**Решения на задачите от глава 16 на книгата "Въведение в програмирането със C#"**

Предлагаме ви решения на задачите от [книгата "Въведение в програ­мирането със C#"](http://www.introprogramming.info), заедно с анализ на задачата, описание на използва­ните идеи, алгоритми, подходи за решаване и тестове.

**Авторски колектив**

**Кирил Попов**

Контакти: kpopov.soft@gmail.com

**Димитър Тачев**

Контакти: cska\_\_forever@abv.bg

**Мариан Маринов**

Контакти: lezetac@gmail.com

**Наталия Илиева**

Контакти: ilieva.natalia@gmail.com

|  |  |
| --- | --- |
| **Задача 1. *Сума и средно аритметично на редица числа.*** | |
| **Условие**  Напишете програма, която прочита от конзолата поредица от цели положителни числа. Програмата трябва да изчислява сумата и средното аритметично на поредицата.  Hints: Може да използвате List<int> и LINQ функциите Average() и SUM(). | |
| **Описание на входа**  Всички положителни цели числа трябва да бъдат прочетени от конзолата, като всяко едно от тях ще бъде на нов ред. Поредицата свършва при въвеждане на празен ред (празен низ). Числата ще бъдат от 0 до 2147483647, а тяхната сума не по-голяма от 2147483647. | |
| **Описание на изхода**  Изходните данните трябва да се изведат на стандартния изход (конзолата).  При празна поредица на единствения ред на стандарния изход да се извежда съобщение:  „Sequence contains no elements“.  При непразна поредица:  На първия ред на стандартния изход трябва да се изведе сумата на всички числа от поредицата.  На втория ред трябда да се изведе средното аритметично на поредицата. (използвайте **Double** за средно аритметичното число). | |
| **Анализ на задачата**  За решаването на задачата е използвана структората от данни **List<T>** и разширяващите методи от **namespace** **System.Linq**: **Sum()** и **Average()**. В един цикъл **while**, който спира при въвеждане на празен ред се чете ред по ред от конзолата, като всеки ред се парсва в число и се добавя към листа. След края на цикъла се извикват методите за сума и средно аритметично число, като резултатите се изписват на конзолата чрез системния метод: **Console.WriteLine(string s)**. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;    class IntegerSequence  {      static void Main(string[] args)      {          string line = null;          List<int> numbers = new List<int>();          line = Console.ReadLine();          while (line != "")          {              int number = int.Parse(line);              numbers.Add(number);              line = Console.ReadLine();          }              if (numbers.Count == 0)              {                  Console.WriteLine("Sequence contains no elements");              }              else              {                  Console.WriteLine(numbers.Sum());                  Console.WriteLine(numbers.Average());              }      }  } | |
| **Тестове**  Интересните случаи за тестване са следните:   * Поредица от само едно число. * Празна поредица. * Поредица с не цяло средо аритметично число. * Поредица включваща 0. * Поредица включваща максималното възможно число. | |
| 0 | 0  0 |
|  | Sequence contains no elements |
| 2147483647 | 2147483647  2147483647 |
| 10  10 | 20  10 |
| 1  2  3  4  5  6  7  8  9  10 | 55  5,5 |
| 5  5  5  5  5 | 25  5 |
| 0  15  150  1500 | 1665  416,25 |
| 0  2147483647 | 2147483647  1073741823,5 |
| 0  0  0  0  0 | 0  0 |
| 1073741823  1073741823  1 | 2147483647  715827882,333333 |

|  |  |
| --- | --- |
| **Задача 2.** ***Редица цели числа в обратен ред.*** | |
| **Условие**  Напишете програма, която прочита N цели числа от конзолата и ги отпечатва в обратен ред. Използвайте класа **Stack<int>**. | |
| **Описание на входа**  На първия ред се въвежда числото N-броят на числата, които ще бъдат прочетени от конзолата. На следващите N-реда се въвежда по едно цяло число. | |
| **Описание на изхода**  Изходът трябва да отпечата въведените числа в обратен ред. Разделени със  „ , “ и подравнени надясно с 5 позиции. При невалидни входни данни програмата отпечатва „Invalid input datа“. При N=0 програмата не отпечатва нищо. | |
| **Анализ на задачата**  За решаването на задачата се използва структура от данни Stack<int>, в която последователно се добавят числата. При отпечване на екрана числата се изваждат едно по едно от стека. | |
| **Решение (сорс код)** | |
| using System; using System.Collections.Generic;  namespace P2.ReverseSequence {  class Program  {  static void Main(string[] args)  {  string lengthAsString = Console.ReadLine();  int sequenceLength;  if (int.TryParse(lengthAsString, out sequenceLength) == false)  {  Console.WriteLine("Invalid input data!");  return;  }  Stack<int> sequence = new Stack<int>();  for (int i = 0; i < sequenceLength; i++)  {  string currentNumberAsString = Console.ReadLine();  int currentNumber;  if (int.TryParse(currentNumberAsString, out currentNumber) == false)  {  Console.WriteLine("Invalid input data!");  return;  }  sequence.Push(currentNumber);  }  if (sequenceLength > 0)  {  while (sequence.Count > 1)  {  Console.Write("{0,5},", sequence.Pop());  }  Console.WriteLine("{0,5}", sequence.Pop());  }  }  } } | |
| **Тестове**  Интересните случаи за тестване са следните:   * 0 елемента – резултатът е празен ред * Валидни входни данни * Препълване на int типа – изходът трябва да отпечатва „Invalid input data“ * Oтрицателно число на първия ред от програмата – изходът трябва да отпечата „Invalid input data“ * невалидни входни данни - изходът трябва да отпечатва „Invalid input data“ | |
| **Вход** | **Изход** |
| 5  1  2  3  4  5 | 5, 4, 3, 2, 1 |
| **Вход** | **Изход** |
| 6  5  4  3  2  1  0 | 0, 1, 2, 3, 4, 5 |
| **Вход** | **Изход** |
| 0 |  |
| **Вход** | **Изход** |
| -1 | Invalid input data |
| **Вход** | **Изход** |
| 1  1001 | 1001 |
| **Вход** | **Изход** |
| 2  101  201 | 201, 101 |
| **Вход** | **Изход** |
| 8  5456456  44565  323423  124124  214124  234214  241242  333333 | 333333,241242,234214,214124,124124,323423,44565,5456456 |
| **Вход** | **Изход** |
| 2.5 | Invalid input data |
| **Вход** | **Изход** |
| 1  9999999999999999999999 | Invalid input data |
| **Вход** | **Изход** |
| 2  123.45678  8765.4321 | Invalid input data |
| **Вход** | **Изход** |
| asf | Invalid input data |

|  |  |
| --- | --- |
| Задача 3. *Сортиране на поредица от числа.* | |
| **Условие**  Напишете програма, която прочита от конзолата поредица от цели положителни числа, поредицата спира когато се въведе празен ред, и ги сортира възходящо. | |
| **Описание на входа**  Входните данни се въвеждат поредица от цели числа, всяко на отделен ред. При срещане на празен ред програмата прекратява въвеждането. | |
| **Описание на изхода**  Изходът ще отпечатва подредена въведената поредица от числа на един ред, разделени с интервал | |
| **Анализ на задачата**  Основния момент тук е, че не знаем колко дълга ще е поредицата, затова не можем да ползваме обикновен масив. За целта използваме списък **List<int>**, който е удобен за работа с динамични масиви.  За самото въвеждане използваме безкаен цикъл **while(true)**. След въвеждането на всеки ред се проверява въведената информация. Ако въведеното е цяло число, то се добавя в лист-а. Ако не е цяло число, прескачаме. Ако реда е празен(не е въведено нищо) се прекъсва безкрайния цикъл.  За сортирането ще използваме **Linq** метода **Sort()**, след което с един **foreach** отпечатваме редицата. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  namespace SortIntSequence  {  class Numbers  {  static void Main(string[] args)  {  List<int> sequence = new List<int>();  string line;  while (true)  {  line = Console.ReadLine();  int number;  if (int.TryParse(line, out number))  {  sequence.Add(number);  }  else if (line == "")  {  break;  }  }  sequence.Sort();  foreach (int num in sequence)  {  Console.Write(num + " ");  }  Console.WriteLine();  }  }  } | |
| **Тестове**  Тестовете за задачата са следните:   * нормална произволна поредица положителни числа; * положителни и отрицателни числа и 0(нула); * равни числа; * дробни числа; * само неправилни символи; * празен вход; * тест за натоварване- 10 000 000 числа, произволно генерирани с **Random** класа(отчита се време); * worst case- числата са в намаляващ ред(отчита се време); * best case- числата са вече подредени(отчита се време); * има 2 цели числа на един ред; * числата са в шестнайдесетична форма; * пред числото има 0(Пр. 0013); * числото се въвежда с предшестващи или следващи интервали, табулации; * числото е извън размера на **int;** * числа и непарвилни символи(на отделни редове); * поредицата има числа, повтарящи се два или повече пъти; | |
| **Вход** | **Изход** |
| 7  5  2  11  40  24  18 | 2 5 7 11 18 24 40 |
| **Вход** | **Изход** |
| -6  11  29  0  1  -15  -80 | -80 -15 -6 0 1 11 29 |
| **Вход** | **Изход** |
| 42  42  42  42 | 42 42 42 42 |
| **Вход** | **Изход** |
| 1.5  2.6  1999.999 | (празен) |
| **Вход** | **Изход** |
| I  am  Goshko  the handsome  forever a bacherlor | (празен) |
| **Вход** | **Изход** |
| (празен) | (празен) |
| **Вход** | **Изход** |
| (произволни 10 000 000 числа) | (сортирането трае 3,5 сек.;  Извеждането на екрана е значително бавно, но правилно) |
| **Вход** | **Изход** |
| **For** цикъл довавя 10 000 000 намаляващи числа. | (сортирането трае 0,5 сек.) |
| **Вход** | **Изход** |
| **For** цикъл довавя 10 000 000 нарастващи числа. | (сортирането трае 0,5 сек.) |
| **Вход** | **Изход** |
| 1  2  3 4  5 | 1 2 5 |
| **Вход** | **Изход** |
| 4А2  11 | 11 |
| **Вход** | **Изход** |
| 01  05  02  09  10  000052 | 1 2 5 9 10 52 |
| **Вход** | **Изход** |
| 12  23  16  1 | 1 12 16 23 |
| **Вход** | **Изход** |
| 999999999999999999999999999999  99999999999999999999999999999999  99999999999999  5  6 | 5 6 |
| **Вход** | **Изход** |
| Imam  4  novi  3-ki  za  2  kurs | 2 4 |
| **Вход** | **Изход** |
| 1  2  2  9  2  16  9  2  11  9  20 | 1 2 2 2 2 9 9 9 11 16 20 |

