



Синтаксис С#

(списки, словари, стеки, очереди)

Андрей Голяков

Список `List<T>`

Класс `List<T>` представляет простейший список однотипных объектов.

Определен в неймспейсе `System.Collections.Generic`.

Среди методов списка можно выделить следующие:

- `void Add(T item)`: добавление нового элемента в список
- `void AddRange(ICollection collection)`: добавление в список коллекции или массива
- `void Clear()`: очищает список
- `int IndexOf(T item)`: возвращает индекс первого вхождения элемента в списке
- `void Insert(int index, T item)`: вставляет элемент `item` в списке на позицию `index`
- `bool Remove(T item)`: удаляет элемент `item` из списка, и если удаление прошло успешно, то возвращает `true`
- `void RemoveAt(int index)`: удаление элемента по указанному индексу `index`
- `void Sort()`: сортировка списка



Добавление элементов в список (Add)

```
var intList = new List<int>();
intList.Add(10);
intList.Add(20);
intList.Add(30);
intList.Add(40);

var l = string.Join(", ", intList);
Console.WriteLine(l);
// 10, 20, 30, 40
```

```
var strList = new List<string>();
strList.Add("one");
strList.Add("two");
strList.Add("three");
strList.Add("four");
strList.Add("four");
strList.Add(null);
strList.Add(null);

var m = string.Join(", ", strList);
Console.WriteLine(m);
// one, two, three, four, four, ,
```

Инициализация списка при объявлении

```
List<int> intList = new List<int> { 10, 20, 30, 40 };
```

```
var l = string.Join(", ", intList);  
Console.WriteLine(l);  
// 10, 20, 30, 40
```

```
var strList = new List<string> { "one", "two", "three", "four" };  
var m = string.Join(", ", strList);  
Console.WriteLine(m);  
// one, two, three, four
```



Изменение набора значений

```
List<int> intList = new List<int> { 10, 20, 30, 40 };  
intList.AddRange(new[] { 50, 60, 70, 80 }); // add new 4 elements  
Console.WriteLine(string.Join(", ", intList));  
// 10, 20, 30, 40, 50, 60, 70, 80
```

```
intList.RemoveRange(4, 2); // remove 2 elements from index 4  
Console.WriteLine(string.Join(", ", intList));  
// 10, 20, 30, 40, 70, 80
```



Изменение набора значений

```
var list = new List<string> { "10", "20", "30", "40" };  
Console.WriteLine(list.IndexOf("30")); // 2  
  
list.Insert(3, "35"); // insert 35 in 3 position of list  
Console.WriteLine(string.Join(", ", list)); // 10, 20, 30, 35, 40  
  
list.Remove("20"); // remove element with value 20 if exists  
Console.WriteLine(string.Join(", ", list)); // 10, 30, 35, 40  
  
list.RemoveAt(0); // remove element in 0 position  
Console.WriteLine(string.Join(", ", list)); // 30, 35, 40  
  
list.Clear(); // clean up all the list  
Console.WriteLine(list.Count); // 0
```



Самостоятельная работа

- Написать приложение, которое будет спрашивать значения типа `double` до тех пор, пока не введено слово “stop”.
- Когда оно введено необходимо завершить цикл запрашивания значений и рассчитать сумму и среднее арифметическое введенных величин.
- Если введено нечисловое значение
 - перехватить исключение,
 - вывести в консоль сообщение об ошибке и остановке программы,
 - пробросить оригинальный эксепшн с помощью ключевого слова `throw`.



Вариант решения

```
var list = new List<double>();
Console.WriteLine("Enter double precision float values (enter \"stop\" to finish");
do
{
    string str = Console.ReadLine();
    if (str.Equals("stop", StringComparison.InvariantCultureIgnoreCase))
        break;

    try
    {
        list.Add(double.Parse(str));
    }
    catch (FormatException)
    {
        Console.WriteLine("Error! Not a number entered! Aborting");
        throw;
    }
} while (true);

double sum = 0;
foreach (double d in list)
    sum += d;
double avg = sum / list.Count;
Console.WriteLine($"Sum: {sum:###.###}, Average: {avg:###.###}");
```



Словарь Dictionary<T1, T2>

Еще один распространенный тип коллекции представляют словари. Словарь хранит объекты, которые представляют пару ключ-значение. Каждый такой объект является объектом структуры `KeyValuePair<TKey, TValue>`.

