Дисциплина «Программирование корпоративных систем» Рабочая тетрадь 2.2

Массивы

|  |
| --- |
| **Теоретический материал** |
| Массив представляет набор однотипных данных. Объявление массива похоже на объявление переменной за тем исключением, что после указания типа ставятся квадратные скобки:  тип\_переменной[] название\_массива;  Определим массив целых чисел:  **int[] numbers;**  После определения переменной массива можно присвоить ей определенное значение:  int[] nums = new int[4];  Также мы сразу можем указать значения для этих элементов:  int[] nums2 = new int[4] { 1, 2, 3, 5 };  int[] nums3 = new int[] { 1, 2, 3, 5 };  int[] nums4 = new[] { 1, 2, 3, 5 };  int[] nums5 = { 1, 2, 3, 5 };  Все перечисленные выше способы будут равноценны.  Начиная с версии C# 12 для определения массивов можно использовать выражения коллекций, которые предполагают заключение элементов массива в квадратные скобки:  int[] nums1 = [ 1, 2, 3, 5 ];  int[] nums2 = []; // пустой массив  Для обращения к элементам массива используются индексы. Индекс представляет номер элемента в массиве, при этом нумерация начинается с нуля, поэтому индекс первого элемента будет равен 0, индекс четвертого элемента - 3.  Используя индексы, можно, как получить элементы массива:  int[] numbers = { 1, 2, 3, 5 };  Console.WriteLine(numbers[3]);  //получение эл-та массива 5  Так и изменить элемент массива по индексу:  numbers[1] = 505;  Console.WriteLine(numbers[1]); // 505  Каждый массив имеет свойство Length, которое хранит длину массива. Например, получим длину массива numbers:  int[] numbers = { 1, 2, 3, 5 };  Console.WriteLine(numbers.Length); // 4  Для перебора массивов можно использовать различные типы циклов. Например, цикл **foreach**:  int[] numbers = { 1, 2, 3, 4, 5 };  foreach (int i in numbers)  {  Console.WriteLine(i);  }  Аналогично подобные действия можно сделать и с помощью цикла **for**:  int[] numbers = { 1, 2, 3, 4, 5 };  for (int i = 0; i < numbers.Length; i++)  {  Console.WriteLine(numbers[i]);  }  Также можно использовать и другие виды циклов, например, **while**:  int[] numbers = { 1, 2, 3, 4, 5 };  int i = 0;  while(i < numbers.Length)  {  Console.WriteLine(numbers[i]);  i++;  }  Массивы, которые имеют два измерения (ранг равен 2) называют двухмерными. Например, создадим одномерный и двухмерный массивы, которые имеют одинаковые элементы:  int[] nums1 = new int[] { 0, 1, 2, 3, 4, 5 };  int[,] nums2 = { { 0, 1, 2 }, { 3, 4, 5 } };  Для генерации случайных чисел в программах, написанных на C#, предназначен класс «Random».  //Создание объекта для генерации чисел  Random rnd = new Random(245);    //Получить случайное число (в диапазоне от 0 до 10)  int value = rnd.Next(0, 10);    //Вывод числа в консоль  Console.WriteLine(value); |

|  |  |
| --- | --- |
|  | |
| **Задание 1** | |
| ***Задача:*** | |
|  | **Калькулятор матриц**  Реализуйте программный продукт средствами языка C# со следующим функционалом:   1. Создание двух матриц размерности n\*m (значения n и m вводятся с клавиатуры); 2. Заполнение матриц значениями с клавиатуры (по выбору пользователя, с последующим выводом результата на экран); 3. Заполнение матриц рандомными числами в диапазоне [a; b] (значения a и b вводятся с клавиатуры) (по выбору пользователя, с последующим выводом результата на экран); 4. Сложение матриц (предусмотреть проверку на возможность выполнения операции, с последующим