|  |  |  |
| --- | --- | --- |
| Задача 4*. Най-дълга последователност от еднакви числа* | | |
| **Условие**  Напишете метод, който намира най-дългата поредица от равни числа в даден **List<int>** и връща като резултат нов **List<int>** със тази поредица. Напишете програма, която проверява дали този метод работи коректно. | | |
| **Описание на входа**  Входните данни се въвеждат като поредица от числа, на всеки ред по един елемент. | | |
| **Описание на изхода**  Изходът се извежда като поредица от числа на един ред с интервали между тях. | | |
| **Анализ на задачата**  Въвеждането на числа се прекратява, когато програма срещне празен ред. Списъкът от входни елементи се подава на метода **FindLongestSubsequence**.  В тялото на метода с помощта на един цикъл **for()** сканираме елементите в подадения списък, като ако текущият елемент е равен по стойност на предходния, увеличаваме с 1 стойността на променливата **currentCount**. Ако текущата стойност на тази променлива е по- голяма от стойността на променливата **MaxCount**, то на **MaxCount** присвояваме стойността на **CurrentCount**.  С помощта на променливата **CurrentIndex** запазваме стойността на текущият индекс. В случай, че на **MaxCount** се присвои **CurrentCount**, то на променливата **bestIndex** се присвоява стойността на **currentIndex**.  В края на метода използваме втори цикъл **for()**, с помощта на който редицата от еднакви елементи се прибавя към списъка **longestSubsequence**.  В главната програма, с помощта на цикъл **for()**, се обхожда резултатния списък и се принтира всеки един от елементите му на конзолата. | | |
| **Решение (сорс код)** | | |
| using System;  using System.Collections.Generic;    class SameNumbers  {      static void Main()      {          List<int> list = new List<int>();          string line;          try          {              do              {                  line = Console.ReadLine();                  if (line != "")                  {                      list.Add(int.Parse(line));                  }              } while (line != "");              List<int> sequence = FindLongestSubsequence(list);              Console.WriteLine("The longest subsequence is: ");              foreach (int num in sequence)              {                  Console.Write("{0,3}", num);              }              Console.WriteLine();          }          catch (OverflowException exception)          {              Console.WriteLine(exception.Message);          }          catch (ArgumentNullException exception)          {              Console.WriteLine(exception.Message);          }          catch (FormatException exception)          {              Console.WriteLine(exception.Message);          }      }        static List<int> FindLongestSubsequence(List<int> list)      {          List<int> longestSubsequence = new List<int>();          int currentIndex = 0;          int bestIndex = int.MinValue;          int currentCount = 1;          int maxCount = 0;            for (int i = 1; i < list.Count; i++)          {              if (list[i] == list[currentIndex])              {                  currentCount++;              }              else              {                  if (currentCount > maxCount)                  {                      maxCount = currentCount;                      bestIndex = currentIndex;                  }                    currentIndex = i;                  currentCount = 1;              }              if (i == list.Count - 1)              {                  if (currentCount > maxCount)                  {                      maxCount = currentCount;                      bestIndex = currentIndex;                  }                }          }          for (int i = 0; i < maxCount; i++)          {              int tempNumber = list[i + bestIndex];              longestSubsequence.Add(tempNumber);          }          return longestSubsequence;      }  } | | |
| **Тестове**  Интересните случаи за тестване са следните:   * Списък, в който всички елементи са различни – тогава се отпечатва единствено първият елемент от списъка * Списък с две еднакво-дълги последователни редици – трябва да се изведе първата от тях * Списък от един елемент – тогава този елемент трябва да е търсеният резултат * Празен списък (нула елемента) – резултатът е празна редица * Списък, състоящ се само от еднакви елементи – търсената редица е първоначалният списък | | |
| **Вход** | **Изход** | |
| 5 4 3 2 1 | 5 | |
| **Вход** | **Изход** | |
| 4 3 3 2 2 1 | 3 3 | |
| **Вход** | **Изход** | |
| 1 | 1 | |
| **Вход** | **Изход** | |
| (празен масив) | (празен масив) | |
| **Вход** | **Изход** | |
| 1 1 1 1 1 | 1 1 1 1 1 | |
| **Вход** | **Изход** | |
| 2 2 2 3 3 | 2 2 2 | |
| **Вход** | **Изход** | |
| 3 3 3 3 3 3 3 3 4 4 4 4 | 3 3 3 3 3 3 3 3 | |
| **Вход** | **Изход** | |
| 4 4 4 4 1 2 3 4 | 4 4 4 4 | |
| **Вход** | **Изход** | |
| 1 1 1 | 1 1 1 | |
| **Вход** | **Изход** | |
| 5 6 8 8 | 8 | |
| **Задача 5. *Извличане на неотрицателни числа от редица.*** | | |
| **Условие**  Напишете програма, която премахва всички отрицателни числа от дадена редица.  Пример: array = {19, -10, 12, -6, -3, 34, -2, 5} 🡪 {19, 12, 34, 5} | | |
| **Описание на входа**  Всички числа трябва да бъдат прочетени от конзолата, като всяко едно от тях ще бъде на нов ред. Поредицата свършва при въвеждане на празен ред (празен низ). Числата ще бъдат в границите на типа **double**. | | |
| **Описание на изхода**  Изходните данните трябва да се изведат на стандартния изход (конзолата).  При празна поредица на единствения ред на стандарния изход да се извежда съобщение:  „Sequence contains no elements“.  При непразна поредица:  На конзолата да се изведат всички числа в реда, в който са постъпили. Всяко число да се изведе само на нов ред. | | |
| **Анализ на задачата**  За решаването на задачата е използвана структората от данни **List<T>.** В един цикъл **while**, който спира при въвеждане на празен ред се чете ред по ред от конзолата, като всеки ред се парсва в число и се добавя към листа. След края на цикъла чрез един **foreach** цикъл се проверява дали числата са отрицателни и ако не са се изписват на конзолата чрез системния метод: **Console.WriteLine(string s)**. | | |
| **Решение (сорс код)** | | |
| using System;  using System.Collections.Generic;    class PrintNonNegative  {      static void Main(string[] args)      {          List<int> numbers = new List<int>();          string line = Console.ReadLine();          while (line != "")          {              int number = int.Parse(line);              numbers.Add(number);              line = Console.ReadLine();          }          if (numbers.Count == 0)          {              Console.WriteLine("Sequence contains no elements");          }          else          {              foreach (double number in numbers)              {                  if (number>=0)                  {                      Console.WriteLine(number);                  }              }          }      }  } | | |
| **Тестове**  Интересните случаи за тестване са следните:   * Поредица от само едно число. * Празна поредица. * Поредица с не цели числа. * Поредица само с отрицателни числа. * Поредица само с положителни числа. * Поредица включваща 0. * Поредица от еднакви числа. * Поредица включваща максималното възможно число. | | |
| 0 | | 0 |
| 5,5 | | 5,5 |
| 2147483647 | | 2147483647 |
| 10  10  10 | | 10  10  10 |
| 1  2  3  4,5  5  6  7  8  9  10 | | 1  2  3  4,5  5  6  7  8  9  10 |
|  | | Sequence contains no elements |
| -1  -2  -3  -4  -2147483647 | | (празен изход) |
| -321  0  32  -456  0  0  0  -3  1 | | 0  32  0  0  0  1 |
| 0,5  -0,5  2,513413  -2,513413  -23  23  0,0001  -0,00000001 | | 0,5  2,513413  23  0,0001 |
| 231431  513  513  642  -531  753  ,642  413  -52  -,52 | | 231431  513  513  642  753  0,642  413 |

|  |  |
| --- | --- |
| **Задача 6*. Числа срещани четен брой пъти.*** | |
| **Условие**  Напишете програма, която при дадена редица изтрива всички числа, които се срещат нечетен брой пъти. | |
| **Описание на входа**  На първият ред програмата прочита числото N – дължината на редицата. Дължината на редицата ще бъде винаги число между 0 и 10000. На следващите N реда програмата прочита по 1 цяло число – n-тия по ред член на редицата. Членовете на редицата трябва да се проверяват дали са от тип int. | |
| **Описание на изхода**  На изхода програмата отпечатва тези елементи от редицата, който се срещат четен брой пъти, сортирани във възходящ ред. Ако няма такива елементи, програмата отпечатва „No elements found“. При невалидни входни данни програмата отпечатва „Invalid input data“.  Пример:  Вход: Изход:  5 1  1 2  1  2  2  2 | |
| **Анализ на задачата**  Програмата прочита входните данни в масив и след това сортира масива във възходящ ред. Обхождаме последователно всички числа от масива и гледаме за всяко число броят на срещанията му. Записваме резулата в списък и го отпечатваме на екрана, ако списъкът е празен отпечатваме „No elements found“. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections;  using System.Collections.Generic;    class OddNumbers  {      static void Main(string[] args)      {          string lengthAsString = Console.ReadLine();          int sequenceLength;          if (int.TryParse(lengthAsString, out sequenceLength) == false)          {              Console.WriteLine("Invalid data!");              return;          }          if (sequenceLength < 0)          {              Console.WriteLine("Invalid data!");          }          int[] sequence = new int[sequenceLength];          for (int i = 0; i < sequenceLength; i++)          {              string currentNumberAsString = Console.ReadLine();              int currentNumber;              if (int.TryParse(currentNumberAsString, out currentNumber) == false)              {                  Console.WriteLine("Invalid data!");                  return;              }              sequence[i] = currentNumber;          }          List<int> outputSequence = RemoveElements(sequence);          PrintSequence(outputSequence);      }        private static List<int> RemoveElements(int[] sequence)      {          List<int> resultSequence = new List<int>();          Array.Sort(sequence);          int numberOcurances = 1;          for (int i = 0; i < sequence.Length - 1; i++)          {              if (sequence[i] == sequence[i + 1])              {                  numberOcurances++;              }              else if ((numberOcurances % 2) == 0)              {                  resultSequence.Add(sequence[i]);                  numberOcurances = 1;              }          }          return resultSequence;      }        private static void PrintSequence(IEnumerable sequence)      {          IEnumerator i = sequence.GetEnumerator();          while (i.MoveNext())          {              Console.WriteLine(i.Current);          }      }  } | |
| **Тестове**  Интересните случаи за тестване са следните:   * 0 елемента – резултатът е “No elements found” * Редицата не съдържа елементи, които се срещат четен брой пъти – изходът трябва да отпечата “No elements found” * Валидни входни данни – “Invalid input data” * Препълване на int типа – изходът трябва да отпечатва „Invalid input data“ * Oтрицателно число на първия ред от програмата – изходът трябва да отпечата „Invalid input data“ * невалидни входни данни - изходът трябва да отпечатва „Invalid input data“ | |
| **Вход**  5  1  1  1  1  2 | **Изход**  1 |
| **Вход**  8  1  1  1  2  2  2  3  3 | **Изход**  3 |
| **Вход**  6  1  1  2  2  2  3 | **Изход**  1 |
| **Вход**  10  1  1  1  1  1  1  2  1  1  1 | **Изход**  1 |
| **Вход**  5  99999999999999999  99999999999999999  99999999999999999  99999999999999999  99999999999999999 | **Изход**  Invalid input data |
| **Вход**  1  15.3 | **Изход**  Invalid input data |
| **Вход**  0 | **Изход**  No elements found |
| **Вход**  1  1 | **Изход**  No elements found |
| **Вход**  10  1  1  1  2  2  2  3  3  3  4 | **Изход**  No elements found |
| **Вход**  5  1000  2000  3000  4000  5000 | **Изход**  No elements found |
| **Вход**  15  1  1  2  2  10  10  8  8  -10  -10  -22222  -22222  2222222  2222222  3333333 | **Изход**  -22222  -10  1  2  8  10  2222222 |
| **Вход**  -1 | **Изход**  Invalid input data |

