**Код до оптимизации и рефакторинга**

using namespace std;

#include <iostream>

#include <random> // для 1-го задания (2 n 3)

void task\_1(int(\*pointMatrix)[2], int\* pSum, int\* pMult1, int\* pMult2);

int task\_2(int(\*pMatrix\_OG), int size);

void task\_3(double\*\* pSource, int rowsM, int colN);

int main()

{

//setlocale(LC\_ALL, "Russian");

start:

int num;

cout << "Задание: ";

cin >> num;

switch (num)

{

case 1:

{

random\_device rd; //the random engine itself.

uniform\_int\_distribution <int> dist(1, 10); //setting the distibution (distance), can change the types to double etc(??)

int matrix[2][2] = {};

int\* pMatrix = &matrix[2][2];

int det = 0, mult1 = 1, mult2;

cout << "Матрица: " << endl;

for (int i = 0; i < 2; i++) //row

{

for (int j = 0; j < 2; j++) //column

{

matrix[i][j] = dist(rd);

cout << matrix[i][j] << " ";

}

cout << "\n";

};

task\_1(matrix, &det, &mult1, &mult2);

cout << "Определитель = " << det << endl;

cout << "Произведение диагональных элементов матрицы: " << mult1 << " и " << mult2 << endl;

break;

}

case 2:

{

random\_device rd;

uniform\_int\_distribution <int> dist(1, 10);

int m, count;

cout << "Введите длинну массива: ";

cin >> m;

int\* matrix\_OG;

matrix\_OG = new int[m];

cout << "Массив: " << endl;

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

{

matrix\_OG[i] = dist(rd);

cout << matrix\_OG[i] << " ";

};

cout << endl;

int count\_1 = task\_2(matrix\_OG, m);

cout << "Наибольший э-т массива: " << \*matrix\_OG << ". Стоит под номером " << count\_1 + 1 << endl;

break;

}

case 3:

{

random\_device rd;

uniform\_real\_distribution <double> dist(-2.0, 2.0);

int m, n;

cout << "Введите число строк (m): ";

cin >> m;

cout << "Введите число столбцов (n): ";

cin >> n;

double\*\* matrix\_OG = new double\* [m]; // создание основной матрицы и транспонированной матрицы. Вторая пустая

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

{

matrix\_OG[i] = new double[n];

}

cout << "Матрица: " << endl;

for (int i = 0; i < m; i++) //row

{

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

{

matrix\_OG[i][j] = dist(rd);

cout << matrix\_OG[i][j] << " ";

}

cout << "\n";

};

task\_3(matrix\_OG, m, n);

break;

}

default:

cout << "invalid input. try again" << endl;

goto start;

break;

}

}

// вычисляющую определитель и произведение диагональных элементов матрицы второго порядка

// Расчет должен производиться в функции, которая получает данные по указателю и возвращает результат по указателю.

void task\_1(int(\*pointMatrix)[2], int\* pSum, int\* pMult1, int\* pMult2) //function that returns void, as we want to modify the values passed by pointers

{

int tempDet = 0, tempSum = 0, tempMult1 = 1, tempMult2 = 1;

tempMult1 = pointMatrix[0][0] \* pointMatrix[1][1];

tempMult2 = pointMatrix[0][1] \* pointMatrix[1][0];

tempDet = tempMult1 - tempMult2;

\*pSum = tempDet;

\*pMult1 = tempMult1;

\*pMult2 = tempMult2;

}

// определяющую максимальный(-ые) элемент одномерного массива, и выводящую его номер

// При вводе/выводе элементов использовать индексы, а обработку элементов массива осуществлять с помощью указателей

// Расчет должен производиться в функции, в которую массив передается по указателю

int task\_2(int(\*pMatrix\_OG), int size)

{

int count = 0;

int max\_value = pMatrix\_OG[0];

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

{

if (pMatrix\_OG[i] > max\_value)

{

max\_value = pMatrix\_OG[i];

count++;

}

}

pMatrix\_OG[0] = max\_value;

return count;

}

// выполняющую транспонирование неквадратной матрицы.

