**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №3**

**по дисциплине «Программирование»**

Тема: **ДВУМЕРНЫЕ СТАТИЧЕСКИЕ МАССИВЫ. УКАЗАТЕЛИ**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 4373 | Некрасова А.В. |  |
| Преподаватель | Глущенко А. Г. |  |

Санкт-Петербург

2024

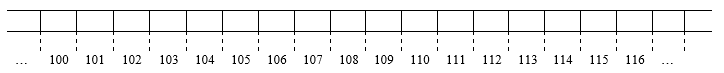
**Цель работы.**

Изучение указателей и работа с ними; получение практических навыков работы с указателями.

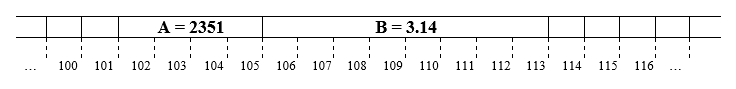
**Основные теоретические положения.**

Указатели и ссылки являются одними из самых важных и достаточно сложных для понимания и использования средств языка программирования. Они ориентированы на прямую работу с памятью компьютера. С помощью этих средств реализуется работа с динамической памятью и динамическими объектами, возвращение из функций измененных данных и многое другое. К использованию указателей и ссылок мы будем неоднократно возвращаться в последующих разделах.

Все данные (переменные, константы и др.) хранятся в памяти. Память представляет собой непрерывную последовательность ячеек (байтов), каждая из которых имеет свой номер – адрес:



При определении, например, некоторой переменной, она располагается в памяти по определенному адресу и занимает столько ячеек, сколько требует тип этой переменной. Пусть, например, имеется переменные int A = 2351 и double B = 3.1 и пусть они располагаются в памяти так:



Говорят, что переменная А располагается по адресу 102 и занимает 4 байта, а переменная B имеет адрес 106 и занимает 8 байт памяти.

Для получения адреса какого-либо программного объекта используется оператор &. Например, если выполнить фрагмент следующей программы (в предположении, что переменные A и B располагаются в памяти, как это показано на предыдущем рисунке):

int A = 2351;

double B = 3.14;

cout <<  “Значение переменной А: ” << A << endl;

cout <<  “Адрес переменной А: ” << &A << endl;

cout <<  “Значение переменной В: ” << В << endl;

cout <<  “Адрес переменной В: ” << &В << endl;

получим следующий результат:

Значение переменной А: 2351

Адрес переменной А: 102

Значение переменной В: 3.14

Адрес переменной В: 106

Правда, значения адресов переменных будут выведены в шестнадцатеричном формате.

Указатели – это тоже обычные переменные, но они служат для хранения адресов памяти.

Указатели определяются в программе следующим образом:

<тип данных> \*<имя переменной>

Здесь <тип данных> определяет так называемый базовый тип указателя.

<Имя переменной> является идентификатором переменной-указателя.

Признаком того, что это переменная указатель, является символ \*, располагающийся между базовым типом указателя и именем переменной-указателя.

Например:

int \*p1;

double \*p2;

Здесь определены две переменные-указатели (или просто – два указателя). Указатель p1 является переменной-указателем на базовый тип int (или, как говорят, переменная p1 указывает  на int - значение), а указатель p2 указывает на double – значение.

Иными словами, переменная p1 предназначена для хранения адресов участков памяти, размер которых соответствует типу int (4 байта), а переменная p2 - для хранения адресов участков памяти, размер которых соответствует типу double (8 байт).

Формально указатели представляют собой обычные целые значения типа int и занимают в памяти 4 байта не зависимо от базового типа указателя. Значения указателей при их выводе на экран представляются как целые значения в шестнадцатеричном формате.

**Постановка задачи.**

Необходимо написать программу, которая:

1)    Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка *N* (6,8,10) случайными числами от 1 до  N\*N согласно схемам, приведенным на рисунках. Пользователь должен видеть процесс заполнения квадратичной матрицы (\*Для манипуляции с элементами используйте только арифметику указателей):



2)    Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами (\*Для манипуляции с элементами используйте только арифметику указателей):



3)    Используя арифметику указателей, сортирует элементы любой сортировкой из списка ниже (если во 2 ПР вы реализовывали одну из представленных сортировок, рекомендуется переиспользовать написанный код и модернизировать его для работы с указатями).   
Варианты сортировок:

* Shaker sort;
* Comb sort;
* Insert sort;
* Quick sort;

4)    Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число (\*Для манипуляции с элементами используйте только арифметику указателей).