|  |  |
| --- | --- |
| Задача 7. *Срещане на числа в масив.* | |
| **Условие**  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}  2 => 2 пъти  3 => 4 пъти  4 => 3 пъти | |
| **Описание на входа**  Програмата няма вход, но има предварително зададени входни данни. | |
| **Описание на изхода**  На изхода се извеждат подредени по големина числата, срещнати в масива. Всяко число се изписва на нов ред и срещу него, броя на срещанията му в масива. | |
| **Анализ на задачата**  Понеже числата, съдържащи се в масива са от 0 до 1000, можем да създадем втори масив с размер 1000, в който ще записваме броя на срещания на елементите в масива. Когато обхождаме 1-я масив, използваме числата му като индекс за 2-я и увеличаваме стойноста на тази позиция във втория с 1. Т.е., ако в 1-я масив срещнем числото 3, увеличаваме стойността във 2-я масив на позиция 3 с 1.  Накрая отпечатваме индексите и числата от 2-я масив, за които стоиността на индекса не е 0; | |
| **Решение (сорс код)** | |
| using System;  using System.Text;  namespace Occurences  {  class Program  {  static void Main(string[] args)  {  int[] occurense;  int[] numbers = new int[] { 3, 4, 4, 2, 3, 3, 4, 3, 2 };  occurense = FindOccurence(numbers);  StringBuilder output = new StringBuilder();  for (int i = 0; i < occurense.Length; i++)  {  if (occurense[i] != 0)  {  output.Append(i + " -> " + occurense[i] + " times\n");  }  }  Console.Write(output);  }  static int[] FindOccurence(int[] arr)  {  int[] occurenseCount = new int[1001];  for (int i = 0; i < arr.Length; i++)  {  occurenseCount[arr[i]]++;  }  return occurenseCount;  }  }  } | |
| **Тестове**   * пирмерния масив * празен масив * масив без повторения * масив с много повторения * масив, в който има 2-те гранични стойности 0 и 1000 | |
| **Вход** | **Изход** |
| { 3, 4, 4, 2, 3, 3, 4, 3, 2 } | 2 -> 2 times  3 -> 4 times  4 -> 3 times |
| **Вход** | **Изход** |
| { } | (празен) |
| **Вход** | **Изход** |
| { 1, 2, 3, 4, 5 } | 1 -> 1 times  2 -> 1 times  3 -> 1 times  4 -> 1 times  5 -> 1 times |
| **Вход** | **Изход** |
| { 1, 2, 2, 2, 2, 2, 2, 2, 2 } | 1 -> 1 times  2 -> 8 times |
| **Вход** | **Изход** |
| { 0, 50, 1000} | 0 -> 1 times  50 -> 1 times  1000 -> 1 times |
| **Вход** | **Изход** |
| { 10, 60, 313, 700} | 10 -> 1 times  60 -> 1 times  313 -> 1 times  700 -> 1 times |
| **Вход** | **Изход** |
| { 1, 1, 1, 1, 1, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,8 ,8, 8, 9, 9, 9, 9, 1, 1, 1, 1, 8, 8,8, 9, 8, 8, 1, 1, 1, 1, 8} | 1 -> 13 times  8 -> 19 times  9 -> 5 times |
| **Вход** | **Изход** |
| {0,0,0,0,0,0,0,0,0} | 0 -> 9 times |
| **Вход** | **Изход** |
| {1000,1000,1000,1000} | 1000 -> 4 times |
| **Вход** | **Изход** |
| {5,5,5,5,5,5,5,5,5,5,5,5,5,5,5  ,5,5,5,5,5,1000} | 5 -> 20 times  1000 -> 1 times |

|  |  |
| --- | --- |
| Задача 8. Мажорант на масив от N елемента | |
| **Условие**  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!”.  Пример: {2, 2, 3, 3, 2, 3, 4, 3, 3} -> 3 | |
| **Описание на входа**  Входните данни се въвеждат като поредица от числа, на всеки ред по един елемент. | |
| **Описание на изхода**  Изходът представлява съобщение, което ни съобщава кое число е мажоранта на масива. Ако такова число не съществува, на конзолата се извежда съобщение „The majorant does not exist!“ | |
| **Анализ на задачата**  Използваме списък, в който ще съхраняваме стойностите на елементите които въвеждаме от конзолата и го сортираме. С помощта на два вложени цикъла **for()** сравняваме съседните едно на друго числа. Докато имаме равенство между съседните стойности, увеличаваме променливата **count** с единица и запазваме индекса на елемента, който последен е бил равен на своя съсед.  Ако променливата **count** е по- голяма или равна на дължината на списъка делена на две плюс едно, то не са ни необходими повече проверки, тъй като е изпълнено условието за мажорант на поредица от елементи. Извеждаме съобщение, че мажорантът е открит заедно със съответната му стойност и прекратяваме изпълнението на програмата. В случай, че това условие не се изпълни, чрез втори оператор **if()** правим проверка дали броячът на външният цикъл е по- голям или равен на дължината на списъка делена на две плюс едно. Ако това е вярно, на конзолата се извежда съобщение, че не съществува мажорант и програмата се преустановява. | |
| **Решение (сорс код)** | |
| using System;  class Majorant  {  static void Main()  {  List<int> inputNumbers = new List<int>();  string line;  do  {  line = Console.ReadLine();  if (line != "")  {  inputNumbers.Add(int.Parse(line));  }  } while (line != "");  inputNumbers.Sort();  int majorant = int.MinValue;  int count = 1;  int tempIndex = 0;  for (int i = 0; i < inputNumbers.Count - 1; i++)  {  majorant = inputNumbers[i];  for (int j = i + 1; j < inputNumbers.Count; j++)  {  tempIndex = j;  if (inputNumbers[i] == inputNumbers[j])  {  count++;  }  else  {  break;  }  }  if (count >= inputNumbers.Count / 2 + 1)  {  Console.WriteLine("The majorant is {0}.", majorant);  break;  }  if (i > inputNumbers.Count / 2 + 1)  {  Console.WriteLine("The majorant doesn`t exist!");  break;  }  count = 1;  i = tempIndex;  }  }  } | |
| **Тестове**  Интересните случаи за тестване са следните:   * Примерният вход от условието на задачата * Редица от елементи, която няма мажорант – няма мажорант * Редица от един елемент – няма мажорант * Празен вход – няма изход | |
| **Вход** | **Изход** |
| 2 2 3 3 2 3 4 3 3 | The majorant is 3. |
| **Вход** | **Изход** |
| 2 3 1 4 4 2 1 | The majorant does not exist! |
| **Вход** | **Изход** |
| 1 | 1 |
| **Вход** | **Изход** |
| Празен вход | Няма изход |
| **Вход** | **Изход** |
| 3 3 3 3 | The majorant is 3. |
| **Вход** | **Изход** |
| 2 2 2 3 3 2 | The majorant is 2. |
| **Вход** | **Изход** |
| 1 1 | The majorant is 1. |
| **Вход** | **Изход** |
| -5 -10 -15 -15 -15 -15 -15 -15 -1 1 10 5 | The majorant is -15. |
| **Вход** | **Изход** |
| 321 321 53 312 652 75 121 1 45 | The majorant is 321. |
| **Вход** | **Изход** |
| 2 31 21 13 5 521 43 542 82 21 21 | The majorant is 21. |

