# Решения на задачите от глава 11 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

**Александър Михайлов Дженков** - **70\_ALDJ – [a.djenkov90@gmail.com](mailto:a.djenkov90@gmail.com), phone: 0883486223**

**Кристиан Димитров Димитров 223\_KRDI –** [**kristian.dimitrov.sz@gmail.com**](mailto:kristian.dimitrov.sz@gmail.com)**, p: 0883540544**

**Александър Владиславов Лазаров -253\_ALLA –** [**sass\_waz@abv.bg**](mailto:sass_waz@abv.bg) **phone: 0883478803**

**Здравко Йорданов Брънзов - 14\_ZDBR –** [**z\_branzov@abv.bg**](mailto:z_branzov@abv.bg) **, phone: 0895700739**

|  |  |
| --- | --- |
| **Задача 1. Leap Year** | |
| **Условие**  Напишете програма ,коят прочита от конзолата година и проверяява дали е високосна | |
| **Описание на входа**  Входните данни са едно число (година) между 0 и 9999 | |
| **Описание на изхода**  Изходът се извежда на един ред и гласи дали въведената година е високосна или не,и дали входът е коректен | |
| **Анализ на задачата**  Първо с IF оператор се проверява дали входът е положителен след което се прави проверка на дадената година с помоща на класа DateTime и функцията към него IsLeapYear ,която връща TRUE или FALSE . | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace LeapYearTask1  {  class LeapYearTask1  {  static void Main(string[] args)  {  int year = int.Parse(Console.ReadLine());  if (year > 0)  {  if (DateTime.IsLeapYear(year)) Console.WriteLine("{0} is leap year", year);  else Console.WriteLine("{0} is not a leap year", year);  }  else Console.WriteLine("Incorrect year");  }  }  } | |
| **Тестове**  Решението е тествано с номинални случаи,отрицателна стойност и няколко валидни години | |
| **Вход** | **Изход** |
| 0 | Incorrect year |
| **Вход** | **Изход** |
| -100 | Incorrect year |
| **Вход** | **Изход** |
| 4 | 4 is a leap year |
| **Вход** | **Изход** |
| 1990 | 1990 is not a leap year |
| **Вход** | **Изход** |
| 2056 | 2056 is a leap year |
| **Вход** | **Изход** |
| 9999999 | Incorrect year |
| **Вход** | **Изход** |
| 11 | 11 is not a leap year |
| **Вход** | **Изход** |
| 2012 | 2012 is a leap year |
| **Вход** | **Изход** |
| 9999 | 9999 is a leap year |
| **Вход** | **Изход** |
| 2011 | 2011 is not a leap year |
|  |  |
| **Задача 2. Random Numbers** | |
| **Условие**  Напишете програма, която генерира и принтира на конзолата 10 случайни числа в интервала [100,200]. | |
| **Описание на входа**  Няма вход ,тъй като се генерират десет произволни числа. | |
| **Описание на изхода**  Изходът извежда на конзолата десет случайни числа в интервала 100-200. | |
| **Анализ на задачата**  В задачата се ползва обект от класа Random на който се извиква функцията.Next; която връща цяло произволно число в интервала от 100 до 20 ,след което това число се принтира на конзолата и се извиква същата функция докато не стигне 10 принтирания. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace RandomNumbersTask2  {  class RandomNumbersTask2  {  static void Main(string[] args)  {  int randomNumber = 0;  int counter = 0;  Random rndGenerator = new Random();  do  {  counter++;  randomNumber = rndGenerator.Next(100, 201);  Console.WriteLine(randomNumber);  } while (counter != 10);  }  }  } | |
| **Тестове**  Тест от конзолата няма как да се направи, тъй като се взима автоматично произволно число в интервала 100-200 | |
| **Вход** | **Изход** |
|  |  |