выводом результата на экран); 5. Умножение матриц (предусмотреть проверку на возможность выполнения операции, с последующим выводом результата на экран); 6. Нахождение детерминанта (определителя) матрицы (предусмотреть проверку на возможность выполнения операции, с последующим выводом результата на экран); 7. Нахождение обратной матрицы (предусмотреть проверку на возможность выполнения операции, с последующим выводом результата на экран); 8. Транспонирование матриц (с последующим выводом результата на экран); 9. Нахождение корней системы уравнений, заданных матрицей (с последующим выводом результата на экран).   При тестировании продемонстрировать успешное выполнение всех пунктов (положительный сценарий), а также обработку следующих ситуаций (негативный сценарий):   1. Невозможность сложения матриц по причине несоответствия их размерностей; 2. Невозможность умножения матриц в связи с их несовместимостью; 3. Невозможность нахождения детерминанта у не квадратных матриц (n!=m); 4. Невозможность нахождения обратной матрицы в случае, если детерминант равен нулю (d=0); 5. Невозможность нахождения корней систему уравнений, если она не имеет решения или не имеет однозначного решения.   Весь функционал должен быть реализован вами, программы, разработанные с использованием сторонних решений (библиотеки, фреймворки и т.д.) реализующих функционал, приниматься не будут. |
| ***Решение:*** | |
|  | using System;  Console.Write("Введите количество строк в первой матрице: ");  int stroka\_first = int.Parse(Console.ReadLine());  Console.Write("Введите количество столбцов в первой матрице: ");  int stolbec\_first = int.Parse(Console.ReadLine());  Console.WriteLine();  Console.Write("Как вы хотите заполнить матрицу? (1 - в ручную, 2 - случайными числами в диапазоне [a, b]): ");  int first\_user\_choice = int.Parse(Console.ReadLine());  int[,] first\_matrix = new int[stroka\_first, stolbec\_first];  if (first\_user\_choice == 1)  {  first\_matrix = CreateManualMatrix(stroka: stroka\_first, stolbec: stolbec\_first);  }  else  {  first\_matrix = CreateRandomMatrix(stroka: stroka\_first, stolbec: stolbec\_first);  }  Console.WriteLine();  Console.WriteLine("Вот первая матрица:");  ShowMatrix(matrix: first\_matrix, stroka: stroka\_first, stolbec: stolbec\_first);  Console.WriteLine();  Console.Write("Введите количество строк во второй матрице: ");  int stroka\_second = int.Parse(Console.ReadLine());  Console.Write("Введите количество столбцов во второй матрице: ");  int stolbec\_second = int.Parse(Console.ReadLine());  Console.WriteLine();  Console.Write("Как вы хотите заполнить матрицу? (1 - в ручную, 2 - случайными числами в диапазоне [a, b]): ");  int second\_user\_choice = int.Parse(Console.ReadLine());  int[,] second\_matrix = new int[stroka\_second, stolbec\_second];  if (second\_user\_choice == 1)  {  second\_matrix = CreateManualMatrix(stroka: stroka\_second, stolbec: stolbec\_second);  }  else  {  second\_matrix = CreateRandomMatrix(stroka: stroka\_second, stolbec: stolbec\_second);  }  Console.WriteLine();  ShowMatrix(matrix: second\_matrix, stroka: stroka\_second, stolbec: stolbec\_second);  Console.WriteLine();  while (true)  {  Console.WriteLine("Выберите, что хотите сделать с этими матрицами: ");  Console.WriteLine("1. Сложение матриц друг с другом");  Console.WriteLine("2. Умножение матриц друг на друга");  Console.WriteLine("3. Нахождение