|  |  |
| --- | --- |
| **Задача 9. *Числа от дадена поредица.*** | |
| **Условие**  Дадена е следната поредица:  S1 = N;  S2 = S1 + 1;  S3 = 2\*S1 + 1;  S4 = S1 + 2;  S5 = S2 + 1;  S6 = 2\*S2 + 1;  S7 = S2 + 2;  ...  Използвайки класа Queue<T> напишете програма, която по дадено N отпечатва на конзолата първите 50 числа от тази поредица.  Пример: N=2 🡪 2, 3, 5, 4, 4, 7, 5, 6, 11, 7, 5, 9, 6, ... | |
| **Описание на входа**  Числото **N** трябва да бъде прочетено от първия и единсвен ред и ще бъде в интервала от  -1000000000 до 1000000000 включително. | |
| **Описание на изхода**  Изходните данните трябва да се изведат на стандартния изход (конзолата).  На конзолата да се изведат първите 50 числа от дадената поредица (започваща с **N**), като всчики числа трябва да бъдат разделени с запетая и интервал, а поредицата да завършва с точка.  Пример: „2, 3, 5, ........, 9.“ | |
| **Анализ на задачата**  За решаването на задачата е използвана структората от данни **Queue<T>** и брояч, който брои на кой от трите типа продължение на поредицата се намираме. В цикъл **for**, който се изпълнява 50 пъти пълним опашката с елементи според стойността на брояча, като на всяка 4та стойност на брояча изкарваме вече използвания елемент от опашката. На края в цикъл **while** докато има елементи в опашката ги извеждаме на екрана със системния метод: **Console.WriteLine(string s)**. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;    class CustomSequence  {      private static Queue<double> queue = new Queue<double>();      private static int counter = 1;      static void Main(string[] args)      {          Initialize();          EnqueueTheQueue();          PrintQueue();      }        private static void EnqueueTheQueue()      {          for (int i = 1; i < 50; i++, counter++)          {              DequeueAndPrintUssedElement(counter);              EnqueueElement();          }      }        private static void PrintQueue()      {          while (queue.Count > 0)          {              if (queue.Count != 1)              {                  Console.Write(queue.Dequeue() + ", ");              }              else              {                  Console.WriteLine(queue.Dequeue() + ".");              }          }      }        private static void EnqueueElement()      {          if (counter == 1)          {              queue.Enqueue(queue.Peek() + 1);          }          else if (counter == 2)          {              queue.Enqueue(queue.Peek() \* 2 + 1);          }          else if (counter == 3)          {              queue.Enqueue(queue.Peek() + 2);          }      }        private static void DequeueAndPrintUssedElement(int currElement)      {          if (counter == 4)          {              Console.Write(queue.Dequeue() + ", ");              counter = 1;          }      }      private static void Initialize()      {          queue.Enqueue(double.Parse(Console.ReadLine()));      }  } | |
| **Тестове**  Интересните случаи за тестване са следните:   * Началното число да е отрицателно. * Началното число да е примерното. * Началното число да не е цяло. * Началното число да е 0. * Началното число да е отрицателно, но поредицата да има и положителни числа. * Началното число да е максималното възможно. * Началното число да е минималното възможно. | |
| -100 | -100, -99, -199, -98, -98, -197, -97, -198, -397,  -197, -97, -195, -96, -97, -195, -96, -196, -393,  -195, -96, -193, -95, -197, -395, -196, -396,  -793, -395, -196, -393, -195, -96, -193, -95,  -194, -389, -193, -95, -191, -94, -96, -193, -95,  -194, -389, -193, -95, -191, -94, -195. |
| 0,5 | 0,5, 1,5, 2, 2,5, 2,5, 4, 3,5, 3, 5, 4, 3,5, 6, 4,5, 3,5, 6, 4,5, 5, 9, 6, 4,5, 8, 5,5, 4, 7, 5, 6, 11, 7, 5, 9, 6, 4,5, 8, 5,5, 7, 13, 8, 5,5, 10, 6,5, 4,5, 8, 5,5, 7, 13, 8, 5,5, 10, 6,5, 6. |
| 0 | 0, 1, 1, 2, 2, 3, 3, 2, 3, 3, 3, 5, 4, 3, 5, 4, 4, 7, 5, 4, 7, 5, 3, 5, 4, 4, 7, 5, 4, 7, 5, 4, 7, 5, 6, 11, 7, 5, 9, 6, 4, 7, 5, 6, 11, 7, 5, 9, 6, 5. |
| -1 | -1, 0, -1, 1, 1, 1, 2, 0, -1, 1, 2, 3, 3, 2, 3, 3, 2, 3, 3, 3, 5, 4, 1, 1, 2, 0, -1, 1, 2, 3, 3, 3, 5, 4, 4, 7, 5, 4, 7, 5, 3, 5, 4, 4, 7, 5, 4, 7, 5, 3. |
| 1000000000 | 1000000000, 1000000001, 2000000001, 1000000002, 1000000002, 2000000003, 1000000003, 2000000002, 4000000003, 2000000003, 1000000003, 2000000005, 1000000004, 1000000003, 2000000005, 1000000004, 2000000004, 4000000007, 2000000005, 1000000004, 2000000007, 1000000005, 2000000003, 4000000005, 2000000004, 4000000004, 8000000007, 4000000005, 2000000004, 4000000007, 2000000005, 1000000004, 2000000007, 1000000005, 2000000006, 4000000011, 2000000007, 1000000005, 2000000009, 1000000006, 1000000004, 2000000007, 1000000005, 2000000006, 4000000011, 2000000007, 1000000005, 2000000009, 1000000006, 2000000005. |
| -1000000000 | -1000000000, -999999999, -1999999999,  -999999998, -999999998, -1999999997,  -999999997, -1999999998, -3999999997,  -1999999997, -999999997, -1999999995,  -999999996, -999999997, -1999999995,  -999999996, -1999999996, -3999999993,  -1999999995, -999999996, -1999999993,  -999999995, -1999999997, -3999999995,  -1999999996, -3999999996, -7999999993,  -3999999995, -1999999996, -3999999993,  -1999999995, -999999996, -1999999993,  -999999995, -1999999994, -3999999989,  -1999999993, -999999995, -1999999991,  -999999994, -999999996, -1999999993,  -999999995, -1999999994, -3999999989,  -1999999993, -999999995, -1999999991,  -999999994, -1999999995. |
| 2 | 2, 3, 5, 4, 4, 7, 5, 6, 11, 7, 5, 9, 6, 5, 9, 6, 8, 15, 9, 6, 11, 7, 7, 13, 8, 12, 23, 13, 8, 15, 9, 6, 11, 7, 10, 19, 11, 7, 13, 8, 6, 11, 7, 10, 19, 11, 7, 13, 8, 9. |
| 43143121 | 43143121, 43143122, 86286243, 43143123, 43143123, 86286245, 43143124, 86286244, 172572487, 86286245, 43143124, 86286247, 43143125, 43143124, 86286247, 43143125, 86286246, 172572491, 86286247, 43143125, 86286249, 43143126, 86286245, 172572489, 86286246, 172572488, 345144975, 172572489, 86286246, 172572491, 86286247, 43143125, 86286249, 43143126, 86286248, 172572495, 86286249, 43143126, 86286251, 43143127, 43143125, 86286249, 43143126, 86286248, 172572495, 86286249, 43143126, 86286251, 43143127, 86286247. |
| 0,001 | 0,001, 1,001, 1,002, 2,001, 2,001, 3,002, 3,001, 2,002, 3,004, 3,002, 3,001, 5,002, 4,001, 3,001, 5,002, 4,001, 4,002, 7,004, 5,002, 4,001, 7,002, 5,001, 3,002, 5,004, 4,002, 4,004, 7,008, 5,004, 4,002, 7,004, 5,002, 4,001, 7,002, 5,001, 6,002, 11,004, 7,002, 5,001, 9,002, 6,001, 4,001, 7,002, 5,001, 6,002, 11,004, 7,002, 5,001, 9,002, 6,001, 5,002. |
| -2,34 | -2,34, -1,34, -3,68, -0,34, -0,34, -1,68, 0,66,  -2,68, -6,36, -1,68, 0,66, 0,32, 1,66, 0,66, 0,32, 1,66, -0,68, -2,36, 0,32, 1,66, 2,32, 2,66, -1,68,  -4,36, -0,68, -5,36, -11,72, -4,36, -0,68, -2,36, 0,32, 1,66, 2,32, 2,66, 1,32, 1,64, 2,32, 2,66, 4,32, 3,66, 1,66, 2,32, 2,66, 1,32, 1,64, 2,32, 2,66, 4,32, 3,66, 0,32. |

|  |  |
| --- | --- |
| **Задача 10. *Най-кратката поредица операции*** | |
| **Условие**  Дадени са числа N и M и следните операции:  N = N+1  N = N+2  N = N\*2  Напишете програма, която намира най-кратката поредица от посочените операции, която започва от N и завършва в M. Използвайте опашка.  Пример: N = 5, M = 16  Поредицата е: 5 ->7 ->8 ->16 | |
| **Описание на входа**  На първият ред програмата прочита цислото N – началото на поредицата. На вторият ред програмта причита цислото M – краят на поредицата. | |
| **Описание на изхода**  Програмата отпечатва най-кратката поредица от посочените операции, която започва от N и завършва в M, като всяко число е подравнено с 4 позиции надясно. Ако входните данни са невалидни (напр. Някое от числата е отрицателно или M < N) програмата отпечатва „Invalid input data“. | |
| **Анализ на задачата**  За решаването на задачата се използва структура от данни опашка. В опашката се слага първо числото N, след това елементите започват да се изваждат на нива. За всеки изваден елемент от опашката добавяме три нови елемента -  ако числото, което сме извадили е X, вкарваме X \* 2, X + 2 и X + 1 .За всеки елемент в опашката пазим пътя, по който сме стигнали до него. За да се оптимизира решението на задачата запомняме елементите, през които вече сме минали. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;    class Sequence  {      static void Main(string[] args)      {          int N = int.Parse(Console.ReadLine());          int M = int.Parse(Console.ReadLine());          if (N < 0 || M < 0 || M < N)          {              Console.WriteLine("Invalid input data");              return;          }          Queue<Node> operations = new Queue<Node>();          operations.Enqueue(new Node(null, N));          HashSet<int> found = new HashSet<int>();          while (operations.Count > 0)          {              Node node = operations.Dequeue();              int current = node.Value;              if (current < M)              {                  if (found.Contains(current + 2) == false)                  {                      operations.Enqueue(new Node(node, current + 2));                      found.Add(current + 2);                  }                  if (found.Contains(current + 1) == false)                  {                      operations.Enqueue(new Node(node, current + 1));                      found.Add(current + 1);                  }                  if (found.Contains(current \* 2) == false)                  {                      operations.Enqueue(new Node(node, current \* 2));                      found.Add(current \* 2);                  }              }              else if (current == M)              {                  node.Print();                  Console.WriteLine();              }          }      }  }    class Node  {      private Node prev;      private int value;        public int Value      {          get { return this.value; }          private set { this.value = value; }      }        public Node(Node prev, int value)      {          this.prev = prev;          this.Value = value;      }        public void Print()      {          if (prev != null)          {              prev.Print();          }          Console.Write("{0,4}", Value);      }  } | |
| **Тестове**  Интересните случаи за тестване са следните:   * Отрицателно число на входа - “Invalid input data” * N = M – програмата отпечатва 1 число * Примерният тест * M < N – изходът трябва да отпечатва „Invalid input data“ * Дълга редица – тест за време | |
| **Вход**  10  15 | **Изход**  10 12 14 15 |
| **Вход**  5  16 | **Изход**  5 7 8 16 |
| **Вход**  8  4 | **Изход**  Invalid input data |
| **Вход**  1  210 | **Изход**  1 3 6 12 13 26 52 104 105 210 |
| **Вход**  1  999 | **Изход**  1 3 5 7 14 15 30 31 62 124 248 249 498 499 998 999 |
| **Вход**  -999  1 | **Изход**  Invalid input data |
| **Вход**  -999  -5 | **Изход**  Invalid input data |
| **Вход**  255  256 | **Изход**  255 256 |
| **Вход**  0  0 | **Изход**  0 |
| **Вход**  3  333 | **Изход**  3 5 10 20 40 41 82 83 166 332 333 |
| **Вход**  210  0 | **Изход**  Invalid input data |