**Выполнение работы.**

**#include <iostream>**

**#include <iomanip> // Для std::setw**

**#include <Windows.h> // Для работы с Windows API**

**#include <thread> // Для использования std::this\_thread::sleep\_for**

**#include <random>**

**#include <chrono>**

**using namespace std;**

**const int MAX\_SIZE = 10;**

**void fillMatrixRandom(int\*\* matrix, int N) {**

**for (int i = 0; i < N; ++i) {**

**for (int j = 0; j < N; ++j) {**

**matrix[i][j] = rand() % (N \* N) + 1;**

**}**

**}**

**}**

**void fillMatrixRandom3x3(int\*\* matrix, int N) {**

**for (int i = 0; i < N; ++i) {**

**for (int j = 0; j < N; ++j) {**

**matrix[i][j] = rand() % 61 - 30;**

**}**

**}**

**}**

**int determinant3x3(int\*\* matrix) {**

**int\* row0 = matrix[0];**

**int\* row1 = matrix[1];**

**int\* row2 = matrix[2];**

**//применяем правило треугольника**

**int det = (\*row0) \* (\*(row1 + 1) \* \*(row2 + 2) - \*(row1 + 2) \* \*(row2 + 1)) -**

**(\*(row0 + 1)) \* (\*(row1) \* \*(row2 + 2) - \*(row1 + 2) \* \*(row2)) +**

**(\*(row0 + 2)) \* (\*(row1) \* \*(row2 + 1) - \*(row1 + 1) \* \*(row2));**

**return det;**

**}**

**void printMatrixx(int\*\* matrix, int N) {**

**for (int i = 0; i < N; ++i) {**

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << matrix[i][j];**

**}**

**cout << endl;**

**}**

**}**

**void setCursorPosition(HANDLE hStdout, int x, int y) {**

**COORD coord;**

**coord.X = x;**

**coord.Y = y;**

**SetConsoleCursorPosition(hStdout, coord);**

**}**

**void fillAndPrintMatrix(int\* matrix, int N) {**

**for (int i = 0; i < N; ++i) {**

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << \*(matrix + i \* N + j);**

**}**

**cout << endl;**

**}**

**}**

**void VerticalSnake(HANDLE hStdout, int N) {**

**random\_device rd; //генератор случайных чисел**

**mt19937 gen(rd());**

**uniform\_int\_distribution<> dis(1, N \* N);**

**int value = 1;**

**int startRow = 1;**

**setCursorPosition(hStdout, 0, startRow);**

**cout << "ЗАДАНИЕ 1" << endl;**

**for (int j = 0; j < N; ++j) {**

**if (j % 2 == 0) { //четные столбцы**

**for (int i = 0; i < N; ++i) {**

**setCursorPosition(hStdout, j \* 5, startRow + i + 2); //позиция вывода**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**}**

**else { //нечетные столбцы**

**for (int i = N - 1; i >= 0; --i) {**

**setCursorPosition(hStdout, j \* 5, startRow + i + 2); //позиция вывода**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**}**

**}**

**}**

**void Spiral(HANDLE hStdout, int N) {**

**random\_device rd; //генератор случайных чисел**

**mt19937 gen(rd());**

**uniform\_int\_distribution<> dis(1, N \* N);**

**int value = 1;**

**int left = 0, right = N - 1, top = 0, bottom = N - 1;**

**while (left <= right and top <= bottom) {**

**//заполняем верхнюю строку**

**for (int i = left; i <= right; ++i) {**

**setCursorPosition(hStdout, i \* 5, top + N + 4); //позиция для вывода**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**top++;**

**//заполняем правый столбец**

**for (int i = top; i <= bottom; ++i) {**

**setCursorPosition(hStdout, right \* 5, i + N + 4); //позиция для вывода**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**right--;**

**//заполняем нижнюю строку**

**if (top <= bottom) {**

**for (int i = right; i >= left; --i) {**

**setCursorPosition(hStdout, i \* 5, bottom + N + 4); //позиция для вывода**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**bottom--;**

**}**

**//заполняем левый столбец**

**if (left <= right) {**

**for (int i = bottom; i >= top; --i) {**

**setCursorPosition(hStdout, left \* 5, i + N + 4); //позиция для вывода**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**left++;**

