**ПРАКТИЧЕСКАЯ РАБОТА № 23**

Массивы объектов . Коллекции

***Цель работы:*** Овладение приемами разработки простейших классов на языке С#. Получение практических навыков написания программ с использованием массива объектов.

|  |  |  |
| --- | --- | --- |
| №  п/п | Ф.И.О. | НОМЕР ВАРИАНТА |
|  | Барбарас Александр | 11,1 |
|  | Бортник Иван Артемович | 6,2 |
|  | Бура Артур Дмитриевич | 10,3 |
|  | Вернер Андрей Уриелович | 7,4 |
|  | Дабежа Максим Сергеевич | 6,5 |
|  | Давискиба Максим Александрович | 9,2 |
|  | Деде Иван Дмитриевич | 1,8 |
|  | Дидур Даниил Юрьевич | 8,10 |
|  | Доцин Даниил Владимирович | 3,11 |
|  | Журба Иван Сергеевич | 9,3 |
|  | Занделов Данил Павлович | 1,12 |
|  | Иордатий Кирилл Валерьевич | 4,5 |
|  | Кирчу Игорь | 7,1 |
|  | Кырлан Александра Олеговна | 6,12 |
|  | Лозинский Даниил Максимович | 2,9 |
|  | Макаренко Данил Александрович | 5,10 |
|  | Мартынов Даниил Анатольевич | 5,11 |
|  | Марущак Анатолий Сергеевич | 4,3 |
|  | Морошан Никита Павлович | 3,7 |
|  | Мотынга Иван Александрович | 8,9 |
|  | Павлин Кирилл Юрьевич | 9,7 |
|  | Розенцвит Михаил Александрович | 4,12 |
|  | Сагайдак Глеб Андреевич | 10,1 |
|  | Сакара Анна Ивановна | 11,12 |
|  | Сулак Екатерина Александровна | 5,2 |
|  | Ушаков Геннадий Геннадьевич | 2,3 |
|  | Чайковский Андриан Андреевич | 6,7 |
|  | Чепалыга Амалия Максимовна | 8,12 |
|  | Шевченко Александр Александрович | 5,7 |
|  | Шеленков Кирилл Вячеславович | 6,11 |

Задания для самостоятельной работы

Итак, класс **ArrayList** представляет коллекцию объектов. И если надо сохранить вместе разнотипные объекты - строки, числа и т.д., то данный класс как раз для этого подходит.

Основные методы класса:

* int Add(object value): добавляет в список объект value
* void AddRange(ICollection col): добавляет в список объекты коллекции col, которая представляет интерфейс ICollection - интерфейс, реализуемый коллекциями.
* void Clear(): удаляет из списка все элементы
* bool Contains(object value): проверяет, содержится ли в списке объект value. Если содержится, возвращает true, иначе возвращает false
* void CopyTo(Array array): копирует текущий список в массив array.
* ArrayList GetRange(int index, int count): возвращает новый список ArrayList, который содержит count элементов текущего списка, начиная с индекса index
* int IndexOf(object value): возвращает индекс элемента value
* void Insert(int index, object value): вставляет в список по индексу index объект value
* void InsertRange(int index, ICollection col): вставляет в список начиная с индекса index коллекцию ICollection
* int LastIndexOf(object value): возвращает индекс последнего вхождения в списке объекта value
* void Remove(object value): удаляет из списка объект value
* void RemoveAt(int index): удаляет из списка элемент по индексу index
* void RemoveRange(int index, int count): удаляет из списка count элементов, начиная с индекса index
* void Reverse(): переворачивает список
* void SetRange(int index, ICollection col): копирует в список элементы коллекции col, начиная с индекса index
* void Sort(): сортирует коллекцию

Кроме того, с помощью свойства Count можно получить количество элементов в списке.

Посмотрим применение класса на примере.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | using System;  using System.Collections;    namespace Collections  {      class Program      {          static void Main(string[] args)          {              ArrayList list = new ArrayList();              list.Add(2.3); // заносим в список объект типа double              list.Add(55); // заносим в список объект типа int              list.AddRange(new string[] { "Hello", "world" }); // заносим в список строковый массив                // перебор значений              foreach (object o in list)              {                  Console.WriteLine(o);              }                // удаляем первый элемент              list.RemoveAt(0);              // переворачиваем список              list.Reverse();              // получение элемента по индексу              Console.WriteLine(list[0]);              // перебор значений              for (int i = 0; i < list.Count; i++)              {                  Console.WriteLine(list[i]);              }                Console.ReadLine();          }      }  } |

Во-первых, так как класс ArrayList находится в пространстве имен System.Collections, то подключаем его (using System.Collections;).

