов

turnir\_Array = turnir(Dom\_data, powerArr, VARIANT, VARIANT, k);//определение турнирных вариантов

std::cout << "Доминирующий механизм" << std::endl;

for (int i = 0; i < dom\_Array.size(); ++i)

{

std::cout << dom\_Array[i] << std::endl;

rating[dom\_Array[i]] += powerArr[k];

}

std::cout << "Блокирующий механизм " << std::endl;

for (int i = 0; i < block\_Array.size(); ++i)

{

std::cout << block\_Array[i] << std::endl;

ratingBlock[block\_Array[i]] += powerArr[k];

}

//данные по турнирному механизму

std::cout << "Турнирный механизм " << std::endl;

for (int i = 0; i < turnir\_Array.size(); ++i)

{

std::cout << turnir\_Array[i] << std::endl;

ratingTurnir[i] += turnir\_Array[i];

}

//данные по K-max механизму

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

{

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

{

kmax[i] += Karray[i][j] \* powerArr[k];

}

}

//данные по K-opt

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

{

if ((kopt\_Array[i] == 1) || (kopt\_Array[i] == 2) ||

(kopt\_Array[i] == 3) || (kopt\_Array[i] == 4))

{

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

{

kopt[i] += Karray[i][j] \* powerArr[k];

}

}

}

distractionArray(Dom\_data, VARIANT);

distractionArray(Karray, VARIANT);

}

int rating\_place[VARIANT] = { 0 };

placeRating(rating, rating\_place);

std::cout << "\_\_\_\_\_Механизм доминирования\_\_\_\_\_" << std::endl;

std::cout << "Баллы вариантов с учетом весовых коэффициентов и места вариантов" << std::endl;

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

{

std::cout << std::setw(0) << names[i] << std::setw(25 - names[i].size()) << rating[i] <<

std::setw(8) << rating\_place[i] << std::endl;

}

int rating\_place\_block[VARIANT] = { 0 };

placeRating(ratingBlock, rating\_place\_block);

std::cout << "\_\_\_\_\_Механизм блокировки\_\_\_\_\_\_" << std::endl;

std::cout << "Баллы вариантов с учетом весовых коэффициентов и места вариантов" << std::endl;

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

{

std::cout << std::setw(0) << names[i] << std::setw(25 - names[i].size()) << ratingBlock[i] <<

std::setw(8) << rating\_place\_block[i] << std::endl;

}

int rating\_place\_turnir[VARIANT] = { 0 };

placeRating(ratingTurnir, rating\_place\_turnir);

std::cout << "\_\_\_\_\_\_Турнирный механизм\_\_\_\_\_\_" << std::endl;

std::cout << "Баллы вариантов с учетом весовых коэффициентов и места вариантов" << std::endl;

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

{

std::cout << std::setw(0) << names[i] << std::setw(25 - names[i].size()) << ratingTurnir[i] <<

std::setw(8) << rating\_place\_turnir[i] << std::endl;

}

int rating\_place\_kmax[VARIANT] = { 0 };

int rating\_place\_kopt[VARIANT] = { 0 };

placeRating(kmax, rating\_place\_kmax);

placeRating(kopt, rating\_place\_kopt);

std::cout << "\_\_\_\_\_\_Механизм K-MAX\_\_\_\_\_\_" << std::endl;

std::cout << "Баллы вариантов с учетом весовых коэффициентов и места вариантов" << std::endl;

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

{

std::cout << std::setw(0) << names[i] << std::setw(25 - names[i].size()) << kmax[i] <<

std::setw(8) << rating\_place\_kmax[i] << std::setw(8) << kopt[i] <<

std::setw(8) << rating\_place\_kopt[i] << std::endl;