|  |  |
| --- | --- |
| **Задача 3. Check Today** | |
| **Условие**  Напишете програма, която показва на конзолата кой ден от седмицата е днес | |
| **Описание на входа**  Няма вход ,тъй като той се взима автоматично от часовникът на компютъра | |
| **Описание на изхода**  Изходът извежда на конзолата името на денят кой е днес на английски | |
| **Анализ на задачата**  В задачата се ползва обект от класа DateTime на който се извиква функцията.Today; която връща цяло число от 1 до 7 ,след което това число с помоща на switch оператор се превежда в дума отговаряща на денят от седмицата | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace TodayTask3  {  class TodayTask3  {  static void Main(string[] args)  {  DateTime dateValue = DateTime.Today;  switch ((int)dateValue.DayOfWeek)  {  case 1: Console.WriteLine("Monday");  break;  case 2: Console.WriteLine("Tuesday");  break;  case 3: Console.WriteLine("Wednesday");  break;  case 4: Console.WriteLine("Thursday");  break;  case 5: Console.WriteLine("Friday");  break;  case 6: Console.WriteLine("Saturday");  break;  case 7: Console.WriteLine("Sunday");  break;  }  }  }  } | |
| **Тестове**  Тест от конзолата няма как да се направи тъй като денят се взима автоматично от часовникът на компютъра.Възможно е да се премести ръчно датата ,за да се установи промяна в резултата | |
| **Вход** | **Изход** |
| 1 | Monday |
| **Задача 4. Time Elapsed** | |
| **Условие**  Напишете програма, която извежда на стандартния изход броя на дните, часовете и минутите, които са изтекли от включването на компютъра до момента на изпълнението на програмата. | |
| **Описание на входа**  Няма вход ,тъй като не може да се предвиди колко време е работил компютъра | |
| **Описание на изхода**  Изходът извежда на конзолата дните, часовете и минутите от включването на компютъра | |
| **Анализ на задачата**  В задачата се ползва обект от класа Environment на който се извиква функцията .TickCount; която връща времето от включването на компютъра до изпълнението на програмата и се дели на минути, часове, и с условни оператори се извежда отговора на конзолата | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace TimeEplasedTask4  {  class TimeEplasedTask4  {  static void Main(string[] args)  {  decimal time = Environment.TickCount;  if (time / 60000 / 60 / 24 < 1) Console.WriteLine("0 days");  else  {  Console.WriteLine("{0} days", time / 60000 / 60 / 24);  time = time / 6000 / 60 / 24;  }  if (time / 60000 / 60 < 1) Console.WriteLine("0 hours");  else  {  Console.WriteLine("{0} hours", time / 60000 / 60);  time = time / 6000 / 60;  }  if (time < 1) Console.WriteLine("0 minutes");  else Console.WriteLine("{0} minutes", time);  }  }  } | |
| **Тестове**  Тест от конзолата няма как да се направи тъй като не може да се предвиди колко време е работил компютъра и да се изведат тестове | |
| **Вход** | **Изход** |
|  |  |