// Матрицы должны храниться в памяти в виде двумерного динамического массива, размерности матриц вводятся пользователем с клавиатуры.

// Предусмотреть генерацию значений матриц как случайных вещественных чисел в диапазоне от -2.0 до 2.0.

void task\_3(double\*\* pSource, int rowsM, int colN)

{

// строки m, столбцы n

// выделяем память для матриц

double\*\* Transported = new double\* [colN];

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

{

Transported[i] = new double[rowsM];

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

{

Transported[i][j] = pSource[j][i];

}

}

cout << "Транспонированная матрица: " << endl;

for (int i = 0; i < colN; i++) //row

{

for (int j = 0; j < rowsM; j++) //column

{

cout << Transported[i][j] << " ";

}

cout << endl;

};

}

**Рефакторинг и оптимизация**

Помимоописанных ниже проблем были удалены некорректные или излишние комментарии. **1. Кусок кода, который необходимо исправить:**

int\* pMatrix = &matrix[2][2];

Что с ним "не так":

* Некорректное использование указателя. &matrix[2][2] обращается за пределы массива.
* Указатель не нужен в этом контексте, поскольку доступ к элементам можно осуществлять напрямую.

Исправление: полностью убрать ненужный указатель

**2. Кусок кода, который необходимо исправить:**

task\_1(matrix, &det, &mult1, &mult2);

Что с ним "не так":

* Указатель &mult2 не инициализирован перед использованием.
* Лучше явно инициализировать переменные перед использованием для ясности.

Исправление:  
int det = 0, mult1 = 1, mult2 = 1; // инициализируем переменные перед передачей по указателю

task\_1(matrix, &det, &mult1, &mult2);

**3. Кусок кода, который необходимо исправить:**

int\* matrix\_OG;

matrix\_OG = new int[m];

Что с ним "не так":

* В коде отсутствует освобождение памяти, выделенной через new. Это может вызвать утечку памяти при большом числе операций.

Исправление:

int\* matrix\_OG = new int[m];

// Освобождаем память после использования

delete[] matrix\_OG;

**4. Кусок кода, который необходимо исправить:**

int count = 0;

int max\_value = pMatrix\_OG[0];

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

{

if (pMatrix\_OG[i] > max\_value)

{

max\_value = pMatrix\_OG[i];

count++;

}

}

Что с ним "не так":

Логика count++ здесь ошибочна, так как она увеличивает счётчик всякий раз, когда находится новый максимальный элемент, а не возвращает индекс максимума.

Это приводит к неверному индексу максимального элемента.

Исправление:

int max\_index = 0;

int max\_value = pMatrix\_OG[0];

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

{

if (pMatrix\_OG[i] > max\_value)

{

max\_value = pMatrix\_OG[i];

max\_index = i;

}

}

return max\_index; // возвращаем правильный индекс максимального элемента

**4. Кусок кода, который необходимо исправить:**

double\*\* Transported = new double\* [colN];

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

{

Transported[i] = new double[rowsM];

}

Что с ним "не так":

* Отсутствует освобождение памяти для динамических массивов.
* Неправильное управление памятью может привести к утечкам.

Исправление:

// Освобождаем память после использования

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

{

delete[] pSource[i];

}

delete[] pSource;

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

{

delete[] Transported[i];

}

delete[] Transported;

Кусок кода, который необходимо исправить:

//default block with goto statement

goto start;

Что с ним "не так":

* Использование goto не рекомендуется, так как это ухудшает читаемость кода и делает его сложным для поддержки.
* Логику лучше переписать через цикл.