|  |
| --- |
| Задача 11*. Двойно свързан динамичен списък.* |
| **Условие**  Реализирайте структурата двойно свързан динамичен списък – списък, чиито елементи имат указател, както към следващия така и към предхождащия го елемент. Реализирайте операциите добавяне, премахване и търсене на елемент, добавяне на елемент на определено място (индекс), извличане на елемент по индекс и метод, който връща масив с елементите на списъка. |
| **Описание на входа**  (няма) |
| **Описание на изхода**  (няма) |
| **Анализ на задачата**  Създаваме аналогия на системния клас **LinkedList<T>**. Създаваме клас, в който ще имаме всичките методи на динамичния списък, заедно с конструктора му, първия и последния му елемент и свойство **Count.** В отделен клас ще е самия възел на динамичния списък.  Възела ще има публични поле **Value,** и 2 указателя към други възли- предишния и следващия, които за жалост също ще са публични заради избраната структура. Възела има само един конструктор, използващ като параметър стойноста му.  Самия Свързан списък се състои от публично св-во **Count**(read-only, private set), и 2 указателя към възли, които са 1-я и последния в списъка. Конструктора е без параметри и инициализира 3-те променливи.  Функцията за добавяне на последен елемент към списъка прави проверка дали не са зададени стойности в списъка. Ако не са, то последния и 1-я елемент са един и същ. Ако вече има списък, то за новосъздадения възел се правят съответните настройки на указателите(за списъка също) с което на практика е добавен, след което се променя **Last** да сочи към него и се увеличава **Count**. Аналогично е за добавяне на 1-и елемент, но там работим с **First**  Функцията за изтриване на елемент работи със неговата стойност и премахва 1-я елемент със същата стойност. За проверка за съвпадение използваме **CompareTo** ф-ята, като предварително е известно, че елементите от списъка имплементират **IComparable.** Тя обхожда списъка, започвайки от 1-я му елемент.Намери ли съвпадение, започва пренасочване на указателите, като има специални проверки дали елемента не е първия или последния. След това само остава да намалим **Count**  Ф-ята за намиране на елемент работи по сходен начин, но с по-малко проверки- тук само се обхожда списъка, почвайки от 1-я елемент, и ако се намери съвпадение, се връща указател към възела. Ако се стигне края на списъка без открито съвпадение, се връща **null**  Функцията за вмъкване на определено място първо прави 2 проверки. Ако индекса е извън размера на списъка, то връща съответното изключение. Ако индекса е последния, то управлението се прехвърля към ф-та за добавяне на края. Ако и това не е случая, то елемента се добавя на мястото на елемента, чийто индекс съответства. Това се прави с преминаване по списъка, като отделна променлива брой поредния индекс. Добавянето вече е сходно със добавянето на елемент на 1-ва позиция. **Count** се увеличава и ф-та връща управление.  Ф-та за намиране на елемент по индекс работи подобно на ф-та за вмъкване що се отнася до намиране на позицията, само дето като я намери връща елемента който и съответства.  Ф-та за въщане на списък просто създава масив с размерност **Count** и обхожда списъка, като на съответния индекс задава съответния елемент от списъка. Накрая връща новия ссисък.  Ф-та за изчистване на списъка на практика просто го преинициализира и е един вид излишна. |
| **Решение (сорс код)** |
| using System;    class MyLinkedList<T> where T : IComparable  {      public static MyLinkedListNode<T> First { get; private set; }      public static MyLinkedListNode<T> Last { get; private set; }      public int Count { get; private set; }        public MyLinkedList()      {          First = null;          Last = null;          Count = 0;      }        public void AddLast(T value)      {          if (Last == null)          {              First = Last = new MyLinkedListNode<T>(value);          }          else          {              Last.Next = new MyLinkedListNode<T>(value);              Last.Next.Previous = Last;              Last = Last.Next;          }          Count++;      }        public void AddFirst(T value)      {          if (First == null)          {              First = Last = new MyLinkedListNode<T>(value);          }          else          {              First.Previous = new MyLinkedListNode<T>(value);              First.Previous.Next = First;              First = First.Previous;          }          Count++;      }        public void Remove(T value)      {          MyLinkedListNode<T> current = First;            while (current != null)          {              if (current.Value.CompareTo(value) == 0)              {                  if (current.Previous != null)                  {                      current.Previous.Next = current.Next;                  }                  else                  {                      First = current.Next;                  }                    if (current.Next != null)                  {                      current.Next.Previous = current.Previous;                  }                  else                  {                      Last = current.Previous;                  }                    current = null;                  Count--;                  break;              }                current = current.Next;          }      }        public MyLinkedListNode<T> Find(T value)      {          MyLinkedListNode<T> current = First;            while (current != null)          {              if (current.Value.CompareTo(value) == 0)              {                  return current;              }                current = current.Next;          }            return null;      }        public void InsertAt(T insertValue, int index)      {          if (index > Count || index < 0)          {              throw new IndexOutOfRangeException();          }          else if (index == Count)          {              this.AddLast(insertValue);          }          else          {              int position = 0;              MyLinkedListNode<T> insertNode = new MyLinkedListNode<T>(insertValue);              MyLinkedListNode<T> current = First;                while (current != null)              {                  if (position == index)                  {                      insertNode.Previous = current.Previous;                      insertNode.Next = current;                        if (current.Previous != null)                      {                          current.Previous.Next = insertNode;                      }                      else                      {                          First = insertNode;                      }                        current.Previous = insertNode;                      Count++;                      break;                  }                  else                  {                      current = current.Next;                      position++;                  }              }            }      }        public MyLinkedListNode<T> ElementAt(int index)      {          MyLinkedListNode<T> current = First;          int position = 0;            if (index < 0 || index >= Count)          {              throw new IndexOutOfRangeException();          }          else          {              while (current != null)              {                  if (position == index)                  {                      return current;                  }                    current = current.Next;                  position++;              }          }            return null;      }        public T[] ToArray()      {          T[] arr = new T[Count];          MyLinkedListNode<T> current = First;            for (int i = 0; i < Count; i++)          {              arr[i] = current.Value;              current = current.Next;          }            return arr;      }        public void Clear()      {          First = null;          Last = null;          Count = 0;      }  }  class MyLinkedListNode<T>  {      public T Value { get; private set; }      public MyLinkedListNode<T> Next { get; internal set; }      public MyLinkedListNode<T> Previous { get; internal set; }        public MyLinkedListNode(T value)      {          this.Value = value;      }  } |
| **Тестове**  Тестовете са представени като поредица от команди в отделен сорс код. |
| **Тестове (сорс код)** |
| using System;   class LinkedListTest {     static void Main(string[] args)     {         MyLinkedList<int> linkedList = new MyLinkedList<int>();         Console.WriteLine("Linked list initialized.");         Console.WriteLine("Count: " + linkedList.Count);           Console.WriteLine("Add first test");         for (int i = 0; i < 10; i++)         {             linkedList.AddFirst(i);         }         Print(linkedList);         linkedList.Clear();         Console.WriteLine("Add last test");         for (int i = 0; i < 10; i++)          {             linkedList.AddLast(i);         }         Print(linkedList);         Console.WriteLine();         Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");         Console.WriteLine("Remove tests");         Console.WriteLine(" -remove existing element");         linkedList.Remove(4);         Print(linkedList);         Console.WriteLine(" -remove nonexisting element");         linkedList.Remove(1112);         Print(linkedList);         Console.WriteLine(" -remove first element");         linkedList.Remove(0);         Print(linkedList);         Console.WriteLine(" -remove last element");         linkedList.Remove(9);         Print(linkedList);         Console.WriteLine(" -remove element in an empty list");         linkedList.Clear();         linkedList.Remove(2);         Print(linkedList);         Console.WriteLine();         for (int i = 0; i < 10; i++)         {             linkedList.AddLast(i);         }          Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");         Console.WriteLine("Find tests");         MyLinkedListNode<int> found;         Console.WriteLine(" -find existing element");         found = linkedList.Find(2);         if (found != null)         {             Console.WriteLine(" Found: " + found.Value);         }         Console.WriteLine(" -find nonexisting element");         found = linkedList.Find(90);         if (found != null)         {             Console.WriteLine(" Found: " + found.Value);         }         Console.WriteLine(" -find first/last element");         found = linkedList.Find(0);         if (found != null)         {             Console.WriteLine(" Found: " + found.Value);         }         found = linkedList.Find(9);         if (found != null)         {             Console.WriteLine(" Found: " + found.Value);         }         Console.WriteLine();         Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");         Console.WriteLine("Insert tests");         Console.WriteLine(" -insert in middle");         linkedList.InsertAt(50, 3);         Print(linkedList);         linkedList.Remove(50);         Console.WriteLine(" -insert at begining/ending");         linkedList.InsertAt(-50, 0);         linkedList.InsertAt(50, 11);         Print(linkedList);         Console.WriteLine(" -insert out of range");         try         {             linkedList.InsertAt(10000, 2054);         }         catch (IndexOutOfRangeException e)         {             Console.WriteLine("Thrown exception: " + e.GetType());         }         Console.WriteLine();         Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");         Console.WriteLine("ElementAt tests");         Console.WriteLine(" -element at middle index");         MyLinkedListNode<int> nodeAt = linkedList.ElementAt(4);         Console.WriteLine("Element [4]: " + nodeAt.Value);         Console.WriteLine(" -element at first/last index");         nodeAt = linkedList.ElementAt(0);         Console.WriteLine("Element [0]: " + nodeAt.Value);         nodeAt = linkedList.ElementAt(linkedList.Count - 1);         Console.WriteLine("Element [{0}]: {1}",   linkedList.Count - 1, nodeAt.Value);         Console.WriteLine(" -element in out of range index");         try         {             nodeAt = linkedList.ElementAt(5000);         }         catch (IndexOutOfRangeException e)         {             Console.WriteLine("Thrown exception: " + e.GetType());         }         Console.WriteLine();     }      static void Print(MyLinkedList<int> list)     {         int[] listArr = list.ToArray();         foreach (var item in listArr)         {             Console.Write(item + " ");         }         Console.WriteLine();     } } |

