

Тема 7. Структуры данных. Массивы.

Одномерные массивы.

Многомерные массивы.

Цикл foreach.

Учебные вопросы:

- 1. Введение в структуры данных.**
- 2. Массивы как тип данных. Одномерные массивы.**
- 3. Цикл foreach.**
- 4. Многомерные массивы.**
- 5. Свойства и методы массивов.**
- 6. Библиотека LINQ. Основные свойства и методы.**

1. Введение в структуры данных.

Что такое структуры данных?

- **Определение:** Структура данных — это способ организации и хранения данных, который позволяет эффективно управлять и обрабатывать информацию. Структуры данных определяют, как данные структурированы и как к ним можно получить доступ.
- **Цель:** Основная цель структур данных — оптимизация операций над данными (поиск, добавление, удаление и т.д.) для повышения производительности программ.

Зачем нужны структуры данных?

- Организация данных: Структуры данных помогают организовать данные таким образом, чтобы они были легко доступны и обрабатывались быстро.
- Эффективность: Использование правильной структуры данных может значительно улучшить производительность программы.
- Моделирование реальных проблем: Многие задачи в программировании требуют моделирования сложных систем, и структуры данных помогают в этом.

Классификация структур данных

1. Линейные структуры данных:

Массивы: Фиксированный размер, быстрый доступ по индексу.

Списки (Linked Lists): Гибкий размер, быстрые вставка и удаление.

Стек (Stack): Принцип LIFO (Last In, First Out), используется в алгоритмах обхода и обработки данных.

Очередь (Queue): Принцип FIFO (First In, First Out), используется в задачах управления задачами и обработки данных.

2. Нелинейные структуры данных:

Деревья (Trees): Иерархическая структура, используется в поисковых алгоритмах и организации данных.

Графы (Graphs): Сложные структуры для представления взаимосвязей между объектами, используются в сетевом анализе и маршрутизации.

Хеш-таблицы (Hash Tables): Быстрый доступ к данным по ключу, используется в задачах поиска.

Примеры использования структур данных

- **Массивы:** Используются для хранения упорядоченных данных, таких как списки товаров в интернет-магазине.
- **Списки:** Применяются в случаях, когда требуется часто изменять размер структуры, например, в задачах управления задачами (task management).
- **Стек:** Используется для обработки обратных вызовов (callback) и реализации алгоритмов поиска (например, обход в глубину).
- **Очередь:** Применяется в задачах планирования процессов, очередях обработки задач.
- **Деревья:** Применяются для реализации файловых систем и баз данных.
- **Графы:** Используются для моделирования сетей (например, социальных сетей или компьютерных сетей).
- **Хеш-таблицы:** Используются для организации словарей (например, для быстрого поиска значений по ключу).

Заключение

Структуры данных являются фундаментальной частью программирования, определяющей, как организованы и обработаны данные.

Правильный выбор структуры данных позволяет существенно оптимизировать выполнение программ и решить задачи более эффективно.

Далее будут рассмотрены конкретные структуры данных, такие как массивы и их разновидности, а также способы их использования.

2. Массивы как тип данных.

Одномерные массивы.

Что такое массивы?

Определение: **Массив** — это структура данных, которая представляет собой коллекцию элементов одного типа, расположенных в непрерывной области памяти. Каждый элемент массива имеет уникальный индекс, который позволяет получить к нему доступ.

Типы данных в массивах: В массиве все элементы должны быть **одного типа данных**, например, все элементы могут быть целыми числами (**int**), числами с плавающей точкой (**double**), символами (**char**), строками (**string**), массивами и т.д.

Массив, состоящий из 5 элементов

123	7	50	-9	24
-----	---	----	----	----

0

1

2

3

4

индексы элементов массива

Преимущества массивов

- Простой доступ к элементам: Элементы массива могут быть быстро доступны по их индексу. Время доступа к элементу по индексу — $O(1)$.
- Эффективное использование памяти: Массивы занимают непрерывный блок памяти, что делает их эффективными в плане использования памяти и кэширования.
- Упорядоченность данных: Элементы массива располагаются в определенном порядке, что делает их удобными для сортировки и поиска.

Недостатки массивов

- **Фиксированный размер:** Размер массива задается при его создании и не может быть изменен. Это может привести к избыточному или недостаточному использованию памяти.
- **Однородность данных:** Массивы могут содержать только элементы одного типа данных, что может быть ограничением в некоторых задачах.
- **Операции вставки и удаления:** массивы не поддерживают вставку и удаление элементов. Для "вставки" элемента нужно создать новый массив большего размера, скопировать элементы из исходного массива, добавить новый элемент и вернуть новый массив.