|  |  |
| --- | --- |
| **Задача 5.Hypotenuse By Two Sides** | |
| **Условие**  Напишете програма, която която по дадени два катета намира хипотенузата на правоъгален триъгалник.Реализирайте въвеждането на дължините на катетите от стандартния вход ,а за пресмятането на хипотенузата използвайте медоти на класа Math. | |
| **Описание на входа**  Въвеждат се три две последователно на отделен ред всяко. | |
| **Описание на изхода**  Изходът представлява един ред на който се печата дължината на хипотенузата | |
| **Анализ на задачата**  Присвояват се две стойности за 2-те страни на триъгалника които трябва да са положителни ,след което с помоща на методите Math.Sqrt(коренуване) и Math.Pow(степенуване) се пресмята хипотенузата по познатата формула | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace TriangleByTwoSidesTask5  {  class TriangleByTwoSidesTask5  {  static void Main(string[] args)  {  double sideA = int.Parse(Console.ReadLine());  double sideB = int.Parse(Console.ReadLine());  if (sideA > 0 && sideB > 0)  {  double sideC = Math.Sqrt(Math.Pow(sideA, 2) + Math.Pow(sideB, 2));  Console.WriteLine(sideC);  }  else Console.WriteLine("Incorect Input");  }  }  } | |
| **Тестове**  Решението е тествано с номинални случаи, въвеждане на отрицателни стойности,цели числа ,дробни числа | |
| **Вход** | **Изход** |
| -1  10 | Incorrect Input |
| **Вход** | **Изход** |
| -1000  -9999 | Incorrect Input |
| **Вход** | **Изход** |
| 1  1 | 1,4142135623731 |
| **Вход** | **Изход** |
| 12  6 | 1,4142135623731 |
| **Вход** | **Изход** |
| 3  5 | 5 |
| **Вход** | **Изход** |
| 100  123 | 158,521291945278 |
| **Вход** | **Изход** |
| 78  44 | 89,5544527089524 |
| **Вход** | **Изход** |
| 50000  0 | Incorrect Input |
| **Вход** | **Изход** |
| 43986724  342906734 | 345716444.663541 |
| **Вход** | **Изход** |
| 600353532  1563467347 | 1674769986.75045 |
|  |  |
| **Задача 6. Class Triangle** | |
| **Условие**  Напишете програма, която пресмята лице на триъгълник по:  А. дължините на трите му страни;  B. дължината на една от страните и височината към нея;  С. дължините на две от страните и ъгъла между тях в градуси. | |
| **Описание на входа**  Входа на програмата представлява:   * на първи ред избор на метод за смятане **(a,b or c)**, който не е case sensitive * на следващите редове в зависимост от избора на метод различен брой променливи от тип **double** всяка на отделен ред.   Входа винаги е правилен. | |
| **Описание на изхода**  Изхода е лицето на триъгълник, пресметнато по една от трите възможности. | |
| **Анализ на задачата**  За първата подточка се използва херонова теорема. За втората подточка се използва формулата S= (a\*ha)/2. За трето подусловие се използва формулата S=(a\*b\*sin(y))/2. За функцията синус е използван класа System.Math и метода .Sin();  За да опишем триъгълник използваме клас с необходимите променливи и три различни конструктора в зависимост от метода на смятане на лицето. Създаваме и метод за принтиране на лицето.  Написваме метод, който вече е извън класа на Триъгълника CalculateResult() чрез който приемаме избора на вход и смятаме спрямо входа през switch резултата и го изписваме. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace ClassTriangleTask6  {  class ClassTriangleTask6  {  class Triangle  {  double x, y, z, surface;  double altitude;  double angle;  public Triangle(double x , double altitude )  {  this.x = x;  this.altitude = altitude;  this.surface = (x \* altitude) / 2;  }  public Triangle(string x, string y, string z)  {  this.x = double.Parse(x);  this.y = double.Parse(y);  this.z = double.Parse(z);  this.surface = (this.x + this.y + this.z) / 2;  }  public Triangle(double x , double y , double angle)  {  this.x = x;  this.y = y;  this.angle = Math.Sin(angle);  this.surface = (this.angle \* x \* y) / 2;  }  public void printSurface()  {  Console.WriteLine(surface);  }  }  static void CalculateResult()  {  char choice = (char)Console.ReadLine()[0];  switch (choice)  {  case 'a':  case 'A':  {  Console.WriteLine("Input three sides:");  string sideA = Console.ReadLine();  string sideB = Console.ReadLine();  string sideC = Console.ReadLine();  Triangle obj2 = new Triangle(sideA, sideB, sideC);  obj2.printSurface();  }  break;  case 'b':  case 'B':  {  Console.WriteLine("Input a side and its altitude:");  double sideA = double.Parse(Console.ReadLine());  double altitudeA = double.Parse(Console.ReadLine());  Triangle obj1 = new Triangle(sideA, altitudeA);  obj1.printSurface();  }  break;  case 'c':  case 'C':  {  Console.WriteLine("Input two sides and angle (in degrees) between:");  double sideA = double.Parse(Console.ReadLine());  double sideB = double.Parse(Console.ReadLine());  double angleDegree = double.Parse(Console.ReadLine());  Triangle obj3 = new Triangle(sideA, sideB, angleDegree);  obj3.printSurface();  }  break;  }  }  static void Main(string[] args)  {  Console.WriteLine("Choose the way you want to calculate the triangle's surface");  Console.WriteLine("A) Three sides. \nB) Side and its altitude. \nC) Two sides and the angle (in degrees) between.");  CalculateResult();  }  }  } | |
| **Тестове**  Решението е тествано за всеки избор от менюто със съответните променливи. | |
| **Вход** | **Изход** |
| a  3  4  5 | 6 |
| **Вход** | **Изход** |
| B  7  3 | 10.5 |
| **Вход** | **Изход** |
| C  4  6  45 | 8.48528137423857 |
| **Вход** | **Изход** |
| A  2.5  3.3  4.5 | 5.15 |
| **Вход** | **Изход** |
| B  12.1  0.5 | 3.025 |
| **Вход** | **Изход** |
| C  3.6  7.5  45.9 | 9.69470501980305 |
| **Вход** | **Изход** |
| C  12  12  60 | 62.3538290724796 |
| **Вход** | **Изход** |
| A  100  100  100 | 150 |
| **Вход** | **Изход** |
| B  2  1 | 1 |
| **Вход** | **Изход** |
| A  1  1  1 | 1.5 |