|  |
| --- |
| Задача 12. *Динамична реализация на стек.* |
| **Условие**  Създайте клас **DynamicStack** представляващ динамична реализация на стек. Добавете методи за необходимите операции. |
| **Описание на входа**  (няма входни данни) |
| **Описание на изхода**  (няма изходни данни) |
| **Анализ на задачата**  Класът за възел **Node** ще има едно публично поле **Element**, както и един указател **Next**, който ще сочи към следващият елемент в стека. На лице е един конструктор, който използва стойността на **Element**.  Класът за динамичен стек **DynamicStack** ще съдържа всичките методи, които са необходими за манипулиране на данните, а също така и конструктора му, първият му елемент, и свойството **Count.**  Функцията **Push()** добавя елемент най- отгоре в стека. Ако стекът е празен, исканият елемент директно бива добавен. Ако стекът не е празен, тогава запазваме стойността на най- горният елемент в една текуща променлива, добавяме исканият от нас елемент и променяме настройките на указателя. Всеки път, когато се добавя нов елемент **Count** се увеличава с 1.  Функцията **Pop()** премахва елементът, който се намира най- отгоре в стека и променя указателя да сочи към следващият елемент в стека. Всеки път, когато се премахне елемент от стека **Count** се намалява с 1.  Функцията **Clear()** премахва всички елементи, намиращи се в стека. От своя страна функцията **Peek()** връща, като стойност елемента, който се намира най- отгоре в стека.  На функцията **Contains()** и се подава, като аргумент елемента, които търсим. Ако този елемент съществува в рамките на стека, то като резултат се извежда търсената стойност. Ако този елемент не съществува, резултатът е null.  С помощта на функцията **ToArray()**, като резултат се връща масив, който съдържа елементите от стека. |
| **Решение (сорс код)** |
| class Node<T>  {  public T Element { get; set; }  public Node<T> Next { get; set; }  public Node(T element)  {  this.Element = element;  }  }  class DynamicStack<T> where T: IComparable  {  public static Node<T> First { get; private set; }  public int Count { get; private set; }  public DynamicStack()  {  First = null;  Count = 0;  }  public void Push(T element)  {  Node<T> newNode = new Node<T>(element);  if (First == null)  {  First = newNode;  }  else  {  Node<T> temp = First;  First = newNode;  First.Next = temp;  }  Count++;  }  public void Pop()  {  Node<T> elementToPop = First;  First = First.Next;  Count--;  }  public void Clear()  {  First = null;  Count = 0;  }  public Node<T> Peek()  {  if (First != null)  {  return First;  }  else  {  return null;  }  }  public Node<T> Contains(T element)  {  Node<T> current = First;  while (current != null)  {  if (current.Element.CompareTo(element) == 0)  {  return current;  }  current = current.Next;  }  return current;  }  public T[] ToArray()  {  T[] myArray = new T[Count];  Node<T> current = First;  for (int i = 0; i < Count; i++)  {  myArray[i] = current.Element;  current = current.Next;  }  return myArray;  }  } |
| class Test  {  static void Main()  {  DynamicStack<int> dynamicStackExample = new DynamicStack<int>();  Console.WriteLine("The dynamic stack is initialized.");  Console.WriteLine("Count: " + dynamicStackExample.Count);  Console.Write("---->Push elements test: ");  for (int i = 0; i < 5; i++)  {  dynamicStackExample.Push(i + 1);  }  Print(dynamicStackExample);  Console.WriteLine();  Console.WriteLine("---->Pop elements test: ");  dynamicStackExample.Pop();  dynamicStackExample.Pop();  Console.Write("The stack after pop: ");  Print(dynamicStackExample);  Console.WriteLine("----> Clear test: ");  dynamicStackExample.Clear();  Print(dynamicStackExample);  for (int i = 0; i < 5; i++)  {  dynamicStackExample.Push(i + 1);  }  Console.WriteLine("----> Peek test: ");  Node<int> peekElement = dynamicStackExample.Peek();  if (peekElement != null)  {  Console.WriteLine("The top element: " + peekElement.Element);  }  else  {  Console.WriteLine("The stack is empty!");  }  Console.WriteLine("----> Contains element:");  Console.Write("If the element exists: ");  Node<int> containsElement = dynamicStackExample.Contains(5);  Console.Write(containsElement.Element);  Console.WriteLine();  Console.Write("If the element doesn`t exist: ");  containsElement = dynamicStackExample.Contains(12);  Console.Write(containsElement);  Console.WriteLine();  }  public static void Print(DynamicStack<int> dynamicStack)  {  int[] myArray = dynamicStack.ToArray();  foreach (var item in myArray)  {  Console.Write("{0} ", item);  }  Console.WriteLine();  }  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Задача 13. *Дек.*** | | | |
| **Условие**  Реализирайте структурата от данни "дек". Това е специфична списъчна структура, подобна на стек и опашка, позволя­ваща елементи да бъдат добавяни и премахвани от двата й края. Нека освен това, елемент поставен от едната страна да може да бъде премахнат само от същата. Реализирайте операции за премах­ване добавяне и изчистване на дека. При невалидна операция подавайте подходящо изключение. | | | |
| **Описание на входа**  На първия ред от конзолата ще бъдат числата, които да добавите в началото на вашия Дек в реда, в който са постъпили разделени с един интервал (ако има такива).  На втория ред от конзолата ще бъдат числата, които да добавите в края на вашия Дек в реда, в който са постъпили разделени с един интервал (ако има такива).  На третия ред ще бъде цяло число указващо броя пъти, които да извиката функцията за триене от началото на Дека (от 0 до 100).  На четвъртия ред ще бъде цяло число указващо броя пъти, които да извикате функцията за триене от края на Дека (от 0 до 100).  Числата, които ще постъпват за добавяне в дека ще бъдат в границите на типа **double**. | | | |
| **Описание на изхода**  Изходните данните трябва да се изведат на стандартния изход (конзолата).  На конзолата трябва да изведете числата върнати от методите за триене от началото и края на  дека, чиито брой сте получили като вход. Ако няма такова число в началото или края (дека е  празен или числата в него не са постъпили от страната, която искате да ги изкарате) изведете  съпбщение: „Invalid command“. Всяко число или съобщение да бъде изписано на отделен ред. | | | |
| **Анализ на задачата**  За решаването на задачата е използвана структората от данни **LinkedList<T>**. С два свързани списъка се представят елементите, които са добавени в началото и края на дека, като при извикване на метод за триене се обръщаме към съответния списък, като така избягваме възможността да извадим елемент не от тази страна, от която е влязъл. За самите функции са използвани функциите на свързаните списъци за триене и добавяне на елемент в тяхното начало или край. | | | |
| **Решение (сорс код)** | | | |
| using System.Collections;  using System.Collections.Generic;  using System;    class Deck<T>  {      private LinkedList<T> startDeck;      private LinkedList<T> endDeck;        public Deck()      {          this.startDeck = new LinkedList<T>();          this.endDeck = new LinkedList<T>();      }        public void AddToStart(T element)      {          this.startDeck.AddFirst(element);      }      public void AddToEnd(T element)      {          this.endDeck.AddLast(element);      }      public T RemoveFromStart()      {          T value = this.startDeck.First.Value;          this.startDeck.RemoveFirst();          return value;      }      public T RemoveFromEnd()      {          T value = this.endDeck.Last.Value;          this.endDeck.RemoveLast();          return value;      }      public void Clear()      {          this.startDeck.Clear();          this.endDeck.Clear();      }  }  class DeckTester  {      static void Main()      {          char[] separators = new char[]{' '};          Deck<double> deck = new Deck<double>();          string line;          line = Console.ReadLine();          string[] elementsToStarStr = line.Split(separators,StringSplitOptions.RemoveEmptyEntries);          if (elementsToStarStr.Length>0)          {              for (int i = 0; i < elementsToStarStr.Length; i++)              {                  deck.AddToStart(double.Parse(elementsToStarStr[i]));              }          }          line = Console.ReadLine();          string[] elementsToEndStr = line.Split(separators, StringSplitOptions.RemoveEmptyEntries);          if (elementsToEndStr.Length>0)          {              for (int i = 0; i < elementsToEndStr.Length; i++)              {                  deck.AddToEnd(double.Parse(elementsToEndStr[i]));              }          }          line = Console.ReadLine();          int elementsToRemoveFromStart = int.Parse(line);          line = Console.ReadLine();          int elementsToRemoveFromEnd = int.Parse(line);          for (int i = 0; i < elementsToRemoveFromStart; i++)          {              try              {                  Console.WriteLine(deck.RemoveFromStart());              }              catch (NullReferenceException)              {                  Console.WriteLine("Invalid command");                  continue;              }          }            for (int i = 0; i < elementsToRemoveFromEnd; i++)          {              try              {                  Console.WriteLine(deck.RemoveFromEnd());              }              catch (NullReferenceException)              {                  Console.WriteLine("Invalid command");                  continue;              }          }      }  } | | | |
| **Тестове**  Интересните случаи за тестване са следните:   * Добавяме елементи само от началото и се опитваме да трием и от края. * Добавяме елементи само в края и се опитваме да трием в началото. * Опитваме се да трием повече от един несъществуващи елемента. * Добавяме голям брой елементи и в началото и в края и следим за тяхната подредба. * Подаваме 0 елемента за триене. | | | |
| 1 2 3  4 5  3  2 | 3  2  1  5  4 | | |
| 1 2 3  4 5  3  3 | 3  2  1  5  4  Invalid command | | |
| 2 3  3  2 | Invalid command  Invalid command  Invalid command  3  2 | | |
| 1 2 3  3  2 | 3  2  1  Invalid command  Invalid command | | |
| 1,5 1,6  321 5136 641  5 | 1,6  1,5  Invalid command  Invalid command  Invalid command  641 | | |
| 5  5 | Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command | | |
| 12,34567 11,32131 42135 531 21  542 431 123 543 214214 0  3  5 | 21  531  42135  0  214214  543  123  431 | | |
| 231 321 5311 61,31 ,531 213,642 213,6427426 2  32,312513 5,4513666 31 531 61451 132312,65  10  10 | 2  213,6427426  213,642  0,531  61,31  5311  321  231  Invalid command  Invalid command  132312,65  61451  531  31  5,4513666  32,312513  Invalid command  Invalid command  Invalid command  Invalid command | | |
| 123,213413 51 14 13 5  214 1231 4531 531 15 16 351  0  0 |  | | |
| 1 2 3 4  5 6 7 8  100  100 | 4  3  2  1  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  8  7  6  5  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command  Invalid command | | |
| **Задача 14. *Зациклена опашка.*** | | | |
| **Условие**  Реализирайте структурата "зациклена опашка" с масив, който при нужда удвоява размера си. Имплементирайте необходимите методи за добавяне към опашката, извличане на елемента, който е наред и поглеждане на елемента, който е наред, без да го премахвате от опашката. При невалидна операция подавайте подходящо изключение. | | | |
| **Описание на входа**  На първия ред от входа програмата прочита числото N. На следващите N реда програмата чете по една команда и я изпълнява. Командите могат да бъдат push, pop и peek. Командата push има един аргумент – цяло число, като командата е разделена от числото с един “ “(интервал). Командите pop и peek не приемат аргументи. | | | |
| **Описание на изхода**  На изхода програмата отпечатва резултата от изпълнението на командите. Ако командата е:  “**push** *number*” – (където *number* е цяло число), програмата отпечатва “*number* pushed*”*  “**pop**” *–* програмата отпечатва *“number” (*където *number* e първото влязло число в опашката). Ако опашката е празна програмата трябва да изписва “Queue is empty”  *“***peek***” -* програмата отпечатва *“number” - (*където *number* e първото влязло число в опашката*).* ). Ако опашката е празна програмата трябва да изписва “Queue is empty” | | | |
| **Анализ на задачата**  За реализацията на структурата от данни се използва масив. Когато стигнем до последния индекс добавяме следващия елемент в началото на масива. За точното пресмятане на индексите използваме остатък от делене на дължината на масива. Ако има нужда да се преоразмерява масива, големината става двойно по-голяма и елементите се преподреждат, така че да могат да се добавят и изваждат елементи без да се губи първоначалната наредба. | | | |
| **Решение (сорс код)** | | | |
| using System;  using System.IO;  class CyclicQueue<T>  {  public const int INITIAL\_CAPACITY = 1000;  private T [] q;  private int headIndex;  private int tailIndex;  private int capacity;  private int count;    public int Count  {  get { return count; }  }    public CyclicQueue(int capacity = INITIAL\_CAPACITY)  {  this.headIndex = 0;  this.tailIndex = 0;  this.capacity = capacity;  this.count = 0;  q = new T[capacity];  }  public void Push(T element)  {  if (this.Count == this.capacity)  {  capacity = capacity \* 2;  T[] bigger = new T[capacity];  int head = headIndex;  for (int i = 0; i < this.Count; i++)  {  bigger[i] = q[head];  if (head == q.Length - 1)  {  head = -1;  }  head++;  }  this.tailIndex = this.Count - 1;  this.headIndex = 0;  q = bigger;  }  this.count++;  this.tailIndex++;  this.tailIndex = tailIndex % capacity;  q[tailIndex] = element;  }  public T Pop()  {  if (count == 0)  {  Console.WriteLine("Queue is empty!");  }  T element = q[headIndex];  q[headIndex] = default(T);  if (headIndex == q.Length - 1)  {  headIndex = 0;  }  else  {  headIndex++;  }  this.count--;  return element;  }  public T Peek()  {  if (count == 0)  {  Console.WriteLine("Queue is empty!");  }  T element = q[headIndex];  return element;  }  } | | | |
| **Тестове**  Интересните случаи за тестване са следните:   * Добавяме елементи в опашката после ги изтриваме напълно и добавяме нови елементи * Случайте, в които има нужда опашката да се уголемява * Случайте, в които опашката е празна и се опитваме да извадим елемент | | | |
| push 5  push 10  push 15  push 20  push 30  push 40 | | pushed 5  pushed 10  pushed 15  pushed 20  pushed 30  pushed 40 | |
| push 10  pop  push 15  push 20  push 30  push 40  peek | | pushed 10  10  pushed 15  pushed 20  pushed 30  pushed 40  40 | |
| push 10  pop  push 15  push 20  push 30  push 40  peek  pop  peek | | pushed 10  10  pushed 15  pushed 20  pushed 30  pushed 40  40  40  30 | |
| push 9  pop  push 9  pop  push 30  pop  peek | | pushed 9  9  pushed 9  9  pushed 30  30  Queue is empty | |
| push 11  push 12  pop  pop  peek  push 6 | | pushed 11  pushed 12  11  12  Queue is empty  pushed 6 | |
| push 8  push 7  pop  pop  peek  push 4 | | pushed 4  pushed 4  4  4  Queue is empty  pushed 4 | |
| peek | | Queue is empty | |
| pop | | Queue is empty | |
| push 3 | | pushed 3 | |
| push 12  pop  push 3  peek | | pushed 12  12  Pushed 3  3 | |