Создание и инициализация массива

Объявление массива:

```
int[] numbers; // Объявление массива целых чисел
```

Создание массива:

```
numbers = new int[5]; // Создание массива из 5 элементов
```

Объявление и создание массива одновременно:

```
int[] numbers = new int[5]; // Одновременное объявление и создание массива
```

Инициализация массива при создании:

```
int[] numbers = { 1, 2, 3, 4, 5 }; // Создание и инициализация массива
```

`byte` – тип элементов массива

имя массива

`[3]` – количество элементов массива

`byte[] array = new byte[3];`

квадратные скобки указывают на то,
что переменная `array` типа `byte` – массив

выражение создания массива

Доступ к элементам массива и их изменение

Доступ к элементам:

```
int firstNumber = numbers[0]; // Доступ к первому элементу массива
```

Изменение значения элемента:

```
numbers[0] = 10; // Изменение значения первого элемента на 10
```

Индексация массива начинается с 0: Первый элемент массива имеет индекс 0, второй — 1 и т.д. Последний элемент массива имеет индекс, равный размеру массива минус 1.

```
// объявляем массив  
int[] age = new int[5];  
  
// инициализируем массив  
age[0] = 12;  
age[1] = 4;  
age[2] = 5;
```

age[0]	age[1]	age[2]	age[3]	age[4]
12	4	5	2	5

Перебор элементов массива

Использование цикла **for**:

```
for (int i = 0; i < numbers.Length; i++)  
{  
    Console.WriteLine(numbers[i]);  
}
```

3. Цикл foreach.

Что такое цикл foreach?

Определение: Цикл foreach — это специальный цикл, который используется для перебора всех элементов коллекции (например, массивов, списков и других типов данных, реализующих интерфейс IEnumerable) без необходимости управления индексами.

Основная идея: Цикл автоматически проходит через каждый элемент коллекции, предоставляя удобный способ обработки элементов без риска выхода за пределы массива или работы с некорректными индексами.

Основной синтаксис:

```
foreach (var element in collection)
{
    // Код, выполняемый для каждого элемента коллекции
}
```

var или конкретный тип данных (int, string, и т.д.) используется для объявления переменной, которая будет представлять текущий элемент коллекции.

element — переменная, которая последовательно принимает значение каждого элемента коллекции.

collection — коллекция или массив, элементы которого нужно перебрать.

Пример использования цикла **foreach**:

```
int[] numbers = { 1, 2, 3, 4, 5 };  
foreach (int number in numbers)  
    Console.WriteLine(number);  
  
string[] cyties= { "Almaty", "Astana", "Aktau" };  
foreach ( string city in cyties)  
    Console.WriteLine(city);
```

Особенности и преимущества использования foreach

- Простота использования: Нет необходимости управлять индексами, что упрощает код и уменьшает вероятность ошибок, таких как выход за пределы массива.
- Чтение данных: Цикл foreach идеально подходит для чтения элементов коллекции, так как не нужно изменять значения элементов или управлять их расположением.
- Неприменимость для модификации: В отличие от цикла for, в цикле foreach нельзя изменять элементы коллекции. Для изменения значений элементов необходимо использовать другой цикл, например, for.

Ограничения цикла foreach

- Только для чтения: Элементы коллекции в цикле `foreach` не могут быть изменены напрямую, так как переменная `element` является копией элемента коллекции. Любые изменения внутри цикла не повлияют на оригинальные данные.
- Не подходит для удаления элементов: Если необходимо удалять элементы из коллекции во время итерации, `foreach` не подходит, так как это может привести к исключениям (например, `InvalidOperationException`).

4. Многомерные массивы.

Определение: Многомерный массив — это массив, элементы которого сами являются массивами.

Многомерные массивы позволяют хранить данные в форме матриц, таблиц или других многомерных структур.

Чаще всего используются двумерные массивы, которые можно представить как таблицу с рядами и столбцами. Однако можно создавать массивы с тремя и более измерениями.