Вначале создаем объект коллекции через конструктор как объект любого другого класса: ArrayList list = new ArrayList();. При необходимости мы могли бы так же, как и с массивами, выполнить начальную инициализацию коллекции, например, ArrayList list = new ArrayList(){1, 2, 5, "string", 7.7};

Далее последовательно добавляем разные значения. Данный класс коллекции, как и большинство других коллекций, имеет два способа добавления: одиночного объекта через метод **Add** и набора объектов, например, массива или другой коллекции через метод **AddRange**

Через цикл foreach мы можем пройтись по всем объектам списка. И поскольку данная коллекция хранит разнородные объекты, а не только числа или строки, то в качестве типа перебираемых объектов выбран тип object: foreach (object o in list)

Многие коллекции, в том числе и ArrayList, реализуют удаление с помощью методов Remove/RemoveAt. В данном случае мы удаляем первый элемент, передавая в метод RemoveAt индекс удаляемого элемента.

В завершении мы опять же выводим элементы коллекции на экран только уже через цикл for. В данном случае с перебором коллекций дело обстоит также, как и с массивами. А число элементов коллекции мы можем получить через свойство Count

С помощью индексатора мы можем получить по индексу элемент коллекции так же, как и в массивах: object firstObj = list[0];

Хотя в языке C# есть массивы, которые хранят в себе наборы однотипных объектов, но работать с ними не всегда удобно. Например, массив хранит фиксированное количество объектов, однако что если мы заранее не знаем, сколько нам потребуется объектов. И в этом случае намного удобнее применять коллекции. Еще один плюс коллекций состоит в том, что некоторые из них реализует стандартные структуры данных, например, стек, очередь, словарь, которые могут пригодиться для решения различных специальных задач. Большая часть классов коллекций содержится в пространстве имен **System.Collections.Generic**.

Класс List<T> из пространства имен System.Collections.Generic представляет простейший список однотипных объектов. Класс List типизируется типом, объекты которого будут хранится в списке.

Мы можем создать пустой список:

|  |  |
| --- | --- |
| 1 | List<string> people = new List<string>(); |

В данном случае объект List типизируется типом **string**. А это значит, что хранить в этом списке мы можем только строки.

Можно сразу при создании списка инициализировать его начальными значениями. В этом случае элементы списка помещаются после вызова конструктора в фигурных скобках

|  |  |
| --- | --- |
| 1 | List<string> people = new List<string>() { "Tom", "Bob", "Sam" }; |

В данном случае в список помещаются три строки

Также можно при создании списка инициализировать его элементами из другой коллекции, например, другого списка:

|  |  |
| --- | --- |
| 1  2 | var people = new List<string>() { "Tom", "Bob", "Sam" };  var employees = new List<string>(people); |

Можно совместить оба способа:

|  |  |
| --- | --- |
| 1  2 | var people = new List<string>() { "Tom", "Bob", "Sam" };  var employees = new List<string>(people){"Mike"}; |

В данном случае в списке employees будет четыре элемента ({ "Tom", "Bob", "Sam", "Mike" }) - три добавляются из списка people и один элемент задается при инициализации.

Начиная с версии **C# 12** для определения списков можно использовать выражения коллекций, которые предполагают заключение элементов коллекции в квадратные скобки:

|  |  |
| --- | --- |
| 1  2 | List<string> people = ["Tom", "Bob", "Sam"];  List<string> employees = [];// пустой список |

Подобным образом можно работать со списками других типов, например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | List<Person> people = new List<Person>()  {      new Person("Tom"),      new Person("Bob"),      new Person("Sam")  };    class Person  {      public string Name { get;}      public Person(string name) => Name = name;  } |

### Установка начальной емкости списка

Еще один конструктор класса List принимает в качестве параметра начальную емкость списка:

|  |  |
| --- | --- |
| 1 | List<string> people = new List<string>(16); |

Указание начальной емкости списка позволяет в будущем увеличить производительность и уменьшить издержки на выделение памяти при добавлении элементов. Поскольку динамическое добавление в список может приводить на низком уровне к дополнительному выделению памяти, что снижает производительность. Если же мы знаем, что список не будет превышать некоторый размер, то мы можем передать этот размер в качестве емкости списка и избежать дополнительных выделений памяти.

Также начальную емкость можно установить с помощью свойства Capacity, которое имеется у класса List.