Исправление:

bool repeat = true;

while (repeat)

{

int num;

cout << "Задание: ";

cin >> num;

switch (num)

{

case 1:

// код задания 1

break;

case 2:

// код задания 2

break;

case 3:

// код задания 3

break;

default:

cout << "invalid input. try again" << endl;

break;

}

cout << "Повторить? (1 - да, 0 - нет): ";

cin >> repeat;

}

**Окончательный вариант программы после рефакторинга и оптимизации**

#include <iostream>

#include <random>

using namespace std;

void task\_1(int(\*pointMatrix)[2], int\* pSum, int\* pMult1, int\* pMult2);

int task\_2(int(\*pMatrix\_OG), int size);

void task\_3(double\*\* pSource, int rowsM, int colN);

int main()

{

bool repeat = true;

while (repeat)

{

int num;

cout << "Task: ";

cin >> num;

switch (num)

{

case 1:

{

random\_device rd;

uniform\_int\_distribution<int> dist(1, 10);

int matrix[2][2];

int det = 0, mult1 = 1, mult2 = 1;

cout << "Matrix: " << endl;

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

matrix[i][j] = dist(rd);

cout << matrix[i][j] << " ";

}

cout << "\n";

}

task\_1(matrix, &det, &mult1, &mult2);

cout << "Determinant = " << det << endl;

cout << "Product of diagonal elements of the matrix: " << mult1 << " and " << mult2 << endl;

break;

}

case 2:

{

random\_device rd;

uniform\_int\_distribution<int> dist(1, 10);

int m;

cout << "Enter the length of the array: ";

cin >> m;

int\* matrix\_OG = new int[m];

cout << "Array: " << endl;

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

{

matrix\_OG[i] = dist(rd);

cout << matrix\_OG[i] << " ";

}

cout << endl;

int max\_index = task\_2(matrix\_OG, m);

cout << "The largest element of the array: " << matrix\_OG[max\_index] << ". It is at position " << max\_index + 1 << endl;

delete[] matrix\_OG;

break;

}

case 3:

{

random\_device rd;

uniform\_real\_distribution<double> dist(-2.0, 2.0);

int m, n;

cout << "Enter the number of rows (m): ";

cin >> m;

cout << "Enter the number of columns (n): ";

cin >> n;

double\*\* matrix\_OG = new double\* [m];

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

{

matrix\_OG[i] = new double[n];

}

cout << "Matrix: " << endl;

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

{

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

{

matrix\_OG[i][j] = dist(rd);

cout << matrix\_OG[i][j] << " ";

}

cout << "\n";

}

task\_3(matrix\_OG, m, n);

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

{

delete[] matrix\_OG[i];

}

delete[] matrix\_OG;

break;

}

default:

cout << "Invalid input. Try again" << endl;

break;

}

cout << "Repeat? (1 - yes, 0 - no): ";

cin >> repeat;

}

}

void task\_1(int(\*pointMatrix)[2], int\* pSum, int\* pMult1, int\* pMult2)

{

int tempDet = 0, tempMult1 = 1, tempMult2 = 1;

tempMult1 = pointMatrix[0][0] \* pointMatrix[1][1];

tempMult2 = pointMatrix[0][1] \* pointMatrix[1][0];

tempDet = tempMult1 - tempMult2;

\*pSum = tempDet;

\*pMult1 = tempMult1;

\*pMult2 = tempMult2;

}

int task\_2(int(\*pMatrix\_OG), int size)

{

int max\_index = 0;

int max\_value = pMatrix\_OG[0];

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

{

if (pMatrix\_OG[i] > max\_value)

{

max\_value = pMatrix\_OG[i];

max\_index = i;

}

}

return max\_index;

}

void task\_3(double\*\* pSource, int rowsM, int colN)

{

double\*\* Transposed = new double\* [colN];

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

{

Transposed[i] = new double[rowsM];

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

{

Transposed[i][j] = pSource[j][i];

}

}

cout << "Transposed matrix: " << endl;

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

{

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

{

cout << Transposed[i][j] << " ";

}

cout << endl;

}

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

{

delete[] Transposed[i];

}

delete[] Transposed;

}