Создание и инициализация многомерных массивов


Объявление и создание двумерного массива:

```
int[,] matrix = new int[3, 3]; // Создание двумерного массива размером 3x3
```

Инициализация двумерного массива:

```
int[,] matrix = {  
    { 1, 2, 3 },  
    { 4, 5, 6 },  
    { 7, 8, 9 }  
};
```


Column 

Row 

	0	1	2
0	1 $x[0, 0]$	2 $x[0, 1]$	3 $x[0, 2]$
1	3 $x[1, 0]$	4 $x[1, 1]$	5 $x[1, 2]$

Создание трехмерного массива:

```
int[, ,] threeDArray = new int[2, 2, 2]; // Создание трехмерного массива 2x2x2
```

Доступ к элементам многомерного массива

Обращение к элементу двумерного массива:

```
int value = matrix[1, 2]; // Доступ ко второму ряду, третьему столбцу (значение 6)
```

Изменение значения элемента:

```
matrix[0, 0] = 10; // Изменение значения первого элемента (левый верхний угол)
```

Перебор элементов многомерного массива

Использование вложенных циклов **for** для двумерного массива:

```
for (int i = 0; i < matrix.GetLength(0); i++) // Перебор рядов
{
    for (int j = 0; j < matrix.GetLength(1); j++) // Перебор столбцов
    {
        Console.Write(matrix[i, j] + " ");
    }
    Console.WriteLine();
}
```

Использование цикла **foreach**:

Цикл `foreach` в C# можно использовать для перебора элементов двумерного массива, хотя при этом нельзя получить доступ к индексам элементов.

Вместо этого перебираются все элементы массива по порядку, начиная с первого элемента первой строки и заканчивая последним элементом последней строки.

Пример:

```
int[,] matrix =  
{  
    { 4, 5, 6 },  
    { 1, 2, 3 },  
    { 7, 8, 9 }  
};  
  
foreach (int element in matrix)  
{  
    // 4 5 6 1 2 3 7 8 9  
    Console.Write(element + " ");  
}
```

Примеры использования многомерных массивов

- Матрицы и таблицы: Двумерные массивы часто используются для хранения данных в виде таблиц. Например, таблица оценок студентов, где строки представляют студентов, а столбцы — предметы.
- Графики и изображения: Двумерные массивы могут использоваться для представления пикселей изображения, где каждый элемент содержит значения цвета пикселя.
- Математические операции: Многомерные массивы часто применяются в математических вычислениях, таких как операции с матрицами (сложение, умножение, транспонирование).

Преимущества и недостатки многомерных массивов

Преимущества:

- Возможность хранения и работы с данными в виде таблиц и многомерных структур.
- Удобство представления данных, имеющих более двух измерений.

Недостатки:

- Сложность управления: Увеличение количества измерений усложняет работу с массивом, включая создание, доступ и перебор элементов.
- Повышенные требования к памяти: Многомерные массивы могут занимать значительно больше памяти, особенно при увеличении числа измерений и размеров массива.

5. Свойства и методы массивов.

Основные свойства массивов

Длина массива (Length):

Свойство **Length** возвращает общее количество элементов в массиве.

Пример:

```
int[] numbers = { 1, 2, 3, 4, 5 };  
int length = numbers.Length; // length будет равно 5
```

Применение: Часто используется для создания циклов, перебирающих элементы массива.

Размерность массива (Rank):

Свойство **Rank** возвращает количество измерений в массиве.

Пример:

```
int[,] matrix = new int[3, 4];  
int rank = matrix.Rank; // rank будет равно 2, так как это двумерный массив
```

Применение: Полезно при работе с многомерными массивами для определения их структуры.

Тип элементов массива (GetType):

Метод **GetType()** возвращает тип данных массива, включая информацию о типе элементов, которые он содержит.

Пример:

```
int[] numbers = { 1, 2, 3 };  
Type type = numbers.GetType(); // Возвращает System.Int32[]
```

Применение: Используется для получения информации о типе данных массива, что может быть полезно для отладки.

Методы для работы с массивами

Сортировка (Array.Sort()):

Метод **Array.Sort()** сортирует элементы массива по возрастанию.

Пример:

```
int[] numbers = { 3, 1, 4, 1, 5 };  
Array.Sort(numbers); // После сортировки: { 1, 1, 3, 4, 5 }
```

Применение: Полезно для упорядочивания данных в массиве.

Обратный порядок (Array.Reverse):

Метод **Array.Reverse()** переворачивает порядок элементов в массиве.

Пример:

```
int[] numbers = { 1, 2, 3, 4, 5 };  
Array.Reverse(numbers); // После обращения: { 5, 4, 3, 2, 1 }
```

Применение: Полезно для упорядочивания данных в массиве.