Благодаря свойствам `Key` и `Value`, которые есть у данной структуры, мы можем получить ключ и значение элемента в словаре.

Определен в неймспейсе `System.Collections.Generic`.

Имеет большинство методов сходных с методами списков.

Для определения есть ли в словаре элемент с заданным ключом, используется метод `HasKey()`

Словарь **не может** хранить элементы с одинаковыми ключами!



Инициализация и работа со словарем

```
Dictionary<int, string> countries = new Dictionary<int, string>(5);  
countries.Add(1, "Russia");  
countries.Add(3, "Great Britain");  
countries.Add(2, "USA");  
countries.Add(4, "France");  
countries.Add(5, "China");
```

```
foreach (KeyValuePair<int, string> keyValue in countries)  
    Console.WriteLine($"{keyValue.Key} - {keyValue.Value}");
```

```
// getting elements by key  
string country = countries[4];  
// changing of the value by index  
countries[4] = "Spain";  
// removing by key  
countries.Remove(2);
```

```
// #4 is "France"
```

```
// #4 has changed, now it "Spain"
```

```
// now "Great Britain" deleted
```



Самостоятельная работа

Написать приложение-игру:

- Программа хранит небольшой список стран и соответствующих им столиц
- Пользователя циклически спрашивают столицу страны в случайном порядке до тех пор, пока он не ошибется
- Если пользователь угадал столицу, его нужно похвалить.
- При ошибке, сообщаем, что пользователь ошибся и выходим из приложения



Вариант решения

```
Dictionary<string, string> countries = new Dictionary<string, string>
{
    { "Россия", "Москва" },
    { "Франция", "Париж" },
    { "Германия", "Берлин" },
    { "Великобритания", "Лондон" }
};

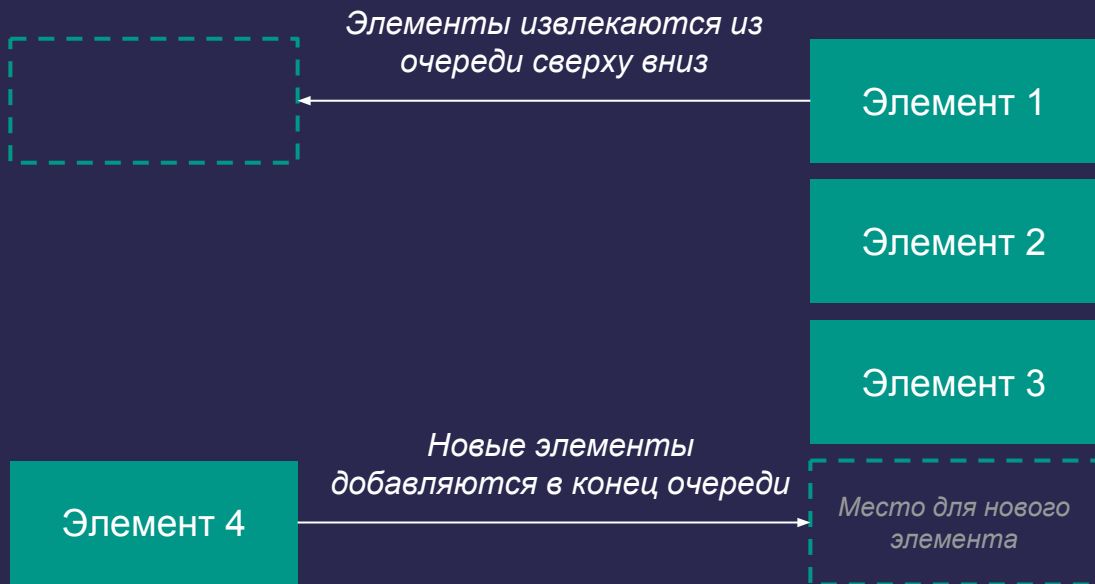
while (true)
{
    int index = (new Random()).Next(4);
    KeyValuePair<string, string> kvp = countries.ElementAtOrDefault(index);
    string country = kvp.Key;
    string capital = kvp.Value;

    Console.Write($"Введите столицу страны \"{country}\": ");
    var answer = Console.ReadLine()?.Trim();
    if (answer == capital)
        Console.WriteLine("Правильно!");
    else
    {
        Console.WriteLine("Вы проиграли :( Выходим...");
        break;
    }
}
```



Очередь `Queue<T>`

Представляет коллекцию объектов, основанную на принципе “**первым** поступил — **первым** обслужен” (First In, First Out: FIFO).