|  |
| --- |
| **Задача 7.Namespace Chapter11** |
| **Условие**  Дефинирайте свое собствено пространство **Chapter11** и поставете в него двата класа **Cat** и **Sequence**, които използвахме в примерите на текущата тема. Направете още едно собствено пространство с име **Chapter11.Examples** и в него направете клас, който извиква класовете **Cat** и **Sequence**. |
| **Описание на входа**  В програмата липсва вход тъй като се изисква просто да се извикат и създадът успешно обекти от различни namespaces |
| **Описание на изхода**  Изход също липсва тъй като няма изискване да се извежда нещо от класовете на екрана |
| **Анализ на задачата**  Създаваме ново именовано пространство в което добавяме 2 нови класа Cat и Sequence след което в първото именовано пространство добавяме новото и създаваме 2 обекта е от тези класове които имат видимост благодарение на public идентификатора си |
| **Решение (сорс код)** |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using NamespaceTask7.Chapter11;  namespace NamespaceTask7.Chapter11  {  public class Cat  {  private string name;  private string color;  public Cat()  {  this.name = "Unnamed";  this.color = "grey";    }  public Cat(string name, string color)  {  this.name = name;  this.color = color;  }  public void SayMiau()  {  Console.WriteLine("Car {0} said :Miauuu",name);  }  }  }  namespace Chapter11.Examples  {  class Example  {  static void Main(string[] args)  {  Cat kitty = new Cat();  Sequence seq = new Sequence();  }  }  }    namespace NamespaceTask7.Chapter11  {  public class Sequence  {    private static int currenValue = 0;  public Sequence()  {  }  public static int NextValue()  {  currenValue++;  return currenValue;  }  }  } |

|  |  |
| --- | --- |
| **Задача 8. Say Miau** | |
| **Условие**  Напишете програма, която създава 10 обекта от тип Cat, дава им имена от вида CatN, където N е уникален пореден номер на обекта, и накрая извиква метода SayMiau() на всеки от тях. За реализацията използвайте вече дефинираното пространство Chapter11. | |
| **Описание на входа**  Използва се метода Next.Value(), съответно вход не може да се даде. | |
| **Описание на изхода**  На изхода излизат името на котката и „miau”. | |
| **Анализ на задачата**  Създава се масив с 10 елемента от тип Cat. Създават се в цикъл 10 обекта от тип Cat ( използва се конструктор с параметри) като ги присвоява на съответните елементи от масива. За пореден номер на обектите се използва метода Next.Value() от класа Sequence. Накрая отново в цикъл се изпълнява метода SayMiau() на всеки от елементите на масива. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using NamespaceTask7.Chapter11;  namespace CatsTask8  {    class CatsTask8  {  static void Main(string[] args)  {  for (int i = 0; i < 10; i++)  {  string name="Cat"+i;  Cat catObj = new Cat(name, "white");  catObj.SayMiau();  }  }  }  } |