Поиск элементов (Array.IndexOf, Array.Find):

Array.IndexOf возвращает индекс первого вхождения элемента в массиве.

Пример:

```
int[] numbers = { 1, 2, 3, 4, 5 };  
int index = Array.IndexOf(numbers, 3); // Возвращает 2 (индекс элемента со значением 3)
```

Применение: Эти методы полезны для поиска элементов в массиве по значению.

Очистка массива (Array.Clear):

Метод **Array.Clear()** обнуляет все элементы массива, задавая им значение по умолчанию.

Пример:

```
int[] numbers = { 1, 2, 3, 4, 5 };  
Array.Clear(numbers, 0, numbers.Length); // Все элементы массива станут равны 0
```

Применение: Применяется, когда нужно быстро очистить массив от всех значений.

6. Библиотека LINQ. Основные свойства и методы.

LINQ (Language Integrated Query) — это мощная библиотека в C#, предоставляющая удобные инструменты для выполнения запросов к коллекциям данных, таким как массивы, списки, базы данных, XML-документы и многое другое.

LINQ позволяет писать запросы непосредственно в C# с использованием синтаксиса, напоминающего SQL-запросы.

Пространство имен LINQ.

Для работы с LINQ необходимо подключить пространство имен:

```
using System.Linq;
```

Основные свойства и методы LINQ.

Sum(). Возвращает сумму всех элементов числового массива.

Пример:

```
int[] numbers = { 1, 2, 3 };  
int sum = numbers.Sum();  
// sum = 6
```

Min(). Возвращает минимальное значение в коллекции.

Пример:

```
int[] numbers = { 5, 2, 8 };  
int min = numbers.Min();  
// min = 2
```

Max(). Возвращает максимальное значение в коллекции.

Пример:

```
int[] numbers = { 5, 2, 8 };  
int max = numbers.Max();  
// max = 8
```

Average(). Возвращает среднее арифметическое элементов числового массива.

Пример:

```
int[] numbers = { 1, 2, 3 };  
double average = numbers.Average();  
// average = 2.0
```

Заключение.

Массивы в C# представляют собой одну из базовых структур данных, которая позволяет хранить и организовывать наборы элементов одного типа в памяти.

Основные свойства массивов:

- Фиксированная структура. После создания массивы имеют **неизменяемую длину, размерность и тип данных**, что делает их эффективными для использования в случаях, когда заранее известен объем данных.
- Многомерные массивы. Помимо одномерных массивов, C# поддерживает многомерные массивы (например, двумерные и трехмерные), которые используются для представления сложных данных, таких как таблицы или матрицы.

- Эффективность работы. Массивы обеспечивают постоянное время доступа к элементам ($O(1)$) по индексу, что делает их очень быстрыми для чтения и записи данных.
- Ограничения. **Фиксированный размер** массива может быть недостатком, если данные меняются динамически. Для таких случаев в C# предусмотрены другие коллекции.
- Гибкость и мощьность. Массивы могут быть использованы для выполнения различных операций, таких как сортировка, поиск, агрегация данных и т.д., с помощью встроенных методов и циклов.

Контрольные вопросы:

- Что такое массив и какие его основные характеристики?
- Каковы преимущества и недостатки использования одномерных массивов?
- Опишите, как работает цикл `foreach` и в чем его основные преимущества.
- Как создаются и инициализируются многомерные массивы?
- Какие методы и свойства массивов вы знаете и для чего они применяются?
- В чем разница между одномерными и многомерными массивами?
- Как осуществляется доступ к элементам массива и их изменение?
- Почему массивы считаются эффективными структурами данных для хранения и обработки информации?
- Объясните, как можно перебирать элементы многомерного массива с помощью цикла `for`.
- Какие альтернативные структуры данных можно использовать, если размер данных изменяется динамически?

Домашнее задание:

1. Повторить материал лекции.
2. Решить задачи: Учебное пособие, с. 52 и с. 58.

Материалы лекций:

<https://github.com/ShViktor72/Education>

Обратная связь:

colledge20education23@gmail.com

Список литературы:

1. Жаксыбаева Н.Н. Основы объектно-ориентированного программирования: язык C#. Часть 1./ Учебное пособие предназначено для учащихся технического и профессионального образования, Алматы, 2010,
2. <https://learn.microsoft.com/ru-ru/dotnet/csharp/>