**}**

**}**

**}**

**void printA(int\* matrix, int N) {**

**int halfN = N / 2;**

**int\* quadrants[4]; //массив указателей**

**//выделяем память для каждого блока**

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

**quadrants[i] = new int[halfN \* halfN];**

**}**

**for (int i = 0; i < halfN; ++i) { //перебор эл-ов блоков и заполнение их зн-ями из исходной матрицы**

**for (int j = 0; j < halfN; ++j) {**

**quadrants[0][i \* halfN + j] = \*(matrix + i \* N + j);**

**quadrants[1][i \* halfN + j] = \*(matrix + i \* N + (j + halfN));**

**quadrants[2][i \* halfN + j] = \*(matrix + (i + halfN) \* N + j);**

**quadrants[3][i \* halfN + j] = \*(matrix + (i + halfN) \* N + (j + halfN));**

**}**

**}**

**cout << "Матрица в порядке 4-1-2-3 (A):" << endl;**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[2][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[0][i \* halfN + j];**

**}**

**cout << endl;**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[3][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[1][i \* halfN + j];**

**}**

**cout << endl;**

**}**

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

**delete[] quadrants[i];**

**}**

**}**

**void printB(int\* matrix, int N) {**

**int halfN = N / 2;**

**//создаем массив для хранения блоков**

**int\* quadrants[4];**

**//выделяем память**

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

**quadrants[i] = new int[halfN \* halfN];**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**quadrants[0][i \* halfN + j] = \*(matrix + i \* N + j);**

**quadrants[1][i \* halfN + j] = \*(matrix + i \* N + (j + halfN));**

**quadrants[2][i \* halfN + j] = \*(matrix + (i + halfN) \* N + j);**

**quadrants[3][i \* halfN + j] = \*(matrix + (i + halfN) \* N + (j + halfN));**

**}**

**}**

**cout << "Матрица в порядке 3-4-1-2 (B):" << endl;**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[3][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[2][i \* halfN + j];**

**}**

**cout << endl;**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[1][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[0][i \* halfN + j];**

**}**

**cout << endl;**

**}**

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

**delete[] quadrants[i];**

**}**

**}**

**void printC(int\* matrix, int N) {**

**int halfN = N / 2;**

**int\* quadrants[4];**

**//выделяем память для каждого блока**

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

**quadrants[i] = new int[halfN \* halfN];**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**quadrants[0][i \* halfN + j] = \*(matrix + i \* N + j);**

**quadrants[1][i \* halfN + j] = \*(matrix + i \* N + (j + halfN));**

**quadrants[2][i \* halfN + j] = \*(matrix + (i + halfN) \* N + j);**

**quadrants[3][i \* halfN + j] = \*(matrix + (i + halfN) \* N + (j + halfN));**

**}**

**}**

**cout << "Матрица в порядке 4-3-2-1 (C):" << endl;**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[2][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[3][i \* halfN + j];**

**}**

**cout << endl;**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[0][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[1][i \* halfN + j];**

**}**

**cout << endl;**

**}**

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

**delete[] quadrants[i];**

**}**

**}**

**void printD(int\* matrix, int N) {**

**int halfN = N / 2;**

**//создаем массив для хранения блоков**

**int\* quadrants[4];**

**//выделяем память**

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

**quadrants[i] = new int[halfN \* halfN];**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**quadrants[0][i \* halfN + j] = \*(matrix + i \* N + j);**

**quadrants[1][i \* halfN + j] = \*(matrix + i \* N + (j + halfN));**

**quadrants[2][i \* halfN + j] = \*(matrix + (i + halfN) \* N + j);**

**quadrants[3][i \* halfN + j] = \*(matrix + (i + halfN) \* N + (j + halfN));**

**}**

**}**

**cout << "Матрица в порядке 2-1-4-3 (D):" << endl;**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[1][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[0][i \* halfN + j];**

**}**

**cout << endl;**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[3][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[2][i \* halfN + j];**

**}**

**cout << endl;**

**}**

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

**delete[] quadrants[i];**

**}**

**}**

**void shakerSort(int\* array, int size) {**

**bool swapped = true;**

**int start = 0;**

**int end = size - 1;**

**while (swapped) {**

**swapped = false;**

**for (int i = start; i < end; ++i) {**

**if (\*(array + i) > \*(array + i + 1)) {**

**swap(\*(array + i), \*(array + i + 1));**

**swapped = true;**

**}**

**}**

**if (!swapped) break;**

**--end;**

**swapped = false;**

**for (int i = end - 1; i >= start; --i) {**

**if (\*(array + i) > \*(array + i + 1)) {**

**swap(\*(array + i), \*(array + i + 1));**

**swapped = true;**

**}**

**}**

**++start;**

**}**

**}**

**void shakerSortMatrix(int\* matrix, int N) {**

**shakerSort(matrix, N \* N);**

**cout << "Отсортированная матрица:" << endl;**

**for (int i = 0; i < N; ++i) {**

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << \*(matrix + i \* N + j);**