|  |  |
| --- | --- |
| **Задача 9. Workdays List** | |
| **Условие**  Напишете програма, която пресмята броя работни дни между днешната дата и дадена друга дата след днешната (включително). Работните дни са всички дни без събота и неделя, които не са официални празници, като по изключение събота може да е работен ден, когато се отра­ботват почивни дни около празниците. Програмата трябва да пази списък от предварително зададени официални празници, както и списък от предварително зададени работни съботи. | |
| **Описание на входа**  Въвеждаме дата различна и по голяма от текущата във формата ГГГГ.М.Д | |
| **Описание на изхода**  На изхода излизат подредени дати в същият формат ,които не са празници и са работни съботи.Празниците и работните съботи са дефинирани в програмата в масив от дади | |
| **Анализ на задачата**  Дефинира се масив от Работни съботи под формата на дати и празници ,след което се дефинира клас Workday в който се взима обект от днешната дата с DateTime класа и обект от датата която сме въвели след което се прави обхождане на всички дати от текущата до зададената и се извеждат на екрана ,ако не присъсват в масива от празници и работни съботи. Всички дати отговарящи на това условие се запазват в List от тип DateTime . | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace WorkdaysTask9  {    class Workdays  {  static DateTime[] holidays = new DateTime[]  {  new DateTime(2012, 3, 3),  new DateTime(2012, 12, 25),  new DateTime(2012, 12, 31),  };  static DateTime[] saturdays = new DateTime[]  {  new DateTime(2012, 3, 10),  new DateTime(2012, 4, 25),  new DateTime(2012, 12, 31),  };  class workday  {  DateTime today = DateTime.Today;  DateTime next = new DateTime();  public workday(DateTime next)  {  this.next = next;  }  public void print()  {  DateTime dateValue = DateTime.Today;  List<DateTime> dateList = new List<DateTime>();  int weekends = (int)dateValue.DayOfWeek;  int isHoliday = 0;  dateList.Add(dateValue);  while (today.AddDays(1) <= next)  {  today = today.AddDays(1);  dateList.Add(today);  }  foreach (var item in dateList)  {  for (int i = 0; i < holidays.Length; i++)  {  if (item == holidays[i]) isHoliday = 1;  }  if (weekends == 6 || weekends == 7) isHoliday = 1;  for (int i = 0; i < saturdays.Length; i++)  {  if (item == saturdays[i]) isHoliday = 0;  }  if (isHoliday == 0) Console.WriteLine("{0}.{1}.{2}", item.Day, item.Month, item.Year);  if (weekends == 7) weekends = 0;  isHoliday = 0;  weekends++;  }  }  }  static void Main(string[] args)  {  DateTime today = DateTime.Today;  DateTime dayInFuture;  dayInFuture = DateTime.Parse(Console.ReadLine());  if (dayInFuture <= today) Console.WriteLine("Date before today");  else  {  workday ob1 = new workday(dayInFuture);  ob1.print();  }  }  }  } | |
| **Тестове**  Тества се с дати преди текущата ,дати в бъдещето и дати които са празник или работна събота | |
| **Вход** | **Изход** |
| 2012.4.30 | 28.3.2012  29.3.2012  30.3.2012  2.4.2012  4.4.2012  5.4.2012  6.4.2012  9.4.2012  10.4.2012  11.4.2012  12.4.2012  13.4.2012  16.4.2012  17.4.2012  18.4.2012  19.4.2012  20.4.2012  23.4.2012  24.4.2012  25.4.2012  26.4.2012  27.4.2012 |
| **Вход** | **Изход** |
| 1990.5.24 | Date before today |
| **Вход** | **Изход** |
| 2.2.2 | Date before today |
| **Вход** | **Изход** |
| 2012.4.16 | 28.3.2012  29.3.2012  30.3.2012  2.4.2012  4.4.2012  5.4.2012  6.4.2012  9.4.2012  10.4.2012  11.4.2012  12.4.2012  13.4.2012  16.4.2012 |
| **Вход** | **Изход** |
| 24.06.2012 | 28.3.2012  29.3.2012  30.3.2012  2.4.2012  3.4.2012  4.4.2012  5.4.2012  6.4.2012  9.4.2012  10.4.2012  11.4.2012  12.4.2012  13.4.2012  16.4.2012  17.4.2012  18.4.2012  19.4.2012  20.4.2012  23.4.2012  24.4.2012  25.4.2012  26.4.2012  27.4.2012  30.4.2012  1.5.2012  2.5.2012  3.5.2012  4.5.2012  7.5.2012  8.5.2012  9.5.2012  10.5.2012  11.5.2012  14.5.2012  15.5.2012  16.5.2012  17.5.2012  18.5.2012  21.5.2012  22.5.2012  23.5.2012  24.5.2012  25.5.2012  28.5.2012  29.5.2012  30.5.2012  31.5.2012  1.6.2012  4.6.2012  5.6.2012  6.6.2012  7.6.2012  8.6.2012  11.6.2012  12.6.2012  13.6.2012  14.6.2012  15.6.2012  18.6.2012  19.6.2012  20.6.2012  21.6.2012  22.6.2012 |
| **Вход** | **Изход** |
| 5.6.2000 | Date before today |
| **Вход** | **Изход** |
| 12.12.02 | Date before today |
| **Вход** | **Изход** |
| 6.6.12 | 24.06.2012  28.3.2012  29.3.2012  30.3.2012  2.4.2012  3.4.2012  4.4.2012  5.4.2012  6.4.2012  9.4.2012  10.4.2012  11.4.2012  12.4.2012  13.4.2012  16.4.2012  17.4.2012  18.4.2012  19.4.2012  20.4.2012  23.4.2012  24.4.2012  25.4.2012  26.4.2012  27.4.2012  30.4.2012  1.5.2012  2.5.2012  3.5.2012  4.5.2012  7.5.2012  8.5.2012  9.5.2012  10.5.2012  11.5.2012  14.5.2012  15.5.2012  16.5.2012  17.5.2012  18.5.2012  21.5.2012  22.5.2012  23.5.2012  24.5.2012  25.5.2012  28.5.2012  29.5.2012  30.5.2012  31.5.2012  1.6.2012  4.6.2012  5.6.2012  6.6.2012  7.6.2012  8.6.2012  11.6.2012  12.6.2012  13.6.2012  14.6.2012  15.6.2012  18.6.2012  19.6.2012  20.6.2012  21.6.2012  22.6.2012 |
| **Вход** | **Изход** |
| 21.12.1321 | Date before today |
| **Вход** | **Изход** |
| 01.02.03 | Date before today |
|  |  |