}

std::cout << "\_\_\_\_\_\_Бальная система\_\_\_\_\_\_" << std::endl;

std::cout << "===============================================================================" << std::endl;

std::cout << " || Блок || Дом || Тур || Sjp || SjM || Сумма баллов ||" << std::endl;

std::cout << "===============================================================================" << std::endl;

int winner = VARIANT + 1 - rating\_place[0] +

VARIANT + 1 - rating\_place\_block[0] +

VARIANT + 1 - rating\_place\_turnir[0] +

VARIANT + 1 - rating\_place\_kmax[0] +

VARIANT + 1 - rating\_place\_kopt[0]

;

int index { 0 };

for (int i = 1; i < VARIANT; ++i)

{

int dom\_value = VARIANT + 1 - rating\_place[i];

int block\_value = VARIANT + 1 - rating\_place\_block[i];

int turn\_value = VARIANT + 1 - rating\_place\_turnir[i];

int kmax\_value = VARIANT + 1 - rating\_place\_kmax[i];

int kopt\_value = VARIANT + 1 - rating\_place\_kopt[i];

int sum = dom\_value + block\_value + turn\_value + kmax\_value +

kopt\_value;

if ( winner < sum ) {

winner = sum;

index = i;

}

}

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

{

int dom\_value = VARIANT + 1 - rating\_place[i];

int block\_value = VARIANT + 1 - rating\_place\_block[i];

int turn\_value = VARIANT + 1 - rating\_place\_turnir[i];

int kmax\_value = VARIANT + 1 - rating\_place\_kmax[i];

int kopt\_value = VARIANT + 1 - rating\_place\_kopt[i];

int sum = dom\_value + block\_value + turn\_value + kmax\_value +

kopt\_value;

if ( i == index ) {

// Unix only (light text).

std::cout << "\033[1;31m" << std::setw(0) << names[i] << std::setw(20 - names[i].size()) << block\_value <<

std::setw(8) << dom\_value << std::setw(9) << turn\_value << std::setw(9) <<

kmax\_value << std::setw(9) << kopt\_value << std::setw(13) << sum << "\033[0m" << std::endl;

} else {

std::cout << std::setw(0) << names[i] << std::setw(20 - names[i].size()) << block\_value <<

std::setw(8) << dom\_value << std::setw(9) << turn\_value << std::setw(9) <<

kmax\_value << std::setw(9) << kopt\_value << std::setw(13) << sum << std::endl;

}

}

return 0;

}

//выделение памяти для динамического двухмерного массива

double\*\* createArr(int n, int m)

{

double\*\* A = new double\* [n];

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

{

A[i] = new double[m];

}

// Fill zeros.

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

for (int j = 0; j < m; j++) {

A[i][j] = 0;

}

}

return A;

}

//считать весовые коэффициенты

void readPower(double\* arr, double n, std::ifstream& in,

std::vector<std::string> &pref\_names)

{

std::string name {""};

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

{

in >> name;

pref\_names.emplace\_back( name );

in >> arr[j];

}

}

//считать массив размером NxM из входящего потока

void readFile(double\*\* arr, double n, double m, std::ifstream& in)

{

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

{

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

{

in >> arr[i][j];

}

}

}

//определение доминирующего варианта

std::vector<int> dominate(double\*\* arr, double n, double m)

{

std::vector<int> dom\_str\_Array;

bool dom\_str;

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

{

dom\_str = true;

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

{

if (i == j) continue;

if (arr[i][j] != 1)

{

dom\_str = false;

break;

}

}

if (dom\_str) dom\_str\_Array.push\_back(i);

}

return dom\_str\_Array;

}

//определение блокирующего варианта

std::vector<int> block(double\*\* arr, double n, double m)

{

std::vector<int> block\_str\_Array;

bool block\_str;

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

{

block\_str = true;

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

{

if (i == j) continue;

if (arr[j][i] != 0)

{

block\_str = false;

break;

}

}

if (block\_str) block\_str\_Array.push\_back(i);

}

return block\_str\_Array;