### Обращение к элементам списка

Как и массивы, списки поддерживают индексы, с помощью которых можно обратиться к определенным элементам:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var people = new List<string>() { "Tom", "Bob", "Sam" };    string firstPerson = people[0]; // получаем первый элемент  Console.WriteLine(firstPerson); // Tom  people[0] = "Mike";     // изменяем первый элемент  Console.WriteLine(people[0]); // Mike |

### Длина списка

С помощью свойства **Count** можно получить длину списка:

|  |  |
| --- | --- |
| 1  2 | var people = new List<string>() { "Tom", "Bob", "Sam" };  Console.WriteLine(people.Count);    // 3 |

### Перебор списка

C# позволяет осуществить перебор списка с помощью стандартного цикла **foreach**:/p>

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | var people = new List<string>() { "Tom", "Bob", "Sam" };    foreach (var person in people)  {      Console.WriteLine(person);  }  // Вывод программы:  // Tom  // Bob  // Sam |

Также можно использовать другие типы циклов и в комбинации с индексами перебирать списки:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var people = new List<string>() { "Tom", "Bob", "Sam" };    for (int i = 0; i < people.Count; i++)  {      Console.WriteLine(people[i]);  } |

### Методы списка

Среди его методов можно выделить следующие:

* **void Add(T item)**: добавление нового элемента в список
* **void AddRange(IEnumerable<T> collection)**: добавление в список коллекции или массива
* **int BinarySearch(T item)**: бинарный поиск элемента в списке. Если элемент найден, то метод возвращает индекс этого элемента в коллекции. При этом список должен быть отсортирован.
* **void CopyTo(T[] array)**: копирует список в массив array
* **void CopyTo(int index, T[] array, int arrayIndex, int count)**: копирует из списка начиная с индекса index элементы, количество которых равно count, и вставляет их в массив array начиная с индекса arrayIndex
* **bool Contains(T item)**: возвращает true, если элемент item есть в списке
* **void Clear()**: удаляет из списка все элементы
* **bool Exists(Predicate<T> match)**: возвращает true, если в списке есть элемент, который соответствует делегату match
* **T? Find(Predicate<T> match)**: возвращает первый элемент, который соответствует делегату match. Если элемент не найден, возвращается null
* **T? FindLast(Predicate<T> match)**: возвращает последний элемент, который соответствует делегату match. Если элемент не найден, возвращается null
* **List<T> FindAll(Predicate<T> match)**: возвращает список элементов, которые соответствуют делегату match
* **int IndexOf(T item)**: возвращает индекс первого вхождения элемента в списке
* **int LastIndexOf(T item)**: возвращает индекс последнего вхождения элемента в списке
* **List<T> GetRange(int index, int count)**: возвращает список элементов, количество которых равно count, начиная с индекса index.
* **void Insert(int index, T item)**: вставляет элемент item в список по индексу index. Если такого индекса в списке нет, то генерируется исключение
* **void InsertRange(int index, collection)**: вставляет коллекцию элементов collection в текущий список начиная с индекса index. Если такого индекса в списке нет, то генерируется исключение
* **bool Remove(T item)**: удаляет элемент item из списка, и если удаление прошло успешно, то возвращает true. Если в списке несколько одинаковых элементов, то удаляется только первый из них
* **void RemoveAt(int index)**: удаление элемента по указанному индексу index. Если такого индекса в списке нет, то генерируется исключение
* **void RemoveRange(int index, int count)**: параметр index задает индекс, с которого надо удалить элементы, а параметр count задает количество удаляемых элементов.
* **int RemoveAll((Predicate<T> match))**: удаляет все элементы, которые соответствуют делегату match. Возвращает количество удаленных элементов
* **void Reverse()**: изменяет порядок элементов
* **void Reverse(int index, int count)**: изменяет порядок на обратный для элементов, количество которых равно count, начиная с индекса index
* **void Sort()**: сортировка списка
* **void Sort(IComparer<T>? comparer)**: сортировка списка с помощью объекта comparer, который передается в качестве параметра

### Добавление в список

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | List<string> people = new List<string> () { "Tom" };    people.Add("Bob"); // добавление элемента  // people = { "Tom", "Bob" };    people.AddRange(new[] { "Sam", "Alice" });   // добавляем массив  // people = { "Tom", "Bob", "Sam", "Alice" };  // также можно было бы добавить другой список  // people.AddRange(new List<string>(){ "Sam", "Alice" });    people.Insert(0, "Eugene"); // вставляем на первое место  // people = { "Eugene", "Tom", "Bob", "Sam", "Alice" };    people.InsertRange(1, new string[] {"Mike", "Kate"}); // вставляем массив с индекса 1  // people = { "Eugene", "Mike", "Kate", "Tom", "Bob", "Sam", "Alice" };    // также можно было бы добавить другой список  // people.InsertRange(1, new List<string>(){ "Mike", "Kate" }); |

### Удаление из списка

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | var people = new List<string> () { "Eugene", "Mike", "Kate", "Tom", "Bob", "Sam", "Tom", "Alice" };    people.RemoveAt(1); //  удаляем второй элемент  // people = { "Eugene", "Kate", "Tom", "Bob", "Sam", "Tom", "Alice" };    people.Remove("Tom"); //  удаляем элемент "Tom"  // people = { "Eugene", "Kate", "Bob", "Sam", "Tom", "Alice" };    // удаляем из списка все элементы, длина строки которых равна 3  people.RemoveAll(person => person.Length == 3);  // people = { "Eugene", "Kate", "Alice" };    // удаляем из списка 2 элемента начиная с индекса 1  people.RemoveRange(1, 2);  // people = { "Eugene"};    // полностью очищаем список  people.Clear();  // people = {  }; |