**}**

**cout << endl;**

**}**

**}**

**void manipulateMatrix(int\* matrix, int N, char operation, int number) {**

**for (int i = 0; i < N \* N; ++i) {**

**switch (operation) {**

**case '+':**

**\*(matrix + i) += number;**

**break;**

**case '-':**

**\*(matrix + i) -= number;**

**break;**

**case '\*':**

**\*(matrix + i) \*= number;**

**break;**

**case '/':**

**if (number != 0)**

**\*(matrix + i) /= number;**

**else**

**cout << "На ноль делить нельзя" << endl;**

**return;**

**}**

**}**

**cout << "Измененная матрица:" << endl;**

**for (int i = 0; i < N; ++i) {**

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << \*(matrix + i \* N + j);**

**}**

**cout << endl;**

**}**

**}**

**int main() {**

**HANDLE hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);**

**setlocale(LC\_ALL, "ru-RU");**

**int N;**

**cout << "Выберите порядок 6 или 8 или 10: ";**

**cin >> N;**

**if (N != 6 and N != 8 and N != 10) {**

**cout << "Введите другое значение" << endl;**

**return 1;**

**}**

**random\_device rd;**

**mt19937 gen(rd());**

**uniform\_int\_distribution<> dis(1, N \* N);**

**int value = 1;**

**int\* matrix = new int[N \* N];**

**//заполнение матрицы по схеме змейки перед разделением на блоки**

**for (int j = 0; j < N; ++j) {**

**if (j % 2 == 0) { //четные**

**for (int i = 0; i < N; ++i) {**

**\*(matrix + i \* N + j) = dis(gen);**

**}**

**}**

**else { //нечетные**

**for (int i = N - 1; i >= 0; --i) {**

**\*(matrix + i \* N + j) = dis(gen);**

**}**

**}**

**}**

**VerticalSnake(hStdout, N);**

**cout << endl;**

**Spiral(hStdout, N);**

**cout << endl << endl << endl;**

**int choice;**

**do {**

**cout << "\nВыберите один из вариантов:\n";**

**cout << "1) Случайная матрица\n";**

**cout << "2) Перестоновка 4-1-2-3 (a)\n";**

**cout << "3) Перестоновка 3-4-1-2 (b)\n";**

**cout << "4) Перестоновка 4-3-2-1 (c)\n";**

**cout << "5) Перестоновка 2-1-4-3 (d)\n";**

**cout << "6) Сортировка shaker sort\n";**

**cout << "7) Операции с матрицей\n";**

**cout << "Ваш выбор: ";**

**cin >> choice;**

**switch (choice) {**

**case 1:**

**cout << "Случайная матрица:\n";**

**fillAndPrintMatrix(matrix, N);**

**break;**

**case 2:**

**printA(matrix, N);**

**break;**

**case 3:**

**printB(matrix, N);**

**break;**

**case 4:**

**printC(matrix, N);**

**break;**

**case 5:**

**printD(matrix, N);**

**break;**

**case 6:**

**shakerSortMatrix(matrix, N);**

**break;**

**case 7:**

**cout << endl;**

**break;**

**default:**

**cout << "Неверный выбор! Попробуйте снова." << endl;**

**}**

**} while (choice != 7);**

**cout << "Исходная матрица:" << endl;**

**for (int i = 0; i < N; ++i) {**

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << \*(matrix + i \* N + j);**