}

//определение турнирного варианта

std::vector<double> turnir(double\*\* arr, const double power[BO], double n,

double m, int number)

{

std::vector<double> turnir\_str\_Array;

bool turnir\_str;

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

{

double sum = 0;

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

{

if (i == j) continue;

if (arr[i][j] == 1)

{

if (arr[j][i] == 0)

{

sum += power[number];

}

else if (arr[j][i] == 1)

{

sum += power[number] / 2;

}

}

}

turnir\_str\_Array.push\_back(sum);

}

return turnir\_str\_Array;

}

//оставление массива для варианта в случае механизма K-max

double\*\* createKarray(double\*\* arr, double n, double m)

{

double\*\* A = createArr(VARIANT, 4);

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

{

double HR0 = 0;

double ER = 0;

double NK = 0;

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

{

if (i == j) continue;

if (arr[i][j] == 1)

{

if (arr[j][i] == 0)

{

HR0 += 1;

}

else if (arr[j][i] == 1)

{

ER += 1;

}

}

if (arr[i][j] == -1)

{

NK += 1;

}

}

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

{

switch (j)

{

case 0:

A[i][j] = HR0 + ER + NK;

break;

case 1:

A[i][j] = HR0 + NK;

break;

case 2:

A[i][j] = HR0 + ER;

break;

case 3:

A[i][j] = HR0;

break;

default:

break;

}

}

}

return A;

};

//определение K-opt вариантов

void createKopt(double\*\* arr, double n, int\* kopt\_Array)

{

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

{

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

{

switch (j)

{

case 0:

if (arr[i][j] == n)

{

kopt\_Array[i] = 1;

}

break;

case 1:

if ((arr[i][j] == (n - 1)) && (arr[i][j] > arr[i][j + 2]))

{

kopt\_Array[i] = 2;

}

break;

case 2:

if ((arr[i][j] == n) && (arr[i][j] > arr[i][j + 1]))

{

kopt\_Array[i] = 3;

}

break;

case 3:

if ((arr[i][j] == (n - 1)) && (arr[i][j] == arr[i][j - 1]) &&

(arr[i][j] == arr[i][j - 2]))

{

kopt\_Array[i] = 4;

}

break;

default:

kopt\_Array[i] = 0;

break;

}

}

}

}

//вывести двумерный массив

void writeArr(double\*\* arr, double n, double m)

{

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

{

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

{

std::cout << arr[i][j] << "\t";

}

std::cout << std::endl;

}

}

//вывести двумерный массив K-opt механизм

void writeArrKopt(double\*\* arr, double n, double m, int\* opt)

{

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

{

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

{

std::cout << arr[i][j] << "\t";

}

switch (opt[i])

{

case 0:

break;

case 1:

std::cout << "максимальный" << "\t";

break;

case 2:

std::cout << "строго максимальный" << "\t";

break;

case 3:

std::cout << "наибольший" << "\t";

break;

case 4:

std::cout << "строго наибольший" << "\t";

break;

default:

break;

}

std::cout << std::endl;

}

}

//уничтожить двумерный массив

void distractionArray(double\*\* arr, int n)

{

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

{

delete[] arr[i];

}

delete[] arr;

}

//расстановка мест

void placeRating(const double arr[VARIANT], int A[VARIANT])

{

double place[VARIANT] = { 0 };

int number[VARIANT];

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

{

number[i] = i + 1;

}

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

{

place[i] = arr[i];

}

std::sort(std::begin(place), std::end(place));

std::reverse(std::begin(place), std::end(place));

int pl = 0;

for (int i = 0; i < VARIANT; ++i)//по массиву place

{

if ((place[i] == place[i - 1]) && (i != 0))

{

continue;

}

for (int j = 0; j < VARIANT; j++)//по массиву arr

{

if (arr[j] == place[i])

{

//A[j] = i + 1;

A[j] = number[pl];

}

}

pl++;

if (place[i] == 0) break;

}

}