|  |  |
| --- | --- |
| **Задача 15. *Сортиране на числа в динамичен свързан списък*.** | |
| **Условие**  Реализирайте сортиране на числа в динамичен свързан списък, без да използвате допълнителен масив. | |
| **Описание на входа**  Понеже в задачата се изисква само да се реализира сортирането, то ще използваме ф-я, която сортира създаден списък от числа. Този списък няма да се въвежда от потребителя, а ще е предварително готов и ще се подава като параметър на функцията. Понеже не е изрично казано че числата са цели, ще използваме списък от **double** елементи. | |
| **Описание на изхода**  При тестовете, на изхода ще се отпечатва сортирания списък, като всички числа от списъка ще са на един ред, отделени със интервал. | |
| **Анализ на задачата**  Най-лесния за имплементиране на решение е сортиране по метода на мехурчето. При него взимаме първия елемент от списъка и го сравняваме със всички следващи елементи. Когато намерим по-малък от него им разменяме местата и продължаваме да сравняваме разменения елемент(който е на първа позиция) с останалите. Така накрая най-малкия от всички елементи е в началото. Продължаваме със втория елемент по същия начин и т.н. Накрая се получава подреден списък с елементи.  Самото обхождане на списъка се прави по стандартния начин за обхождане на динамичен свързан списък. | |
| **Решение (сорс код)** | |
| private static void BubbleSort(LinkedList<double> array)  {  if (array == null)  {  return;  }  LinkedListNode<double> node = array.First;  while (node != array.Last)  {  for (LinkedListNode<double> afterNode = node.Next; afterNode != null; afterNode = afterNode.Next)  {  if (afterNode.Value < node.Value)  {  double nodeOldValue = node.Value;  node.Value = afterNode.Value;  afterNode.Value = nodeOldValue;  }  }  node = node.Next;  }  } | |
| **Тестове**  За тестове ще се подават коректни списъци, за които имаме:   * списък със произволни дробни числа * списък с цели и дробни числа * празен списък * списък с равни елементи * списък с подредени елементи * списък с намаляващи елементи * списък с много елементи(пр. 10 000) (отчита се време) * списък с крайните за **double** стойности * списък с положителни и отрицателни числа и нула * списък с един единствен елемент   Подчертавам, че тези списъци са **готови, а не въведени от козолата.** | |
| **Вход** | **Изход** |
| {16.86, 1.46, 13.33, 15.06, 10.38, 3.62} | 1.46 3.62 10.38 13.33 15.06 16.86 |
| **Вход** | **Изход** |
| { 16.86, 1.00, 13.33, 15.00, 10.38, 3.00 } | 1.00 3.00 10.38 13.33 15.00 16.86 |
| **Вход** | **Изход** |
| { } | (празно) |
| **Вход** | **Изход** |
| { 1.0, 2.0, 3.0, 4.0} | 1.00 2.00 3.00 4.00 |
| **Вход** | **Изход** |
| { 4.0, 3.0, 2.0, 1.0} | 1.00 2.00 3.00 4.00 |
| **Вход** | **Изход** |
| { 1.5, 1.5, 1.5, 0.2 } | 0.20 1.50 1.50 1.50 |
| **Вход** | **Изход** |
| (генерирани 10 000 подредени дроби) | (сортираха се за 1,18 сек.) |
| **Вход** | **Изход** |
| (генерирани 10 000 намаляващи дроби) | (сортираха се за 2,7 сек.) |
| **Вход** | **Изход** |
| { 4.0, double.MinValue, double.MaxValue, 1.0} | -1797..., 1.00, 4.00, 1797... |
| **Вход** | **Изход** |
| { 4.0, -2.0, 0.0, 1.0} | -2.00 0.00 1.00 4.00 |
| **Вход** | **Изход** |
| { 1.0 } | 1.00 |

|  |
| --- |
| Задача 16. *Обхождане на директории на твърдия диск* |
| **Условие**  Използвайки опашка реализирайте пълно обхождане на всички директории на твърдия ви диск и ги отпечатвайте на конзолата. Реализирайте алгоритъма „обхождане в ширина“ – Breadth- First- Search (BFS) – можете да намерите стотици статии за него в Интернет. |
| **Описание на входа**  (няма входни данни) |
| **Описание на изхода**  На конзолата се извеждат имената на директориите при обхождане на твърдият диск в ширина. |
| **Анализ на задачата**  За решаването на проблема се използва структурата от данни **Queue(T)**. Използва се алгоритъма „обхождане в широчина“, като самото обхождане започва от C:. Всички директории се добавят в опашката и отпечатваме посетените такива, докато опашката не остане празна. |
| **Решение (сорс код)** |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.IO;  namespace BFS  {  class TraverseDir  {  static void Main(string[] args)  {  Queue<DirectoryInfo> dirQueue = new Queue<DirectoryInfo>();  DirectoryInfo currDir = new DirectoryInfo("C:\\");  dirQueue.Enqueue(currDir);  while (dirQueue.Count > 0)  {  currDir = dirQueue.Dequeue();  Console.WriteLine(currDir.FullName);  try  {  foreach (DirectoryInfo directory in currDir.GetDirectories())  {  dirQueue.Enqueue(directory);  }  }  catch (UnauthorizedAccessException)  {  continue;  }  }  }  }  } |
| **Тестове**  Не съществува директория, която да е обща за всички потребители. Поради тази причина задачата няма как да бъде тествана. |

|  |
| --- |
| **Задача 17. *Обхождане в дълбочина.*** |
| **Условие**  Използвайки стек реализирайте пълно обхождане на всички дирек­тории на твърдия ви диск и ги отпечатвайте на конзолата. Реализи­райте алгоритъма "обхождане в дълбочина" – Depth-First-Search (DFS) – може да намерите стотици статии за него в Интернет. |
| **Описание на входа**  (няма) |
| **Описание на изхода**  Изведете имената на директориите при обхождането в дълбочина на твърдия ви диск. |
| **Анализ на задачата**  За решаването на задачата е използвана структората от данни **Stack(T)**, като започвайки от C: използвайки алгоритъма за обхождане в дълбочина (DFS) добавяме всички директории в стека и отпечатваме посетените докато стека не остане празен. |
| **Решение (сорс код)** |
| using System;  using System.Collections.Generic;  using System.IO;    class TraverseDirDFS  {      static void Main(string[] args)      {          Stack<DirectoryInfo> dirStack = new Stack<DirectoryInfo>();          DirectoryInfo currDir = new DirectoryInfo("C:\\");          dirStack.Push(currDir);          while (dirStack.Count > 0)          {              currDir = dirStack.Pop();              Console.WriteLine(currDir.FullName);              try              {                  foreach (DirectoryInfo directory in currDir.GetDirectories())                  {                      dirStack.Push(directory);                  }              }              catch (UnauthorizedAccessException)              {                  continue;              }          }      }  } |
| **Тестове**  Няма директория, която да е обща за всички потребители, поради това задачата няма как да бъде тествана. |