Очередь `Queue<T>`

Основные свойства и методы очереди:

- Свойство `Count` возвращает количество элементов в очереди
- Метод `Clear()` очищает очередь
- Метод `Contains()` проверяет очередь на наличие указанного элемента
- Метод `Dequeue()` читает и удаляет элемент из головы очереди.
Если на момент вызова метода `Dequeue()` элементов в очереди больше нет, генерируется исключение `InvalidOperationException`
- Метод `Enqueue()` добавляет элемент в конец очереди
- Метод `Peek()` читает элемент из головы очереди, но не удаляет его



Инициализация и работа с очередью

```
Queue<string> numbers = new Queue<string>();
numbers.Enqueue("one");
numbers.Enqueue("two");
numbers.Enqueue("three");
numbers.Enqueue("four");
numbers.Enqueue("five");

// A queue can be enumerated without disturbing its contents.
foreach (string number in numbers)
{
    Console.WriteLine(number);
}

while (numbers.Count > 0)
{
    string n = numbers.Dequeue();
    Console.Write($"Processing \"{n}\"... ");
    // here we can really do something with dequeued element :)
    Console.WriteLine("OK");
}
```



Самостоятельная работа

Написать приложение которое будет запрашивать у пользователя целые числа для отложенного вычисления (по команде) квадратного корня до тех пор, пока пользователь не введет одну из двух команд:

- При вводе команды “run” программа должна вывести на экран расчеты по всем задачам, накопившимся в очереди.
- При вводе команды “exit” программа выводит число оставшихся задач в очереди на момент выхода и завершается.



Вариант решения

```
Queue<int> queue = new Queue<int>();

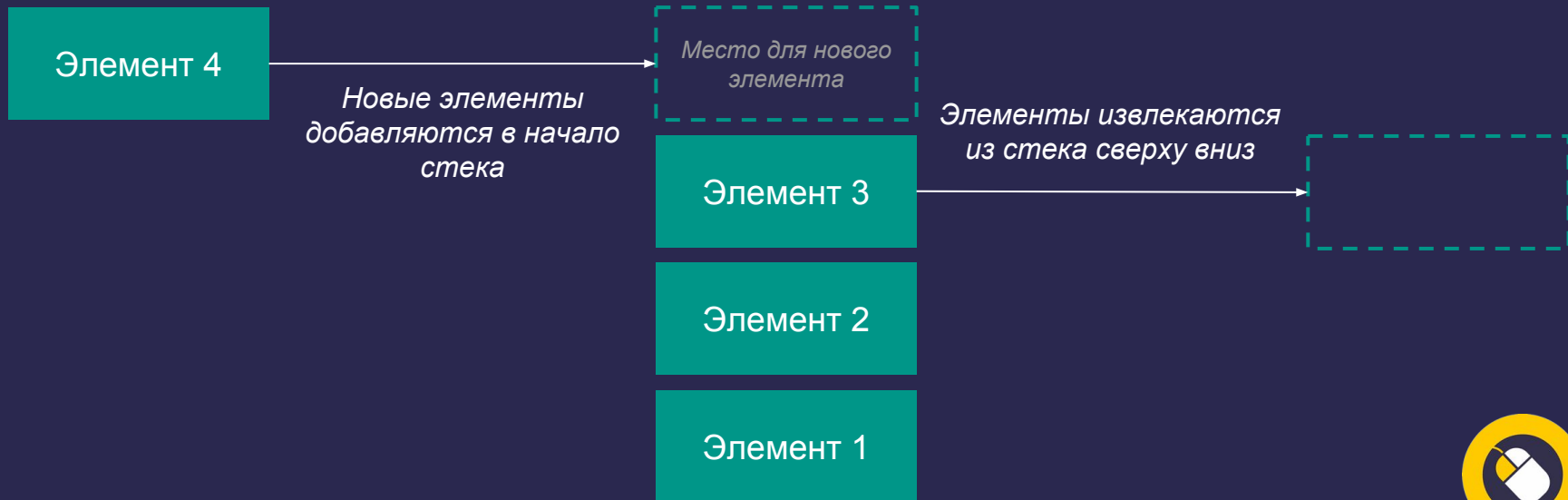
Console.WriteLine("Вводите числа для создания отложенных задач возведения их в квадрат.");
Console.WriteLine("Введите \"run\" чтобы запустить расчет, или \"exit\" чтобы завершить работу.");
string data;
while (true)
{
    data = Console.ReadLine().ToLower();
    if (data == "run")
    {
        while (queue.Count > 0)
        {
            int number = queue.Dequeue();
            Console.WriteLine($"{number}^2 = {number * number}");
        }
        continue;
    }
    else if (data == "exit")
    {
        Console.WriteLine($"Выходим. Число задач в очереди на момент выходы: {queue.Count}");
        break;
    }

    queue.Enqueue(int.Parse(data));
}
```