### Поиск и проверка элемента

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | var people = new List<string> () { "Eugene", "Mike", "Kate", "Tom", "Bob", "Sam" };    var containsBob = people.Contains("Bob");     //  true  var containsBill = people.Contains("Bill");    // false    // проверяем, есть ли в списке строки с длиной 3 символа  var existsLength3 = people.Exists(p => p.Length == 3);  // true    // проверяем, есть ли в списке строки с длиной 7 символов  var existsLength7 = people.Exists(p => p.Length == 7);  // false    // получаем первый элемент с длиной в 3 символа  var firstWithLength3 = people.Find(p => p.Length == 3); // Tom    // получаем последний элемент с длиной в 3 символа  var lastWithLength3 = people.FindLast(p => p.Length == 3);  // Sam    // получаем все элементы с длиной в 3 символа в виде списка  List<string> peopleWithLength3 = people.FindAll(p => p.Length == 3);  // peopleWithLength3 { "Tom", "Bob", "Sam"} |

### Получение диапазона и копирование в массив

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | List<string> people = new List<string>() {"Eugene", "Tom", "Mike", "Sam", "Bob" };    // получаем диапазон со второго по четвертый элемент  var range = people.GetRange(1, 3);  // range = { "Tom", "Mike", "Sam"};    // копируем в массив первые три элемента  string[] partOfPeople = new string[3];  people.CopyTo(0, partOfPeople, 0, 3);  // partOfPeople = { "Eugene", "Tom", "Mike"}; |

### Расположение элементов в обратном порядке

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | var people = new List<string> () { "Eugene", "Tom", "Mike", "Sam", "Bob" };    // переворачиваем весь список  people.Reverse();  // people = { "Bob","Sam", "Mike", "Tom", "Eugene"};    var people2 = new List<string>() { "Eugene", "Tom", "Mike", "Sam", "Bob" };  // переворачиваем часть только 3 элемента с индекса 1  people2.Reverse(1, 3);  // people2 = { "Eugene","Sam", "Mike", "Tom", "Bob" }; |

**Практическая часть**

**Описать класс, соответствующий заданию, содержащий поля, свойства, конструктор с параметрами. При необходимости вложить в свойства дополнительную логику проверки значений и написать методы, реализующие операции с экземплярами класса. Создать массив из *N* объектов данного класса. Создать коллекцию. Написать программу решения задачи.**

**- Показать работу любых семи методов класса List<T>**

**- Осуществить следующие операции: Добавление, удаление, поиск, проверка,** **Получение диапазона и копирование в массив**

1. Ввести информацию по *N* школьникам (Ф.И.О, пол, год рождения). Определить количество мальчиков и девочек. Вывести список каждых.

2. Ввести информацию по *N* студентам (№ группы, Ф.И.О. и три оценки). Вывести сведения о студентах, сдавших экзамены на 4 и 5.

3. Ввести информацию по *N* спортсменам (Ф.И.О., рост, вес). Вывести сведения о спортсменах, чей вес превышает 70 кг. Определить их количество.

4. Ввести информацию по *N* книгам (автор, название, издательство, год, количество страниц). Вывести список книг заданного автора.

5. Ввести информацию по *N* пациентам (фамилия, имя, отчество, номер медицинской карты, диагноз). Вывести список пациентов, имеющих заданный диагноз.

6. Ввести информацию по *N* перевозкам (№ рейса, пункт назначения, вес). Вывести сведения о перевозке с минимальным весом. Найти суммарный объем всех перевозок.

7. Ввести информацию по *N* фильмам (название, режиссер, год выпуска). Вывести сведения обо всех фильмах заданного года выпуска.

8. Ввести информацию о *N* рейсах самолетов (№ рейса, время вылета, время прилета, пункт назначения). Вывести информацию обо всех рейсах в заданный город.

9. Ввести информацию по *N* книгам (автор, название, издательство, год, количество страниц). Вывести список книг, изданных после заданного года.

10. Ввести информацию по *N* ученикам (порядковый номер, Ф.И.О. и три оценки). Определить количество отличников, хорошистов, троечников. Вывести список каждых.

11. Ввести информацию по *N* участникам соревнований (Ф.И.О., год рождения, результат). Вывести сведения о победителе (победителях) соревнований.

12. Ввести информацию о *N* предметах (порядковый номер, наименование, вес). Вывести информацию о предметах, чей вес больше среднего арифметического.