Лекция №6
по курсу
«Системное программирование»

Лектор: д.т.н., Оцоков Шамиль Алиевич,

email: otsokovShA@mpei.ru

Москва, 2021

Коллекции

Коллекция в языке C# — это объединение произвольного количества объектов, возможно, объектов разного типа.

Коллекция — это динамическая структура данных. Память под коллекцию не фиксируется, данные могут добавляться или удаляться по мере необходимости. Кроме того, для многих видов коллекций возможно включение в коллекцию данных разного типа.

Коллекция — это более гибкая организация данных, чем обычный массив. Но работа с массивом всегда ведётся быстрее, чем с коллекцией.

Для работы с коллекциями используется область имён System.Collections. Наиболее популярными видами коллекций являются:

Наиболее часто используемые коллекций .NET

- **List (список)**
- **Dictionary (словарь)**
- **ArrayList (список с разнотипными элементами)**
- **Hashtable (хэш-таблица)**
- **Stack (стэк)**
- **Queue (очередь)**

Коллекции

List (список)

Список позволяет связанно хранить однотипные элементы, динамически выделяя память для последующих элементов. Это значит, что вам не нужно задавать размер списка при его инициализации в отличии от массивов. Доступ к элементу списка возможен по индексу, но обычно элементы списка обрабатываются последовательно.

Упражнения

№1 Напишите программу, которая считывает с консоли последовательность положительных целых чисел. Последовательность заканчивается, когда вводится пустая строка. Вычислите и распечатайте сумму и среднее значение последовательности. Сохраните последовательность в List <int>.

№2 Напишите программу, которая считывает с консоли последовательность положительных целых чисел. Последовательность заканчивается, когда вводится пустая строка. Распечатать последовательность, отсортированную по возрастанию

Упражнения

№1 Напишите метод, который находит самую длинную подпоследовательность равных чисел в данном List <int> и возвращает результат как новый List <int>. Напишите программу, чтобы проверить, правильно ли работает метод.

№2 Напишите программу, которая удаляет все отрицательные числа из последовательности.

Коллекции

Dictionary (словарь)

Словарь — это подходящая коллекция для хранения и последующего поиска однотипных элементов. В словаре элементы хранятся в виде пары «ключ-значение». Это значит, что вы можете сохранить значение, и в дальнейшем используя ключ быстро получить значение из словаря обратно. Тип ключа задается пользователем и может быть любым: числом, строкой или просто объектом.

Коллекции

Главной отличительной особенностью списка Dictionary от списка List заключается в том, что элементы в Dictionary представлены парой Ключ/значение, где каждому ключу соответствует значение.

В отличие от List, Dictionary получает доступ к элементу по его ключу.

Описание коллекции

Dictionary<тип_ключа, тип_значения> имя=new Dictionary<тип_ключа, тип_значения>();

Пример:

```
Dictionary<string, int> ведомость = new Dictionary<string, int> ();
ведомость.Add(«Иванов»,4);
ведомость.Add(«Петров»,5);
ведомость.Add(«Сидоров»,4);
```

Упражнения

№1 Напишите программу, которая удаляет из заданной последовательности все числа, которые встречаются нечетное количество раз.

№2. Напишите программу, которая находит в заданном массиве целых чисел (в диапазоне [0… 1000]), сколько раз встречается каждое из них.

Пример: array = {3, 4, 4, 2, 3, 3, 4, 3, 2}

2 -> 2

3->4

Коллекции

ArrayList (список с разнотипными элементами)

Этот вид списка, в отличие от List, позволяет хранить разнотипные элементы. Это означает, что в одном и том же списке могут одновременно храниться как числа, структуры, так и строки, объекты и null значения. Что делает его значительно более медленным при поиске, чтении и записи элементов, чем его более строго типизированный аналог — List.

Коллекции

Hashtable (хэш-таблица)

По принципу работы хэш-таблица схожа с Dictionary, за тем исключением, что позволяет одновременно использовать разнотипные ключи и хранить разнотипные элементы. Например для первого элемента хэш-таблицы можно использовать ключ в виде числового значения, а для второго, все той же таблицы, — в виде строки.

Коллекции

Stack (стэк)

Стэк — это коллекция для организации хранения элементов по модели LIFO (last-in-first-out), что означает «последний пришел, первый вышел». Коллекция типа стэк незаменима для множества алгоритмов, среди наиболее известных — это алгоритм разбора (парсинга) xml документа.

Стэк

Класс Stack <T> - основные операции

Реализованы все основные операции по работе со стеком:

- Push (T) - добавляет новый элемент вверху стека
- Pop () - возвращает самый высокий элемент и удаляет его из стека
- Peek () - возвращает самый высокий элемент, не удаляя его
- Count - возвращает количество элементов в стеке
- Clear () - извлекает все элементы из стека
- Содержит (T) - проверить, есть ли в стеке элемент
- ToArray () - возвращает массив, содержащий все элементы стека

Стэк

Напишите программу, которая считывает с консоли N целых чисел и выводит их в обратном порядке. Используйте класс Stack <int>.

```
static void Main()
  {
  Stack<string> stack = new Stack<string>();
  stack.Push("1. John");
  stack.Push("2. Nicolas");
  stack.Push("3. Mary");
  stack.Push("4. George");
  Console.WriteLine("Top = " + stack.Peek());
  while (stack.Count > 0)
  {
  string personName = stack.Pop();
  Console.WriteLine(personName);
  }
  }
```

Стэк

Напишите программу, которая проверяет, правильно ли помещены круглые скобки в арифметическое выражение. Пример выражения с правильно расставленными скобками: ((a + b) / 5-d). Пример неверного выражения:) (a + b)).

Указание.

Спецификация стека позволяет нам проверить, имеет ли встреченная скобка соответствующая закрывающая скобка. Когда мы встречаем открывающуюся скобку, мы добавляем ее в стек. Когда мы встречаем закрывающую скобку, мы удаляем элемент из стека. Если стек становится пустым до завершения программы в момент, когда нам нужно удалить элемент, скобки ставятся неправильно. То же самое остается, если в конце выражения есть элементы в стеке

Коллекции

Queue (очередь)

Очередь реализует другую модель добавления и удаления элементов — FIFO (first-in-first-out), что означает «первый пришел, первый вышел». Но в отличие от стэка очередь не так популярна, так как ту же функциональность может обеспечить коллекция List с лучшей производительностью. Очередь обычно используется для изящности кода при небольшой размере коллекции.

Коллекции

```
static void Main()
{
Queue<string> queue = new Queue<string>();
queue.Enqueue("Message One");
queue.Enqueue("Message Two");
queue.Enqueue("Message Three");
queue.Enqueue("Message Four");
while (queue.Count > 0)
{
string msg = queue.Dequeue();
Console.WriteLine(msg);
}
}
```

Коллекции

Коллекции подразделяются на универсальные и неуниверсальные. Универсальные коллекции могут быть типизированы и содержать объекты определенного типа. Это позволяет выигрывать в производительности, особенно, если коллекции хранят типы значений (int, bool, double и т.п.). В неуниверсальных коллекциях происходит упаковка хранимых объектов в тип Object, что требует их последующей распаковки при работе с ними.

Методы Коллекций List и ArrayList

Добавление элементов

- **Add(элемент_коллекции);** - добавляет элемент в коллекцию. В случае универсальной коллекции тип элемента должен совпадать с типом коллекции (пример был приведен выше)
- **AddRange(Коллекция);** - добавляет в текущую коллекцию совокупность элементов заданной коллекции. При этом, коллекция, указанная в качестве параметра, должна реализовывать интерфейс ICollection (стандартный интерфейс коллекции, замечание значимо при использовании пользовательской коллекции).
- **Insert(место, элемент);** - вставляет в заданное место (индекс элемента) заданный элемент
- **InsertRange(место, коллекция);** - вставляет в заданное место в коллекции заданную коллекцию

Методы Коллекций List и ArrayList

Удаление элементов

- **Remove(элемент);** - удаляет элемент из коллекции (первое вхождение элемента)
- **RemoveAt(позиция);** - удаляет элемент на заданной позиции
- **RemoveRange(позиция, количество);** - удаляет заданное количество элементов, начиная с заданной позиции
- **Clear();** - удаляет все элементы из коллекции

Методы Коллекций List и ArrayList

Поиск элементов

- bool **Contains(элемент);** - возвращает, содержится ли заданный элемент в коллекции
- int **IndexOf(элемент);** - возвращает позицию, на которой содержится в коллекции заданный элемент (первое вхождение)
- int **IndexOf(элемент, позиция);** - возвращает позицию, на которой содержится заданный элемент (первое вхождение). Поиск осуществляется по коллекции, начиная с заданной и заканчивая концом коллекции)

Методы Коллекций List и ArrayList

Поиск элементов

- int **IndexOf(элемент, стартовая_позиция, финишная_позиция);** - возвращает позицию, на которой содержится заданный элемент (первое вхождение). Поиск осуществляется по коллекции, начиная с заданной стартовой позиции и заканчивая заданной финишной позицией.
- int **LastIndexOf(элемент);** - возвращает позицию, на которой содержится в коллекции заданный элемент (последнее вхождение). Метод перегружен, его перегруженные версии копируют перегрузки метода IndexOf(), рассмотренного выше
- int **BinarySearch(элемент);** - возвращает позицию, на которой находится заданный элемент в коллекции. По коллекции осуществляется бинарный (двоичный) поиск. Коллекция должная быть отсортирована. Метод содержит несколько перегрузок, можно также задавать компаратор (метод сравнения) для элементов.

Методы Коллекций List и ArrayList

Дополнительные методы для работы с коллекциями

- **Sort(); -** сортирует коллекцию, используя стандартный компаратор. Если тип элементов коллекции является пользовательским, то компаратор (сравнение элементов) нужно реализовать в соответствующем классе этого типа.
- string **ToString(); -** стандартный метод, переводящий коллекцию в строку
- **Reverse(); -** переворачивает коллекцию, точнее, порядок элементов в коллекции. Первый элемент меняется с последним и т.д.
- **Reverse(стартовая_позиция, количество_элементов);** - переворачивает в коллекции заданное количество элементов, начиная с заданного номера элемента

Методы Коллекций List и ArrayList

Дополнительные методы для работы с коллекциями

- **GetRange(стартовая_позиция, количество элементов);** - возвращает коллекцию, элементами которой является заданное количество элементов данной коллекции, начиная с заданной стартовой позиции
- **GetType(); -** возвращает тип элементов коллекции
- **CopyTo(массив);** - копирует элементы коллекции в заданный массив
- **CopyTo(массив, номер_позиции);** - копирует элементы коллекции в заданный массив, начиная с заданной позиции в массиве
- **CopyTo(массив, номер_позиции, количество_элементов);** - копирует заданное количество элементов коллекции в заданный массив, начиная с заданной позиции в массиве

Методы Коллекций List и ArrayList

Перебор элементов списка

Перебрать все значения списка можно с помощью операторов **foreach:**

```
var list = new List<string>();
  list.Add("Я ");
  list.Add("люблю ");
  list.Add("C#");
  foreach (var item in list) // Перебираем элементы списка
                             //  с помощью foreach.
        { Console.WriteLine(item);
        }
```

Локальным переменным вместо определенного типа может быть задан неявный "тип" var. Ключевое слово var сообщает компилятору необходимости определения типа переменной из выражения, находящегося с правой стороны оператора инициализации.

Методы Коллекций List и ArrayList

```
static void Main(string[] args)
    {
        string s;
        StreamReader readfl;
        List<string> st = new List<string>();
        if (File.Exists(@"d:\строки.txt"))
            readfl = new StreamReader(@"d:\строки.txt");
        else
        {
            Console.WriteLine("Файл не найден");
            Console.ReadKey();
            return;
        }
```

Методы Коллекций List и ArrayList

```
while (readfl.EndOfStream == false)
        {  s = readfl.ReadLine();
           st.Add(s);
        }
 foreach (var x in st)
           Console.WriteLine(x);
 Console.WriteLine();
 st.Sort();
 st.Remove("группа");
 foreach (var x in st)
           Console.WriteLine(x);
 Console.ReadKey();
     }
```