**}**

**cout << endl;**

**}**

**char operation;**

**int number;**

**cout << "Выберите операцию (+, -, \*, / ): ";**

**cin >> operation;**

**cout << "Введите число для операции: ";**

**cin >> number;**

**manipulateMatrix(matrix, N, operation, number);**

**srand(static\_cast<unsigned>(time(0)));**

**int\*\* matrix1 = new int\* [3];**

**N = 3;**

**for (int i = 0; i < N; ++i) {**

**matrix1[i] = new int[N];**

**}**

**fillMatrixRandom3x3(matrix1, N);**

**cout << "\nИДЗ ЗАДАНИЕ 10\nМатрица 3x3:" << endl;**

**for (int i = 0; i < N; ++i) {**

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << matrix1[i][j];**

**}**

**cout << endl;**

**}**

**int det = determinant3x3(matrix1);**

**cout << "\nОпределитель матрицы 3x3: " << det;**

**cout << "\n";**

**for (int i = 0; i < N; ++i) {**

**delete[] matrix1[i];**

**}**

**delete[] matrix1;**

**delete[] matrix;**

**return 0;**

**}**

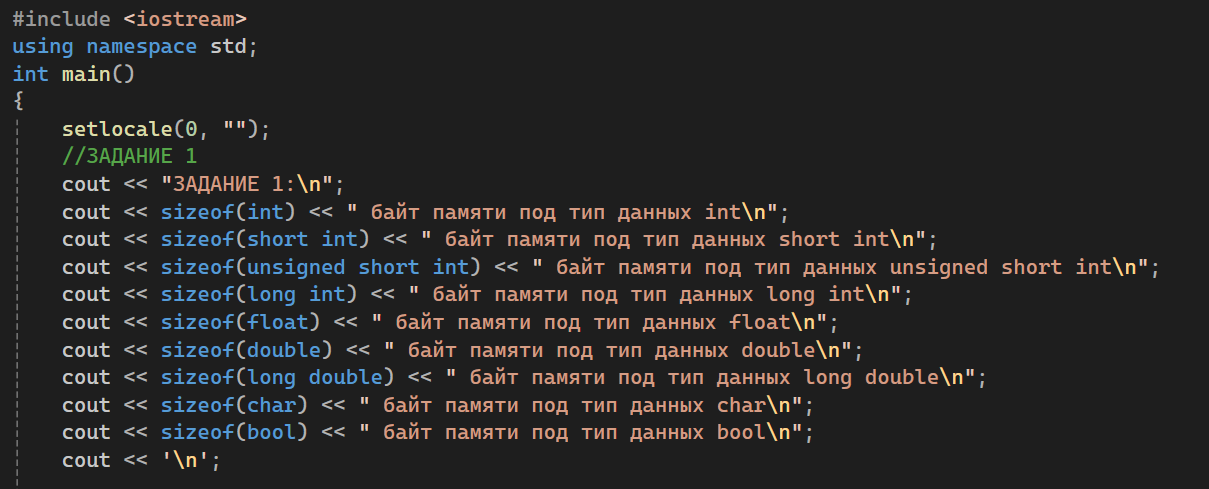
**Выводы.**

Я изучила указатели и поняла, как они представлены в памяти компьютера, научилась работать с ними.

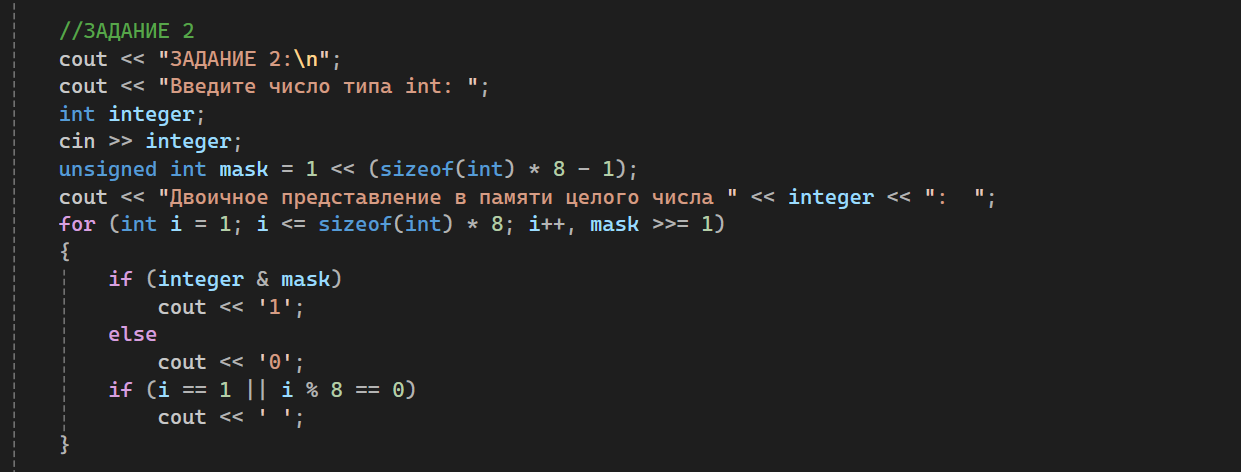
Приложение А

рабочий код

Первое задание:



Второе задание:



Третье задание:



Четвертое задание:

