Модул 4. Увод в алгоритмите и структурите от данни

Тема 1. Въведение в алгоритмите

Задача 1.1. Принадлежи ли дадено число на масив

Определете сложността (максималния брой стъпки) на алгоритъма:

Напишете програма, която чете от конзолата последователност от цели числа на един ред, разделени с интервал и на втори ред число, което се проверява дали съществува в масива от първия ред. Ако числото съществува в масива, се извежда "{number} Exists in the List", в противен случай - "{Number} Not exists in the List".

Решение

```
namespace NumberInArray
{
    internal class Program
    {
        static void Main(string[] args)
        {
            List<int> list = Console.ReadLine().Split().Select(int.Parse).ToList();

            int number = int.Parse(Console.ReadLine());

            if (list.Contains(number)) Console.WriteLine($"{number} Exists in the
List", number);
            else Console.WriteLine($"{number} Not exists in the ", number);
        }
    }
}
```

Задача 1.2. Метод Insert

Определете сложността (максималния брой стъпки) на алгоритъма:

Напишете програма, която чете от конзолата възходяща последователност от цели числа на един ред, разделени с интервал и на втори ред число, което се вмъква на такава позиция, че ново получения масив отново да е възходящо подреден. Изведете:

- Ново получения масив
- Двата масива – този, преди вмъкването и другия – след вмъкването

Решение

```
namespace MethodInsert
{
    internal class Program
    {
        static void Main(string[] args)
        {
```

```
            List<int> list = Console.ReadLine().Split().Select(int.Parse).ToList();

            int number = int.Parse(Console.ReadLine());

            bool inserted = false;
            List<int> newList = new List<int>();
            foreach (var item in list)
            {
                if (inserted)
                {
                    newList.Add(item);
                    continue;
                }
                if (number > item) newList.Add(item);
                else
                {
                    newList.Add(number);
                    newList.Add(item);
                    inserted = true;
                }
            }

            Console.WriteLine(string.Join(" ", newList));
        }
    }
}
```

Задача 1.3. Сбор и Средно аритметично

Определете сложността (максималния брой стъпки) на алгоритъма:

Напишете програма, която чете от конзолата последователност от числа (на един ред, разделени с интервал). Изчислява и отпечатате сбора и средната стойност на елементите на последователността. Запазва последователността в List<int>. Закръглете средната стойност до втората цифра след десетичния разделител.

Примери

Вход	Изход
4 5 6	Sum=15; Average=5.00
1 1	Sum=2; Average=1.00
	Sum=0; Average=0.00
10	Sum=10; Average=10.00
2 2 1	Sum=5; Average=1.67

Решение

```
namespace SumAndAverage
{
    internal class Program
    {
```

```
        static void Main(string[] args)
        {
            List<int> list = Console.ReadLine().Split().Select(int.Parse).ToList();

            int sum = 0;
            float avr = 0;
            foreach (var item in list)
            {
                sum = sum + item;
            }
            avr = sum / list.Count;

            Console.WriteLine("Sum={0}; Average={1:f2}", sum, avr);
        }
    }
}
```

Задача 1.4. Намиране на най-малко число

Определете сложността (максималния брой стъпки) на програма, която чете от конзолата последователност от цели числа на един ред, разделени с интервал. Намерете най-малкото от тях и го изведете.

Подсказка

1. В променлива minimum запишете първото число
2. Сравнявайте това число с всички останали (от второто до последното) и ако някое от тях е по-малко, то в minimum запазете неговата стойност
3. Изведете стйността на minimum.

Решение

```
namespace MinimalNumber
{
    internal class Program
    {
        static void Main(string[] args)
        {
            List<int> list = Console.ReadLine().Split().Select(int.Parse).ToList();

            var minimum = list.First();

            for (int i = 1; i < list.Count; i++)
            {
                if (list[i] < minimum)
                {
                    minimum = list[i];
                }
            }

            Console.WriteLine(minimum);
        }
    }
}
```

Задача 1.5. Наредени двойки

Определете сложността (максималния брой стъпки) на програма, която чете от конзолата последователност от две цели числа m и n . Да се изведат всички възможни наредени двойки цели числа (p, q) които се менят съответно за p [1..m], q в [1..n]

Решение

```
namespace OrderedPairs
{
    internal class Program
    {
        static void Main(string[] args)
        {
            int m = int.Parse(Console.ReadLine());
            int n = int.Parse(Console.ReadLine());

            for (int p = 1; p <= m; p++)
            {
                for (int q = 1; q <= n; q++)
                {
                    Console.WriteLine($"{p} {q}");
                }
            }
        }
    }
}
```

Задача 1.6. Подреждане на думи

Определете сложността на програма, която чете от конзолата последователност от думи (символни низове на един ред, разделени с интервал). Подредете ги по азбучен ред. Запазете последователността в List<string>

Подсказка

1. Използвайте алгоритъма от задачата „Намиране на най-малко число“, променете и допълнете
2. Намерете „най-малката“ по азбучен ред дума и я запазете в променлива minimum и запомнете поицията ѝ в променлива minimumPos
3. На позиция minimumPos в списъка запишете първата дума от списъка
4. На първа позиция в списъка запишете стойността minimum
5. Повторете стъпки от 1 до 3 за елементите на списъка от втора до последа позиция

Вход	Изход
wow softuni alpha	alpha softuni wow
hi	hi

Решение

```
namespace WordsArrangement
{
    internal class Program
    {
        static void Main(string[] args)
        {
            List<string> words = Console.ReadLine().Split().ToList();

            List<string> result = new List<string>();
            var minimum = words.First();
            for (int m = 0; m < words.Count; m++)
            {
                for (int i = 0; i < words.Count; i++)
                {
                    if (words[i].CompareTo(minimum) < 0)
                    {
                        minimum = words[i];
                    }
                }
                result.Add(minimum);
                words.Remove(minimum);
                minimum = words.First();
            }
            result.Add(words.First());

            Console.WriteLine(String.Join(" ", result));
        }
    }
}
```

Задача 1.7. Най-дълга последователност

Определете сложността на метод, който намира най-дългата последователност от равни числа в даден списък от цели числа List<int> и връща резултата като нов List<int>. Ако няколко поредици имат същата най-дълга дължина, върнете най-лявата от тях.

Примери

Вход	Изход
12 2 7 4 3 3 8	3 3
2 2 2 3 3 3	2 2 2
4 4 5 5 5	5 5 5
1 2 3	1
0	0
4 2 3 4 4	4 **4**

Решение

```
namespace LongestSequence
```

```
{
    internal class Program
    {
        static void Main(string[] args)
        {
            List<int> list = Console.ReadLine().Split().Select(int.Parse).ToList();
            List<int> result = new List<int>();

            int number = 0, max = 0; // 6
            for (int i = 0; i < list.Count - 1; i++)
            {
                int counter = 1;
                for (int j = i + 1; j < list.Count; j++)
                {
                    if (list[i] == list[j]) counter++;
                    else break;
                }
                if (counter > max)
                {
                    number = list[i];
                    max = counter;
                }
            }

            for (int i = 0; i < max; i++) result.Add(number);

            Console.WriteLine(string.Join(" ", result));
        }
    }
}
```

Задача 1.8. Remove/Add Method

Определете сложността (максималния брой стъпки) на програма, която чете от конзолата входящ списък от цели числа на един ред, разделени с интервал и на втори ред число, за което се проверява дали е в списъка или не. Ако е, то то се премахва от него , а ако го няма – се добавя на такова място, че списъка отново да е подреден. Изведете:

1. Новополучения списък
2. Двата списъка – входния и новополучения

Решение

```
namespace RemoveAddMethod
{
    internal class Program
    {
        static void Main(string[] args)
        {
            List<int> list = Console.ReadLine().Split().Select(int.Parse).ToList();
            List<int> newList = new List<int>();
            newList.AddRange(list);

            int number = int.Parse(Console.ReadLine());
```

```
            if (list.Contains(number)) list.Remove(number);
            else newList.Add(number);
            newList.Sort();

            Console.WriteLine(String.Join(" ", list));
            Console.WriteLine(String.Join(" ", newList));
        }
    }
}
```

Тема 2. Линейни структури от данни

Задача 2.1. Имплементация на разтеглив масив ArrayList<T>

Имплементирайте структура от данни ArrayList<T> която съдържа поредица от елементи от шаблонен тип T. Структурата трябва да пази поредица от елементи в масив. Структурата трябва да има капацитет, който расте двойно, когато се препълни, като в началото винаги има 2 елемента. Масивът трябва да поддържа следните операции:

- **`int Count`** → дава броя на елементите в структурата
- **`T this[int index]`** → индексатор, който служи за достъпване на елементите по техния индекс (в интервал **`0`** … **`Count-1`**)
- **`void Add(T item)`** → добавя елемент към поредицата и удвоява капацитета на масива, ако е запълнен
- **`T RemoveAt(int index)`** → премахва елемента по неговия индекс (в интервала **`0`** … **`Count-1`**) и връща елемента

Винаги тествайте имплементираните операции преди да продължите с реализацията на следващата

Решение

```
public class ArrayList<T> : IEnumerable<T>
{
    private T[] items;

    public int Count { get; private set; }

    public int Capacity { get; private set; }

    public ArrayList()
    {
        this.Count = 0;
        this.Capacity = 2;
        this.items = new T[this.Capacity];
    }

    public T this[int index]
    {
        get
        {
            OutOfRange(index);
```

```
            return items[index];
        }
        set
        {
            OutOfRange(index);
            this.items[index] = value;
        }
    }

    public void Add(T item)
    {
        if (this.Count == this.Capacity)
        {
            this.Capacity *= 2;
            var temp = this.items;
            this.items = new T[this.Capacity];
            for (int i = 0; i < temp.Count(); i++)
            {
                this.items[i] = temp[i];
            }
        }
        this.items[this.Count] = item;
        this.Count++;
    }

    public void Resize()
    {
        T[] copy = new T[this.items.Length * 2];

        for (int i = 0; i < this.items.Length; i++)
        {
            copy[i] = this.items[i];
        }

        this.items = copy;
    }

    public T RemoveAt(int index)
    {
        OutOfRange(index);

        var item = this.items[index];
        this.items = items.Take(index).Concat(items.Skip(index + 1)).Concat(new
T[1]).ToArray();
        this.Count--;

        if (this.items.Count() <= this.Capacity / 2)
        {
            if (this.Count < 2) return item;
            this.Capacity /= 2;
            var temp = this.items;
            this.items = new T[this.Capacity];
            for (int i = 0; i < temp.Count(); i++)
            {
                this.items[i] = temp[i];
```

```
            }
        }

        return item;
    }

    private void OutOfRange(int index)
    {
        if (index < 0 || index >= this.Count)
        {
            throw new IndexOutOfRangeException();
        }
    }

    public IEnumerator<T> GetEnumerator()
    {
        for (int i = 0; i < this.items.Count(); i++)
        {
            yield return this.items[i];
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return this.GetEnumerator();
    }
}

class Program
{
    static void Main(string[] args)
    {
        ArrayList<int> numbers = new ArrayList<int>();
        numbers.Add(112);
        numbers.Add(911);
        numbers.Add(166);
        numbers.Add(150);
        Console.WriteLine(string.Join(" ", numbers));

        ArrayList<string> text = new ArrayList<string>();
        text.Add("Hello");
        text.Add("World");
        Console.WriteLine(string.Join(" ", text));
    }
}
```

Задача 2.2. Статична реализация на списък

Създайте статична реализация на списък. Използвайте следната структура за класа:

```
public class CustomArrayList
{
      private object[] arr;
```

```
        private int count;

        public int Count
        {
              get
              {
                    return count;
              }
        }

        private static readonly int INITIAL_CAPACITY = 4;

        public CustomArrayList()
        {
              arr = new object[INITIAL_CAPACITY];
              count = 0;
        }
      public void Add(object item)
      {
      }

      public void Insert(int index, object item)
      {
       }

      public int IndexOf(object item)
      {
      }

      public void Clear()
      {
      }

      public bool Contains(object item)
      {
      }

      public object this[int index]
      {
      }

      public object Remove(int index)
      {
      }

      public int Remove(object item)
      {
```

```
        }
}
```

Ако сте дефинирали класа и сте реализирали методите правилно, тестът трябва да мине.

Решение

```
class CustomArrayList
{
    private const int INITIAL_CAPACITY = 4;
    private object[] arr;

    private int count;
    public int Count
    {
        get { return this.count; }
        private set { this.count = value; }
    }

    private int capacity;
    public int Capacity
    {
        get { return this.capacity; }
        private set { this.capacity = value; }
    }

    public CustomArrayList(int size = INITIAL_CAPACITY)
    {
        arr = new object[size];
        capacity = size;
        count = 0;
    }

    private object this[int index]
    {
        get
        {
            OutOfRange(index);
            return this.arr[index];
        }
        set
        {
            OutOfRange(index);
            this.arr[index] = value;
            this.count++;
        }
    }

    public void Add(object item)
    {
        if (capacity == count)
        {
            capacity *= 2; // Разтягане
```

```
            object[] temp = arr;
            arr = new object[capacity];
            for (int i = 0; i < temp.Length; i++) arr[i] = temp[i];
            arr[count] = item;
        }
        else arr[count] = item;
        count++;
    }

    public object Get(int index)
    {
        OutOfRange(index);
        return arr[index];
    }

    public object Remove(int index)
    {
        OutOfRange(index);
        var temp = arr[index];
        arr = arr.Take(index).Concat(arr.Skip(index + 1)).ToArray();
        count--;
        return temp;
    }

    public void Insert(int index, object item)
    {
        OutOfRange(index);
        arr = arr.Take(index).Concat((new object[1] { item
}).Concat(arr.Skip(index))).ToArray();
    }

    public void Clear()
    {
        arr = new object[capacity];
        return;
    }

    private void OutOfRange(int index)
    {
        if (index < 0 || index > count)
            throw new IndexOutOfRangeException();
    }
}

class Program
{
    static void Main(string[] args)
    {
        CustomArrayList list = new CustomArrayList();

        Console.WriteLine("Count = {0}", list.Count);
        Console.WriteLine("Capacity = {0}", list.Capacity);

        list.Add(100);
        list.Add(200);
```

```
        list.Add(300);

        list.Clear();

        list.Add(400);
        list.Add(500);

        list.Insert(0, 69696969);

        Console.WriteLine("Count = {0}", list.Count);
        Console.WriteLine("Capacity = {0}", list.Capacity);
    }

}
```

Задача 2.3. Динамична реализация на списък

Създайте динамична реализация на списък. Използвайте следната структура за класовете:

```
public class DynamicList
{
        private class Node
        {
                private object element;
                private Node next;

                public object Element
                {
                        get { return element; }
                        set { element = value; }
                }

                public Node Next
                {
                        get { return next; }
                        set { next = value; }
                }

                public Node(object element, Node prevNode)
                {
                }

                public Node(object element)
                {
                }
        }

        private Node head;
        private Node tail;
        private int count;
```

```
      public DynamicList()
      {
            this.head = null;
            this.tail = null;
            this.count = 0;
      }

      public void Add(object item)
      {
      }
      public object Remove(int index)
      {
      }


      public int Remove(object item)
      {
      }
      public int IndexOf(object item)
      {
      }

      public bool Contains(object item)
      {
      }
      public object this[int index]
      {
      }

      public int Count{
            get
            {
                  return count;
            }
      }
}
```

Решение

```
class Node
{
    private object element;
    public object Element
    {
        get { return this.element; }
        set { this.element = value; }
    }

    private Node next;
    public Node Next
```

```
    {
        get { return this.next; }
        set { this.next = value; }
    }

    public Node(object element, Node prevNode)
    {
        this.element = element;
        prevNode.next = this;
    }

    public Node(object element)
    {
        this.element = element;
        next = null;
    }
}

class DynamicList
{
    private Node head;
    private Node tail;

    private int count;
    public int Count
    {
        get { return this.count; }
        private set { this.count = value; }
    }

    public DynamicList()
    {
        this.head = null;
        this.tail = null;
        this.count = 0;
    }

    public void Add(object item)
    {
        if (this.head == null)
        {
            Node next = new Node(item);
            this.head = next;
            this.tail = next;
        }
        else
        {
            Node next = new Node(item, tail);
            this.tail = next;
        }
        this.count++;
    }

    public int IndexOf(object item)
    {
```

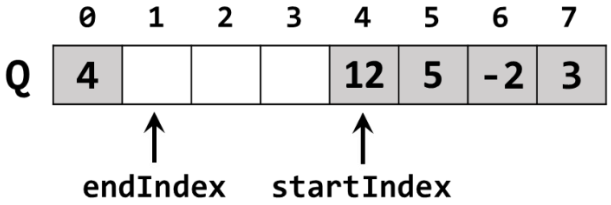
```
        Node current = head;
        int index = 0;
        while (current != null)
        {
            if (current.Element.Equals(item)) return index;
            index++;
            current = current.Next;
        }
        return -1;
    }

    public object Remove(int index)
    {
        Object item = null;
        Node current = head;
        Node previous = null;
        int i = 0;
        while (current != null)
        {
            if (index == i)
            {
                item = current.Element;
                previous.Next = current.Next;
                break;
            }
            i++;
            previous = current;
            current = current.Next;
        }
        return item;
    }

    public int Remove(object item)
    {
        int index = 0;
        Node current = head;
        while (current != null)
        {
            if (current.Element.Equals(item))
            {
                Remove(index);
                return index;
            }
            index++;
            current = current.Next;
        }
        return -1;
    }

    public bool Contains(object item)
    {
        Node current = head;
        while (current != null)
        {
            if (current.Element.Equals(item)) return true;
```

```
            current = current.Next;
        }
        return false;
    }

    public object this[int index]
    {
        get
        {
            Node current = head;
            int i = 0;
            while (current != null)
            {
                if (i == index) return current.Element;
                i++;
                current = current.Next;
            }
            return null;
        }
        set
        {
            Node current = head;
            int i = 0;
            while (current != null)
            {
                if (i == index)
                {
                    current.Element = value;
                    break;
                }
                i++;
                current = current.Next;
            }
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        var list = new DynamicList();

        list.Add(new Node(42));
        list.Add(new Node("yes"));
        list.Add(new Node(true));

        for (int i = 0; i < list.Count; i++)
        {
            var item = ((Node)list[i]).Element.ToString();
            Console.WriteLine(item);
        }

        Console.WriteLine(list.Remove("yes"));
        Console.WriteLine(((Node)list.Remove(1)).Element.ToString());
```

```
    }
}
```

Задача 2.4. Имплементация на опашка

Имплементирайте кръгова опашка, базирана на масив в C# – структура от данни, която съдържа елементи и следва принципа FIFO (First In, First Out – първи вътре, първи вън), като използвате фиксиран вътрешен капацитет, който се удвоява, когато се запълни:



На фигурата по-горе, елементите {12, 5, -2, 3, 4} стоят в масив с фиксиран капацитет от 8 елемента. Капацитета на опашката е 8, броят на елементите е 5, а 3 клетки стоят празни. startIndex ни показва първият непразен елемент в опашката. endIndex ни показва мястото точно след последния непразен елемент в опашката – мястото, където следващият елемент ще бъде добавен към опашката. Забележете, че опашката е кръгова: след елемента на последна позиция 7 идва елемент на позиция 0.

CircularQueue<T>

Използвайте следният скелет за класа:

```
public class CircularQueue<T>
{
    private const int DefaultCapacity = 4;
    public int Count { get; private set; }
    public CircularQueue(int capacity = DefaultCapacity) { … }
    public void Enqueue(T element) { … }
    public T Dequeue() { … }
    public T[] ToArray() { … }
}
```

Създайте вътрешната информация за опашката

Първата стъпка е да създадете вътрешна информация, която пази елементите, както и началният+крайният индекс:

- `T[] elements` – масив, който държи елементите на опашката
  - Непразните клетки пазят елементите
  - Празните клетки са свободни за добавяне на нови елементи
  - Дължината на масива (Length) пази капацитета на опашката
- `int startIndex` – пази началния индекс (индекса на първия влезнал елемент в опашката)

- `int endIndex` – пази крайния индекс (индекса в масива, който е непосредствено след последния добавен елемент)
- `int Count` – пази информация за броя елементи в опашката

Направете конструктор

Сега, нека да имплементираме конструктор. Неговата цел е да заделя място за масива в рамките на CircularQueue<T> класа. Ще имаме два конструктора:

- Конструктор без параметри – трябва да задели 16 елемента (16 е капацитета по подразбиране в началото за опашката)
- Конструктор с параметър **`capacity`** – заделя масива с конкретен капацитет

Имплементиране на Enqueue(...) метод

Нека да имплементираме Enqueue(element) метода, който добавя нов елемент в края на опашката.

Как работи? Първо, ако опашката е пълна, увеличава я (т.е. нейния капацитет става двойно по-голям). След това, добавя новият елемент на позиция endIndex (индексът, който е точно след последния елемент), а след това премества индекса с една позиция надясно, както и увеличава вътрешния брояч Count.

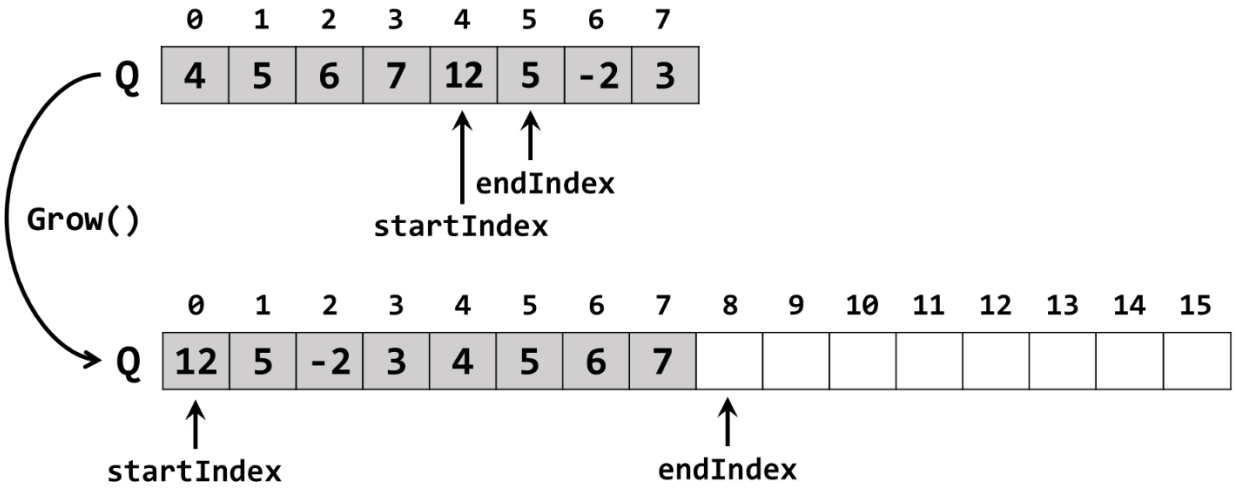
Забележете, че опашката е кръгова, така че елемента след последния елемент (this.elements.Length-1) е 0.

Така стигаме до формула: Елементът следващ p е на позиция (p + 1) % capacity. В кода имаме:

```
(this.endIndex + 1) % this.elements.Length
```

Имплементиране на Grow() метод

Grow() методът се извиква, когато опашката е със запълнен капацитет (capacity == Count) и искаме да добавим нов елемент. Grow() методът трябва да задели нов масив с удвоен капацитет и да премести всички елементи от стария масив в новия масив:



Важна част от "уголемяването" е да се копират елементите от стария масив в новия. *Имплементиране на Dequeue() метод*

Сега е ред на Dequeue() метода. Неговата цел е да се върне и да се премахне от опашката първият добавен елемент (той се намира на позиция startIndex).

Как работи? Ако опашката е празна, се хвърля изключение. В противен случай, първият елемент от опашката се взема; startIndex се отмества нататък; Count се намаля.

Имплементиране на ToArray() Method
Сега нека си направим и ToArray() метод. Той трябва да заделя масив с размер this.Count и да копира всички елементи от опашката в него. Ние вече имаме метод за копиране на елементите, така че този път ще се справим по-лесно и кратко. Кодът е замъглен нарочно. Опитайте се сами.

Решение
```
public class CircularQueue<T> : IEnumerable<T>
{
    private const int DefaultCapacity = 4;

    public int Count { get; private set; }

    private T[] elements;

    private int startIndex = 0;

    private int endIndex = 0;

    public CircularQueue(int capacity = DefaultCapacity)
    {
        this.elements = new T[capacity];
        this.Count = 0;
    }
```

```
public void Enqueue(T element)
{
    if (this.Count >= this.elements.Length)
    {
        this.Glow();
    }
    this.elements[this.endIndex] = element;
    this.endIndex = (this.endIndex + 1) % this.elements.Length;
    this.Count++;
}

private void Glow()
{
    var newElements = new T[this.elements.Length * 2];
    this.CopyAllElementsTo(newElements);
    this.elements = newElements;
    this.startIndex = 0;
    this.endIndex = this.Count;
}

private void Shrink()
{
    var newElements = new T[this.elements.Length / 2];
    this.CopyAllElementsTo(newElements);
    this.elements = newElements;
    this.startIndex = 0;
    this.endIndex = this.Count;
}

private void CopyAllElementsTo(T[] resultArr)
{
    int sourceIndex = this.startIndex;
    int destinationIndex = 0;
    for (int i = 0; i < this.Count; i++)
    {
        resultArr[destinationIndex] = this.elements[sourceIndex];
        sourceIndex = (sourceIndex + 1) % this.elements.Length;
        destinationIndex++;
    }
}

public T Dequeue()
{
    if (this.Count == 0)
    {
        throw new InvalidOperationException("Empty!");
    }

    var result = this.elements[this.startIndex];
    this.startIndex = (this.startIndex + 1) % this.elements.Length;
    this.Count--;

    if (this.Count <= this.elements.Length / 2 && this.Count >= DefaultCapacity)
    {
```

```
            this.Shrink();
        }

        return result;
    }

    public T[] ToArray()
    {
        T[] result = new T[this.Count];
        int sourceIndex = this.startIndex;
        int destinationIndex = 0;
        for (int i = 0; i < this.Count; i++)
        {
            result[destinationIndex++] = this.elements[sourceIndex];
            sourceIndex = (sourceIndex + 1) % this.elements.Length;
        }
        return result;
    }

    public IEnumerator<T> GetEnumerator()
    {
        for (int i = 0; i < this.elements.Count(); i++)
        {
            yield return this.elements[i];
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return this.GetEnumerator();
    }
}

class Program
{
    static void Main(string[] args)
    {
        CircularQueue<int> numbers = new CircularQueue<int>();
        numbers.Enqueue(112);
        numbers.Enqueue(911);
        numbers.Enqueue(166);
        numbers.Enqueue(150);
        Console.WriteLine(string.Join(" ", numbers));

        CircularQueue<string> text = new CircularQueue<string>();
        text.Enqueue("Hello");
        text.Enqueue("World");
        Console.WriteLine(string.Join(" ", text));
    }
}
```

Задача 2.5. Статична имплементация на стек

Имплементирайте статично стек ArrayStack<T>, който пази елементите си в масив:

```
public class ArrayStack<T>
{
    private T[] elements;
    public int Count { get; private set; }
    private const int InitialCapacity = 16;

    public ArrayStack(int capacity = InitialCapacity) { … }
    public void Push(T element) { … }
    public T Pop() { … }
    public T[] ToArray() { … }
    private void Grow() { … }
}
```

Подсказки

- Капацитета на стека е `this.elements.Length`
- Пазете размера на стека (брой елементи) в `this.Count`
- `Push(element)` запазва елемента в `elements[this.Count]` и увеличава `this.Count`
- `Push(element)` трябва да извика `Grow()`, в случай че `this.Count == this.elements.Length`
- `Pop()` намаля `this.Count` и връща `this.elements[this.Count]`
- `Grow()` заделя нов масив `newElements` с размер `2 * this.elements.Length` и копира първите `this.Count` елемента от `this.elements` до `newElements`. Накрая, присвоете `this.elements = newElements`
- `ToArray()` създава и връща масив от `this.elements[0…this.Count-1]`
- `Pop()` трябва да хвърля `InvalidOperationException` (или `IllegalArgumentException`) при празен стек

Решение

```
public class ArrayStack<T> : IEnumerable<T>
{
    private T[] elements;

    public int Count { get; private set; }

    private const int InitialCapacity = 2;

    public ArrayStack(int capacity = InitialCapacity)
    {
        this.elements = new T[capacity];
        this.Count = 0;
    }

    public void Push(T element)
    {
        if (this.Count == this.elements.Length)
        {
            this.Grow();
        }
        this.elements[this.Count] = element;
```

```
        this.Count++;
    }

    public T Pop(T element)
    {
        if (this.Count == 0)
        {
            throw new InvalidOperationException("Empty!");
        }

        var item = this.elements[this.Count - 1];
        this.Count--;

        if (this.Count <= this.elements.Length / 2 && this.Count >= InitialCapacity)
        {
            this.Shrink();
        }

        return item;
    }

    public T[] ToArray()
    {
        return this.elements.ToArray();
    }

    private void Grow()
    {
        T[] copy = new T[this.elements.Length * 2];
        for (int i = 0; i < this.elements.Length; i++)
        {
            copy[i] = this.elements[i];
        }
        this.elements = copy;
    }

    private void Shrink()
    {
        T[] copy = new T[this.elements.Length / 2];
        for (int i = 0; i < this.elements.Length / 2; i++)
        {
            copy[i] = this.elements[i];
        }
        this.elements = copy;
    }

    public IEnumerator<T> GetEnumerator()
    {
        for (int i = 0; i < this.elements.Count(); i++)
        {
            yield return this.elements[i];
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
```

```
    {
        return this.GetEnumerator();
    }
}

class Program
{
    static void Main(string[] args)
    {
        ArrayStack<int> numbers = new ArrayStack<int>();
        numbers.Push(112);
        numbers.Push(911);
        numbers.Push(166);
        numbers.Push(150);
        Console.WriteLine(string.Join(" ", numbers));

        ArrayStack<string> text = new ArrayStack<string>();
        text.Push("Hello");
        text.Push("World");
        Console.WriteLine(string.Join(" ", text));
    }
}
```

Задача 2.6.Имплементиране на ReversedList<T>

Имплементирайте структурата данни ReversedList<T> която пази поредица от елементи от тип T. Тя трябва да пази и поредица от елементи в обратен ред. Структурата трябва да има някакъв капацитет, който расте двойно при препълване, като винаги започва с капацитет 2. Обърнатият списък трябва да поддържа следните операции:

- **Add(T item)** → добавя елемент към поредицата (расте двойно, ако се препълни)
- **Count** → връща броя елементи в структурата
- **Capacity** → връща капацитета на масива, който пази елементите
- **this[index]** → индексатор за достъпване на елементите по индекс (в интервала **0** … **Count-1**) в обратен ред на добавянето им
- **RemoveAt(index)** → премахва елемент на индекс (в интервала **0** … **Count-1**) в обратен ред на добавянето им
- **IEnumerable<T>** → имплементира enumerator, за да позволи лесно обхождане на структурата чрез **foreach** цикъл в обратен ред на добавянето им

Решение

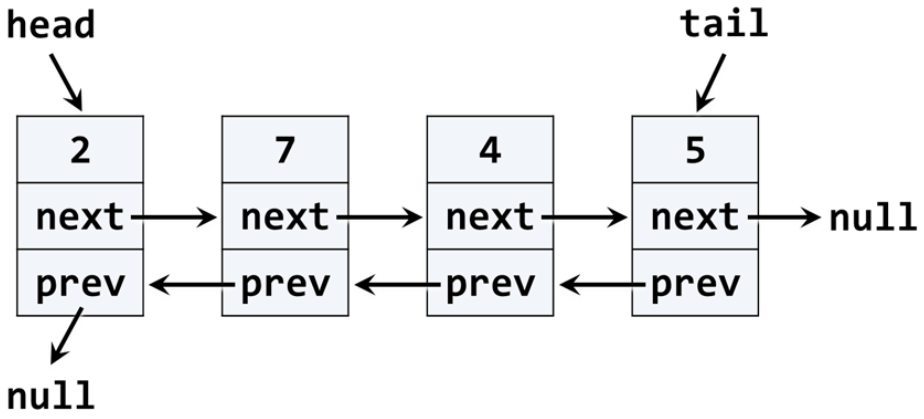
```
public class ReversedList<T> : IEnumerable<T>
{
    private T[] items;

    public int Capacity
    {
        get { return capacity; }
        private set { capacity = value; }
    }
```

```
    private int capacity;

    public int Count
    {
        get { return count; }
        private set { count = value; }
    }
    private int count;

    public ReversedList(int capacity = 2)
    {
        this.count = 0;
        this.capacity = capacity;
        this.items = new T[capacity];
    }

    public void Add(T item)
    {
        if (this.Capacity == this.Count)
        {
            this.Capacity *= 2;
            T[] temp = new T[Capacity];
            for (int i = 0; i < this.Count; i++)
            {
                temp[i] = this.items[i];
            }
            this.items = temp;
        }

        this.items[this.Count] = item;
        this.Count++;
    }

    public T RemoveAt(int index)
    {
        OutOfRange(index);

        T element = this.items[index];

        var temp = items.Take(Count).Reverse();
        this.items = temp.Take(index).Concat(temp.Skip(index +
1)).Reverse().Concat(new T[Capacity - Count + 1]).ToArray();
        this.Count--;

        return element;
    }

    public T this[int index]
    {
        get
        {
            OutOfRange(index);
            return items[index];
        }
        set
```

```
        {
            OutOfRange(index);
            this.items[index] = value;
        }
    }

    private void OutOfRange(int index)
    {
        if (index < 0 || index >= this.Count)
        {
            throw new IndexOutOfRangeException();
        }
    }

    public IEnumerator<T> GetEnumerator()
    {
        for (int i = Count - 1; i >= 0; i--)
        {
            yield return this.items[i];
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return this.GetEnumerator();
    }
}

class Program
{
    static void Main(string[] args)
    {
        ReversedList<int> numbers = new ReversedList<int>();
        numbers.Add(112);
        numbers.Add(911);
        numbers.Add(166);
        numbers.Add(150);
        Console.WriteLine(string.Join(" ", numbers));

        ReversedList<string> text = new ReversedList<string>();
        text.Add("Hello");
        text.Add("World");
        Console.WriteLine(string.Join(" ", text));
    }
}
```

Задача 2.7. Имплементиране на DoublyLinkedList<T>

Трябва да реализирате двусвързан списък в C# – структура от данни, която има възли (nodes), където всеки възел съдържа информация, както за следващия, така и за предишния възел:



Операциите върху двусвързан списък са добавяне (add) / премахване (remove) на елемент от двата края и обхождане. По дефиниция, двусвързания списък има head (начало на списъка) и tail (край на списъка). Да започваме!

Реализация на ListNode<T>

Първата стъпка при имеплементиране на свързан / двусвързан списък е да **разберем, че ни трябват два класа:**

- **`ListNode<T>`** клас, който съдържа един възел (стойност + следващ възел + преден възел)
- **`DoublyLinkedList<T>`** клас, който съдържа целия списък

Сега, нека да напишем класа за възела. Той трябва да пази Value и информация за неговия предишен и следващ възел. Може да бъде и вътрешен клас, защото ще ни трябва само вътрешно от другия.

ListNode<T> се нарича рекурсивна структура от данни, защото „сочи“ сам към себе си рекурсивно. Той използва шаблонен аргумент T за да избегне по-късна конкретика за даден тип данни.

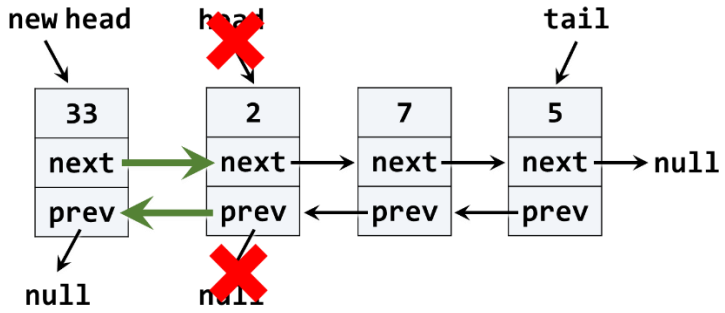
Имплементирайте Head, Tail и Count

Сега нека си дефинираме head и tail на двусвързания списък:

Имплементиране на AddFirst(T) метода

Добавянето на елемент към началото на списъка има два случая:

- Празен списък → добавя нов елемент като свой **`head`** и **`tail`** едновременно.
- Непразен списък → Добавя нов елемент като нов head и задава стария head като втори елемент, след head.



Графиката визуализира процеса на добавянето на нов възел в началото (head) на списъка. Червените стрелки отбелязват премахнатите указатели от стария head. Зелените стрелки отбелязват новите указатели към новият head.

Имплементирайте ForEach(Action) метод

Ние имаме двусвързан списък. Можем да добавяме елементи към него, но не можем да видим какво има вътре, защото списъкът все още няма метод за обхождане на елементите му. Сега, нека да дефинираме ForEach(Action<T>) метод. Той взема за параметър функция (действие, action), която ще бъде извикана за всеки един от елементите в списъка. Алгоритъмът зад този метод е прост: започвайки от head и преминавайки напред към следващия елемент, докато не стигнем до последният елемент (неговият следващ елемент е null).

Имплементиране на AddLast(T) метод

Сега, имплементирайте AddLast(T element) метод, за добавяне на нов елемент като tail. Това трябва да бъде много подобно на AddFirst(T element) метода. Логиката вътре в него е точно същата, но елемента се добавяне към tail, вместо към head.

Имплементирайте RemoveFirst() метода

Сега, нека да имплементираме RemoveFirst() → T. Той трябва да премахва първия елемент и да премества head към втория елемент. Премахнатият елемент трябва да бъде върнат като резултат от метода. В случай на празен списък, методът трябва да хвърли изключение. Трябва да разгледаме следните случаи:

- Празен списък → хвърляме изключение.
- Единствен елемент в списъка → прави списъка празен (**`head == tail == null`**).
- Множество елементи в списъка → премахваме първия елемент и задаваме head да сочи към втория елемент (**`head = head.NextNode`**).

Имплементация на RemoveLast() метода

Сега, нека добавим и метод RemoveLast() → T. Той трябва да махне последния елемент от списъка и да промени неговия tail, така щото да сочи към

елемента преди последния. Идеята е много подобна на RemoveFirst(), така че се опитайте сами.

Имплементация на ToArray() метод

Сега, имплементирайте следващия метод: ToArray() → T[]. Той трябва да копира всички елементи на свързания списък към масив със същия размер. Може да използвате следните стъпки, за да реализирате този метод:

- Заделете място за масив **`T[]`** с размер **`this.Count`**.
- Обходете всички елементи в списъка (от **`head`** към **`tail`**) и ги присвоете в **`T[0]`**, **`T[1]`**, ..., **`T[Count-1]`**.
- Върнете масива като резултат.

Имплементирайте IEnumerable<T>

Класовете за колекции в C# и .NET Framework (като масиви, списъци и т.н.) имплементират системния интерфейс (повече за интерфейси в курсът за ООП) IEnumerable<T>. Това се прави, за да може да направим обхождане с foreach по елементите.

Този метод връща IEnumerator<T>, който може да мине към следващия елемент и да прочете текущия елемент. Това е известно още и като итератор (enumerator).

Ще използване "yield return" от C#, за да опростим имплементацията на итератора:

Кодът по-горе ще ви позволи да използвате DoublyLinkedList<T> и foreach цикли.

Решение

```
public class ListNode<T>
{
    public T Value { get; private set; }

    public ListNode<T> Prev { get; set; }

    public ListNode<T> Next { get; set; }

    public ListNode(T value)
    {
        this.Value = value;
    }
}

public class DoubleLinkedList<T> : IEnumerable<T>
{
    private ListNode<T> Head { get; set; }

    private ListNode<T> Tail { get; set; }

    public int Count { get; private set; }
```

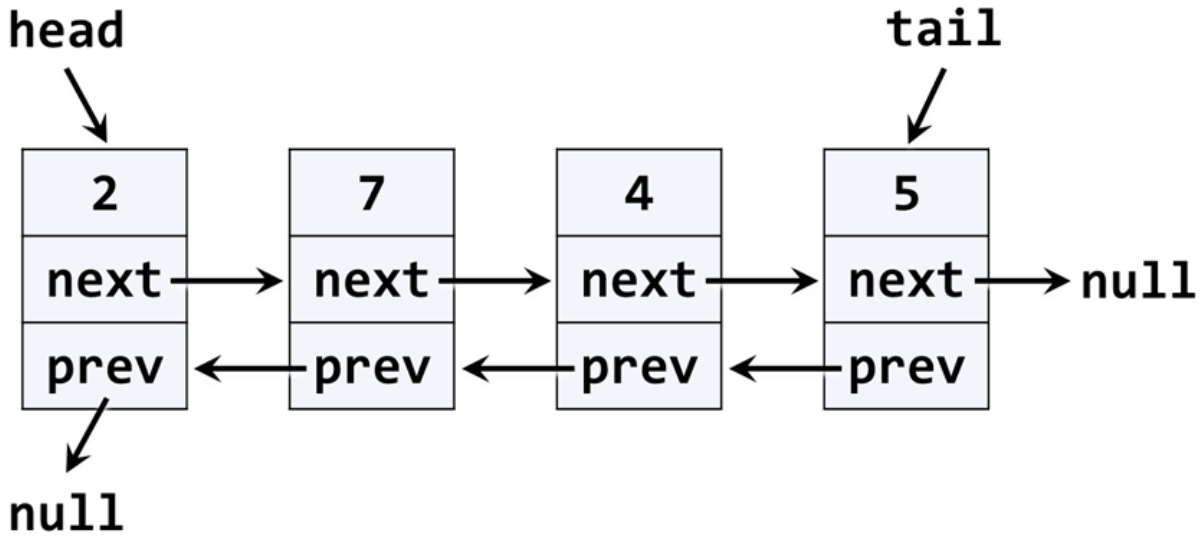
```
public void AddFirst(T element)
{
    if (this.Count == 0)
    {
        this.Head = this.Tail = new ListNode<T>(element);
    }
    else
    {
        var newone = new ListNode<T>(element);
        newone.Next = this.Head;
        newone.Prev = newone;
        this.Head = newone;
    }
    this.Count++;
}

public void AddLast(T element)
{
    if (this.Count == 0)
    {
        this.Head = this.Tail = new ListNode<T>(element);
    }
    else
    {
        var newone = new ListNode<T>(element);
        newone.Prev = this.Tail;
        this.Tail.Next = newone;
        this.Tail = newone;
    }
    this.Count++;
}

public T RemoveFirst()
{
    if (this.Count == 0)
    {
        throw new InvalidOperationException("Empty List!");
    }
    var first = this.Head.Value;
    this.Head = this.Head.Next;
    if (this.Head != null)
    {
        this.Head.Prev = null;
    }
    else
    {
        this.Tail = null;
    }
    this.Count--;
    return first;
}

public T RemoveLast()
{
```

```
        if (this.Count == 0)
        {
            throw new InvalidOperationException("Empty List!");
        }
        T last = this.Tail.Value;
        this.Tail = this.Tail.Prev;
        this.Tail.Next = null;
        this.Count--;
        return last;
    }

    public T[] ToArray()
    {
        var array = new T[this.Count];
        int index = 0;
        var current = this.Head;
        while (current != null)
        {
            array[index++] = current.Value;
            current = current.Next;
        }
        return array;
    }

    public IEnumerator<T> GetEnumerator()
    {
        var current = this.Head;
        while (current != null)
        {
            yield return current.Value;
            current = current.Next;
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return this.GetEnumerator();
    }
}

class Program
{
    static void Main(string[] args)
    {
        DoubleLinkedList<int> numbers = new DoubleLinkedList<int>();
        numbers.AddFirst(112);
        numbers.AddFirst(911);
        numbers.AddFirst(166);
        numbers.AddFirst(150);
        Console.WriteLine(string.Join(" ", numbers));

        DoubleLinkedList<string> text = new DoubleLinkedList<string>();
        text.AddFirst("Hello");
        text.AddFirst("World");
        Console.WriteLine(string.Join(" ", text));
```

```
    }
}
```

Задача 2.8. Свързана опашка

Имплементирайте опашката използвайки "двусвързан списък ":



Използвайте този код като скелет:

```
public class LinkedQueue<T>
{
    public int Count { get; private set; }
    public void Enqueue(T element) { … }
    public T Dequeue() { … }
    public T[] ToArray() { … }

    private class QueueNode<T>
    {
        public T Value { get; private set; }
        public QueueNode<T> NextNode { get; set; }
        public QueueNode<T> PrevNode { get; set; }
    }
}
```

Разгледайте и модифицирайте кода за DoublyLinkedList<T> класа. Ако опашката е празна, Dequeue() трябва да хвърля InvalidOperationException.

Решение

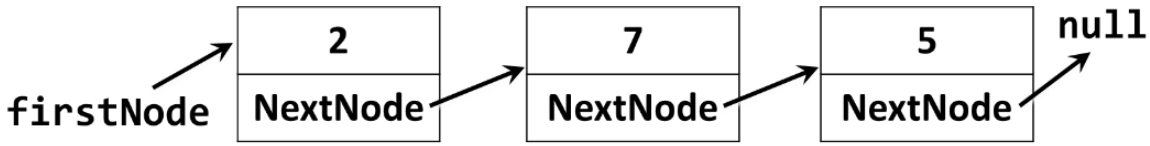
```
public class QueueNode<T>
{
    public T Value { get; private set; }

    public QueueNode<T> Next { get; set; }
```

```
    public QueueNode<T> Prev { get; set; }

    public QueueNode(T value)
    {
        this.Value = value;
    }
}

public class LinkedQueue<T> : IEnumerable<T>
{
    public int Count { get; private set; }
    private QueueNode<T> head;
    private QueueNode<T> tail;

    public LinkedQueue()
    {
        this.head = null;
        this.tail = null;
        this.Count = 0;
    }

    public void Enqueue(T item)
    {
        if (this.Count == 0)
        {
            this.head = new QueueNode<T>(item);
            this.tail = new QueueNode<T>(item);
        }
        else
        {
            QueueNode<T> newTail = new QueueNode<T>(item);
            newTail.Prev = this.tail;
            this.tail.Next = newTail;
            this.tail = newTail;
            if (this.Count == 1) this.head.Next = newTail;
        }
        this.Count++;
    }

    public T Dequeue()
    {
        if (this.Count == 0)
        {
            throw new InvalidOperationException("Empty!");
        }

        QueueNode<T> firstElement = this.head;
        this.head = this.head.Next;
        if (this.head != null)
        {
            this.head.Prev = null;
        }
        else
        {
```

```
            this.tail = null;
        }
        this.Count--;
        return firstElement.Value;
    }

    public IEnumerator<T> GetEnumerator()
    {
        var currentNode = this.head;
        while (currentNode != null)
        {
            T val = currentNode.Value;
            currentNode = currentNode.Next;
            yield return val;

        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return this.GetEnumerator();
    }
}

class Program
{
    static void Main(string[] args)
    {
        LinkedQueue<int> numbers = new LinkedQueue<int>();
        numbers.Enqueue(112);
        numbers.Enqueue(911);
        numbers.Enqueue(166);
        numbers.Enqueue(150);
        Console.WriteLine(string.Join(" ", numbers));

        LinkedQueue<string> text = new LinkedQueue<string>();
        text.Enqueue("Hello");
        text.Enqueue("World");
        Console.WriteLine(string.Join(" ", text));
    }
}
```

Задача 2.9. Свързан стек

Имплементирайте стек чрез "свързан списък":



Използвайте следния код като за начало:

```
public class LinkedStack<T>
{
    private Node<T> firstNode;
    public int Count { get; private set; }
    public void Push(T element) { … }
    public T Pop() { … }
    public T[] ToArray() { … }

    private class Node<T>
    {
        private T value;
        public Node<T> NextNode { get; set; }
        public Node(T value, Node<T> nextNode = null) { … }
    }
}
```

Push(element) операцията трябва да създаде нов Node<T> и да го зададе като firstNode: this.firstNode = new Node<T>(element, this.firstNode).

Pop() операцията трябва да върне firstNode и да го замени с firstNode.NextNode. Ако стекът е празен, то трябва да се хвърли InvalidOperationException.

Решение

```
public class Node<T>
{
    public T Value { get; set; }

    public Node<T> Next { get; set; }

    public Node(T value, Node<T> next = null)
    {
        this.Value = value;
        this.Next = next;
    }
}

public class LinkedStack<T> : IEnumerable<T>
{
    private Node<T> first;

    public int Count { get; private set; }

    public void Push(T element)
    {
        if (Count == 0)
        {
            this.first = new Node<T>(element);
        }
        else
        {
            Node<T> next = this.first;
```

```
            this.first = new Node<T>(element, next);
        }
        Count++;
    }

    public T Pop()
    {
        Node<T> node = this.first;
        this.first = first.Next;
        this.Count--;
        return first.Value;
    }

    public T[] ToArray()
    {
        T[] result = new T[Count];
        Node<T> current = this.first;
        int ind = 0;
        while (current != null)
        {
            result[ind] = current.Value;
            current = current.Next;
            ind++;
        }
        return result.ToArray();
    }

    public IEnumerator<T> GetEnumerator()
    {
        var items = this.ToArray();
        for (int i = 0; i < items.Length; i++)
        {
            yield return items[i];
        }
    }
    IEnumerator IEnumerable.GetEnumerator()
    {
        return this.GetEnumerator();
    }
}

class Program
{
    static void Main(string[] args)
    {
        LinkedStack<int> numbers = new LinkedStack<int>();
        numbers.Push(112);
        numbers.Push(911);
        numbers.Push(166);
        numbers.Push(150);
        Console.WriteLine(string.Join(" ", numbers));

        LinkedStack<string> text = new LinkedStack<string>();
        text.Push("Hello");
        text.Push("World");
```

```
        Console.WriteLine(string.Join(" ", text));
    }
}
```

Тема 3. Алгоритми върху линейни структури от данни

Задача 3.1. Принадлежи ли дадено число на масив

Напишете програма, която чете от конзолата последователност от цели числа на един ред, разделени с интервал и на втори ред число, което се проверява дали съществува в масива от първия ред. Ако числото съществува в масива, се извежда "{number} Exists in the List", в противен случай - "{Number} Not exists in the List".

Решение

```
class Program
{
    static void Main(string[] args)
    {
        List<int> list = Console.ReadLine().Split().Select(int.Parse).ToList();

        int number = int.Parse(Console.ReadLine());

        if (list.Contains(number)) Console.WriteLine($"{number} Exists in the List",
number);
        else Console.WriteLine($"{number} Not exists in the ", number);
    }
}
```

Задача 3.2. Метод Insert

Напишете програма, която чете от конзолата възходяща последователност от цели числа на един ред, разделени с интервал и на втори ред число, което се вмъква на такава позиция, че ново получения масив отново да е възходящо подреден. Изведете:

a) Ново получения масив
b) Двата масива – този, преди вмъкването и другия – след вмъкването

Решение

```
class Program
{
    static void Main(string[] args)
    {
        List<int> list = Console.ReadLine().Split().Select(int.Parse).ToList();

        int number = int.Parse(Console.ReadLine());

        bool inserted = false;
        List<int> newList = new List<int>();
        foreach (var item in list)
        {
            if (inserted)
            {
```

```
                newList.Add(item);
                continue;
            }
            if (number > item) newList.Add(item);
            else
            {
                newList.Add(number);
                newList.Add(item);
                inserted = true;
            }
        }

        Console.WriteLine(string.Join(" ", newList));
    }
}
```

Задача 3.3. Търсене в подреден масив

Определете сложността (максималния брой стъпки) на алгоритъма:

Напишете програма, която чете от конзолата възходяща последователност от цели числа на един ред, разделени с интервал и на втори ред число, което се проверява дали съществува в масива от първия ред. Ако числото съществува в масива, се извежда "{number} Exists in the Array", в противен случай - "{Number} Not exists in the Array".

Решение

```
namespace Solution
{
    public class Program
    {
        static void Main(string[] args)
        {
            int[] array = Console.ReadLine().Split().Select(int.Parse).ToArray();

            int number = int.Parse(Console.ReadLine());

            if (array.Contains(number))
            {
                Console.WriteLine($"{number} Exists in the Array");
            }
            else
            {
                Console.WriteLine($"{number} Not exists in the Array");
            }
        }
    }
}
```

Задача 3.4. Търсене в подреден списък

Определете сложността (максималния брой стъпки) на алгоритъма:

Напишете програма, която чете от конзолата възходяща последователност от цели числа на един ред, разделени с интервал и на

втори ред число, което се проверява дали съществува в списъка от първия ред. Ако числото съществува в масива, се извежда "{number} Exists in the List", в противен случай - "{Number} Not exists in the List".

Решение

```
namespace Solution
{
    public class Program
    {
        static void Main(string[] args)
        {
            List<int> array = Console.ReadLine().Split().Select(int.Parse).ToList();

            int number = int.Parse(Console.ReadLine());

            if (array.Contains(number))
            {
                Console.WriteLine($"{number} Exists in the List");
            }
            else
            {
                Console.WriteLine($"{number} Not exists in the List");
            }
        }
    }
}
```

Задача 3.5. Сбор и Средно аритметично

Напишете програма, която чете от конзолата последователност от числа (на един ред, разделени с интервал). Изчислява и отпечатате сбора и средната стойност на елементите на последователността. Запазва последователността в List<int>. Закръглете средната стойност до втората цифра след десетичния разделител.

Вход	Изход
4 5 6	Sum=15; Average=5.00
1 1	Sum=2; Average=1.00
	Sum=0; Average=0.00
10	Sum=10; Average=10.00
2 2 1	Sum=5; Average=1.67

Решение

```
class Program
{
    static void Main(string[] args)
    {
        List<int> list = Console.ReadLine().Split().Select(int.Parse).ToList();

        int sum = 0;
        float avr = 0;
```

```
        foreach (var item in list)
        {
            sum = sum + item;
        }
        avr = sum / list.Count;

        Console.WriteLine("Sum={0}; Average={1:f2}", sum, avr);
    }
}
```

Задача 3.6. Намиране на най-малко число

Напишете програма, която чете от конзолата последователност от цели числа на един ред, разделени с интервал. Намерете най-малкото от тях и го изведете.

Подсказка

1. В променлива minimum запишете първото число
2. Сравнявайте това число с всички останали (от второто до последното) и ако някое от тях е по-малко, то в minimum запазете неговата стойност
3. Изведете стйността на minimum.

Решение

```
class Program
{
    static void Main(string[] args)
    {
        List<int> list = Console.ReadLine().Split().Select(int.Parse).ToList();

        var minimum = list.First();

        for (int i = 1; i < list.Count; i++)
        {
            if (list[i] < minimum)
            {
                minimum = list[i];
            }
        }

        Console.WriteLine(minimum);
    }
}
```

Задача 3.7. Наредени двойки

Напишете програма, която чете от конзолата последователност от две цели числа m и n . Да се изведат всички възможни наредени двойки цели числа (p, q) които се менят съответно за p [1..m], q в [1..n]

Решение

```
class Program
{
    static void Main(string[] args)
    {
```

```
        int m = int.Parse(Console.ReadLine());
        int n = int.Parse(Console.ReadLine());

        for (int p = 1; p <= m; p++)
        {
            for (int q = 1; q <= n; q++)
            {
                Console.WriteLine($"{p} {q}");
            }
        }
    }
}
```

Задача 3.8. Сливане на списъци

На два реда от конзолата се въвеждат два подредени списъка от цели числа List<int> с разделител интервал. Да се изведе нов списък List<int>, в който да са слети двата списъка, отново подредени.

Решение

```
class Program
{
    static void Main(string[] args)
    {
        List<int> firstList = Console.ReadLine().Split().Select(int.Parse).ToList();
        List<int> secondList =
Console.ReadLine().Split().Select(int.Parse).ToList();

        firstList.AddRange(secondList);
        firstList.Sort();

        Console.WriteLine(String.Join(" ", firstList));
    }
}
```

Задача 3.9. Подреждане на думи

Определете сложността на програма, която чете от конзолата последователност от думи (символни низове на един ред, разделени с интервал). Подредете ги по азбучен ред. Запазете последователността в **List<string>**

Подсказка

1. Използвайте алгоритъма от задачата „Намиране на най-малко число“, променете и допълнете
2. Намерете „най-малката“ по азбучен ред дума и я запазете в променлива minimum и запомнете позицията ѝ в променлива minimumPos
3. На позиция minimumPos в списъка запишете първата дума от списъка
4. На първа позиция в списъка запишете стойността minimum
5. Повторете стъпки от 1 до 3 за елементите на списъка от втора до последа позиция

Вход	Изход

wow softuni alpha	alpha softuni wow
hi	hi

Решение

```
class Program
{
    static void Main(string[] args)
    {
        List<string> words = Console.ReadLine().Split().ToList();

        List<string> result = new List<string>();
        var minimum = words.First();
        for (int m = 0; m < words.Count; m++)
        {
            for (int i = 0; i < words.Count; i++)
            {
                if (words[i].CompareTo(minimum) < 0)
                {
                    minimum = words[i];
                }
            }
            result.Add(minimum);
            words.Remove(minimum);
            minimum = words.First();
        }
        result.Add(words.First());

        Console.WriteLine(String.Join(" ", result));
    }
}
```

Задача 3.10. Най-дълга последователност

Съставете програма, която намира най-дългата последователност от равни числа в даден списък от цели числа List<int> и връща резултата като нов List<int>. Ако няколко поредици имат същата най-дълга дължина, върнете най-лявата от тях.

Вход	Изход
12 2 7 4 3 3 8	3 3
2 2 2 3 3 3	2 2 2
4 4 5 5 5	5 5 5
1 2 3	1
0	0
4 2 3 4 4	4 **4**

Решение

```
class Program
{
```

```
    static void Main(string[] args)
    {
        List<int> list = Console.ReadLine().Split().Select(int.Parse).ToList();
        List<int> result = new List<int>();

        int number = 0, max = 0; // 6
        for (int i = 0; i < list.Count - 1; i++)
        {
            int counter = 1;
            for (int j = i + 1; j < list.Count; j++)
            {
                if (list[i] == list[j]) counter++;
                else break;
            }
            if (counter > max)
            {
                number = list[i];
                max = counter;
            }
        }

        for (int i = 0; i < max; i++) result.Add(number);

        Console.WriteLine(string.Join(" ", result));
    }
}
```

Задача 3.11. Remove/Add Method

Определете сложността (максималния брой стъпки) на програма, която чете от конзолата възходящ списък от цели числа на един ред, разделени с интервал и на втори ред число, за което се проверява дали е в списъка или не. Ако е, то то се премахва от него , а ако го няма – се добавя на такова място, че списъка отново да е подреден. Изведете:

1. Ново получения списък
2. Двата списъка – входния и ново получения

Решение

```
class Program
{
    static void Main(string[] args)
    {
        List<int> list = Console.ReadLine().Split().Select(int.Parse).ToList();
        List<int> newList = new List<int>();
        newList.AddRange(list);

        int number = int.Parse(Console.ReadLine());

        if (list.Contains(number)) list.Remove(number);
        else newList.Add(number);
        newList.Sort();

        Console.WriteLine(String.Join(" ", list));
        Console.WriteLine(String.Join(" ", newList));
```

```
    }
}
```

Задача 3.12. Средно аритметично и сума на списък

Напишете програма, която прочита от конзолата поредица от цели положителни числа. Поредицата спира когато се въведе празен ред. Програмата трябва да изчислява сумата и средното аритметично на поредицата. Използвайте List<int>.

Решение

```
class Program
{
    static void Main(string[] args)
    {
        List<int> list = Console.ReadLine().Split().Select(int.Parse).ToList();
        Console.WriteLine("Sum = {0}", list.Sum());
        Console.WriteLine("Average = {0}", list.Average());
    }
}
```

Задача 3.13. Обръщане на последователността

Напишете програма, която прочита N цели числа от конзолата и ги отпечатва в обратен ред. Използвайте класа Stack<int>.

Решение

```
class Program
{
    static void Main(string[] args)
    {
        Stack<int> stack = new Stack<int>();

        int n = int.Parse(Console.ReadLine());
        while (n > 0)
        {
            var number = int.Parse(Console.ReadLine());
            stack.Push(number);
            n--;
        }

        foreach (var item in stack)
        {
            Console.Write("{0} ", item);
        }
    }
}
```

Задача 3.14. Филтриране

Напишете програма, която премахва всички отрицателни числа от дадена редица. Пример: array = {19, -10, 12, -6, -3, 34, -2, 5} → {19, 12, 34, 5}

Решение

```
class Program
{
```

```
    static void Main(string[] args)
    {
        Queue<int> queue = new Queue<int>();

        var list = Console.ReadLine().Split().Select(int.Parse).ToList();

        foreach (var item in list)
        {
            if (item > 0)
            {
                queue.Enqueue(item);
            }
        }

        Console.WriteLine(string.Join(" ", queue));
    }
}
```

Задача 3.15. Филтриране на нечетен броя срещания

Напишете програма, която при дадена редица изтрива всички числа, които се срещат нечетен брой пъти.

Вход	Изход
4, 2, 2, 5, 2, 3, 2, 3, 1, 5, 2	5, 3, 3, 5

Решение

```
class Program
{
    static void Main(string[] args)
    {
        var list = Console.ReadLine().Split().Select(int.Parse).ToList();

        int[] counter = new int[10];
        for (int i = 0; i < counter.Length; i++)
        {
            counter[i] = 0;
        }
        for (int i = 0; i < list.Count; i++)
        {
            counter[list[i]]++;
        }

        Queue<int> queue = new Queue<int>();
        foreach (var item in list)
        {
            if (counter[item] % 2 == 0)
            {
                queue.Enqueue(item);
            }
        }

        Console.WriteLine(string.Join(" ", queue));

        // v2
```

```
        // List<int> list = Console.ReadLine().Split('
').Select(int.Parse).ToList();
        // List<int> res = list.Where(x => list.Count(c => c == x) % 2 ==
0).ToList();
        // Console.WriteLine(string.Join(" ",res));
    }
}
```

Задача 3.16. Честота на срещания

Напишете програма, която по даден масив от цели числа в интервала [0..1000], намира по колко пъти се среща всяко число.

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

Вход	Изход
3, 4, 4, 2, 3, 3, 4, 3, 2	2 - 2 пъти 3 - 4 пъти 4 - 3 пъти

Решение

```
class Program
{
    static void Main(string[] args)
    {
        var list = Console.ReadLine().Split().Select(int.Parse).ToList();

        SortedDictionary<int, int> counter = new SortedDictionary<int, int>();
        foreach (var item in list)
        {
            if (counter.ContainsKey(item))
            {
                counter[item]++;
            }
            else
            {
                counter.Add(item, 1);
            }
        }

        foreach (var item in counter.Keys)
        {
            Console.WriteLine("{0} -> {1}", item, counter[item]);
        }
    }
}
```

Задача 3.17. Файлове на твърдия диск

Използвайки опашка реализирайте пълно обхождане на всички директории на твърдия ви диск и ги отпечатвайте на конзолата.

Решение

```
class Program
{
```

```
    private static Queue<string> folders = new Queue<string>();

    private static void ProcessFolder(string path)
    {
        string[] dirs = Directory.GetDirectories(path);
        foreach (var dir in dirs)
        {
            folders.Enqueue(dir);
        }
    }

    public static void Main()
    {
        ProcessFolder(@"C:\");

        while (folders.Count > 0)
        {
            try
            {
                var folder = folders.Dequeue();
                Console.WriteLine(folder);
                ProcessFolder(folder);
            }
            catch (Exception ex)
            {
                Console.WriteLine("ERROR: {0}", ex.Message);
            }
        }
    }
}
```

Задача 3.18. Мажорант на масив

Мажорант на масив от N елемента е стойност, която се среща поне N/2+1 пъти. Напишете програма, която по даден масив от числа намира мажоранта на масива и го отпечатва. Ако мажоранта не съществува – отпечатва "The majorant does not exists!".

Вход	Изход
2, 2, 3, 3, 2, 3, 4, 3, 3	3
2, 3, 4, 5, 6, 7, 8, 3	The majorant does not exists!

Решение

```
class Program
{
    static void Main(string[] args)
    {
        List<int> list = Console.ReadLine().Split(' ').Select(int.Parse).ToList();
        int res = list.Where(x => list.Count(c => c == x) >= list.Count() / 2 +
1).ToList().Distinct().FirstOrDefault();
        if (res != 0) Console.WriteLine(res);
        else Console.WriteLine("The majorant does not exists!”.");
    }
```

```
}
```

Задача 3.19. Мода на масив

Мода на масив от N елемента е стойност, която се среща най-често. Напишете програма, която по даден масив от числа намира модата на масива и го отпечатва. Ако има няколко моди се извежда средно аритметичната им стойност

Вход	Пояснения	Изход
2, 2, 3, 3, 2, 3, 4, 3, 3		3
3, 3, 4, 5, 6, 7, 4, 2, 2	3, 4 и 5 се срещат по два пъти => (3+4+5)/2=6	5

Решение

```
class Program
{
    static void Main(string[] args)
    {
        List<int> list = Console.ReadLine().Split(' ').Select(int.Parse).ToList();
        bool moreThanOne = false;
        int maxCount = 0;
        int number = -1;
        foreach (var item in list)
        {
            int counter = list.Count(c => c == item);
            if (counter >= maxCount && number != item)
            {
                if (counter == maxCount && !moreThanOne)
                {
                    moreThanOne = true;
                }
                else
                {
                    maxCount = counter;
                    number = item;
                }
            }
        }
        double mod = 0;
        if (moreThanOne)
        {
            mod = (double)list.Where(x => list.Count(c => c == x) ==
maxCount).ToList().Distinct().Sum() / maxCount;
        }
        else
        {
            mod = number;
        }
        Console.WriteLine(mod);
    }
}
```

Задача 3.20. Обръщане на числа със стека

Напишете програма, която чете N цели числа от конзолата и ги обръща в обратен на въвеждането ред, чрез стек. Използвайте Stack<int> класа от .NET Framework. Просто поставете (put) въвежданите числа в стека и ги вземете (pop) после от стека.

Примери

Вход	Изход
1 2 3 4 5	5 4 3 2 1
1	1
(empty)	(empty)
1 -2	-2 1

Решение

```
class Program
{
    static void Main(string[] args)
    {
        Stack<int> stack = new Stack<int>();
        int n = int.Parse(Console.ReadLine());
        for (int i = 0; i < n; i++)
        {
            stack.Push(int.Parse(Console.ReadLine()));
        }
        for (int i = 0; i < n; i++)
        {
            Console.WriteLine(stack.Pop());
        }
    }
}
```

Задача 3.21. Изчислете редицата с опашка

Дадена е следната последователност от числа:

- $S_1 = N$
- $S_2 = S_1 + 1$
- $S_3 = 2*S_1 + 1$
- $S_4 = S_1 + 2$
- $S_5 = S_2 + 1$
- $S_6 = 2*S_2 + 1$
- $S_7 = S_2 + 2$
- …

Използвайте класа Queue<T> и напишете програма, която извежда първите 50 члена за даденото N

Примери:

Вход	Изход
2	2, 3, 5, 4, 4, 7, 5, 6, 11, 7, 5, 9, 6, …
-1	-1, 0, -1, 1, 1, 1, 2, …
1000	1000, 1001, 2001, 1002, 1002, 2003, 1003, …

Решение

```
class Program
{
    static void Main(string[] args)
    {
        int n = int.Parse(Console.ReadLine());
        Console.WriteLine(n);
        int poped = n;
        Queue<int> q = new Queue<int>();
        for (int i = 1; i < 50; i++)
        {
            int first = poped + 1;
            int second = poped * 2 + 1;
            int third = poped + 2;
            q.Enqueue(first);
            q.Enqueue(second);
            q.Enqueue(third);
            poped = q.Dequeue();
            Console.WriteLine(first);
            Console.WriteLine(second);
            Console.WriteLine(third);
        }
    }
}
```

Задача 3.22. Редица N → M

Дадени са числата n и m и следните операции:

a) n → n + 1
b) n → n + 2
c) n → n * 2

Напишете програма, която намира най-късата редица от операции от списъка по-долу, който започва от n и завършва в m. Ако съществуват няколко най-къси редици, намерете първата от тях.

Примери

Вход	Изход
3 10	3 -> 5 -> 10
5 -5	(няма решение)
10 30	10 -> 11 -> 13 -> 15 -> 30

Подсказка: използвайте опашка и следващия алгоритъм:

1. създайте опашка от числа
2. опашка ← n
3. докато (опашката не е празна)
    1. опашка → e
    2. ако (e < m)
        i. опашка ← e + 1
        ii. опашка ← e + 2
        iii. оопшка ← e * 2
    3. ако (e == m) Print-Solution; край

С по-горния алгоритъм ще намерите решение, или ще откриете, че то не съществува. Той не може да отпечата числата, включващи редицата n → m.

За да отпечатате редицата от стъпки, за да достигне м, започвайки от n, ще трябва да запазите също и предишния елемент. Вместо с опашка от числа, използвайте опашка от елементи. Всеки елемент ще запази число и указател към предишния елемент. Промените в алгоритъма са примерно такива:

Алгоритъм Find-Sequence (n, m):

1. създайте опашка от елемент {стойност, предходен_елемент }
2. опашка ← { n, null }
3. докато (опашката не е празна)
    1. опашка → елемент
    2. ако (елемент.стойност < m)
        i. опашка ← { елемент.стойност + 1, елемент}
        ii. queue ← { елемент.стойност + 2, елемент }
        iii. queue ← { елемент.стойност * 2, елемент }
    3. ако (елемент.стойност == m) Print-Solution; край

Алгоритъм Print-Solution (item):

1. докато (елемента не е null)
    1. отпечатай елемент.стойност
    2. елемент=елемент.предходен_елемент

Решение

```
public class Node
{
    public int Element { get; set; }

    public Node Previous { get; set; }

    public Node(int e, Node p)
    {
        this.Element = e;
        this.Previous = p;
    }
```

```
}

class Program
{

    public static Stack<int> FindSolution(Node e)
    {
        Stack<int> stack = new Stack<int>();
        while (e != null)
        {
            stack.Push(e.Element);
            e = e.Previous;
        }
        return stack;
    }

    static void Main(string[] args)
    {
        Console.Write("n=");
        int n = int.Parse(Console.ReadLine());
        Console.Write("m=");
        int m = int.Parse(Console.ReadLine());

        Queue<Node> queue = new Queue<Node>();
        queue.Enqueue(new Node(n, null));

        while (queue.Count > 0)
        {
            Node e = queue.Dequeue();
            if (e.Element < m)
            {
                queue.Enqueue(new Node(e.Element + 1, e));
                queue.Enqueue(new Node(e.Element + 2, e));
                queue.Enqueue(new Node(e.Element * 2, e));
            }
            if (e.Element == m)
            {
                var stack = FindSolution(e);
                Console.WriteLine(string.Join(" -> ", stack));
                return;
            }
        }

        Console.WriteLine("No Solution");
    }
}
```

Тема 4. Алгоритми за сортиране

Задача 4.1. Сортиране чрез пряка селекция

Сортирайте един масив от елементи с помощта на алгоритъма за пряка селекция (Selection Sort).

Примери

Вход	Изход
5 4 3 2 1	1 2 3 4 5

Подсказки

Първо разгледайте описанието на алгоритъма от Уикипедия и визуализацията му във Visualgo.net. Или пък се облегнете на стола и му се насладете под формата на цигански народен танц, представен от AlgoRythmics.

В реализацията опишете механизъма на самото сортиране. Алгоритъмът накратко е:

- Намираме най-малката стойност в масива
- Разменяме я с първата несортирана стойност (в началото това е първия елемент, после е втория и т.н.)
- Това се повтаря върху останалата част от масива (без сортираните стойности). Тоест:
  - търсим най-малката стойност от втория елемент до края и я поставяме на втора позиция
  - търсим най-малката стойност от третия елемент до края и я поставяме на трета позиция
  - и т.н., докато стигнем да сравняваме предпоследния и последния елемент

Решение

```
public class Help
{
    public static void Swap<T>(T[] collection, int from, int to)
    {
        T temp = collection[from];
        collection[from] = collection[to];
        collection[to] = temp;
    }

    public static bool IsLess(IComparable first, IComparable second)
    {
        return first.CompareTo(second) < 0;
    }
}

public class SelectionSort
{
    public static void Sort<T>(T[] collection) where T : IComparable
    {
        for (int index = 0; index < collection.Length; index++)
        {
            int min = index;
            for (int curr = index + 1; curr < collection.Length; curr++)
            {
                if (Help.IsLess(collection[curr], collection[min]))
```

```
                {
                    min = curr;
                }
            }
            Help.Swap(collection, index, min);
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("SelectionSort:");

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

        Console.WriteLine(String.Join(" ", array));

        SelectionSort.Sort(array);

        Console.WriteLine(String.Join(" ", array));
    }
}
```

Задача 4.2. Визуализация на сортирането

Модифицирайте кода на предната задача, така че да служи за визуализация на метода на сортиране чрез пряка селекция. За целта на конзолата отпечатвайте всяка съществена стъпка от алгоритъма, кои елементи ще се разменят и как изглежда масивът след всяка размяна. Обсъдете в клас коя визуализация се е получила най-прегледна и разбираема и защо.

Подсказки

Модифицирайте метода Swap( ), така че да извежда кои елементи се разменят и да отпечатва масива след размяната.

Задача 4.3. Класиране на приема

От ръководството на вашето училище са ви помолили да участвате в комисията по приема на нови ученици. Ще ви предоставят лист с точките на всеки един ученик, кандидат за дадена паралелка, а вие трябва да направите програма, която ги извежда сортирани в не нарастващ ред, за да може комисията да определи кои ученици са приети в паралелката, кои са резерви и т.н.

Пример

Вход	Изход
95 98 103 109 48 92 25 106 160	160 **109 106 103 98 95 92 48 25**

Решение

```
public static class Help
{
    public static void Swap<T>(T[] elements, int first, int second)
    {
        T temp = elements[first];
        elements[first] = elements[second];
        elements[second] = temp;
    }

    public static bool IsLess(IComparable first, IComparable second)
    {
        return first.CompareTo(second) < 0;
    }

    public static bool IsSorted<T>(T[] elements) where T : IComparable
    {
        for (int i = 1; i < elements.Length; i++)
        {
            if (elements[i - 1].CompareTo(elements[i]) > 0)
            {
                return false;
            }
        }
        return true;
    }
}

public static class Sort
{
    public static void Selection<T>(T[] elements) where T : IComparable
    {
        for (int i = 0; i < elements.Length; i++)
        {
            int min = i;
            for (int j = i + 1; j < elements.Length; j++)
            {
                if (Help.IsLess(elements[min], elements[j]))
                {
                    min = j;
                }
            }
            Help.Swap(elements, min, i);
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        int[] arr = { 95, 98, 103, 109, 48, 92, 25, 106, 160 };

        Console.WriteLine(string.Join(" ", arr));
```

```
        Sort.Selection<int>(arr);

        Console.WriteLine(string.Join(" ", arr));
    }
}
```

Задача 4.4. Ученици по физическо

Вашият учител по физическо строява класовете винаги по височина. Можете ли да направите програма, която може да определи колко е височината на произволни двама от така подредените ученици при указване на поредния им номер в строя?

Вход

- Входните данни трябва да се прочетат от конзолата.
- На първия ред се подават неподредени цели числа, отделени едно от друго с интервал. Това са височините в сантиметри на всички ученици, преди да се подредят в строя.
- На втория ред се подават две цели числа, всяко в интервала от едно до броят на числата на първия ред. Това са позициите на тези ученици в подредения строй, на които искаме да разберем височините.
- Входните данни винаги ще са валидни и в описания формат. Не е необходимо да бъдат изрично проверявани.

Изход

- Изходните данни трябва да бъдат отпечатани на конзолата.
- На първия и единствен ред трябва да бъдат отпечатани две цели числа, разделени с интервал - височините на учениците, чиято позиция сме указали във входните данни.

Пример

Вход	Изход	Коментар
165 182 173 173 168 2 5	173 165	Класът, подреден по височина е 182 173 173 168 165. Височините на вторият и петият ученик са удебелени.

Подсказки

Първо сортирайте масива и после изведете стойностите на елементите с указаните номера. Помислете може ли кодът да се оптимизира така, че да не се налага сортиране на целия масив.

Решение

```
class Program
{
    static void Main(string[] args)
    {
        int[] arr = Console.ReadLine().Split().Select(int.Parse).ToArray();
        int[] pos = Console.ReadLine().Split().Select(int.Parse).ToArray();
        int left = pos[0] - 1, right = pos[1] - 1;
```

```
        arr = arr.OrderByDescending(x => x).ToArray();
        Console.WriteLine($"{arr[left]} {arr[right]}");
    }
}
```

Задача 4.5. Кросово бягане

Миналата седмица във вашия град бе проведено масово кросово бягане. Тъй като участниците са били наистина много, те са били разпределени да се състезават в два различни дни, по един и същ маршрут. На финиша е имало съдии, които са записали времето на пристигането на всеки състезател в секунди. На вас се пада честта да обедините тези два списъка в един и да оформите общата класация. Напишете програма, която да ви улесни в изпълнението на задачата.

Вход

- Входните данни трябва да се прочетат от конзолата.
- На първия ред ще има цели числа, подредени в ненамаляващ ред и отделени едно от друго с интервал. Това са времената на състезателите, участвали в кроса през първия ден.
- На втория ред ще са времената на състезателите, участвали в кроса през втория ден. Те са в същия формат, както числата от първия ред, но не е сигурно, че са същия брой.
- Входните данни винаги ще са валидни и в описания формат. Не е необходимо да бъдат изрично проверявани.

Изход

- Изходните данни (сортираният масив с класацията на всички състезатели, участвали в кроса, подредени в ненамаляващ ред) трябва да бъдат отпечатани на конзолата.

Пример

Вход	Изход	Коментар
956 989 1037 1095 948 992 1025 1062 1160	948 956 989 992 1025 1037 1062 1095 1160	За яснота участниците от първия ден са удебелени

Решение

```
class Program
{
    static void Main(string[] args)
    {
        int[] day1 = Console.ReadLine().Split().Select(int.Parse).ToArray();
        int[] day2 = Console.ReadLine().Split().Select(int.Parse).ToArray();
        int[] result = day1.Concat(day2).ToArray().OrderBy(x => x).ToArray();
        Console.WriteLine(string.Join(" ", result));
    }
}
```

Задача 4.6. Сортиране чрез метода на мехурчето

Сортирайте един масив от елементи с помощта на метода на мехурчето (Bubble Sort).

Примери

Вход	Изход
5 4 3 2 1	1 2 3 4 5

Подсказки

Първо разгледайте описанието на алгоритъма от Уикипедия и визуализацията му във Visualgo.net. Или пък изберете по-нестандартната визуализация като унгарски народен танц, представен от AlgoRythmics.

В реализацията на метода опишете механизъма на самото сортиране:

- Обхождаме целия масив и сравняваме всеки 2 съседни елемента
- Ако първия е по-голям от втория, те разменят местата си.
- Това се повтаря до подреждането на масива.

Решение

```
public static class Help
{
    public static void Swap<T>(T[] elements, int first, int second)
    {
        T temp = elements[first];
        elements[first] = elements[second];
        elements[second] = temp;
    }

    public static bool IsLess(IComparable first, IComparable second)
    {
        return first.CompareTo(second) < 0;
    }

    public static bool IsSorted<T>(T[] elements) where T : IComparable
    {
        for (int i = 1; i < elements.Length; i++)
        {
            if (elements[i - 1].CompareTo(elements[i]) > 0)
            {
                return false;
            }
        }
        return true;
    }
}

public static class Sort
{
    public static void Bubble<T>(T[] elements) where T : IComparable
    {
        for (int i = 0; i < elements.Length; i++)
```

```
        {
            for (int j = 0; j < elements.Length; j++)
            {
                if (Help.IsLess(elements[i], elements[j]))
                {
                    Help.Swap(elements, i, j);
                }
            }
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        int[] numbers = Console.ReadLine().Split().Select(int.Parse).ToArray();
        Sort.Bubble(numbers);
        Console.WriteLine(string.Join(" ", numbers));
    }
}
```

Задача 4.7. Сортиране чрез вмъкване

Сортирайте един масив от елементи с помощта на метода на сортиране чрез вмъкване (Insertion Sort). Но първо разгледайте описанието на алгоритъма от Уикипедия и визуализацията му във Visualgo.net или пък мази под формата на румънски народен танц, представен от AlgoRythmics. ☺ Можете да направите сравнение между този и другите алгоритми за сортиране, които познавате, и чрез чудесната визуализация на David Galles от USF .

Примери

Вход	Изход
5 4 3 2 1	1 2 3 4 5

Подсказки

Използвайте указанията на предната задача. Алгоритъмът накратко:

- **Преместваме първия несортиран елемент наляво на мястото му**
- **Повтаряме това докато не останат несортирани елементи**

Решение

```
public static class Help
{
    public static void Swap<T>(T[] elements, int first, int second)
    {
        T temp = elements[first];
        elements[first] = elements[second];
        elements[second] = temp;
    }

    public static bool IsLess(IComparable first, IComparable second)
```

```
    {
        return first.CompareTo(second) < 0;
    }

    public static bool IsSorted<T>(T[] elements) where T : IComparable
    {
        for (int i = 1; i < elements.Length; i++)
        {
            if (elements[i - 1].CompareTo(elements[i]) > 0)
            {
                return false;
            }
        }
        return true;
    }
}

public static class Sort
{
    public static void Insertion<T>(T[] elements) where T : IComparable
    {
        for (int i = 0; i < elements.Length; i++)
        {
            int prev = i - 1;
            int curr = i;
            while (true)
            {
                if (prev < 0 || Help.IsLess(elements[prev], elements[curr]))
                {
                    break;
                }
                Help.Swap(elements, curr, prev);
                curr--;
                prev--;
            }
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        int[] numbers = Console.ReadLine().Split().Select(int.Parse).ToArray();
        Sort.Insertion(numbers);
        Console.WriteLine(string.Join(" ", numbers));
    }
}
```

Задача 4.8. Визуализация на сортирането

Модифицирайте кода на предните две задачи, така че да служи и за визуализация на сортирането по тези два метода. За целта на конзолата отпечатвайте всяка съществена стъпка от алгоритъма, кои елементи ще

се разменят и как изглежда масивът след всяка размяна. Обсъдете в клас коя визуализация се е получила най-прегледна и разбираема и защо.

Подсказки

Модифицирайте метода Swap( ), така че да извежда кои елементи се разменят и да отпечатва масива след размяната.

Задача 4.9. Уравновесен масив

Да се напише програма, която позволява да се въведе размер на масив и стойностите на неговите елементи и после го подрежда така, че стойностите на елементите да нарастват от началото до средата на масива и след това отново да намаляват от средата до края. Елементите от първата половина на масива не трябва да преминават във втората му половина. Полученият масив трябва да бъде отпечатан.

Вход

- Входните данни трябва да се прочетат от конзолата.
- На първия ред се подава цяло четно число N, съдържащо броят на числата в масива
- На втория ред се подават N цели числа, отделени едно от друго с интервал. Това са стойностите на елементите в масива.
- Входните данни винаги ще са валидни и в описания формат. Не е необходимо да бъдат изрично проверявани.

Изход

- Изходните данни („уравновесеният" масив) трябва да бъдат отпечатани на конзолата.

Пример

Вход	Изход	Коментар
10 4 2 6 3 8 1 7 4 2 9	2 3 4 6 8 9 7 4 2 1	За яснота елементите от първата половина на масива са удебелени.

Решение

```
class Program
{
    static void Main(string[] args)
    {
        int N = int.Parse(Console.ReadLine());
        int[] arr = new int[N];
        arr = Console.ReadLine().Split().Select(int.Parse).ToArray();
        var left = arr.Take(arr.Length / 2).OrderBy(x => x).ToArray();
        var right = arr.Skip(arr.Length / 2).Take(arr.Length - arr.Length /
2).OrderByDescending(x => x).ToArray();
        Console.WriteLine(string.Join(" ", left.Concat(right).ToArray()));
    }
}
```

Задача 4.10. Сортиране по произволна колона

Напишете програма, която сортира таблица от числови стойности по произволна колона.

Вход

- Входните данни трябва да се прочетат от конзолата.
  - На първия ще ви бъдат подадени три цели числа R, C и S, разделени с интервал. R е броят на редовете в таблицата, C - броят на колоните, а S - номерът на колоната, по която трябва да бъде сортиран масива.
  - На следващите R реда ще са числата от всеки от редовете в таблицата, C на брой, разделени с интервали.
- Входните данни винаги ще са валидни и в описания формат.

Изход

- Изходните данни трябва да бъдат отпечатани на конзолата.
- На R реда трябва да бъдат изведени числата от таблицата, сортирани по указаната колона.

Подсказки

- Ще се наложи да ползвате двумерен масив или масив от масиви.
- Тествайте задачата с различно големи масиви и различни алгоритми за сортиране.
- Обърнете внимание дали при стабилни и нестабилни алгоритми се получава еднакъв резултат.
- Коментирайте в клас има ли начин за минимизиране на размяната на редовете при сортирането.

Примери

Вход	Изход	Коментар	Вход	Изход	Коментар
3 4 1 1 2 3 4 3 1 2 4 2 3 1 2	1 2 3 4 2 3 1 2 3 1 2 4	Има 3 реда и 4 колони. Вторият и третият ред трябва да разменят местата си, ако сортираме по колона 1.	4 2 2 1 2 3 1 2 3 4 4	3 1 1 2 2 3 4 4	Има 4 реда и 2 колони. Първият и вторият ред трябва да разменят местата си, ако сортираме по колона 2.

Решение

```
class Program
{
    static void Main(string[] args)
    {
        var input = Console.ReadLine().Split().Select(int.Parse).ToArray();
        int R = input[0], C = input[1], S = input[2];
        int[][] array = new int[R][];
        for (int i = 0; i < R; i++)
        {
            array[i] = Console.ReadLine().Split().Select(int.Parse).ToArray();
```

```
        }
        for (int i = 0; i < R; i++)
        {
            int minInd = 0;
            for (int j = 0; j < R; j++)
            {
                if (array[i][S] > array[j][S])
                {
                    minInd = j;
                }
            }
            int[] temp = array[i];
            array[i] = array[minInd];
            array[minInd] = temp;
        }
        for (int i = 0; i < R; i++)
        {
            Console.WriteLine(string.Join(" ", array[i]));
        }
    }
}
```

Задача 4.11. Сортиране чрез сливане

Сортирайте масив от елементи с помощта на популярния алгоритъм за сортиране чрез сливане (Merge Sort)

Примери

Вход	Изход
5 4 3 2 1	1 2 3 4 5

Решение

```
public static class Help
{
    public static void Swap<T>(T[] elements, int first, int second)
    {
        T temp = elements[first];
        elements[first] = elements[second];
        elements[second] = temp;
    }

    public static bool IsLess(IComparable first, IComparable second)
    {
        return first.CompareTo(second) < 0;
    }

    public static bool IsSorted<T>(T[] elements) where T : IComparable
    {
        for (int i = 1; i < elements.Length; i++)
        {
            if (elements[i - 1].CompareTo(elements[i]) > 0)
            {
                return false;
            }
```

```
        }
        return true;
    }
}

public static class Sort
{
    public static void Merge<T>(T[] elements) where T : IComparable
    {
        MergeAlgo(elements, 0, elements.Length);
    }

    private static void MergeAlgo<T>(T[] elements, int left, int right) where T :
IComparable
    {
        int diff = right - left;

        if (diff <= 1) return;

        int mid = left + diff / 2;
        MergeAlgo(elements, left, mid);
        MergeAlgo(elements, mid, right);

        T[] temp = new T[diff];
        int i = left, j = mid;
        for (int k = 0; k < diff; k++)
        {
            if (i == mid) temp[k] = elements[j++];
            else if (j == right) temp[k] = elements[i++];
            else if (Help.IsLess(elements[j], elements[i])) temp[k] = elements[j++];
            else temp[k] = elements[i++];
        }
        for (int k = 0; k < diff; k++)
        {
            elements[left + k] = temp[k];
        }
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        var numbers = Console.ReadLine().Split().Select(int.Parse).ToArray();
        Sort.Merge(numbers);
        Console.WriteLine(string.Join(" ", numbers));
    }
}
```

Задача 4.12. Бързо сортиране

Сортирайте един масив от елементи с помощта на популярния алгоритъм за бързо сортиране (QuickSort)

Примери

Вход	Изход
5 4 3 2 1	1 2 3 4 5

Подсказки

Можете да научите повече за алгоритъма QuickSort от Уикипедия. Страхотен инструмент за визуализиране на този алгоритъм (а и на много други) ще намерите на Visualgo.net.

Алгоритъмът накратко:

- Бързото сортиране взима несортирани части на масив и ги сортира
- Избираме опорен елемент
  - Избираме първият елемент от несортираната част и го местим по такъв начин, че всички по-малки елементи да са му отляво и всички по-големи - отдясно
- С опорния елемент на правилната позиция, сега имаме две несортирани части - една отляво и една отдясно
- Извикваме рекурсивно процедурата за всяка част
- Дъното на рекурсията е **когато частта е с размер 1 елемент, който по дефиниция е сортиран**

Решение

```
public static class Help
{
    public static void Swap<T>(T[] elements, int first, int second)
    {
        T temp = elements[first];
        elements[first] = elements[second];
        elements[second] = temp;
    }

    public static bool IsLess(IComparable first, IComparable second)
    {
        return first.CompareTo(second) < 0;
    }

    public static bool IsSorted<T>(T[] elements) where T : IComparable
    {
        for (int i = 1; i < elements.Length; i++)
        {
            if (elements[i - 1].CompareTo(elements[i]) > 0)
            {
                return false;
            }
        }
        return true;
    }
}

public static class Sort
```

```
{
    public static void Quick<T>(T[] elements) where T : IComparable
    {
        QuickAlgo(elements, 0, elements.Length - 1);
    }

    private static void QuickAlgo<T>(T[] elements, int left, int right) where T :
IComparable
    {
        if (left > right || left < 0 || right < 0) return;
        int pivot = QuickPartitionSort(elements, left, right);
        if (pivot != -1)
        {
            QuickAlgo(elements, left, pivot - 1);
            QuickAlgo(elements, pivot + 1, right);
        }
    }

    private static int QuickPartitionSort<T>(T[] elements, int left, int right)
where T : IComparable
    {
        if (left > right) return -1;
        int end = left;
        T pivot = elements[right];
        for (int i = left; i < right; i++)
        {
            if (Help.IsLess(elements[i], pivot))
            {
                Help.Swap(elements, i, end);
                end++;
            }
        }
        Help.Swap(elements, end, right);
        return end;
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        var numbers = Console.ReadLine().Split().Select(int.Parse).ToArray();
        Sort.Quick(numbers);
        Console.WriteLine(string.Join(" ", numbers));
    }
}
```

Задача 4.13. Визуализация на сортирането

Модифицирайте кода на предните две задачи, така че да служи и за визуализация на сортирането по тези два метода. За целта на конзолата отпечатвайте всяка съществена стъпка от алгоритъма, кои елементи ще се разменят и как изглежда масивът след всяка размяна. Обсъдете в клас коя визуализация се е получила най-прегледна и най-разбираема и защо.

Задача 4.14. Брой на инверсиите

Приемаме, че брой на инверсиите е колко далеч (или близо) е масивът от това да бъде сортиран. Ако списъкът е вече сортиран, тогава броят на инверсиите е 0. Ако списъкът е сортиран в обратен ред, тогава броят на инверсиите е в своя максимум.

Два елемента a[i] и a[j] формират инверсия ако a[i] > a[j] и i < j.

Открийте и отпечатайте броят на всички инверсии в даден масив от входящи данни.

Примери

Вход	Изход	Инверсии
2 4 1 3 5	3	2 1 4 1 4 3
5 4 3 2 1	10	5 4 5 3 5 2 5 1 4 3 4 2 4 1 3 2 3 1

Подсказки

Използвайте модифицирана версия на сортиране чрез сливане.

Полезно четиво: http://www.geeksforgeeks.org/counting-inversions/

Решение

```
public class Program
{
    public static void Main(string[] args)
    {
        int[] a = Console.ReadLine().Split(' ').Select(int.Parse).ToArray();
        int count = 0;
        for (int i = 0; i < a.Length; i++)
        {
            for (int j = 0; j < a.Length; j++)
            {
                if (a[i] > a[j] && i < j) count++;
            }
        }
        Console.WriteLine(count);
    }
}
```

Задача 4.15. Най-често срещано число

Да се напише програма, която в масив от непоредни положителни двуцифрени числа намира числото, което се среща най-често в масива и извежда кое е то и колко пъти се среща. Ако повече от едно число се срещат максимален брой пъти, извежда най-голямото от най-често повтарящите числа.

Вход

- Входните данни трябва да се прочетат от конзолата.
- На първия и единствен ред се подават неподредени положителни цели двуцифрени числа, отделени едно от друго с интервал.
- Входните данни винаги ще са валидни и в описания формат. Не е необходимо да бъдат изрично проверявани.

Изход

- Изходните данни трябва да бъдат отпечатани на конзолата.
- На първия ред трябва да бъдат изведени две числа, разделени с интервал - броят повторения на най-голямото от най-често срещаните числа и самото това число.

Примери

Вход	Изход	Коментари
`10 13 10 99`	`2 10`	`Най-често се повтаря 10 - 2 пъти.`
`10 13 99 10 99`	`2 99`	`И 10, и 99 се срещат по 2 пъти, но 99 е по-голямо от 10.`
`10 13 99 13 99 13`	`3 13`	`13 се повтаря най-много - 3 пъти.`

Подсказки

Използвайте модифицирана версия на алгоритъма за сортиране чрез броене.

Решение

```
public class Program
{
    public static void Main(string[] args)
    {
        int[] sequence = Console.ReadLine().Split(' ').Select(int.Parse).ToArray();
        int maxCount = 0;
        int repaintingNumber = 0;
        for (int i = 0; i < sequence.Length; i++)
        {
            int currentCount = 0;
            for (int j = i; j < sequence.Length; j++)
            {
                if (sequence[i] == sequence[j])
                {
                    currentCount++;
```

```
                }
            }
            if (currentCount > maxCount)
            {
                repaintingNumber = sequence[i];
                maxCount = currentCount;
            }
        }
        Console.WriteLine(maxCount + " " + repaintingNumber);
    }
}
```

Задача 4.16. Най-бърз подреждач

Напишете програма, която сравнява бързодействието на различните алгоритми за сортиране в четири кръга и определя кой е победителят за всеки кръг, както и финалния победител (този с най-малко време общо за четирите кръга). Използвайте два или повече от следните алгоритми за сортиране:

- Сортиране чрез вмъкване
- Метод на мехурчето
- Алгоритъм на Шел
- Сортиране чрез сливане
- Бързо сортиране
- Bucket сортиране

Във всеки кръг, всеки от алгоритмите за сортиране трябва да бъде тестван върху един от следните четири масива:

1. Масив от поредни числа от 1 до N, т.е подредени в нарастващ ред (1, 2, 3, … N)
2. Масив от поредни числа от N до 1, т.е подредени в намаляващ ред (N, N-1, N-2, … 3, 2, 1)
3. Масив от N случайни числа, всяко в диапазона от 1 до N включително.
4. Масив от N числа, съдържащ многократно повторение на числата от 1 до 10 до запълване на масива.

Тестването ще става чрез сортирането на копие на масива в нарастващ ред с помощта на всеки от алгоритмите. Засича се времето за сортиране на целия масив. Тествайте с различно големи масиви. Коментирайте в клас получените резултати за различните методи.

Вход

- Входните данни трябва да се прочетат от конзолата.
- На първия и единствен ред ще ви бъде подадено число N, указващо броят на числата в масива, върху който ще тествате методите за сортиране.
- Входните данни винаги ще са валидни и в описания формат. Не е необходимо да бъдат изрично проверявани.

Изход

- Изходните данни трябва да бъдат отпечатани на конзолата.
- На четири реда трябва да бъдат изведени от 2 до 6 числа, разделени с интервал, съобразно броят на реализираните алгоритми за сортиране. Всяко число представлява времето на съответния алгоритъм сортиране на масива, използван в този кръг.
- Под тях изведете общото време за изпълнението на 4-те горни кръга за всеки от тестваните алгоритми.

Решение

```
public static class Help
{
    public static Random random = new Random();

    public static void Shuffle<T>(T[] elements)
    {
        for (int i = 0; i < elements.Length; i++)
        {
            int r = i + random.Next(0, elements.Length - i);
            Help.Swap(elements, i, r);
        }
    }

    public static void Swap<T>(T[] elements, int first, int second)
    {
        T temp = elements[first];
        elements[first] = elements[second];
        elements[second] = temp;
    }

    public static bool IsLess(IComparable first, IComparable second)
    {
        return first.CompareTo(second) < 0;
    }

    public static bool IsSorted<T>(T[] elements) where T : IComparable
    {
        for (int i = 1; i < elements.Length; i++)
        {
            if (elements[i - 1].CompareTo(elements[i]) > 0)
            {
                return false;
            }
        }
        return true;
    }
}

public static class Sort
{
    public static void Bubble<T>(T[] elements) where T : IComparable
    {
        for (int i = 0; i < elements.Length; i++)
```

```
        {
            for (int j = 0; j < elements.Length; j++)
            {
                if (Help.IsLess(elements[i], elements[j]))
                {
                    Help.Swap(elements, i, j);
                }
            }
        }
    }

    public static void Insertion<T>(T[] elements) where T : IComparable
    {
        for (int i = 0; i < elements.Length; i++)
        {
            int prev = i - 1;
            int curr = i;
            while (true)
            {
                if (prev < 0 || Help.IsLess(elements[prev], elements[curr]))
                {
                    break;
                }
                Help.Swap(elements, curr, prev);
                curr--;
                prev--;
            }
        }
    }

    // TODO: Add more sorting algorithms ....
}

public class Program
{
    public static Random random = new Random();

    public static void MesureTime(Action method)
    {
        Stopwatch timer = new Stopwatch();
        timer.Start();

        method();

        timer.Stop();
        Console.WriteLine("Time = {0} ms", timer.ElapsedMilliseconds);
    }

    public static void Main(string[] args)
    {
        const int N = 10000;
        int[] numbers = new int[N];
        for (int i = 0; i < N; i++) numbers[i] = random.Next(0, N);

        Console.Write("Bubble Sort ... ");
```

```
        MesureTime(() => Sort.Bubble(numbers));

        Console.Write("Shuffle ... ");
        MesureTime(() => Help.Shuffle(numbers));

        Console.Write("Insertion Sort ... ");
        MesureTime(() => Sort.Insertion(numbers));
    }
}
```

Задача 4.17. Топове плат

Студенти решили в свободното си време да работят като общи работници в голям склад за платове. Складът разполага с голямо количество платове от най-различни десени. Когато дойде някой клиент, той си поръчва желаното количество плат от някой десен, този плат се развива от топа, отрязва се и се предава на клиента, а останалото от топа се връща в склада. За да се оптимизира работата в склада, управителят на склада помолил студентите да сортират платовете по дължина - първо тези с останал 1 метър плат, после тези с 2 метра и т.н. Те решили да напишат програма, която да работи за всички подобни складове и да ги улесни работата на работниците в тях. Проблемът е, че складовете и работниците в тях може да са различни: в някои складове платовете са прекалено фини и нежни, а работниците груби и неуки и измерването и сравняването на дължините на платовете е най-тежката и бавна операция, а в други складове платовете са груби и на големи топове, а работниците слаби и хилави и пренасянето на топовете от едно място на друго е най-тежката и бавна операция. Направете програма, която да помага да се сортират платовете максимално бързо и с цената на минимум усилие. Сравнете в клас кой какъв метод е използвал и какъв резултат е получил за едни и същи входни данни. Уверете се, че има поне двама, които ползват един и същ метод за сравнение. Обсъдете получените резултати.

Вход

- Входните данни трябва да се прочетат от конзолата.
- На първия ред се подават неподредени цели числа, отделени едно от друго с интервал. Всяко число указва дължината на плата в някой от топовете в склада.
- На втория ред се подават две цели числа, указващи условното време за сравняване на два топа Tc и времето за размяна на два топа Ts.
- На третия ред се подават две цели числа, указващи условното усилие за сравняване на два топа Ec и усилието за размяна на два топа Es.
- Входните данни винаги ще са валидни и в описания формат. Не е необходимо да бъдат изрично проверявани.

Изход

- Изходните данни трябва да бъдат отпечатани на конзолата.

- На първия ред трябва да бъдат изведени две числа - броят сравнения CC и броят разместванията CS за този метод и този масив.
- На вторият ред трябва да бъдат изведени две числа:
  - общото време за сортиране на масива (равно на Tc * CC + Ts * CS)
  - общото усилие за сортиране на масива (равно на Ec * CC + Es * CS)

Подсказки

Модифицирайте методите Less( ) и Swap( ), така че да натрупвате броят на сравняванията и размените за всеки алгоритъм.

Решение

```
public class Program
{
    public static void Swap<T>(T[] elements, int first, int second)
    {
        T temp = elements[first];
        elements[first] = elements[second];
        elements[second] = temp;
        Cs++;
    }

    public static bool IsLess(IComparable first, IComparable second)
    {
        Cc++;
        return first.CompareTo(second) < 0;
    }

    public static void Selection<T>(T[] elements) where T : IComparable
    {
        for (int i = 0; i < elements.Length; i++)
        {
            int min = i;
            for (int j = i + 1; j < elements.Length; j++)
            {
                if (IsLess(elements[j], elements[min]))
                {
                    min = j;
                }
            }
            if (i != min)
                Swap(elements, min, i);
        }
    }

    public static int Cc = 0, Cs = 0;

    public static void Main(string[] args)
    {
        var numbers = Console.ReadLine().Split(' ').Select(int.Parse).ToArray();
        var time = Console.ReadLine().Split(' ').Select(int.Parse).ToArray();
        int Tc = time[0], Ts = time[1];
        var effort = Console.ReadLine().Split(' ').Select(int.Parse).ToArray();
        int Ec = effort[0], Es = effort[1];
```

```
        Selection(numbers);
        Console.WriteLine($"{Cc} {Cs}");
        Console.WriteLine($"{Tc * Cc + Ts * Cs} {Ec * Cc + Es * Cs}");
    }
}
```

Тема 5. Алгоритми за търсене

Задача 5.1. Линейно търсене

Реализирайте алгоритъм, който намира индекса на елемент в неподреден масив от цели числа с помощта на линейно търсене. Прочетете поредица от числа на първия ред и едно число на втория ред от конзолата. Намерете индекса на числото в дадения масив. Върнете -1, ако елементът не присъства в масива.

Примери

Вход	Изход
1 2 3 4 5 1	0
1 2 3 4 5 6	-1

Решение

```
public class Search
{
    public static int Linear<T>(T[] elements, T key) where T : IComparable
    {
        for (int index = 0; index < elements.Count(); index++)
        {
            if (elements[index].CompareTo(key) == 0)
            {
                return index;
            }
        }
        return -1;
    }
}

class Program
{
    static void Main(string[] args)
    {
        int[] array = { 1, 2, 3, 4, 5 };
        Console.WriteLine(Search.Linear(array, 3));
    }
}
```

Задача 5.2. Реализиране на двоично търсене

Реализирайте алгоритъм, който намира индекса на елемент в подреден масив от цели числа за логаритмично време

Примери

Вход	Изход	Коментари
1 2 3 4 5 1	0	Индексът на 1 е 0
-1 0 1 2 4 1	2	Индексът на 1 е 2

Подсказки

Първо, ако не сте запознати с принципа му на работа, прочетете за двоичното търсене в Wikipedia.

Тук може да намерите инструмент, който показва нагледно как се осъществява търсенето.

Накратко, ако имаме сортирана колекция от сравними елементи, вместо да правим последователно търсене (което отнема линейно нарастващо с броя на елементите време), можем да елиминираме половината от елементите на всяка стъпка и да свършим за логаритмично време.

Двоичното търсене е алгоритъм от типа разделяй-и-владей; започваме в средата на колекцията - ако не намерим елемента там, имаме три възможности:

- Елементът, който търсим е по-малък – тогава търсим наляво от текущия елемент, защото знаем, че всички надясно от него са по-големи;
- Елементът, който търсим е по-голям – тогава търсим надясно от текущия елемент;
- Елементът, който търсим не е наличен - в този случай, по традиция, връщаме -1.

Решение

```
static class Search
{
    public static int Binary<T>(T[] elements, T item) where T : IComparable
    {
        int low = 0;
        int high = elements.Length - 1;

        while (low <= high)
        {
            int mid = (low + high) / 2;

            if (IsLess(item, elements[mid]))
                high = mid - 1;
            else if (IsLess(elements[mid], item))
                low = mid + 1;
            else
                return mid;
        }
```

```
            return -1;
        }

        private static bool IsLess(IComparable first, IComparable second)
        {
            return first.CompareTo(second) < 0;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            int[] array = { 1, 2, 3, 4, 5 };
            Console.WriteLine(Search.Binary<int>(array, 3));
        }
    }
```

Задача 5.3. Фибоначи търсене

Реализирайте алгоритъм, който намира индекса на елемент в подреден масив от цели числа с помощта на Фибоначи търсене. Прочетете поредица от числа на първия ред и едно число на втория ред от конзолата. Намерете индекса на числото в дадения масив. Върнете -1, ако елементът не присъства в масива.

Примери

Вход	Изход
1 2 3 4 5 1	0
1 2 3 4 5 6	-1

Решение

```
public class Search
{
    public static int Fibonacci(int[] arr, int x, int n)
    {
        int fibMMm2 = 0;
        int fibMMm1 = 1;
        int fibM = fibMMm2 + fibMMm1;

        while (fibM < n)
        {
            fibMMm2 = fibMMm1;
            fibMMm1 = fibM;
            fibM = fibMMm2 + fibMMm1;
        }

        int offset = -1;
```

```
        while (fibM > 1)
        {
            int i = Math.Min(offset + fibMMm2, n - 1);

            if (arr[i] < x)
            {
                fibM = fibMMm1;
                fibMMm1 = fibMMm2;
                fibMMm2 = fibM - fibMMm1;
                offset = i;
            }

            else if (arr[i] > x)
            {
                fibM = fibMMm2;
                fibMMm1 = fibMMm1 - fibMMm2;
                fibMMm2 = fibM - fibMMm1;
            }

            else return i;
        }

        if (fibMMm1 != 0 && arr[offset + 1] == x) return offset + 1;

        return -1;
    }
}

class Program
{
    static void Main(string[] args)
    {
        int[] array = { 1, 2, 3, 4, 5 };
        Console.WriteLine(Search.Fibonacci(array, 3, array.Length));
    }
}
```

Задача 5.4. Състезание за търсене

Напишете програма, която сравнява бързодействието на различните алгоритми за търсене в три кръга и определя кой е победителят за всеки кръг, както и финалния победител (този с най-малко време общо за трите кръга). Използвайте два или повече от следните алгоритми за търсене:

- Линейно търсене
- Двоично търсене
- Фибоначи търсене
- Търсене чрез интерполация

Във всеки кръг, всеки от алгоритмите за търсене трябва да бъде тестван върху един от следните три масива:

1. Масив от поредни числа от 1 до N, подредени в нарастващ ред (т.е. 1, 2, 3, ... N)

2. Масив от поредни числа от N до 1, подредени в намаляващ ред (т.е. N, N-1, N-2, ... 3, 2, 1)
3. Масив от N случайни числа, всяко в диапазона от 1 до N включително.

Тестването ще става чрез търсенето на R на брой случайни числа (всяко в диапазона от 1 до N включително) в съответния масив с помощта на всеки от алгоритмите. Масивът, в който се търсят числата, също е един и същ за всеки от алгоритмите. Засича се общото време за откриването на всичките R на брой числа. Ако даден алгоритъм се нуждае от сортиран масив за намиране на числата, то това време се добавя веднъж към времето за търсене на всички числата за съответния алгоритми. Масивът се сортира само веднъж и се ползва за всички алгоритми. Изберете алгоритъм за сортиране, който смятате, че ще е най-удачен за съответния масив от числа.

Вход

- Входните данни трябва да се прочетат от конзолата.
- На първия и единствен ред ще ви бъдат подадени целите числа N и R, разделени с интервал.
  - N е броят на числата в масива, който ще тествате и максимално допустима стойност за всяко число. Минималната е 1.
  - R е броят на случайните числа, които ще търсите в масива. Всяко трябва да е в диапазона от 1 до N.
- Входните данни винаги ще са валидни и в описания формат. Не е необходимо да бъдат изрично проверявани.

Изход

- Изходните данни трябва да бъдат отпечатани на конзолата.
- На четири реда трябва да бъдат изведени от 2 до 4 числа, разделени с интервал, съобразно броят на реализираните алгоритми. Всяко число представлява времето на съответния алгоритъм за намирането на R случайни числа в:
  - масива от поредни числа от 1 до N
  - масива от поредни числа от N до 1
  - масива от N случайни числа
  - общото време за изпълнението на 3-те горни кръга за съответния алгоритъм.

Подсказки

- Тествайте задачата с различно големи масиви, различни алгоритми за сортиране и различен брой търсени числа.
- Коментирайте в клас получените резултати.

Решение

```
using System.Diagnostics;

public class Search
```

```
{
    public static int Linear<T>(T[] elements, T key) where T : IComparable
    {
        for (int index = 0; index < elements.Count(); index++)
        {
            if (elements[index].CompareTo(key) == 0)
            {
                return index;
            }
        }
        return -1;
    }

    public static int Binary<T>(T[] elements, T key) where T : IComparable
    {
        int start = 0, end = elements.Count() - 1;
        while (end >= start)
        {
            int mid = (start + end) / 2;

            if (elements[mid].CompareTo(key) > 0)
            {
                end = mid - 1;
            }
            else if (elements[mid].CompareTo(key) < 0)
            {
                start = mid + 1;
            }
            else
            {
                return mid;
            }
        }
        return -1;
    }

    public static int Interpolational(int[] sortedArray, int key)
    {
        int low = 0;
        int high = sortedArray.Length - 1;
        while (sortedArray[low] <= key && sortedArray[high] >= key)
        {
            int mid = (int)(low + ((double)(key - sortedArray[low]) * (high - low))
/ (sortedArray[high] - sortedArray[low]));
            if (sortedArray[mid] < key) low = mid + 1;
            else if (sortedArray[mid] > key) high = mid - 1;
            else return mid;
        }
        if (sortedArray[low] == key) return low;
        else return -1;
    }

    public static long Fibonacci(int[] arr, int x, int n)
    {
        long fibMMm2 = 0;
```

```
        long fibMMm1 = 1;
        long fibM = fibMMm2 + fibMMm1;
        while (fibM < n)
        {
            fibMMm2 = fibMMm1;
            fibMMm1 = fibM;
            fibM = fibMMm2 + fibMMm1;
        }
        long offset = -1;
        while (fibM > 1)
        {
            long i = offset + fibMMm2 > n - 1 ? n - 1 : offset + fibMMm2;
            if (arr[i] < x)
            {
                fibM = fibMMm1;
                fibMMm1 = fibMMm2;
                fibMMm2 = fibM - fibMMm1;
                offset = i;
            }
            else if (arr[i] > x)
            {
                fibM = fibMMm2;
                fibMMm1 = fibMMm1 - fibMMm2;
                fibMMm2 = fibM - fibMMm1;
            }
            else return i;
        }
        if (fibMMm1 != 0 && arr[offset + 1] == x) return offset + 1;
        return -1;
    }
}

class Program
{
    private static long MesureTime(Action metod)
    {
        Stopwatch timer = new Stopwatch();
        timer.Start();

        metod();

        timer.Stop();
        Console.WriteLine("Time = {0} ticks", timer.ElapsedTicks);
        return timer.ElapsedTicks;
    }

    static void Main(string[] args)
    {
        Console.Write("n=");
        int n = int.Parse(Console.ReadLine());
        Console.Write("r=");
        int r = int.Parse(Console.ReadLine());

        Random rnd = new Random();
        int[] test1 = new int[n];
```

```
        int[] test2 = new int[n];
        int[] test3 = new int[n];
        for (int i = 0; i < n; i++)
        {
            test1[i] = i + 1;
            test2[i] = n - i - 1;
            test3[i] = rnd.Next(1, n);
        }

        Stopwatch timer = new Stopwatch();
        timer.Start();
        int[] sortedTest2 = test2.OrderBy(x => x).ToArray();
        timer.Stop();
        long sortingTimeTest2 = timer.ElapsedMilliseconds;
        timer.Restart();
        int[] sortedTest3 = test3.OrderBy(x => x).ToArray();
        timer.Stop();
        long sortingTimeTest3 = timer.ElapsedMilliseconds;

        long totalLinearTime = 0, totalBinaryTime = 0, totalFibonacciTime = 0,
totalInterpolationTime = 0;


        long linearTime = 0, binaryTime = 0, fibonacciTime = 0, interpolationTime =
0;
        Console.WriteLine("\n\nSearch on Sorted Array:\n");
        for (int i = 0; i < r; i++)
        {
            int tester = rnd.Next(1, n);
            Console.Write($"{i + 1}. Linear Search ");
            linearTime += MesureTime(() => Search.Linear(test1, tester));
            Console.Write($"{i + 1}. Binary Search ");
            binaryTime += MesureTime(() => Search.Binary(test1, tester));
            Console.Write($"{i + 1}. Fibоncci Search ");
            fibonacciTime += MesureTime(() => Search.Fibonacci(test1, tester,
test1.Length));
            Console.Write($"{i + 1}. Interpolation Search ");
            interpolationTime += MesureTime(() => Search.Interpolational(test1,
tester));
        }
        totalBinaryTime += binaryTime; totalFibonacciTime = fibonacciTime;
totalInterpolationTime = interpolationTime; totalLinearTime = linearTime;
        Console.WriteLine("\nTotal Time: (Linear,Binary,Fibonacci,Interpolation)");
        Console.WriteLine($"{linearTime} {binaryTime} {fibonacciTime}
{interpolationTime}\n\n");

        linearTime = 0; binaryTime = fibonacciTime = interpolationTime =
sortingTimeTest2;
        Console.WriteLine("Search on Reversed Sorted Array:\n");
        for (int i = 0; i < r; i++)
        {
            int tester = rnd.Next(1, n);
            Console.Write($"{i + 1}. Linear Search ");
            linearTime += MesureTime(() => Search.Linear(test2, tester));
            Console.Write($"{i + 1}. Binary Search ");
```

```
            binaryTime += MesureTime(() => Search.Binary(sortedTest2, tester));
            Console.Write($"{i + 1}. Fibоncci Search ");
            fibonacciTime += MesureTime(() => Search.Fibonacci(sortedTest2, tester,
sortedTest2.Length));
            Console.Write($"{i + 1}. Interpolation Search ");
            interpolationTime += MesureTime(() =>
Search.Interpolational(sortedTest2, tester));
        }
        totalBinaryTime += binaryTime; totalFibonacciTime += fibonacciTime;
totalInterpolationTime += interpolationTime; totalLinearTime += linearTime;
        Console.WriteLine("\nTotal Time: (Linear,Binary,Fibonacci,Interpolation)");
        Console.WriteLine($"{linearTime} {binaryTime} {fibonacciTime}
{interpolationTime}\n\n");

        linearTime = 0; binaryTime = fibonacciTime = interpolationTime =
sortingTimeTest3;
        Console.WriteLine("Search on Random Array:\n");
        for (int i = 0; i < r; i++)
        {
            int tester = rnd.Next(1, n);
            Console.Write($"{i + 1}. Linear Search ");
            linearTime += MesureTime(() => Search.Linear(test3, tester));
            Console.Write($"{i + 1}. Binary Search ");
            binaryTime += MesureTime(() => Search.Binary(sortedTest3, tester));
            Console.Write($"{i + 1}. Fibоncci Search ");
            fibonacciTime += MesureTime(() => Search.Fibonacci(sortedTest3, tester,
sortedTest3.Length));
            Console.Write($"{i + 1}. Interpolation Search ");
            interpolationTime += MesureTime(() =>
Search.Interpolational(sortedTest3, tester));
        }
        totalBinaryTime += binaryTime; totalFibonacciTime += fibonacciTime;
totalInterpolationTime += interpolationTime; totalLinearTime += linearTime;
        Console.WriteLine("\nTotal Time: (Linear,Binary,Fibonacci,Interpolation)");
        Console.WriteLine($"{linearTime} {binaryTime} {fibonacciTime}
{interpolationTime}");

        Console.WriteLine("\n\n\nTotal results:
(Linear,Binary,Fibonacci,Interpolation)");
        Console.WriteLine($"{totalLinearTime} {totalBinaryTime} {totalFibonacciTime}
{totalInterpolationTime}\n");
    }
}
```

Задача 5.5. Игли

В тази задача трябва да намерите вярното място на числата в дадена поредица. От конзолата ще прочетете поредица от ненамаляващи цели числа, между които на случаен принцип има „дупки" (представени чрез нули).

Ще ви бъдат дадени иглите – числа, които трябва да бъдат вмъкнати в поредицата, така че тя да остане ненамаляваща (без да се вземат под внимание „дупките"). За всяка игла намерете най-левия възможен индекс, където може да бъде вмъкната.

Вход

- Входните данни трябва да се прочетат от конзолата.
- На първия ред ще ви бъдат дадени числата C и N, разделени с интервал.
- На втория ред ще ви бъдат дадени C неотрицателни цели числа, формиращи ненамаляваща редица (като не вземаме в предвид нулите).
- На третия ред ще ви бъдат дадени N положителни цели числа, иглите.
- Въведените данни винаги ще са валидни и в описания формат. Не е необходимо да бъдат изрично проверявани.

Изход

- Отговорът трябва да се отпечата на конзолата. Трябва да бъде на един ред.
- На единствения изведен ред отпечатайте N числа, разделени с интервал. Всяко число представлява най-левият възможен индекс, където може да бъде вмъкната съответната игла.

Ограничения

- Всички входни числа ще бъдат 32-битови цели числа със знак.
- N ще бъдат в обхвата [1 ... 1000].
- C ще бъдат в обхвата [1 ... 50000].
- Допустимо време за работа на вашата програма: 0.1 секунди. Допустима памет: 16 MB.

Примери

Вход
23 9 3 5 11 0 0 0 12 12 0 0 0 12 12 70 71 0 90 123 140 150 166 190 0 5 13 90 1 70 75 7 188 12
Изход
1 13 15 0 13 15 2 21 3
Коментари
5 отива при индекс 1 – между 3 и 5 13 отива при индекс 13 – 12 и 70 90 отива при индекс 15 – между 71 и 0 1 отива при индекс 0 – преди 3 Etc.
Вход
11 4 2 0 0 0 0 0 0 0 0 0 3 4 3 2 1
Изход
11 1 0 0

Решение

```
class Program
{
    public static int RealNumber(int startIndex, int[] arr)
    {
        for (int i = startIndex; i < arr.Length; i++)
        {
```

```
            if (arr[i] != 0) return arr[i];
        }
        return -1;
    }

    static void Main(string[] args)
    {
        int[] array = Console.ReadLine().Split(' ').Select(int.Parse).ToArray();
        int N = array[0], C = array[1];
        int[] numbers = new int[N];
        numbers = Console.ReadLine().Split(' ').Select(int.Parse).ToArray();
        int[] needles = new int[C];
        needles = Console.ReadLine().Split(' ').Select(int.Parse).ToArray();
        int[] newArray = new int[C + N];
        for (int i = 0; i < C; i++)
        {
            for (int y = 0; y < N - 1; y++)
            {
                if (needles[i] <= numbers[0] && numbers[0] != 0)
                {
                    Console.Write(0 + " "); break;
                }
                if (needles[i] > numbers[numbers.Length - 1] &&
numbers[numbers.Length - 1] != 0)
                {
                    Console.Write(numbers.Length + " "); break;
                }
                if (numbers[y] != 0 && needles[i] >= numbers[y] && needles[i] <=
RealNumber(y + 1, numbers))
                {
                    Console.Write(y + 1 + " "); break;
                }
            }
        }
    }
}
```

Тема 6. Допълнителни задачи

Задача 6.1. Обръщане последователността на масив

Напишете програма, която обръща и отпечатва масив. Използвайте рекурсия.

Примери

Вход	Изход
1 2 3 4 5 6	6 **5 4 3 2 1**

Решение

```
public class Program
{
    private static void Reverse(int[] numbers, int pos)
    {
        if (pos >= numbers.Count()) return;
        Reverse(numbers.Skip(1).ToArray(), pos++);
```

```
        Console.Write("{0} ", numbers[pos - 1]);
    }

    public static void Main(string[] args)
    {
        int[] numbers = Console.ReadLine().Split().Select(int.Parse).ToArray();
        Reverse(numbers, 0);
    }
}
```

Задача 6.2. Вложени цикли и рекурсия

Напишете програма, която симулира изпълнението на n вложени цикъла от 1 до n, която отпечатва стойностите на всичките си итерационни променливи по всяко време на един ред. Използвайте рекурсия.

Примери

Вход	Изход	Решение с вложени цикли (приемаме, че n е положително)

2	1 1
	1 2
	2 1
	2 2
		`int n = 2;`
		
		`for (int i1 = 1; i1 <= n; i1++)`
		`{`
		`  for (int i2 = 1; i2 <= n; i2++)`
		`  {`
		`  Console.WriteLine($"{i1} {i2}");`
		`  }`
		`}`

3	1 1 1
	1 1 2
	1 1 3
	1 2 1
	1 2 2
	…
	3 2 3
	3 3 1
	3 3 2
	3 3 3
		`int n = 3;`
		
		`for (int i1 = 1; i1 <= n; i1++)`
		`{`
		`  for (int i2 = 1; i2 <= n; i2++)`
		`  {`
		`  for (int i3 = 1; i3 <= n; i3++)`
		`  {`
		`  Console.WriteLine($"{i1} {i2} {i3}");`
		`  }`
		`  }`
		`}`

Решение

```
public class Program
{
```

```
    private static int n = int.Parse(Console.ReadLine());

    private static int[] loops = new int[n];

    private static void Permutation(int index)
    {
        if (index == loops.Length)
        {
            foreach (var item in loops)
            {
                Console.Write($"{item} ");
            }
            Console.WriteLine();
            return;
        }
        else
        {
            for (int i = 1; i <= loops.Length; i++)
            {
                loops[index] = i;
                Permutation(index + 1);
            }
        }
    }

    public static void Main(string[] args)
    {
        Permutation(0);
    }
}
```

Задача 6.3. Комбинации с повторения

Напишете рекурсивна програма за генериране и отпечатване на всички комбинации с повторения на k елемента от набор от n елементи (k < = n). В комбинациите, редът на елементите няма значение, следователно (1 2) и (2 1) са една и съща комбинация, което означава, че след като получите (1 2), (2 1) вече не е валидно.

Примери

Вход	Иход	Коментари	Решение с вложени цикли

3
2
	1 1
	1 2
	1 3
	2 2
	2 3
	3 3
		• **n=3 => имаме множество от 3 елемента {1, 2, 3}**
		• **k=2 => избираме два от три елемента всеки път**
		• **Повтренията са позволени, което значи, че (1 1) е валидна комбинаация**
			`int n = 3;`
			`int k = 2;`
			
			`// k == 2 => 2 nested for-loops`
			`for (int i1 = 1; i1 <= n; i1++)`
			`{`
			`  for (int i2 = i1; i2 <= n; i2++)`
			`  {`
			`  Console.WriteLine($"({i1} {i2})");`
			`  }`
			`}`

5
3
	1 1 1
	1 1 2
	1 1 3
	1 1 4
	1 1 5
	1 2 2
	…
	3 5 5
	4 4 4
	4 4 5
	4 5 5
	5 5 5
		Избираме 3 елемента от общо 5 – {1, 2, 3, 4, 5}, общо 35 комбинации
		(1 2 1) не е валидна комбинация, тъй като е същата като (1 1 2)
			`int n = 5;`
			`int k = 3;`
			
			`// k == 3 => 3 вложени цикъла`
			`for (int i1 = 1; i1 <= n; i1++)`
			`{`
			`  for (int i2 = i1; i2 <= n; i2++)`
			`  {`
			`  for (int i3 = i2; i3 <= n; i3++)`
			`  {`
			`  Console.WriteLine($"({i1} {i2} {i3})");`
			`  }`
			`  }`
			`}`

Решение

```
public class Program
{
    private static void Combinations(int[] arr, int index, int start, int end)
    {
        if (index >= arr.Length)
        {
            for (int i = 0; i < arr.Length; i++)
            {
                Console.Write("{0} ", arr[i]);
            }
            Console.WriteLine();
            return; // Exit
        }
        else
        {
            for (int i = start; i <= end; i++)
            {
                arr[index] = i;
                Combinations(arr, index + 1, i, end);
            }
        }
    }

    public static void Main(string[] args)
    {
        int n = int.Parse(Console.ReadLine());
        int k = int.Parse(Console.ReadLine());

        int[] arr = new int[k];
        Combinations(arr, 0, 1, n);
    }
}
```

Задача 6.4. Ханойски кули

Вашата цел е да преместите всички дискове от пръчката източника на пръчката местоназначение. Има няколко правила:

1. само един диск може да бъде преместен в даден момент
2. само най-горния диск може да бъде преместен
3. един диск може да бъдат поставен само върху по-голям диск, или върху празна пръчка

Примери

Вход	Изход
3	Source: 3, 2, 1 Destination: Spare: Step #1: Moved disk 1 Source: 3, 2 Destination: 1 Spare: Step #2: Moved disk 2 Source: 3 Destination: 1 Spare: 2 Step #3: Moved disk 1 Source: 3 Destination: Spare: 2, 1 Step #4: Moved disk 3 Source: Destination: 3 Spare: 2, 1 Step #5: Moved disk 1 Source: 1 Destination: 3 Spare: 2 Step #6: Moved disk 2 Source: 1 Destination: 3, 2 Spare: Step #7: Moved disk 1 Source: Destination: 3, 2, 1 Spare:

Решение

```
public class Program
{
    static void Main(string[] args)
    {
        int numberOfDisks = int.Parse(Console.ReadLine());

        Stack<int> source = new Stack<int>(Enumerable.Range(1,
numberOfDisks).Reverse());
        Stack<int> destination = new Stack<int>();
        Stack<int> spare = new Stack<int>();

        PrintRods(source, destination, spare);
        MoveDisks(numberOfDisks, source, destination, spare);
        PrintRods(source, destination, spare);
    }

    private static void PrintRods(Stack<int> source, Stack<int> destination,
Stack<int> spare)
    {
        Console.WriteLine("Source: {0}", string.Join(", ", source.Reverse()));
        Console.WriteLine("Spare: {0}", string.Join(", ", spare.Reverse()));
        Console.WriteLine("Destination: {0}", string.Join(", ",
destination.Reverse()));
        Console.WriteLine();
    }

    private static void MoveDisks(int bottomDisk, Stack<int> source, Stack<int>
destination, Stack<int> spare)
    {
        if (bottomDisk == 1)
        {
            destination.Push(source.Pop());
            PrintRods(source, destination, spare);
            return;
        }

        MoveDisks(bottomDisk - 1, source, spare, destination);
        PrintRods(source, destination, spare);

        destination.Push(source.Pop());
        PrintRods(source, destination, spare);

        MoveDisks(bottomDisk - 1, spare, destination, source);
        PrintRods(source, destination, spare);
    }
}
```

Задача 6.5. Комбинации без повторения

Промяна на предишната програма да пропуснете копия, например (1 1) не е валиден.

Примери

Вход	Изход	Коментари	Решение с вложени цикли

3
2
	1 2
	1 3
	2 3
		• **n=3 => имаме множество от 3 елемента{1, 2, 3}**
		• **k=2 => избираме два от три елемента**
		• **Повторенията не са позвлени, сиреч (1 1) не е валидна комбинация.**
			`int n = 3;`
			`int k = 2;`
			
			`// k == 2 => 2 вложени цикли`
			`for (int i1 = 1; i1 <= n; i1++)`
			`{`
			`  for (int i2 = i1 + 1; i2 <= n; i2++)`
			`  {`
			`  Console.WriteLine($"({i1} {i2})");`
			`  }`
			`}`

5
3
	1 2 3
	1 2 4
	1 2 5
	1 3 4
	1 3 5
	1 4 5
	2 3 4
	2 3 5
	2 4 5
	3 4 5
		Избираме три от пет – {1, 2, 3, 4, 5}, всико 10 комбинции	`int n = 5;`
			`int k = 3;`
			
			`// k == 3 => 3 вложени цикъла`
			`for (int i1 = 1; i1 <= n; i1++)`
			`{`
			`  for (int i2 = i1 + 1; i2 <= n; i2++)`
			`  {`
			`  for (int i3 = i2 + 1; i3 <= n; i3++)`
			`  {`
			
			`Console.WriteLine($"({i1}  {i2} {i3})");`
			`  }`
			`  }`
			`}`

Решение

```
public class Program
{
    private static int n = int.Parse(Console.ReadLine());
    private static int k = int.Parse(Console.ReadLine());
    private static int[] arr = new int[k];

    private static void Combinations(int index, int border = 1)
    {
        if (index == arr.Length)
        {
            Console.WriteLine(String.Join(" ", arr));
            return;
        }
```

```
        else
        {
            for (int i = border; i <= n; i++)
            {
                arr[index] = i;
                Combinations(index + 1, i + 1);
            }
        }
    }

    public static void Main(string[] args)
    {
        Combinations(0);
    }
}
```

Задача 6.6. Свързани масиви в матрица

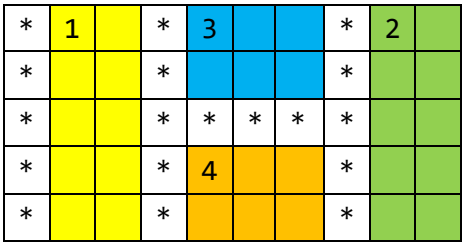
Нека да определим свързани масиви в една матрица като област от клетки, в които има път между всеки две клетки.

Напишете програма, която намира всички свързани области в една матрица. Изведете на екрана общия брой намерени площи и на отделен ред информация за всяка от областите – позиция (горния ляв ъгъл) и размер. Подредете областите по размер (в низходящ ред), така че най-голямата площ се отпечатва първо. Ако няколко области имат същия размер, подредете ги по своята позиция, първа е тази, чийто горен ляв ъгъл е по-горе и/или по-вдясно на реда. Така че ако има две свързани области със същия размер, която е над и/или вляво от другата ще бъде отпечатана първо.

- На първия ред вие ще получите броя на редовете
- На втори ред вие ще получите броя на колоните
- Останалата част от вход ще бъде действителната матрица.

Примери

Примерно разположение	Изход
4 9 1 - - * 2 - - * 3 - - - * - - - * - - - - * - - - * - - - - - * - * - -	Обща площ: 3 Площ #1 at (0, 0), размер: 13 Площ #2 at (0, 4), размер: 10 Площ #3 at (0, 8), размер: 5

5 10 	Общо намерени области : 4 Площ #1 at (0, 1), размер: 10 Площ #2 at (0, 8), размер: 10 Площ #3 at (0, 4), размер: 6 Площ #4 at (3, 4), размер: 6

Подсказки

- Създаване на метод за намиране на първата „проходима“ клетка, която не е била посетена. Тя ще бъде в горния ляв ъгъл на свързаната област. Ако няма такива клетки, това означава, че всички области са били открити.
- Можете да създадете клас, който да държи информация за свързаната област (своята позиция и размер). Освен това можете да реализирате IComparable и да съхранявате всички намерени области в подредено множество.

Решение

```
class Region
{
    public int X { get; set; }
    public int Y { get; set; }
    public int Area { get; set; }
    public Region(int x, int y)
    {
        X = x;
        Y = y;
        Area = 1;
    }
}

class Program
{
    private static int row = int.Parse(Console.ReadLine());
    private static int col = int.Parse(Console.ReadLine());
    private static string[] table = new string[row];
    private static List<Region> regions = new List<Region>();
    private static int Counter = 0;

    private static void ReadTable()
    {
        for (int i = 0; i < row; i++)
        {
            table[i] = Console.ReadLine();
        }
    }

    private static void FindStartingPoint()
    {
```

```
        for (int x = 0; x < row; x++)
        {
            for (int y = 0; y < col; y++)
            {
                if (table[x][y] == '-')
                {
                    if (x == 0 && y == 0) regions.Add(new Region(y, x));
                    else if (y == 0) { if (table[x - 1][y] == '*') regions.Add(new
Region(y, x)); }
                    else if (x == 0) { if (table[x][y - 1] == '*') regions.Add(new
Region(y, x)); }
                    else if (table[x - 1][y] == '*' && table[x][y - 1] == '*')
regions.Add(new Region(y, x));
                }
            }
        }
    }

    private static void PrintRegions()
    {
        foreach (var region in regions)
        {
            Console.WriteLine($"{region.X} {region.Y} {region.Area}"); // ?
        }
    }

    private static void RegionCheck(int x, int y)
    {
        if (IsCellInReagionBounds(x, y) && table[y][x] != '*')
        {
            MarkCellInRegion(x, y);
            RegionCheck(x, y + 1);
            RegionCheck(x + 1, y);
            RegionCheck(x, y - 1);
            RegionCheck(x - 1, y);
        }
    }

    private static bool IsCellInReagionBounds(int x, int y)
    {
        if (x < 0 || y >= row || y < 0 || x >= col) return false;
        else return true;
    }

    private static void MarkCellInRegion(int x, int y)
    {
        var temp = table[y].ToCharArray();
        temp[x] = '*';
        table[y] = string.Join("", temp);
        Counter++;
    }

    public static void Main(string[] args)
    {
        ReadTable();
```

```
        FindStartingPoint();
        for (int i = 0; i < regions.Count; i++)
        {
            int x = regions[i].X;
            int y = regions[i].Y;
            Counter = 0;
            RegionCheck(x, y);
            regions[i].Area = Counter;
        }
        regions = regions.Where(x => x.Area != 0).OrderByDescending(y =>
y.Area).ToList();
        PrintRegions();
    }
}
```

Задача 6.7. Рекурсивна сума на масив

Напишете програма, която намира сумата на всички елементи на масив от цели числа. Използвайте рекурсия. Забележка: На практика рекурсия трябва да не се използва тук (вместо това използвайте итеративно решение), това е просто упражнение.

Примери

Вход	Изход
1 2 3 4	10
-1 0 1	0

Подсказки

Напишете рекурсивна метод. Той ще вземе като аргументи входния масив и текущия индекс.

- **Методът трябва да върнете текущия елемент + сумата от всички следващите елементи (получени чрез рекурсивно, наричайки я).**
- **Рекурсията трябва да спре, когато няма повече елементи в масив.**

```
private static int Sum(int[] array, int index)
{
    // TODO: Set bottom of recursion
    // TODO: Return the sum of current element + sum of elements to the right
}
```

Решение

```
public class Program
{
    private static int Sum(int[] array, int index)
    {
        int sum = 0;
        if (index <= 0) return 0;
        return Sum(array, index - 1) + array[index - 1];
    }
```

```
    public static void Main(string[] args)
    {
        int[] array = Console.ReadLine().Split().Select(int.Parse).ToArray();
        Console.WriteLine(Sum(array, array.Length));
    }
}
```

Задача 6.8. Рекурсивен факториел

Напишете програма, която намира факториела на дадено число. Използвайте рекурсия.

Забележка: На практика рекурсия не трябва да се използва тук (вместо това използвайте итерация), това е просто упражнение.

Примери

Вход	Изход
5	120
10	3628800

Подсказки

Напишете рекурсивна метод.Той взима като аргумент цяло число.

- Методът трябва да връща текущия елемент * резултатът от изчисляване факториела на текущия елемент - 1 (получени чрез рекурсивно извикване).
- Рекурсията трябва да спре, когато няма повече елементи в масива.

```
static long Factorial(int n)
{
    // TODO: Set bottom of recursion
    // TODO: Return the multiple of current n and factorial of n - 1
}
```

Решение

```
public class Program
{
    private static long Fak(int n)
    {
        if (n <= 0) return 1;
        else return n * Fak(n - 1);
    }

    public static void Main(string[] args)
    {
        int n = int.Parse(Console.ReadLine());
        Console.WriteLine(Fak(n));
    }
}
```

Задача 6.9. Рекурсивно чертаене

Напишете програма, която прави фигурата по-долу в зависимост от n. Ползвайте рекурсия.

Примери

Вход	Изход
2	** * # ##
5	***** **** *** ** * # ## ### #### #####

Решение

```
public class Program
{
    private static void Draw(int n)
    {
        int c = n;
        if (n <= 1) Console.WriteLine(new string('*', n));
        else
        {
            Console.WriteLine(new string('*', n));
            Draw(n - 1);
        }
        if (c <= n) Console.WriteLine(new string('#', n));
    }

    public static void Main(string[] args)
    {
        int n = int.Parse(Console.ReadLine());
        Draw(n);
    }
}
```

Задача 6.10. Генериране на 0/1 вектори

Генерирайте всички n битови вектори от нули и единици в азбучен ред.

Примери

Вход	Изход

3	000 001 010 011 100 101 110 111
5	00000 00001 00010 … 11110 11111

Решение

```
public class Program
{
    private static void Print(int[] a, int n)
    {
        for (int i = 0; i < n; i++)
        {
            Console.Write(a[i]);
        }
        Console.WriteLine();
    }

    private static void Gen01(int n, int[] a, int i)
    {
        if (i == n)
        {
            Print(a, n);
            return;
        }
        a[i] = 0;
        Gen01(n, a, i + 1);
        a[i] = 1;
        Gen01(n, a, i + 1);
    }

    public static void Main(string[] args)
    {
        int n = int.Parse(Console.ReadLine());
        int[] a = new int[n];
        Gen01(n, a, 0);
    }
}
```

Задача 6.11. Генериране на комбинации

Генерира всички комбинации от n числа в групи по k. Прочетете множеството от n елементи, а след това броят k на елементите за избор.

Примери

Вход	Изход
1 2 3 4 2	1 2 1 3 1 4 2 3 2 4 3 4
1 2 3 4 5 3	1 2 3 1 2 4 1 2 5 … 3 4 5

Решение

```
public class Program
{
    private static int[] array =
Console.ReadLine().Split().Select(int.Parse).ToArray();
    private static int n = int.Parse(Console.ReadLine());
    private static int[] loops = new int[n];

    private static void Combinations(int index, int[] array, int border = 0)
    {
        if (index == loops.Length)
        {
            Console.WriteLine(String.Join(" ", loops));
            return; // Exit
        }
        for (int i = border; i < array.Length; i++)
        {
            loops[index] = array[i];
            Combinations(index + 1, array, i + 1);
        }
    }

    public static void Main(string[] args)
    {
        Combinations(0, array);
    }
}
```

Задача 6.12. Задачата за 8 Царици

Ще реализираме рекурсивен алгоритъм за решаване на задачата за "8 царици". Нашата цел е да се напише програма за намиране на всички

възможни разположения на шахматната дъска на 8 царици, така че да няма две царици, които да могат да се нападат взаимно (по ред, колона или диагонал).

Примери

Вход	Изход
(no input)	* - - - - - - - - - - - * - - - - - - - - - - * - - - - - * - - - - * - - - - - - - - - - - * - - * - - - - - - - - - * - - - - * - - - - - - - - - - - - * - - - - - - - - - * - - * - - - - - - - - - - - * - - - - * - - - - - * - - - - - - - - - - * - - - … (над 90 решения)

Научете за задачата

Научете повече за "8 царици" пъзел, например от Уикипедия: http://en.wikipedia.org/wiki/Eight_queens_puzzle.

Определете структурата от данни, която ще държи шахматната дъска

Първо нека да се определи структурата на данните, която ще да държи шахматната дъска. Тя трябва да се състои от 8 x 8 клетки, всяка е или заета от царица или празна. Нека също така да се определи размера на шахматната дъска като константа:

```
class EightQueens
{
    const int Size = 8;
    static bool[,] chessboard = new bool[Size, Size];
}
```

Определяне на структурата от данни, която държи атакуваните позиции

Ние трябва да държим атакуваните позиции в някаква структура от данни. Във всеки момент от изпълнението на програмата ние трябва да знаем дали

определена позиция {ред, колона} е под атака от една царица, или не. Има много начини за съхраняване на атакуваните позиции:

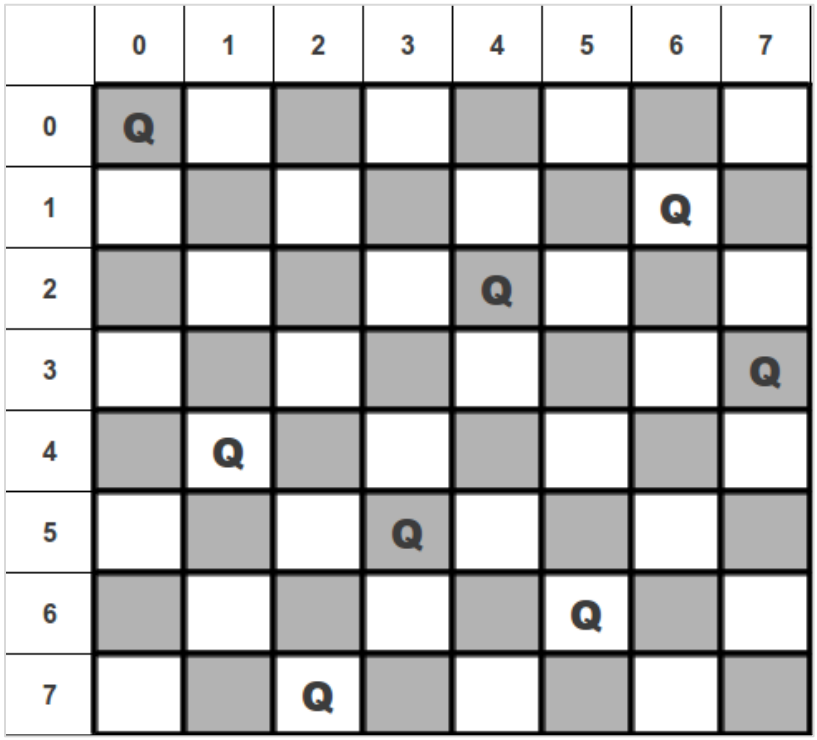
- Като пазим всички в момента поставени царици и проверяваме дали новата позиция е в конфликт с някои от тях.
- Чрез запазване в int [,] матрица от всички атакувани позиции и проверка на новата позиция директно в него. Това ще бъде сложно да се поддържа, защото в матрицата трябва да се променят много позиции след всяка поставяне/премахване на царица.
- Като пазим множества от всички атакувани редове, колони и диагонали. Нека опитаме тази идея:

```
static HashSet<int> attackedRows = new HashSet<int>();
static HashSet<int> attackedColumns = new HashSet<int>();
static HashSet<int> attackedLeftDiagonals = new HashSet<int>();
static HashSet<int> attackedRightDiagonals = new HashSet<int>();
```

Горните определения имат следните предположения:

- редове са 8, номерирани от 0 до 7.
- Колоните са 8, номерирани от 0 до 7.
- левите диагонали са 15, номерирани от -7 до 7. Можем да използваме следната формула за изчисляване на номера на левия диагонал по ред и колона: leftDiag = col - row.
- десните диагонали са 15, номерирани от 0 до 14 по формулата: rightDiag = col + ред.

Да вземем като пример следната шахматната дъска с 8 царици поставени върху ѝ в следните позиции: {0, 0}; {1, 6}; {2, 4}; {3, 7}; {4, 1}; {5, 3}; {6, 5}; {7, 2}



Следвайки определенията по-горе за нашия пример царица {4, 1} заема ред 4, колона 1, левия диагонал -3 и десния диагонал 5.

Напишете Backtracking алгоритъм

Сега е време да се напише на рекурсивен backtraking алгоритъм за разполагането на 8 царици. Алгоритъм започва от ред 0 и се опитва да постави кралица в някоя колона в ред 0. При успех той се опитва да постави следващата царица в ред 1, после следващата царица на ред 2 и т.н. до последния ред. Кодът за поставянето на следващата царица в определен ред може да изглежда така:

```
static void PutQueens(int row)
{
    if (row == Size)
    {
        PrintSolution();
    }
    else
    {
        for (int col = 0; col < Size; col++)
        {
            if (CanPlaceQueen(row, col))
            {
                MarkAllAttackedPositions(row, col);
                PutQueens(row + 1);
                UnmarkAllAttackedPositions(row, col);
            }
        }
    }
}
```

Първоначално ние извикваме този метод от ред 0:

```
static void Main()
{
    PutQueens(0);
}
```

Проверка, ако позицията е свободна

Сега нека да напишем код, за да проверим дали е свободна определена позиция. Позицията е свободна, когато не е под атака от всички други царици. Това означава, че ако някои редове, колони и диагонали вече са заети от други царици, то позицията е заета. В противен случай тя е свободна. Примерно, кода може да изглежда така:

```
static bool CanPlaceQueen(int row, int col)
{
    var positionOccupied =
         attackedRows.Contains(row) ||
         attackedColumns.Contains(col) ||
         attackedLeftDiagonals.Contains(col - row) ||
         attackedRightDiagonals.Contains(row + col);
    return !positionOccupied;
}
```

Спомнете си, че col-row е номера на левия диагонал и че row+col е номера на десния диагонал.

Маркиране / Демаркиране на полетата, които царицата атакува

След като царицата е поставена, ние трябва да маркирате като заети всички редове, колони и диагонали, които тя може да атакува

```
static void MarkAllAttackedPositions(int row, int col)
{
    attackedRows.Add(row);
    attackedColumns.Add(col);
    attackedLeftDiagonals.Add(col - row);
    attackedRightDiagonals.Add(row + col);
    chessboard[row, col] = true;
}
```

При преместване на царица ще ни трябва метод, който да маркира като свободни всички редове, колони и диагонали, които са били нападнати от него. Напишете сами:

```
static void UnmarkAllAttackedPositions(int row, int col)
{
    // TODO
}
```

Отпечатайте решението

Когато е намерено решение, то трябва да бъде отпечатано в конзолата. Първо въведе брояч на решения да се опрости проверка дали намерените решения са правилни:

```
class EightQueens
{
    static int solutionsFound = 0;
```

След това преминете през всички редове и всички колони на шахматната дъска и ги отпечатайте:

```
static void PrintSolution()
{
    for (int row = 0; row < Size; row++)
    {
        for (int col = 0; col < Size; col++)
        {
            // TODO: print '*' or '-' depending on scoreboard[row, col]
        }
        Console.WriteLine();
    }
    Console.WriteLine();

    solutionsFound++;
}
```

Testing the Code

Задачата за "8 царици" има 92 различни решения. Проверете дали вашия код генерира и отпечатва всички тях правилно. SolutionsFound брояч ще ви помогне да проверите броя на решения.

Оптимизирайте решението

Сега можем да оптимизираме нашия код:

- премахване на множеството attackedRows. Не е необходимо, защото всички кралици са поставени последователно в редове 0... 7. Опитайте се да използвате bool [] масив за attackedColumns, attackedLeftDiagonals и attackedRightDiagonals вместо множества. Обърнете внимание, че масивите са индексирани от 0 до техния размер и не може да има отрицателни индекси.

Решение

```
public static class EightQueens
{

    public const int Size = 8;
    public static bool[,] chessboard = new bool[Size, Size];
}

public class Program
{
    static HashSet<int> attackedRows = new HashSet<int>();
    static HashSet<int> attackedColumns = new HashSet<int>();
    static HashSet<int> attackedLeftDiagonals = new HashSet<int>();
    static HashSet<int> attackedRightDiagonals = new HashSet<int>();

    static void PutQueen(int row)
    {
        if (row == EightQueens.Size)
        {
            PrintSolution();
```

```
        }
        else
        {
            for (int col = 0; col < EightQueens.Size; col++)
            {
                if (CanPlaceQueen(row, col))
                {
                    MarkAllAttacketPositions(row, col);
                    PutQueen(row + 1);
                    UnmarkAllAttackedPositions(row, col);
                }
            }
        }
    }

    static bool CanPlaceQueen(int row, int col)
    {
        var positionOccupied =
            attackedRows.Contains(row) ||
            attackedColumns.Contains(col) ||
            attackedLeftDiagonals.Contains(col - row) ||
            attackedRightDiagonals.Contains(row + col);
        return !positionOccupied;
    }

    static void MarkAllAttacketPositions(int row, int col)
    {
        attackedRows.Add(row);
        attackedColumns.Add(col);
        attackedLeftDiagonals.Add(row - col);
        attackedRightDiagonals.Add(row + col);

        EightQueens.chessboard[row, col] = true;
    }

    static void UnmarkAllAttackedPositions(int row, int col)
    {
        attackedRows.Remove(row);
        attackedColumns.Remove(col);
        attackedLeftDiagonals.Remove(row - col);
        attackedRightDiagonals.Remove(row + col);

        EightQueens.chessboard[row, col] = false;
    }

    static void PrintSolution()
    {
        for (int row = 0; row < EightQueens.Size; row++)
        {
            for (int col = 0; col < EightQueens.Size; col++)
            {
                if (EightQueens.chessboard[row, col] == true)
                {
                    Console.Write('*');
```

```
                }
                else
                {
                    Console.Write('-');
                }
            }
            Console.WriteLine();
        }
        Console.WriteLine();
    }

    public static void Main(string[] args)
    {
        PutQueen(0);
    }
}
```

Решение с пермутации

Опитайте се да приложите по-ефективни пермутации, както в задачата "8 царици". Погледнете този код да се възползвате от идеята: http://introcs.cs.princeton.edu/java/23recursion/Queens.java.html.

Задача 6.13. Намиране на път в лабиринт

Даден е лабиринт. Целта ви е да намерите всички пътища от самото начало (клетка 0, 0) към изхода, маркирана с 'e'.

- Празните клетки са отбелязани с тире "-".
- Стените са отбелязани със звездичка "*".

На първия ред ще получите размерите на лабиринта. На следващите ще получите самия лабиринт. Редът на пътища не е от значение.

Примери

Input	Output
3 3 --- -*- --e	RRDD DDRR
3 5 -**-e ----- *****	DRRRRU DRRRUR

Подсказка

Създаване на методи за четене и намирането на всички пътища в лабиринта.

```
static void Main(string[] args)
{
    lab = ReadLab();
    FindPaths(0, 0, 'S');
}
```

Създаване на статичен списък, който ще съдържа всички пътища

```
static List<char> path = new List<char>();
```

Намиране на всички пътища трябва да е рекурсивно

```
private static void FindPaths(int row, int col, char direction)
{
    if (!IsInBounds(row, col))
        return;

    path.Add(direction);

    if (IsExit(row, col))
    {
        PrintPath();
    }
    else if (!IsVisited(row, col) && IsFree(row, col))
    {
        Mark(row, col);
        FindPaths(row, col + 1, 'R');
        FindPaths(row + 1, col, 'D');
        FindPaths(row, col - 1, 'L');
        FindPaths(row - 1, col, 'U');
        Unmark(row, col);
    }

    path.RemoveAt(path.Count - 1);
}
```

Изпълнете всички helper методи, които се намират в горния код.

Решение

```
public class Program
{
    static int height = int.Parse(Console.ReadLine());
    static int width = int.Parse(Console.ReadLine());
    static string[] lab = new string[height];
```

```
static List<char> path = new List<char>(); // U=Up, D=Down, L=Left, R=Right

static void ReadLab()
{
    for (int i = 0; i < height; i++)
    {
        lab[i] = Console.ReadLine();
    }
}

static void FindPaths(int row, int col, char direction)
{
    if (!IsInBounds(row, col)) return; // Ебит

    path.Add(direction);

    if (IsExit(row, col))
    {
        Console.WriteLine(string.Join("", path));
    }
    else if (!IsVisited(row, col) && IsFree(row, col, direction))
    {
        Mark(row, col);
        FindPaths(row, col + 1, 'R');
        FindPaths(row + 1, col, 'D');
        FindPaths(row, col - 1, 'L');
        FindPaths(row - 1, col, 'U');
        Unmark(row, col);
    }

    path.RemoveAt(path.Count - 1);
}

static bool IsFree(int row, int col, char direction)
{
    switch (direction)
    {
        case 'D': // Down
            {
                if
                (
                    row < width - 1 &&
                    col > 0 &&
                    col < height - 1 &&
                    lab[row][col + 1] == '*' &&
                    lab[row][col - 1] == '*' &&
                    lab[row + 1][col] == '*'
                ) return false;
                else return true;
                break;
            }

        case 'R': // Right
            {
                if
```

```
                (
                    row < width - 1 &&
                    row > 0 &&
                    col < height - 1 &&
                    lab[row][col + 1] == '*' &&
                    lab[row - 1][col] == '*' &&
                    lab[row + 1][col] == '*'
                ) return false;
                else return true;
                break;
            }

        case 'L': // Left
            {
                if
                (
                    row < width - 1 &&
                    col > 0 &&
                    row > 0 &&
                    lab[row - 1][col] == '*' &&
                    lab[row][col - 1] == '*' &&
                    lab[row + 1][col] == '*'
                ) return false;
                else return true;
                break;
            }

        case 'U': // Up
            {
                if
                (
                    col > 0 &&
                    row > 0 &&
                    col < height - 1 &&
                    lab[row][col + 1] == '*' &&
                    lab[row][col - 1] == '*' &&
                    lab[row - 1][col] == '*'
                ) return false;
                else return true;
                break;
            }
    }
    return true;
}

static bool IsInBounds(int row, int col)
{
    if (row < 0 || col >= width || col < 0 || row >= height) return false;
    else return true;
}

static bool IsVisited(int row, int col)
{
    if (lab[row][col] == '*') return true;
    else return false;
```

```
    }

    static bool IsExit(int row, int col)
    {
        if (lab[row][col] == 'e') return true;
        else return false;
    }

    static void Mark(int row, int col)
    {
        var copy = lab[row].ToCharArray();
        if (lab[row][col] != 'e') copy[col] = '*';
        lab[row] = string.Join("", copy);
    }

    static void Unmark(int row, int col)
    {
        var copy = lab[row].ToCharArray();
        if (lab[row][col] != 'e') copy[col] = '-';
        lab[row] = string.Join("", copy);
    }

    public static void Main(string[] args)
    {
        ReadLab();
        FindPaths(0, 0, 'S');
    }
}
```

Задача 6.14. Думи

Подадена ви е поредица от латински букви. Напишете програма, която намира броя на всички думи, не съдържащи две последователни еднакви букви, които могат да бъдат съставени чрез пренареждане на дадените букви. Генерираните думи трябва да съдържат всички подадени букви. Ако подадена дума отговаря на изискванията, тя трябва да бъде включена в броя.

Вход

- Входните данни трябва да се прочетат от конзолата.
- На единствения ред с входни данни ще има само една дума, съдържаща всички букви, които трябва да използвате за съставяне на думите.
- Входните данни винаги ще са валидни и в описания формат. Не е необходимо да бъдат изрично проверявани.

Изход

- Изходните данни трябва да бъдат отпечатани на конзолата.
- На единствения изведен ред напишете броя намерени думи.

Ограничения

- Броят на подадените букви ще бъде между 1 и 10 включително.
- Всички подадени букви ще бъдат малки латински букви ('a' – 'z')

- Допустимо време за работа на вашата програма: 0.35 секунди. Допустима памет: 32 MB.

Примери

Вход	Примерен изход	Коментари
xy	2	Две възможни думи: "xy" и "yx"
xxxy	0	Невъзможно е да се състави дума с тези букви.
aahhhaa	1	Единствената възможна дума е "ahahaha".
nopqrstuvw	3628800	Има 3628800 възможни думи.

Решение

```
public class Program
{
    public static List<string> final = new List<string>();
    public static int Counter = 0;
    public static bool distincted = true;

    public static void CheckWord(char[] w)
    {
        bool goodToGo = true;

        for (int i = 0; i < w.Length - 1; i++)
        {
            if (w[i] == w[i + 1])
            {
                goodToGo = false;
                break;
            }
        }

        if (goodToGo)
        {
            if (distincted) Counter++;
            else
            {
                final.Add(String.Join("", w));
                final = final.Distinct().ToList();
            }
        }
    }

    public static void Swap(ref char a, ref char b)
    {
        if (a == b) return; // Exit
        a ^= b;
        b ^= a;
        a ^= b;
    }

    public static void GetPermutations(char[] w)
    {
```

```
        int m = w.Length - 1;
        GetPermutations(w, 0, m);
    }

    public static void GetPermutations(char[] w, int k, int m)
    {
        if (k == m)
        {
            CheckWord(w); // Exit
        }
        else
        {
            for (int i = k; i <= m; i++)
            {
                Swap(ref w[k], ref w[i]);
                GetPermutations(w, k + 1, m);
                Swap(ref w[k], ref w[i]);
            }
        }
    }

    public static void Main(string[] args)
    {
        string s = Console.ReadLine();
        char[] w = s.ToCharArray();

        if (w.Count() != w.Distinct().Count())
        {
            distincted = false;
        }

        GetPermutations(w);

        if (distincted)
        {
            Console.WriteLine(Counter);
        }
        else
        {
            final.Count();
        }
    }
}
```

Тема 7. Подготовка за изпит

Задача 7.1. Влакче

Общ преглед

Вие управлявате една малка жп гара. Въпреки че гарата е много малка от нея тръгват влакове - както пътнически, така и товарни. Един от коловозите на гарата е зает с вагони за композициите. По другия може да се минава, за да се напусне гарата или пък, за да мине машина, която иска да вземе вагони от другия коловоз.

Вашата задача е по дадена информация за всеки влак да създадете разписание, което спазва правилата за приоритет за преминаване през свободния коловоз.

Приемете, че преминаването по коловоза отнема винаги еднакво време и се случва винаги с еднаква скорост, а след като влакът е преминал по него той вече не може да предизвика конфликт с друг влак.

Приоритетът на преминаване е следният:

Винаги разглеждаме последните два добавени влака. От тях влакът се определя по следния приоритет:

1. Товарни влакове с повече от 15 вагона
2. Пътнически влакове
3. Товарни влакове с по-малко от 15 вагона

Ако само един влак е в списък на чакащите, то минава той без значение какъв е

Подзадача 1: Структура на Влак

Train

Всички влакове имат номер, име, вид, брой вагони:

- `Number` – **цяло число, съдържащо до 5 цифри**
- `Name` – **текстов низ, без интервали**
- `Type` – **вид на влака, буква "P" – обозначава пътнически, буква "F" – обозначава товарен**
- `Cars` – **цяло число, показващо брой вагони на композицията**

Реализирайте и конструктор:

- Train(int number, string name, string type, int cars) – този конструктор трябва да приема номер на влака, името на влака, видът му и брой вагони. За справка вижте следната схема за Product.cs:

Train.cs

```
public Train(int number, string name, string type, int cars) {
    //TODO: Добавете вашия код тук …
}
//TODO: Добавете ToString() и свойства…
```

Структура за съхранение на чакащите влакове

Трябва да реализирате специална структура, която има два края (Deque). Тази структура трябва да позволява добавяне в края и началото, както и извличане на елементите от края и началото, а също така и премахването им. Идеята е пътническите влакове да са в началото, а товарните в края.

Deque.cs

```
public class Deque<T> : IList<T> {
	public Deque() : this(defaultCapacity) {
		//празен конструктор, задава капацитета на дека на стойността по
подразбиране (16)
	}
	public Deque(int capacity) {
		//създава дека с точно зададен капацитет
	}
	public Deque(IEnumerable<T> collection)
	     : this(collection.Count()){
		//създава дека с капацитет съответстващ на посочената колекция и
прехвърля елементите от колекцията в дека
	}
	public int Capacity; //показва капацитета
	public int Count; //показва броят елементи
	public void AddFront(T item) {
		//добавя елемент отпред
	}
	public void AddBack(T item) {
		//добавя елемент отзад
	}
	public T RemoveFront() {
		//връща и премахва елемента отпред
	}
	public T RemoveBack() {
		//връща и премахва елемента отзад
	}
	public T GetFront() {
		//връща, без да премахва, елемента отпред
	}
	public T GetBack() {
		//връща, без да премахва, елемента отзад
	}
}
```

Управление на влаковете

Команди

Вашето приложение трябва да реализира следните команди:

- Add <номер на влак> <име на влак> <вид> - добавя нов влак, който трябва да премине през коловоза
- Travel - разрешава отпътуването на следващия по ред влак. При тази команда, вие трябва да изведете информацията за влака, който отпътува и да го премахнете от информацията с чакащи влакове
- Next - при тази команда трябва да отпечатате съобщение “Next train: <номер на влак> <име на влак> <вид>”

Подзадача 2: Още една команда

Трябва да реализирате още команди:

- History - при тази команда трябва да отпечатате информация за вече преминалите влакове по обратния ред на тяхното преминаване, т.е. последният преминал се изпечатва най-напред. Удобно е тази информация да се съхранява в Stack<T>

Забележка: Освен горепосочените методи трябва да реализирате и необходимите свойства за всеки от класовете. Възможно е да е удачно да реализирате допълнителни полета, свойства и методи, по ваша преценка.

Скелет

За да се ориентирате по-добре в структурата на решението получавате помощен скелет TrainsSkeleton

Ограничения

- Броят на влаковете, които може да получите ще бъде не повече от 100
- Номерът на влака ще е цяло число с 5 цифри
- Името на влакът няма да съдържа интервали
- Видът на влака ще се отбелязва с буква - P за пътнически и F за товарен
- Командите Next и History може да бъдат извикани дори когато няма информация за чакащи влакове. В такъв случай, вие трябва да игнорирате съответното извикване на командата

Вход / Изход

Вход

- Програмата ще получава множество редове с информация. Всеки ред представлява команда. Самият вход се обработва изцяло от примерния Program.cs.

- Всички команди приключват с въвеждането на End

Изход

За някои от командите не е нужно да извеждате нищо. За останалите формата е:

Travel

Ако има влак, който да премине:

{номер} {име} {вид} {вагони}

Next

Ако има влак, който да премине:

Next Train: {номер} {име} {вид} {вагони}

History

За всеки един от влаковете, които вече са минали изведете по един ред в следния формат:

{номер} {име} {вид} {вагони}

Примери

Вход	Изход	Обяснение
Add 85611 Karlovo P 2 Add 82223 Sofia P 3 Add 12123 Varna P 4 Next Travel Next Add 12125 Plovdiv P 4 Add 12126 Karnobat P 4 Travel Next Travel Add 12127 Karnobat P 2 End	12123 Varna P 4 12123 Varna P 4 82223 Sofia P 3 12126 Karnobat P 4 12125 Plovdiv P 4 12125 Plovdiv P 4	В този пример варненският влак напуска гарата, но след напускането му добавяме два влака – пловдивски и карнобатски, като има точно 2 команди Travel – те придвижват карнобатския и пловдивския влак. След това добавяме и още един карнобатски влак, но понеже преди End командата не извикваме Travel отново – тези влакове не се появяват в изхода
Add 85611 Karlovo P 2 Add 82223 Sofia P 3 Add 12123 Varna P 4 Next Travel Next Add 12125 Plovdiv P 4 Add 12126 Karnobat P 4 Travel History Next Travel History End	12123 Varna P 4 12123 Varna P 4 82223 Sofia P 3 12126 Karnobat P 4 12126 Karnobat P 4 12123 Varna P 4 12125 Plovdiv P 4 12125 Plovdiv P 4 12125 Plovdiv P 4 12126 Karnobat P 4 12123 Varna P 4	В този пример първо заминава варненския влак, после добавяме пловдивски и карнобатски, след което карнобатския заминава. При справката за история изкарваме карнобатския и варненския влак. След това продължаваме с преминаване на пловдивския влак. Отново изакрваме справка, в която фигурират пловдивския, карнобатския и варненския влак – според реда на преминаваме (от най-скорошен към най-отдавнашен)
Add 31233 BobovDol F 30 Add 85611 Karlovo P 3 Add 22222 Tulovo F 10 Travel Travel Travel Add 31234 BobovDol F 30 Add 85612 Karlovo P 3 Travel History End	85611 Karlovo P 3 22222 Tulovo F 10 31233 BobovDol F 30 31234 BobovDol F 30 31234 BobovDol F 30 31233 BobovDol F 30 22222 Tulovo F 10 85611 Karlovo P 3	В този случай първо добавяме 3 влака – товарен за Бобов Дол, пътнически за Карлово и един товарен за Тулово с 10 вагона. Понеже сравняваме само последните 2 влака – пътническият е с приоритет пред товарния, т.като той е с по-малко от 10 вагона – затова карловския минава пръв. След това остават само двата товарни, следователно минава туловския и едва след това този за Бобов Дол. След това добавяме два влака – товарен за Бобов Дол с 30 вагона, а след него пътнически с 3 вагона – понеже дългите товарни влакове са с приоритет над пътническите – влакът за Бобов Дол минава. В историческата справка излизат

		двата влака за Бобов Дол, влакът за Тулово и първият влак за Карлово

Точки

Разбивката по точки е следната:

1. 30 точки се присъждат, ако вашата програма работи коректно в случаяй, че всички влакове са пътнически. Важно е за тази цел командата Travel да работи коректно
2. 30 точки се присъждат, ако вашата програма работи коректно в случай, че влаковете са от какъв да е вид. Важно за тази цел е командата Travel и командата Next да работят коректно
3. 40 точки се присъждат, ако вашата програма отпечатва коректно историята на преминалите влакове

Общ брой точки: 100

Решение

```
class Deque<T>
{
    private T[] frontBuffer = new T[16];
    private T[] backBuffer = new T[16];

    public int BackCount { get; private set; }
    public int FrontCount { get; private set; }
    public int Count
    {
        get
        {
            return BackCount + FrontCount;
        }
    }

    public void AddBack(T item)
    {
        if (this.BackCount == this.backBuffer.Length)
        {
            GrowBack();
        }
        this.backBuffer[this.BackCount] = item;
        this.BackCount++;
    }

    private void GrowBack()
    {
        T[] newBuffer = new T[backBuffer.Length * 2];
        backBuffer.CopyTo(newBuffer, 0);
        backBuffer = newBuffer;
    }

    public void AddFront(T item)
    {
        if (this.FrontCount == this.frontBuffer.Length)
```

```
        {
            GrowFront();
        }
        this.frontBuffer[this.FrontCount] = item;
        this.FrontCount++;
    }
    private void GrowFront()
    {
        T[] newBuffer = new T[frontBuffer.Length * 2];
        frontBuffer.CopyTo(newBuffer, 0);
        frontBuffer = newBuffer;
    }

    public T GetFront()
    {
        if (0 == this.FrontCount)
        {
            return default(T);
        }

        T result = frontBuffer[this.FrontCount - 1];
        return result;
    }

    public T GetBack()
    {
        if (0 == this.BackCount)
        {
            return default(T);
        }

        T result = backBuffer[this.BackCount - 1];
        return result;
    }

    public T RemoveBack()
    {
        if (0 == this.BackCount)
        {
            throw new InvalidOperationException("The Deque is empty");
        }

        int endIndex = this.BackCount - 1;
        T result = backBuffer[endIndex];
        backBuffer[endIndex] = default(T);
        this.BackCount--;

        return result;

    }

    public T RemoveFront()
    {
        if (0 == this.FrontCount)
        {
```

```
            throw new InvalidOperationException("The Deque is empty");
        }

        int endIndex = this.FrontCount - 1;
        T result = frontBuffer[endIndex];
        frontBuffer[endIndex] = default(T);
        this.FrontCount--;

        return result;
    }
}

class Train
{
    private int number;
    private string name;
    private string type;
    private int cars;

    public int Number
    {
        get
        {
            return this.number;
        }

        set
        {
            this.number = value;
        }
    }

    public string Name
    {
        get
        {
            return this.name;
        }

        set
        {
            this.name = value;
        }
    }
    public string Type
    {
        get
        {
            return this.type;
        }

        set
        {
            this.type = value;
        }
```

```
    }

    public int Cars
    {
        get
        {
            return this.cars;
        }

        set
        {
            this.cars = value;
        }
    }

    public Train(int number, string name, string type, int cars)
    {
        this.Number = number;
        this.Name = name;
        this.Type = type;
        this.Cars = cars;
    }

    public override string ToString()
    {
        return this.Number + " " + this.Name + " " + this.Type + " " + this.Cars;
    }
}

class Program
{
    private static Deque<Train> trains = new Deque<Train>();
    private static Stack<Train> history = new Stack<Train>();

    private static void Add(int number, string name, string type, int cars)
    {
        if (type == "F")
        {
            //Add freight trains to back
            trains.AddBack(new Train(number, name, type, cars));
        }
        else
        {
            trains.AddFront(new Train(number, name, type, cars));
        }
    }

    private static void Travel()
    {
        if (trains.Count > 0)
        {

            Train frontTrain = trains.GetFront();
            Train backTrain = trains.GetBack();
            if (backTrain != null && backTrain.Type == "F" && backTrain.Cars > 15)
```

```
            {
                Console.WriteLine(trains.RemoveBack());
                history.Push(backTrain);
            }
            else if (frontTrain != null && frontTrain.Type == "P")
            {
                Console.WriteLine(trains.RemoveFront());
                history.Push(frontTrain);
            }
            else if (backTrain != null && backTrain.Type == "F")
            {
                Console.WriteLine(trains.RemoveBack());
                history.Push(backTrain);
            }
        }
    }

    private static void Next()
    {
        if (trains.Count > 0)
        {

            Train frontTrain = trains.GetFront();
            Train backTrain = trains.GetBack();
            if (backTrain != null && backTrain.Type == "F" && backTrain.Cars > 15)
            {
                Console.WriteLine(backTrain);
            }
            else if (frontTrain != null && frontTrain.Type == "P")
            {
                Console.WriteLine(frontTrain);
            }
            else if (backTrain != null && backTrain.Type == "F")
            {
                Console.WriteLine(backTrain);
            }
        }
    }

    private static void History()
    {
        Train[] trainHistory = history.ToArray();
        foreach (var train in trainHistory)
        {
            Console.WriteLine(train);
        }
    }

    static void Main(string[] args)
    {
        string[] command;
        do
        {
            command = Console.ReadLine().Split(' ').ToArray();
            switch (command[0])
```

```
            {
                case "Add":
                    Add(int.Parse(command[1]), command[2], command[3],
int.Parse(command[4]));
                    break;
                case "Travel":
                    Travel();
                    break;
                case "Next":
                    Next();
                    break;
                case "History":
                    History();
                    break;
            }
        } while (command[0] != "End");
    }
}
```

Задача 7.2. Игра

Иванчо и Марийка много обичат да играят на играта "Колекционирай числата". Играта е много проста, но в същото време доста занимателна за тях. В началото на играта се определят капацитет на колекцията. Това е броят елементи, които могат да се колекционират. Колекционирането се случва по доста прост начин - чрез хвърляне на два зара от съответния играч. Така получената двойка се записва в колекцията.

В играта основна важност играе операцията сумиране на двойки – това е операция при която се събират съответните елементи на две двойки и се получава нова двойка.

Разгледайте следната схема, за да се ориентирате как правилно да сумирате двойките:

```
 2  +  3  =  5
-------------
 5     3     8
```

При запълване на капацитета на колекцията, всички двойки числа се сумират, така че да се получи една двойка и тя става нов член на колекцията, а останалата част от двойките, които са били получени чрез заровете изчезват. **Сумирането на двойките се случва в момента на добавянето на нова двойка, за която няма място!**

Когато това се случи, играта все пак може да продължи. Новите елементи се добавят след двойката, получена от сумиране и при достигане на капацитета отново, двойките получени от заровете се сумират по същия начин, а резултатът се добавя след предната двойка получена от сумиране. **ВАЖНО: В сумирането на участват двойките, които са получени като сумиране, а само тези, които са добавени от заровете.**

Процесът се повтаря отново и отново, докато не се запълни цялата колекция с двойки, получени от сумиране или пък ако не бъде дадена команда за край на играта.

За да се определи финалния резултат на всеки участник се вземат всички негови двойки, получени като резултат от сумиране и се извършва сумиране върху тях, така че да се получи една крайна двойка. Победител е този, при който абсолютната стойност на разликата между двете числа е **най-малка**.