определителя первой матрицы");  Console.WriteLine("4. Нахождение определителя второй матрицы");  Console.WriteLine("5. Нахождение обратной матрицы для первой матрицы");  Console.WriteLine("6. Нахождение обратной матрицы для второй матрицы");  Console.WriteLine("7. Транспонирование первой матрицы");  Console.WriteLine("8. Транспонирование второй матрицы");  Console.WriteLine("9. Нахождение корней системы уравнений для первой матрицы");  Console.WriteLine("10. Нахождение корней системы уравнений для второй матрицы");  Console.WriteLine("0. Выход");  Console.Write("Выберите, что вы хотите сделать с матрицами: ");  int user\_choice\_matrix = int.Parse(Console.ReadLine());  Console.WriteLine();  if (user\_choice\_matrix == 0) break;  switch (user\_choice\_matrix)  {  case 1:  AddictionMatrix(first\_matrix, second\_matrix, stroka\_first, stolbec\_first, stroka\_second, stolbec\_second);  break;  case 2:  MultiplyMatrix(first\_matrix, second\_matrix, stroka\_first, stolbec\_first, stroka\_second, stolbec\_second);  break;  case 3:  if (first\_matrix.GetLength(0) != first\_matrix.GetLength(1))  {  Console.WriteLine("Ошибка: матрица должна быть квадратной для вычисления определителя.");  break;  }  else  {  int just\_determinator1 = FindDeterminateMatrix(first\_matrix);  Console.WriteLine($"Определитель матрицы равен: {just\_determinator1}");  }  break;  case 4:  if (second\_matrix.GetLength(0) != second\_matrix.GetLength(1))  {  Console.WriteLine("Ошибка: матрица должна быть квадратной для вычисления определителя.");  break;  }  else  {  int just\_determinator2 = FindDeterminateMatrix(second\_matrix);  Console.WriteLine($"Определитель матрицы равен: {just\_determinator2}");  }  break;  case 5:  if ((first\_matrix.GetLength(0) != first\_matrix.GetLength(1)))  {  Console.WriteLine("Ошибка: чтобы найти обратную матрицу, она должна быть квадратной.");  break;  }  else  {  int reverse\_det = FindDeterminateMatrix(first\_matrix);  if (reverse\_det != 0)  {  ShowReverseMatrix(CreateReverseMatrix(first\_matrix, stroka\_first, stolbec\_first, reverse\_det), stroka\_first, stolbec\_first, reverse\_det);  }  else  {  Console.WriteLine("Ошибка: определитель равен 0 - обратную матрицу невозможно найти");  }  }  break;  case 6:  if ((second\_matrix.GetLength(0) != second\_matrix.GetLength(1)))  {  Console.WriteLine("Ошибка: чтобы найти обратную матрицу, она должна быть квадратной.");  break;  }  else  {  int reverse\_det = FindDeterminateMatrix(second\_matrix);  if (reverse\_det != 0)  {  ShowReverseMatrix(CreateReverseMatrix(second\_matrix, stroka\_second, stolbec\_second, reverse\_det), stroka\_second, stolbec\_second, reverse\_det);  }  else  {  Console.WriteLine("Ошибка: определитель равен 0 - обратную матрицу невозможно найти");  }  }  break;  case 7:  ShowMatrix(TransponirovanieMatrix(first\_matrix, stroka\_first, stolbec\_first), stolbec\_first, stroka\_first);  break;  case 8:  ShowMatrix(TransponirovanieMatrix(second\_matrix, stroka\_second, stolbec\_second), stolbec\_second, stroka\_second);  break;  case 9:  case 10:  int[,] matrixToSolve = (user\_choice\_matrix == 9) ? first\_matrix : second\_matrix;  SolveAndPrintResult(matrixToSolve);  break;  }  }  int[,] CreateRandomMatrix(int stroka, int stolbec)  {  int[,] matrix = new int[stroka, stolbec];  Console.Write("Введите нижнюю границу для случайных чисел: ");  int a = int.Parse(Console.ReadLine());  Console.Write("Введите верхнюю границу для случайных чисел: ");  int b = int.Parse(Console.ReadLine());  Random rnd = new Random();  for (int i = 0; i < stroka; i++)  {  for (int j = 0; j < stolbec; j++)  {  matrix[i, j] = rnd.Next(a, b + 1);  }  }  return matrix;  }  int[,] CreateManualMatrix(int stroka, int stolbec)  {  int[,] matrix = new int[stroka, stolbec];  Console.WriteLine();  for (int i = 0; i < stroka; i++)  {  for (int j = 0; j < stolbec; j++)  {  Console.Write($"Введите число для {i + 1} строки и {j + 1} столбца: ");  matrix[i, j] = int.Parse(Console.ReadLine());  }  }  return matrix;  }  void ShowMatrix(int[,] matrix, int stroka, int stolbec)  {  for (int i = 0; i < stroka; i++)  {  for (int j = 0; j < stolbec; j++)  {  Console.Write($"{matrix[i, j]} ");  }  Console.WriteLine();  }  }  void AddictionMatrix(int[,] matrix1, int[,] matrix2, int stroka1, int stolbec1, int stroka2, int stolbec2)  {  if ((stroka1 == stroka2) && (stolbec1 == stolbec2))  {  int[,] new\_matrix = new int[stroka1, stolbec1];  for (int i = 0; i < stroka1; i++)  {  for (int j = 0; j < stolbec1; j++)  {  new\_matrix[i, j] = matrix1[i, j] + matrix2[i, j];  }  }  Console.WriteLine("В ходе сложения получилась следующая матрица: ");  Console.WriteLine();  ShowMatrix(new\_matrix, stroka1, stolbec1);  }  else  {  Console.WriteLine("К сожалению, сложение невозможно, так как матрицы разных размеров!");  Console.WriteLine();  }  }  void MultiplyMatrix(int[,] matrix1, int[,] matrix2, int stroka1, int stolbec1, int stroka2, int stolbec2)  {  if (stolbec1 == stroka2)  {  int[,] new\_matrix = new int[stroka1, stolbec2];  for (int i = 0; i < stroka1; i++)  {  for (int j = 0; j < stolbec2; j++)  {  new\_matrix[i, j] = 0;  for (int k = 0; k < stolbec1; k++)  {  new\_matrix[i, j] += matrix1[i, k] \* matrix2[k, j];  }  }  }  Console.WriteLine("В ходе умножения получилась следующая матрица: ");  Console.WriteLine();  ShowMatrix(new\_matrix, stroka1, stolbec2);  }  else  {  Console.WriteLine("К сожалению, умножение невозможно, так как количество столбцов в первой матрице не равно количеству строк во второй!");  Console.WriteLine();  }  }  int FindDeterminateMatrix(int[,] matrix)  {  int n = matrix.GetLength(0);  if (n == 1)  return matrix[0, 0];  if (n == 2)  return matrix[0, 0] \* matrix[1, 1] - matrix[0, 1] \* matrix[1, 0];  int det = 0;  for (int j = 0; j < n; j++)  {  det += (j % 2 == 1 ? -1 : 1) \* matrix[0, j] \* FindDeterminateMatrix(GetMinor(matrix, 0, j));  }  return det;  }  int[,] GetMinor(int[,] matrix, int row, int col)  {  int n = matrix.GetLength(0);  int[,] minor = new int[n - 1, n - 1];  int m = 0, k = 0;  for (int i = 0; i < n; i++)  {  if (i == row) continue;  k = 0;  for (int j = 0; j < n; j++)  {  if (j == col) continue;  minor[m, k] = matrix[i, j];  k++;  }  m++;  }  return minor;  }  int[,] TransponirovanieMatrix(int[,] matrix, int stroka, int stolbec)  {  int[,] new\_matrix = new int[stolbec, stroka];  for (int i = 0; i < stolbec; i++)  {  for (int j = 0; j < stroka; j++)  {  new\_matrix[i, j] = matrix[j, i];  }  }  return new\_matrix;  }  int[,] CreateReverseMatrix(int[,] matrix, int stroka, int stolbec, int determinator)  {  int[,] new\_matrix = TransponirovanieMatrix(FindAlgebraiticMatrix(matrix, stroka, stolbec, determinator), stolbec, stroka);  return new\_matrix;  }  int[,] FindAlgebraiticMatrix(int[,] matrix, int