|  |  |
| --- | --- |
| **Задача 18. *Най-кратък път в лабиринт*** | |
| **Условие**  Даден е лабиринт с размери N x N. някои от клетките на лабиринта са празни (0) а други са запълнени (х). Можем да се движим от празна клетка до друга празна клетка, ако двете имат обща стена. При дадена начална позиция (\*) изчислете и попълнете лабиринта с минималната дължина от началната позиция до всяка друга. Ако някоя клетка не може да бъде достигната я попълнете с "u”. | |
| **Описание на входа**  На първият ред програмата прочита числото N – размера на лабиринта. N винаги ще бъде между 5 и 200. На следващите N реда програмата прочита N символа от тип char разделени със “ ”(интервал). Символите могат да бъдат “x”- запълнена клетка, “0” – празна клетка или “\*”- началната позиция. | |
| **Описание на изхода**  Програмата отпечатва на екрана разстоянието от началната клетка до всички други проходими клетки. Ако някоя клетка не може да бъде достигната програмата я маркира с “u”. Всички отпечатани клетки трябва да бъдат подравнени 3 позиции надясно.  Пример: | |
| **Анализ на задачата**  Първо прочитаме входа и намираме позицията на клетката, от която стартираме. Дефинираме структура „Cell“, в която пазим реда и колоната на клетката и разстоянието от нея до клетката, от която стартираме. Пускаме BFS от стартовата клетка. Във всяка непосетена съседна клетка на текущата клетка, записваме текущото разстояние + 1. След изпълнението на BFS проверяваме дали има проходими клетки (0), през които не сме преминали и ги маркираме като “u” (unreachable). | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  namespace P15.ShortestPath  {  class Program  {  static void Main(string[] args)  {  char[,] matrix = InitializeMatrixFromConsole();  PathFinder pathfinder = new PathFinder(matrix);  pathfinder.SolveWithBFS();  pathfinder.PrintMatrix();  }  private static char[,] InitializeMatrixFromConsole()  {  string sizeAsString = Console.ReadLine();  int size = int.Parse(sizeAsString);  char[,] matrix = new char[size,size];  for (int row = 0; row < size; row++)  {  string rowAsString = Console.ReadLine();  string[] entireRow = rowAsString.Split(' ');  for (int col = 0; col < entireRow.Length; col++)  {  char[] ch = entireRow[col].ToCharArray();  matrix[row, col] = ch.First();  }  }  return matrix;  }  }  class PathFinder  {  private char[,] matrix;  private Cell startCell;  private const char PASSABLE = '0';  private const char NON\_PASSABLE = 'x';  private const char STARTCELL\_IMAGE = '\*';  private const char UNREACHABLE = 'u';  public PathFinder(char[,] matrix)  {  this.matrix = new char[matrix.GetLength(0), matrix.GetLength(1)];  for (int row = 0; row < matrix.GetLength(0); row++)  {  for (int col = 0; col < matrix.GetLength(1); col++)  {  this.matrix[row, col] = matrix[row, col];  if (matrix[row, col] == STARTCELL\_IMAGE)  {  startCell = new Cell(row, col, 0);  }  }  }  }  private bool IsInMatrix(int row, int col)  {  bool isInMatrix = ((row < matrix.GetLength(0) && row >= 0) && (col < matrix.GetLength(1) && col >= 0));  return isInMatrix;  }  private bool IsPassable(int row, int col)  {  return (matrix[row, col] == PASSABLE);  }  private void MarkUnreachableCells()  {  for (int row = 0; row < matrix.GetLength(0); row++)  {  for (int col = 0; col < matrix.GetLength(1); col++)  {  if (matrix[row, col] == PASSABLE)  {  matrix[row, col] = UNREACHABLE;  }  }  }  matrix[startCell.Row, startCell.Col] = STARTCELL\_IMAGE;  }  public void SolveWithBFS()  {  Queue<Cell> visitedCells = new Queue<Cell>();  visitedCells.Enqueue(startCell);  while (visitedCells.Count > 0)  {  Cell cell = visitedCells.Dequeue();  int row = cell.Row;  int col = cell.Col;  int dist = cell.Distance;  matrix[row, col] = (char)(dist);  if(IsInMatrix(row + 1, col) && IsPassable(row + 1, col))  {  visitedCells.Enqueue(new Cell(row + 1, col, dist + 1));  }  if (IsInMatrix(row, col + 1) && IsPassable(row, col + 1))  {  visitedCells.Enqueue(new Cell(row, col + 1, dist + 1));  }  if (IsInMatrix(row - 1, col) && IsPassable(row - 1, col))  {  visitedCells.Enqueue(new Cell(row - 1, col, dist + 1));  }  if (IsInMatrix(row , col - 1) && IsPassable(row, col - 1))  {  visitedCells.Enqueue(new Cell(row , col - 1, dist + 1));  }  }  MarkUnreachableCells();  }  public void PrintMatrix()  {  for (int row = 0; row < matrix.GetLength(0); row++)  {  for (int col = 0; col < matrix.GetLength(1); col++)  {  if (matrix[row, col] == UNREACHABLE || matrix[row, col] == NON\_PASSABLE)  {  Console.Write("{0,3}",matrix[row, col]);  }  else  {  if (matrix[row, col] == STARTCELL\_IMAGE)  {  Console.Write("{0,3}",matrix[row,col]);  }  else  {  Console.Write("{0,3}", (int)matrix[row, col]);  }  }  }  Console.WriteLine();  }  }  }  class Cell  {  public int Row { set; get; }  public int Col { set; get; }  public int Distance { set; get; }    public Cell(int x, int y, int distance)  {  Row = x;  Col = y;  Distance = distance;  }  }  } | |
| **Тестове** | |
| Интересните случаи за тестване са следните:   * Примерният тест * Голяма матрица – тест за време | |
| **Вход**  6  0 0 0 x 0 x  0 x 0 x 0 x  0 \* x 0 x 0  0 x 0 0 0 0  0 0 0 x x 0  0 0 0 x 0 x | **Изход**  3 4 5 x u x  2 x 6 x u x  1 \* x 8 x 10  2 x 6 7 8 9  3 4 5 x x 10  4 5 6 x u x |
| **Вход**  4  0 0 0 0  0 0 0 0  0 0 0 0  0 0 0 \* | **Изход**  6 5 4 3  5 4 3 2  4 3 2 1  3 2 1 \* |
| **Вход**  8  0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0  \* 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 | **Изход**  6 7 8 9 10 11 12 13  5 6 7 8 9 10 11 12  4 5 6 7 8 9 10 11  3 4 5 6 7 8 9 10  2 3 4 5 6 7 8 9  1 2 3 4 5 6 7 8  \* 1 2 3 4 5 6 7  1 2 3 4 5 6 7 8 |
| **Вход**  15  0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x x x x 0 0 0 0 0 0 0  0 0 0 0 x 0 0 x 0 0 0 0 0 0 0  0 0 0 0 x 0 0 x 0 0 0 0 0 0 0  0 0 0 0 x \* 0 x 0 0 0 0 0 0 0  0 0 0 0 x x x x 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | **Изход**  u u u u x u u u u u u u u u u  u u u u u u u u u u u u u u u  u u u u u u u u u u u u u u u  u u u u x x x x u u u u u u u  u u u u x 2 3 x u u u u u u u  u u u u x 1 2 x u u u u u u u  u u u u x \* 1 x u u u u u u u  u u u u x x x x u u u u u u u  u u u u u u u u u u u u u u u  u u u u u u u u u u u u u u u  u u u u u u u u u u u u u u u  u u u u u u u u u u u u u u u  u u u u u u u u u u u u u u u  u u u u u u u u u u u u u u u  u u u u u u u u u u u u u u u |
| **Вход**  15  0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x x x x 0 0 0 0 0 0 0  0 0 0 0 x 0 0 x 0 0 0 0 0 0 0  0 0 0 0 x 0 0 x 0 0 0 0 0 0 0  0 0 0 0 x \* 0 x 0 0 0 0 0 0 0  0 0 0 0 x x 0 x 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x x x x x x x x 0 0 0  0 0 0 0 0 0 0 0 0 0 0 x x x x  0 0 0 0 0 0 0 x x x x x 0 0 0  0 0 0 0 0 0 0 x 0 0 0 x 0 0 0  0 0 0 0 0 0 0 x x x x x 0 0 0  0 0 0 0 0 0 0 0 0 0 0 x 0 0 0 | **Изход**  17 16 15 14 x 16 15 14 13 14 15 16 17 18 19  16 15 14 13 14 15 14 13 12 13 14 15 16 17 18  15 14 13 12 13 14 13 12 11 12 13 14 15 16 17  14 13 12 11 x x x x 10 11 12 13 14 15 16  13 12 11 10 x 2 3 x 9 10 11 12 13 14 15  12 11 10 9 x 1 2 x 8 9 10 11 12 13 14  11 10 9 8 x \* 1 x 7 8 9 10 11 12 13  10 9 8 7 x x 2 x 6 7 8 9 10 11 12  9 8 7 6 5 4 3 4 5 6 7 8 9 10 11  10 9 8 7 x x x x x x x x 10 11 12  11 10 9 8 9 10 11 12 13 14 15 x x x x  12 11 10 9 10 11 12 x x x x x u u u  13 12 11 10 11 12 13 x u u u x u u u  14 13 12 11 12 13 14 x x x x x u u u  15 14 13 12 13 14 15 16 17 18 19 x u u u |
| **Вход**  30  0 0 0 0 x 0 0 0 0 0 0 0 0 0 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 0 0 0 0 0 0 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 0 0 0 0 x x x x x x x 0 0 0 0 0 0 0 x x 0 x  0 0 0 0 x 0 0 0 0 0 0 0 0 0 0 0 0 0 0 x 0 0 0 0 0 0 0 0 x 0  0 0 0 0 x 0 0 0 0 0 0 0 0 x 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 0 0 0 0 0 0 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x x x x x x x x x x x x x x x x x x x 0 0 0 0 0 0 0  0 0 0 0 x 0 0 x x x x x x 0 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 0 0 0 0 0 0 0 0 0 0 0 x x x x x x x x x x x  0 0 0 0 x 0 0 0 0 0 0 0 x x x x x x x 0 0 0 0 0 0 0 0 0 0 x  0 0 0 0 x 0 0 0 0 0 0 0 0 0 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 0 0 0 0 0 0 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 x x x x x x 0 0 0 0 0 x 0 0 0 0 0 0 0 x x 0  0 0 0 0 x 0 0 0 x 0 0 0 0 x 0 0 0 0 0 x 0 0 0 0 0 0 0 0 x 0  0 0 0 0 x 0 0 0 x 0 0 0 0 x 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 x 0 0 0 0 x 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 x 0 0 0 0 x 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 x 0 0 0 0 x 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 x 0 0 0 0 x 0 0 0 0 0 x x x x x x x x x x x  0 0 0 0 x 0 0 0 x 0 0 0 0 x 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 x 0 0 0 0 x 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 x x x x x x 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 0 0 x 0 0 0 0 0 0 0 0 x 0 0 0 0 0 0 0 x x 0  0 0 0 0 x 0 0 0 0 0 x 0 0 0 0 0 \* 0 0 x 0 0 0 0 0 0 0 0 x 0  0 0 0 0 x 0 0 0 0 0 x 0 0 x x x x x x x x 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 0 0 x 0 0 0 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 0 0 0 0 0 x 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 0 0 0 0 0 0 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0  0 0 0 0 x 0 0 0 0 0 0 0 0 0 0 0 0 0 0 x x x x x x x x x x x  0 0 0 0 x 0 0 0 0 0 0 0 0 0 0 0 0 0 0 x 0 0 0 0 0 0 0 0 0 0 | **Изход**  u u u u x u u u u u u u u u u u u u u x u u u u u u u u u u  u u u u x u u u u u u u u u u u u u u x u u u u u u u u u u  u u u u x u u u u u u u x x x x x x x u u u u u u u x x u x  u u u u x u u u u u u u u u u u u u u x u u u u u u u u x u  u u u u x u u u u u u u u x u u u u u x u u u u u u u u u u  u u u u x u u u u u u u u u u u u u u x u u u u u u u u u u  u u u u x x x x x x x x x x x x x x x x x x x u u u u u u u  u u u u x 27 26 x x x x x x 23 24 25 26 27 28 x u u u u u u u u u u  u u u u x 26 25 24 23 22 21 20 21 22 23 24 25 26 27 x x x x x x x x x x x  u u u u x 25 24 23 22 21 20 19 x x x x x x x u u u u u u u u u u x  u u u u x 24 23 22 21 20 19 18 17 16 15 14 13 14 15 x u u u u u u u u u u  u u u u x 23 22 21 20 19 18 17 16 15 14 13 12 13 14 x u u u u u u u u u u  u u u u x 24 23 22 x x x x x x 13 12 11 12 13 x u u u u u u u x x u  u u u u x 25 24 23 x u u u u x 12 11 10 11 12 x u u u u u u u u x u  u u u u x 26 25 24 x u u u u x 11 10 9 10 11 x u u u u u u u u u u  u u u u x 25 24 23 x u u u u x 10 9 8 9 10 x u u u u u u u u u u  u u u u x 24 23 22 x u u u u x 9 8 7 8 9 x u u u u u u u u u u  u u u u x 23 22 21 x u u u u x 8 7 6 7 8 x u u u u u u u u u u  u u u u x 22 21 20 x u u u u x 7 6 5 6 7 x x x x x x x x x x x  u u u u x 21 20 19 x u u u u x 6 5 4 5 6 x u u u u u u u u u u  u u u u x 20 19 18 x u u u u x 5 4 3 4 5 x u u u u u u u u u u  u u u u x 19 18 17 x x x x x x 4 3 2 3 4 x u u u u u u u u u u  u u u u x 18 17 16 15 14 x 6 5 4 3 2 1 2 3 x u u u u u u u x x u  u u u u x 17 16 15 14 13 x 5 4 3 2 1 \* 1 2 x u u u u u u u u x u  u u u u x 16 15 14 13 12 x 6 5 x x x x x x x x u u u u u u u u u  u u u u x 15 14 13 12 11 x 7 6 7 8 9 10 11 12 x u u u u u u u u u u  u u u u x 14 13 12 11 10 9 8 7 x 9 10 11 12 13 x u u u u u u u u u u  u u u u x 15 14 13 12 11 10 9 8 9 10 11 12 13 14 x u u u u u u u u u u  u u u u x 16 15 14 13 12 11 10 9 10 11 12 13 14 15 x x x x x x x x x x x  u u u u x 17 16 15 14 13 12 11 10 11 12 13 14 15 16 x u u u u u u u u u u |
| **Вход**  5  0 x x x 0  0 x \* x 0  0 x x x 0  0 0 0 0 0  0 0 0 0 0 | **Изход**  u x x x u  u x \* x u  u x x x u  u u u u u  u u u u u |
| **Вход**  6  x x x x x x  x x x x x x  x x \* x x x  x x x x x x  x x x x x x  x x x x x x | **Изход**  x x x x x x  x x x x x x  x x \* x x x  x x x x x x  x x x x x x  x x x x x x |
| **Вход**  6  x x x x x x  x 0 0 0 0 0  x 0 0 \* x x  x x x x x x  0 0 0 0 0 0  0 0 0 0 0 0 | **Изход**  x x x x x x  x 3 2 1 2 3  x 2 1 \* x x  x x x x x x  u u u u u u  u u u u u u |
| **Вход**  3  0 0 0  0 \* 0  0 0 0 | **Изход**  2 1 2  1 \* 1  2 1 2 |
| **Вход**  1  \* | **Изход**  \* |