Стек `Stack<T>`

Представляет коллекцию переменного размера экземпляров одинакового заданного типа, обслуживаемую по принципу "**последним** пришел — **первым** вышел" (Last In, First Out: LIFO)



Стек `Stack<T>`

Основные свойства и методы стека:

- Свойство `Count` возвращает количество элементов в стеке
- Метод `Clear()` очищает стек
- Метод `Contains()` проверяет стек на наличие указанного элемента
- Метод `Push()` добавляет элемент на вершину стека
- Метод `Pop()` читает и удаляет элемент с вершины стека.
Если на момент вызова метода `Dequeue()` элементов в очереди больше нет, генерируется исключение `InvalidOperationException`
- Метод `Peek()` читает элемент с вершины стека, но не удаляет его



Инициализация и работа со стеком

```
Stack<string> numbers = new Stack<string>();
numbers.Push("one");
numbers.Push("two");
numbers.Push("three");
numbers.Push("four");
numbers.Push("five");

// A stack can be enumerated without disturbing its contents.
foreach (string number in numbers)
{
    Console.WriteLine(number);
}

while (numbers.Count > 0)
{
    string n = numbers.Pop();
    Console.Write($"Processing \"{n}\"... ");
    // here we can really do something with popped element :)
    Console.WriteLine("OK");
}
```



Самостоятельная работа

Написать приложение, которое будет запрашивать у пользователя одну из трех команд – “wash”, “dry”, или “exit”.

- Если пользователь вводит “wash”, то мы кладем в стек очередную “тарелку”.
- Если пользователь вводит “dry” мы смотрим, есть ли тарелки в стеке и если есть, то удаляем “тарелку” с вершины стека.
- Если пользователь вводит “exit”, завершаем работу программы.
- После ввода каждой команды программа должна выводить количество тарелок в стопке на вытирание.
- Если вы хотите вытереть тарелку, а тарелок в стопке для вытирания нет, выведите сообщение “Стопка тарелок пуста!”

Считаем, что в раковине бесконечное число тарелок :)



Вариант решения

```
Stack<bool> stack = new Stack<bool>();
Console.WriteLine("Введите \"wash\", чтобы добавить тарелку в стопку помытых.");
Console.WriteLine("Введите \"dry\" чтобы взять тарелку с вершины стопки для вытирания.");
Console.WriteLine("Введите \"exit\" чтобы завершить работу.");
string data;
bool continueWork = true;

while (continueWork)
{
    data = Console.ReadLine().ToLower();
    switch (data)
    {
        case "wash":
            stack.Push(true);
            break;
        case "dry":
            if (stack.Count > 0)
                stack.Pop();
            else
                Console.WriteLine("Стопка тарелок пуста!");
            break;
        case "exit":
            continueWork = false;
            break;
    }

    Console.WriteLine($"Тарелок стопке на вытирание: {stack.Count}");
}
```



Домашнее задание

Написать консольное приложение, которое будет проверять расстановку круглых и квадратных скобок в строке на “правильность” по следующему алгоритму:

Строка считается корректной, если закрывающаяся скобка соответствует последней открытой, но не закрытой скобке.

Проверить алгоритм на таких примерах:

```
var s1 = "()";           // True
var s2 = "[]()";        // True
var s3 = "[[]()]";      // True
var s4 = "([([])])()[]"; // True

var s5 = "(";           // False
var s6 = "[[])";        // False
var s7 = "[()]";        // False
var s8 = "(()[]]";      // False
```



Спасибо за внимание.