Вашата задача е да създадете програма, която да следи играта за n на брой играчи и да отговаря на определени команди:

Първа подзадача (30 точки)
В рамките на тази подзадача трябва да реализирате две комадни: - **Dice** - тази команда задава на кой играч какви числа са се паднали. - **CurrentPairSum** - тази команда показва сумата от всички двойки във формат: (a, b), където a и b са съответно първия и втория елемент на получената двойка.

Втора подзадача (30 точки)
В рамките на тази подзадача реализирайте следната команда: - **CurrentState** - тази команда отпечатва всички двойки, които са съхранени в колекцията на дадения играч, без значение как са получени

Трета подзадача (20 точки)

- **Winner** - тази команда отпечатва победителя във формат: "{играч} wins the game!". Командата ще бъде викана само тогава, когато може да се определи еднозначно победител.

Четвърта подзадача (20 точки)

- **CurrentStanding** - тази команда отпечатва класиране в намалящ ред, в което на всеки ред има информация за един играч във формат: "{играч} - (a, b)". Където a и b са числата от двойката на съответния играч, с която той участва в класирането.

За ваше удобство получвате скелет за решението на тази задача.

Pair.cs Този клас трябва да описва всяка двойка. Класът трябва да поддържа свойства, които съхраняват първата и втората стойност на двойката. Конструктор с два параметъра – числата от двойката. ToString метод и Difference метод, който връща абсолютната стойност на разликата между първия и втория елемент на двойката.

```
class Pair
{
    public int First { get; set; }
    public int Last { get; set; }

    public Pair(int first, int last)
    {
        //TODO: Add code
```

```
    }

    public override string ToString()
    {
        //TODO: Add code
    }

    public int Difference()
    {
        //TODO: Add code
    }
}
```

CapacityList.cs Този клас ще описва структурата, която съхранява всички двойки. По своята същност тя прилича на статичен списък, като мази структура трябва да има масив от обекти от клас Pair. Освен това трябва да разполага с конструктор, който създава масив от Pair обекти. Трябва да поддържате свойство Count, което да показва броят на двойките в масива, които могат да участват в класирането. Методът SumIntervalPairs() трябва да може да сумира всички двойки, започвайки от зададен стартов индекс – този метод ще се използва вътрешно от класа, за да определя сумата от двойките за всеки следващ елемент, който запълва. Методът Sum() трябва да може да сумира всички двойки, които имат право да участват в сумата за класирането. Методът Add(Pair item) има за цел да добавя нова двойка към структурата и да извиква методът за сумиране, когато структурата е на път да се запълни. PrintCurrentState() отпечатва информация за текущите двойки в масива.

```
class CapacityList
{
    private const int InitialCapacity = 2;
    private Pair[] items;

    private int startIndex = 0; //показва първият индекс, от който започваме да
сумираме текущите елементи
    private int nextIndex = 0; //показва поредният индекс, на който можем да
поставим елемент

    public CapacityList(int capacity = InitialCapacity)
    {
        this.items = new Pair[capacity];
    }

    public int Count
    {
        get;
        private set;
    }

    public Pair SumIntervalPairs()
    {
        //TODO: сумирайте двойките от startIndex до nextIndex
    }
```

```
    public Pair Sum()
    {
        //TODO: сумирайте двойките от 0 до this.Count – всички двойки, които имат
право да участват в класирането
    }

    public void Add(Pair item)
    {
        //TODO: Добавяне на двойката
    }

    public void PrintCurrentState()
    {
        //TODO: отпечатайте всички двойки от 0 до nextIndex
    }
}
```

Вход

- На първи ред ще въведете капацитета на структурата. На всеки от следващите редове ще получавате някоя от командите.
- Въвеждането на команди приключва с командата **End**.

Изход

- За всяка от командите изведете изхода посочен по-горе.

Примери

Вход	Изход
3 Dice Gosho 1 3 Dice Ivancho 5 4 Dice Mimi 2 3 Dice Gosho 2 5 Dice Ivancho 3 3 CurrentPairSum Gosho Dice Gosho 1 1 CurrentPairSum Gosho Dice Gosho 3 2 CurrentPairSum Gosho Dice Ivancho 2 1 Dice Gosho 3 3 Dice Ivancho 2 2 CurrentPairSum Ivancho Dice Mimi 6 6 Dice Mimi 3 2 Dice Mimi 1 6 CurrentPairSum Mimi End	(0, 0) (0, 0) (4, 9) (10, 8) (11, 11)

- Първият път сумата на Гошо е (0, 0), т. като досега той не е изтеглил достатъчно двойки, за да формира двойка за сума по правилата. Вторият път Гошо вече има 3 двойки, но те не се сумират, докато не

се прибави и четвърта. Едва на третия път Гошо вече има получена сума.

- Иванчо има двойка (10, 8) - тя е от двойките (5, 4)+(3, 3)+(2, 1), а двойката (2, 2) не участва тъй като тя е само добавена, а не получена от сумиране.
- Мими има двойка (11, 11) - тя е получена от двойките (2, 3)+(6, 6)+(3, 2), а (1, 6) не участва в сумата.

Вход	Изход
3 Dice Ivancho 1 3 Dice Ivancho 5 4 Dice Ivancho 2 3 Dice Ivancho 2 5 Dice Ivancho 3 3 Dice Ivancho 1 1 CurrentState Ivancho Dice Gosho 3 3 Dice Gosho 1 1 Dice Gosho 3 2 Dice Gosho 2 1 Dice Gosho 3 3 Dice Gosho 2 2 CurrentState Gosho End	(8, 10) (5, 8) (1, 1) (7, 6) (5, 4) (2, 2)

- Първите три двойки са тези на Иванчо, като първата двойка е получена като сума на: (1, 3)+(5, 4)+(2, 3), а втората като сума на (2, 5)+(3, 3), третата двойка е двойката, която се е паднала от заровете.
- Вторите три двойки са на Гошо. Първата е сума на (3, 3)+(1, 1)+(3, 2), втората е сума на (2, 1)+(3, 3).

Вход	Изход
3 Dice Ivancho 1 3 Dice Ivancho 5 4 Dice Ivancho 2 3 Dice Ivancho 2 5 Dice Ivancho 3 3 Dice Ivancho 1 1 CurrentState Ivancho Dice Gosho 3 3 Dice Gosho 1 1 Dice Gosho 3 2 Dice Gosho 2 1 Dice Gosho 3 3 Dice Gosho 2 2	(8, 10) (5, 8) (1, 1) (7, 6) (5, 4) (2, 2) (13, 18) (12, 10) Gosho wins the game!

CurrentState Gosho CurrentPairSum Ivancho CurrentPairSum Gosho Winner End	

- В сумата за изчисляване на победител в този случай не участват (1, 1) за Иванчо и (2, 2) за Гошо, т.като не е въведена 7 двойка, която да принуди структурата да извърши сумиране и затова сумата за изчисляване на победител се получава само от по 2-те двойки, които са получени като резултат от сумиране.
- Гошо печели играта, т.като при него разликата е 2, а при Иванчо - 5.

Вход	**Изход**
3 Dice Gosho 1 3 Dice Ivancho 5 4 Dice Mimi 2 3 Dice Gosho 2 5 Dice Ivancho 3 3 CurrentPairSum Gosho Dice Gosho 1 1 CurrentPairSum Gosho Dice Gosho 3 2 CurrentPairSum Gosho Dice Ivancho 2 1 Dice Gosho 3 3 Dice Ivancho 2 2 CurrentPairSum Ivancho Dice Mimi 6 6 Dice Mimi 3 2 Dice Mimi 1 6 CurrentPairSum Mimi CurrentStanding Winner End	(0, 0) (0, 0) (4, 9) (10, 8) (11, 11) Mimi - (11, 11) Ivancho - (10, 8) Gosho - (4, 9) Mimi wins the game!

Тестът е аналогичен на първия, но с добавени команди за класиране и победител, като този път победител е Мими, т.като нейната разлика е 0.

Решение

```
public class Pair
{
    public int First { get; set; }

    public int Last { get; set; }

    public bool Summmed { get; set; }

    public Pair(int first, int last, bool summed = false)
```

```
    {
        this.First = first;
        this.Last = last;
        this.Summmed = summed;
    }

    public override string ToString()
    {
        return $"{this.First} {this.Last}";
    }

    public int Difference()
    {
        return Math.Abs(this.First - this.Last);
    }
}

public class CapacityList
{
    private const int InitialCapacity = 2;
    private Pair[] items;

    private int startIndex = 0;
    private int nextIndex = 0;

    public int Count { get; private set; }

    public CapacityList(int capacity = InitialCapacity)
    {
        this.items = new Pair[capacity];
        this.Count = 0;
    }

    public Pair SumIntervalPairs()
    {
        Pair sum = new Pair(items[startIndex].First, items[startIndex].Last);
        for (int i = startIndex + 1; i < nextIndex; i++)
        {
            sum = new Pair
            (
                sum.First + items[i].First,
                sum.Last + items[i].Last,
                true
            );
        }
        return sum;
    }

    public Pair Sum()
    {
        int part1 = 0, part2 = 0;
        for (int i = 0; i < this.Count; i++)
        {
            if (this.items[i].Summmed == false)
            {
```

```
                part1 += items[i].First;
                part2 += items[i].Last;
            }
        }
        return new Pair(part1, part2, true);
    }

    public void Add(Pair item)
    {
        if (nextIndex == items.Length)
        {
            int capacity = items.Count();

            var newItems = this.items.Where(x => x.Summmed == true).ToList();
            newItems.Add(Sum());
            newItems.Add(item);

            items = new Pair[capacity];
            for (int i = 0; i < newItems.Count; i++)
            {
                items[i] = newItems[i];
            }
            Count = newItems.Count();
            nextIndex = newItems.Count();
        }
        else
        {
            items[nextIndex] = item;
            nextIndex++;
            Count++;
        }
    }

    public void PrintCurrentState()
    {
        for (int i = 0; i < nextIndex; i++)
        {
            Console.WriteLine(items[i].ToString());
        }
    }

    public void PrintCurrentStateSum()
    {
        bool printed = false;
        for (int i = 0; i < nextIndex; i++)
        {
            if (items[i].Summmed == true)
            {
                Console.WriteLine(items[i].ToString());
                printed = true;
            }
        }
        if (!printed) Console.WriteLine("(0,0)");
    }
}
```

```
public class Program
{
    public static int capacity = 0;

    public static Dictionary<string, CapacityList> players = new Dictionary<string,
CapacityList>();

    public static void Dice(string player, int dice1, int dice2)
    {
        if (players.ContainsKey(player))
        {
            players[player].Add(new Pair(dice1, dice2));
        }
        else
        {
            var list = new CapacityList(capacity);
            list.Add(new Pair(dice1, dice2));
            players.Add(player, list);
        }
    }

    public static void CurrentPairSum(string player)
    {
        players[player].PrintCurrentStateSum();
    }

    public static void CurrentState(string player)
    {
        players[player].PrintCurrentState();
    }

    public static void Winner()
    {
        string winner = String.Empty;
        int min = players.FirstOrDefault().Value.SumIntervalPairs().Difference();
        foreach (var player in players)
        {
            var temp = player.Value.SumIntervalPairs().Difference();
            if (temp <= min)
            {
                min = temp;
                winner = player.Key;
            }
        }
        Console.WriteLine($"{winner} wins the game!");
    }

    public static void CurrentStanding()
    {
        var list = players.OrderByDescending(x =>
x.Value.SumIntervalPairs().Difference()).ToList();
        foreach (var player in list)
        {
            Console.Write($"{player.Key} - ");
```

```
            player.Value.PrintCurrentState();
        }
    }

    static void Main(string[] args)
    {
        capacity = int.Parse(System.Console.ReadLine());
        var line = Console.ReadLine().Split().ToArray();
        while (line[0] != "End")
        {
            switch (line[0])
            {
                case "Dice":
                    {
                        Dice(line[1], int.Parse(line[2]), int.Parse(line[3]));
                        break;
                    }
                case "CurrentState":
                    {
                        CurrentState(line[1]);
                        break;
                    }
                case "CurrentPairSum":
                    {
                        CurrentPairSum(line[1]);
                        break;
                    }
                case "Winner":
                    {
                        Winner();
                        break;
                    }
                case "CurrentStanding":
                    {
                        CurrentStanding();
                        break;
                    }
            }
            line = Console.ReadLine().Split().ToArray();
        }
    }
}
```

Задача 7.3. Продукти

Общ преглед

Във вашата фирма постъпва проект за създаване на приложение, обслужващо „ресторант".

Вашият софтуер трябва да описва хладилник (Fridge) и продукт (Product).

Трябва да реализирате функционалност, която да позволява добавяне и премахване на продукти, проверка за налиности и приготвяне на ястията с определени продукти – всичко това ще работи чрез команди, които вие

ще получавате. Поредицата от команди приключва с „END". За ваше удобство ще получите готов Program.cs файл (вж. структурата му по-долу), и ще трябва да реализирате само необходимите класове Fridge.cs и Product.cs.

Основната идея се базира на това, че т.нар. хладилник е структура, която ще съдържа n на брой продукти. Структурата не трябва да пази продуктите в колекция! Всеки продукт пази референция към следващия в поредицата.

Program.cs

```
class Program
{
    static Fridge fridge = new Fridge();

    static void Main(string[] args)
    {
        string line;

        while ("END" != (line = Console.ReadLine()))
        {
            string[] cmdArgs = line.Split(' ');

            switch (cmdArgs[0])
            {
                case "Add":
                    AddProduct(cmdArgs[1]);
                    break;
                case "Check":
                    CheckProductIsInStock(cmdArgs[1]);
                    break;
                case "Remove":
                    try
                    {
                        int index = int.Parse(cmdArgs[1]);
                        RemoveProductByIndex(index);
                    }
                    catch (FormatException e)
                    {
                        RemoveProductByName(cmdArgs[1]);
                    }
                    break;
                case "Print":
                    ProvideInformationAboutAllProducts();
                    break;
                case "Cook":
                    CookMeal(cmdArgs.Skip(1).ToArray());
                    break;
            }
        }
    }

    private static void CookMeal(string[] indexes)
```

```
    {
        string[] products = fridge.CookMeal(int.Parse(indexes[0]),
int.Parse(indexes[1]));
        Console.WriteLine("Meal cooked. Used Products: " + string.Join(", ",
products));
    }

    private static void ProvideInformationAboutAllProducts()
    {
        string[] info = fridge.ProvideInformationAboutAllProducts();
        foreach (var item in info)
        {
            Console.WriteLine(item);
        }
    }

    private static void RemoveProductByName(string name)
    {
        string ProductName = fridge.RemoveProductByName(name);
        if (ProductName != null)
        {
            Console.WriteLine("Removed: " + ProductName);
        }
        else
        {
            Console.WriteLine("Product not found!");
        }
    }

    private static void RemoveProductByIndex(int index)
    {
        string ProductName = fridge.RemoveProductByIndex(index);
        if (ProductName != null)
        {
            Console.WriteLine("Removed: " + ProductName);
        }
        else
        {
            Console.WriteLine("Product not found!");
        }
    }

    private static void CheckProductIsInStock(string name)
    {
        bool isInStock = fridge.CheckProductIsInStock(name);

        Console.WriteLine(isInStock ? $"Product {name} is in stock."
            : "Not in stock");
    }

    private static void AddProduct(string name)
    {
        fridge.AddProduct(name);
    }
}
```

Подзадача 1: 30 точки

Product

Всички продукти имат име и референция към следващ продукт: - name = низ, съставен от малки и/или големи латински букви - next = референция към следващ продукт

Product.cs

```
private string name;
private Product next;

public Product(string name)
{
    // TODO: Добавете вашия код тук …
}

public string Name
{
    // TODO: Добавете вашия код тук …
}

public Product Next
{
    // TODO: Добавете вашия код тук …
}

public override string ToString()
{
    // TODO: Добавете вашия код тук …
}
```

Fridge

Всички хладилници Product head, Product tail, Count: - head = Product, първи в поредицата - tail = Product, последен в поредицата - count = Брой продукти

Fridge.cs

```
private Product head;
private Product tail;
private int count;

public Fridge() { }

public int Count
{
    // TODO: Добавете вашия код тук …
}

public void AddProduct(string ProductName)
{
    // TODO: Добавете вашия код тук …
}
```

```
public string[] CookMeal(int start, int end)
{
    // TODO: Добавете вашия код тук …
}

public string RemoveProductByIndex(int index)
{
    // TODO: Добавете вашия код тук …
}

public string RemoveProductByName(string name)
{
    // TODO: Добавете вашия код тук …
}

public bool CheckProductIsInStock(string name)
{
    // TODO: Добавете вашия код тук …
}

public string[] ProvideInformationAboutAllProducts()
{
    // TODO: Добавете вашия код тук …
}
```

Команда за добавяне на продукт

Вашето приложение трябва да обслужва следната команда за добавяне на продукти: - **Add** - тази команда има за цел да добави продукт с неговото име.

Команда за извеждане на информация

Вашето приложение във всеки един момент може да получи заявка да отпечата информация за всички налични продукти. Командата за това е следната:

- **Print** - отпечатва информация за всички продукти в структурата във формат: **Product {name}**
- За успешна реализация трябва да реализирате ваша версия на **ToString()** метода за класа **Product**.
- **RemoveProductByIndex** - Трябва да бъде премахнат елемент, който се намира на посочения индекс. Тъй като вашата структура не използва индексиране, удобен похват би бил използването на брояч. При успешно намиране и премахване на Product трябва да върнете неговото име, което ще бъде изпечатано на конзолата от Main метод-а. При ненамиране на такъв Product, трябва да бъде върната null стойност.
- **RemoveProductByName** - Трябва да бъде премахнат първият елемент, на който името отговаря на подаденото. При успешно намиране и премахване на Product трябва да върнете неговото име, което ще бъде изпечатано на конзолата от Main метод-а. При ненамиране на такъв Product, трябва да бъде върната null стойност.

- **CheckProductIsInStock** - Трябва да бъде намерен елемент, на който името отговаря на подаденото. При успешно намиране Product трябва да върнете **true** в обратен случай **false**
- **CookMeal<int startIndex, int endIndex>** - Трябва да бъдат намерени всички продукти от **startIndex** до **endIndex**. Имената на всички намерени продукти трябва да бъдат събрани в стрингов масив, който да бъде върнат от метода.

Подзадача 2: 30 точки

Подзадача 3: 20 точки

Подзадача 4: 20 точки

Команди

Вашето приложение трябва да реализира следните команди:

- **Add** - Добавя продукт към структурата
- **Print** – изпечатва се информация за всички налични продукти
- **Remove** - Трябва да бъде премахнат елемент, който се намира на посочения индекс. Тъй като вашата структура не използва индексиране, удобен похват би бил използването на брояч. При успешно намиране и премахване на Product трябва да върнете неговото име, което ще бъде изпечатано на конзолата от Main метод-а. При ненамиране на такъв Product, трябва да бъде върната null стойност.
- **Remove** - Трябва да бъде премахнат първият елемент, на който името отговаря на подаденото. При успешно намиране и премахване на Product трябва да върнете неговото име, което ще бъде изпечатано на конзолата от Main метод-а. При ненамиране на такъв Product, трябва да бъде върната null стойност.
- **Check** - При намерен продукт – **Product is in stock** в обратен случай – **Not in stock**
- **Cook <int startIndex, int endIndex>** - "Приготва се ястие", в контекста на програмата, това означава да извадите имената на всички продукти, който се намират от startIndex ДО endIndex.

В случай, че endIndex е след последния елемент, вземете колкото продукти имате от startIndex

Вход

- Програмата ще получава множество редове с информация. Всеки ред представлява команда. Самият вход се обработва изцяло от примерния Program.cs.
- Всички команди приключват с въвеждането на End

Изход

- За някои от командите не е нужно да извеждате нищо. За други е необходимо форматиране на изход – напр. Product.ToString(), Product.Name()

Ограничения

- Всички цели числа ще бъдат в диапазона **–10000** до **+10000**
- Имената няма да съдържат интервал

Примери

Вход	Изход
Add cherry Add salami Print END	Product cherry Product salami
Add cherry Add salami Add eggs Remove 1 Remove eggs Print Check dadadada Check cherry Check eggs Add eggs Cook 0 2 Cook 0 25 Remove 0 Print END	Removed: salami Removed: eggs Product cherry Not in stock Product cherry is in stock. Not in stock Meal cooked. Used Products: cherry, eggs Meal cooked. Used Products: cherry, eggs Removed: cherry Product eggs

Решение

```
public class Product
{
    private string name;
    private Product next;

    public Product(string name, Product next = null)
    {
        this.name = name;
        this.next = next;
    }

    public string Name
    {
        get { return this.name; }
        set { this.name = value; }
    }

    public Product Next
    {
        get { return this.next; }
        set { this.next = value; }
    }

    public override string ToString()
    {
        return $"Product {this.name}";
```

```
    }
}

public class Fridge
{
    private Product head;
    private Product tail;
    private int count;

    public Fridge()
    {
        this.head = null;
        this.tail = null;
        this.count = 0;
    }

    public int Count
    {
        get { return this.count; }
        private set { this.count = value; }
    }

    public void AddProduct(string name)
    {
        if (head == null)
        {
            var next = new Product(name); // 1x
            head = next;
            tail = next;
        }
        else
        {
            var next = new Product(name, head); // 2x
            head = next;
        }
        Count++;
    }

    public string[] CookMeal(int start, int end)
    {
        var result = new List<string>();
        int index = 0;
        var current = head;
        while (current != null)
        {
            if (index >= start && index <= end)
            {
                result.Add(current.Name);
            }
            index++;
            current = current.Next;
        }
        return result.ToArray();
    }
```

```
public string RemoveProductByIndex(int index)
{
    Product current = head;
    Product previous = null;
    int i = 0;
    while (current != null)
    {
        if (index == 0 && index == i)
        {
            var name = head.Name;
            head = head.Next;
            return name;
        }
        if (index == i)
        {
            var name = current.Name;
            previous.Next = current.Next;
            return name;
        }
        previous = current;
        current = current.Next;
        i++;
    }
    return String.Empty;
}

public string RemoveProductByName(string name)
{
    Product current = head;
    int i = 0;
    while (current != null)
    {
        if (current.Name == name)
        {
            return current.Name;
        }
        current = current.Next;
        i++;
    }
    return String.Empty;
}

public bool CheckProductIsInStock(string name)
{
    Product current = head;
    while (current != null)
    {
        if (current.Name == name)
        {
            return true;
        }
        current = current.Next;
    }
    return false; // Not Found
}
```

```
    public string[] ProvideInformationAboutAllProducts()
    {
        var list = new List<string>();
        Product current = head;
        while (current != null)
        {
            list.Add(current.Name);
            current = current.Next;
        }
        return list.ToArray();
    }
}

public class Program
{
    static Fridge fridge = new Fridge();

    static void Main(string[] args)
    {
        string line;

        while ("END" != (line = Console.ReadLine()))
        {
            string[] cmdArgs = line.Split(' ');

            switch (cmdArgs[0])
            {
                case "Add":
                    AddProduct(cmdArgs[1]);
                    break;
                case "Check":
                    CheckProductIsInStock(cmdArgs[1]);
                    break;
                case "Remove":
                    try
                    {
                        int index = int.Parse(cmdArgs[1]);
                        RemoveProductByIndex(index);
                    }
                    catch (FormatException e)
                    {
                        RemoveProductByName(cmdArgs[1]);
                    }
                    break;
                case "Print":
                    ProvideInformationAboutAllProducts();
                    break;
                case "Cook":
                    CookMeal(cmdArgs.Skip(1).ToArray());
                    break;
            }
        }
    }
```

```
    private static void CookMeal(string[] indexes)
    {
        string[] products = fridge.CookMeal(int.Parse(indexes[0]),
int.Parse(indexes[1]));
        Console.WriteLine("Meal cooked. Used Products: " + string.Join(", ",
products));
    }

    private static void ProvideInformationAboutAllProducts()
    {
        string[] info = fridge.ProvideInformationAboutAllProducts();
        foreach (var item in info)
        {
            Console.WriteLine($"Product {item.ToString()}");
        }
    }

    private static void RemoveProductByName(string name)
    {
        string ProductName = fridge.RemoveProductByName(name);
        if (ProductName != null)
        {
            Console.WriteLine("Removed: " + ProductName);
        }
        else
        {
            Console.WriteLine("Product not found!");
        }
    }

    private static void RemoveProductByIndex(int index)
    {
        string ProductName = fridge.RemoveProductByIndex(index);
        if (ProductName != null)
        {
            Console.WriteLine("Removed: " + ProductName);
        }
        else
        {
            Console.WriteLine("Product not found!");
        }
    }

    private static void CheckProductIsInStock(string name)
    {
        bool isInStock = fridge.CheckProductIsInStock(name);

        Console.WriteLine(isInStock ? $"Product {name} is in stock."
            : "Not in stock");
    }

    private static void AddProduct(string name)
    {
        fridge.AddProduct(name);
    }
```

```
}
```

Съдържание