stroka, int stolbec, int determinator)  {  int[,] new\_matrix = new int[stroka, stolbec];  for (int i = 0; i < stroka; i++)  {  for (int j = 0; j < stolbec; j++)  {  new\_matrix[i, j] = (((i + j) % 2 == 1 ? -1 : 1) \* FindDeterminateMatrix(GetMinor(matrix, i, j)));  }  }  return new\_matrix;  }  void ShowReverseMatrix(int[,] matrix, int stroka, int stolbec, int determinate)  {  int flag = 0;  for (int i = 0; i < stroka; i++)  {  for (int j = 0; j < stolbec; j++)  {  if (((i + 1) \* 2 > stroka) && (j == stolbec - 1) && (flag == 0))  {  flag = 1;  if (determinate < 1)  {  Console.Write($"{matrix[i, j]} \* -1/{Math.Abs(determinate)}");  }  else  {  Console.Write($"{matrix[i, j]} \* 1/{determinate}");  }  }  else  {  Console.Write($"{matrix[i, j]} ");  }  }  Console.WriteLine();  }  }  double[] SolveEquationSystem(int[,] matrix)  {  int rows = matrix.GetLength(0);  int cols = matrix.GetLength(1);  if (cols != rows + 1)  {  Console.WriteLine("Ошибка: матрица должна иметь размер n x (n+1), где n - количество неизвестных.");  return null;  }  int[,] A = new int[rows, rows];  double[] b = new double[rows];  for (int i = 0; i < rows; i++)  {  for (int j = 0; j < rows; j++)  {  A[i, j] = matrix[i, j];  }  b[i] = matrix[i, rows];  }  int det = FindDeterminateMatrix(A);  if (Math.Abs(det) < 1e-10)  {  Console.WriteLine("Определитель матрицы A равен нулю.");  if (IsConsistent(A, b))  {  Console.WriteLine("Система имеет бесконечно много решений.");  }  else  {  Console.WriteLine("Система не имеет решений.");  }  return null;  }  return GaussMethod(A, b);  }  double[] GaussMethod(int[,] A, double[] b)  {  int n = b.Length;  double[] x = new double[n];  double[,] augmentedMatrix = new double[n, n + 1];  for (int i = 0; i < n; i++)  {  for (int j = 0; j < n; j++)  {  augmentedMatrix[i, j] = A[i, j];  }  augmentedMatrix[i, n] = b[i];  }  for (int k = 0; k < n; k++)  {  int maxRow = k;  for (int i = k + 1; i < n; i++)  {  if (Math.Abs(augmentedMatrix[i, k]) > Math.Abs(augmentedMatrix[maxRow, k]))  {  maxRow = i;  }  }  if (maxRow != k)  {  for (int j = k; j <= n; j++)  {  double temp = augmentedMatrix[k, j];  augmentedMatrix[k, j] = augmentedMatrix[maxRow, j];  augmentedMatrix[maxRow, j] = temp;  }  }  if (Math.Abs(augmentedMatrix[k, k]) < 1e-10)  {  Console.WriteLine("Матрица вырождена, точное решение невозможно.");  return null;  }  for (int i = k + 1; i < n; i++)  {  double factor = augmentedMatrix[i, k] / augmentedMatrix[k, k];  for (int j = k; j <= n; j++)  {  augmentedMatrix[i, j] -= factor \* augmentedMatrix[k, j];  }  }  }  for (int i = n - 1; i >= 0; i--)  {  x[i] = augmentedMatrix[i, n];  for (int j = i + 1; j < n; j++)  {  x[i] -= augmentedMatrix[i, j] \* x[j];  }  x[i] /= augmentedMatrix[i, i];  }  return x;  }  bool IsConsistent(int[,] A, double[] b)  {  int n = A.GetLength(0);  for (int i = 0; i < n; i++)  {  bool allZero = true;  for (int j = 0; j < n; j++)  {  if (Math.Abs(A[i, j]) > 1e-10)  {  allZero = false;  break;  }  }  if (allZero && Math.Abs(b[i]) > 1e-10)  {  return false;  }  }  return true;  }  void SolveAndPrintResult(int[,] matrix)  {  double[] solution = SolveEquationSystem(matrix);  if (solution != null)  {  Console.WriteLine("Решение системы уравнений:");  for (int i = 0; i < solution.Length; i++)  {  Console.WriteLine($"x{i + 1} = {solution[i]:F4}");  }  }  } |
| ***Ответ:*** | |
|  |  |