|  |  |
| --- | --- |
| **Задача 10. Sum From Array** | |
| **Условие**  Дадена е последователност от цели положителни числа, записани едно след друго като символен низ, разделени с интервал. Да се напише програма, която пресмята сумата им. | |
| **Описание на входа**  Въвеждаме числа, които разделяме едно от друго с интервал. Входа свършва когато се натисне Enter. | |
| **Описание на изхода**  Резултата от сумирането на всички числа се изписва на конзолата | |
| **Анализ на задачата**  Използван е String.Split(‘ ‘), за да се раздели символния низ по интервалите, след което с Int32.Parse() се извличат отделните числа от получения масив от символни низове. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace ConsoleApplication22  {  class Sum  {  string numbers;  int sum;  public Sum(string numbers)  {  this.numbers = numbers;  }  public int calculate()  {  string[] split = numbers.Split(new Char[] { ' ' });  foreach (string s in split)  {  if (s.Trim() != "")  {  sum += int.Parse(s);  }  }  return sum;  }  }  class Program  {  static void Main(string[] args)  {  Sum ob1 = new Sum(Console.ReadLine());  Console.WriteLine(ob1.calculate());  }  }  } | |
| **Тестове**  Тества се дали събира правилно положителни числа, отрицателни числа, еднакви числа с различни знаци. | |
| **Вход** | **Изход** |
| 1 2 3 4 | 10 |
| **Вход** | **Изход** |
| 43 68 9 23 318 | 461 |
| **Вход** | **Изход** |
| 1 -1 | 0 |
| **Вход** | **Изход** |
| 100 | 100 |
| **Вход** | **Изход** |
| 1000000 100 1000 | -998900 |
| **Вход** | **Изход** |
| 123456 654321 123456 654321 | 1555554 |
| **Вход** | **Изход** |
| -111 -222 -333 -444 -555 -666 | -2331 |
| **Вход** | **Изход** |
| 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 | 135 |
| **Вход** | **Изход** |
| 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 01 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 | 270 |
| **Вход** | **Изход** |
| -5 +6 -1234 +787975 -213435 +23646437 36243 | 24255987 |
|  |  |

|  |  |
| --- | --- |
| **Задача 11. Random Sentences** | |
| **Условие**  Напишете програма, която генерира случайно рекламно съобщение за някакъв продукт. Съобщенията трябва да се състоят от хвалебст­вена фраза, следвани от хвалебствена случка, следвани от автор (първо и второ име) и град, които се избират от предварително подготвени списъци. Например, нека имаме следните списъци:  -     Хвалебствени фрази: {"Продуктът е отличен.", "Това е страхотен продукт.", "Постоянно ползвам този продукт.", "Това е най-добрият продукт от тази категория."}.  -     Хвалебствени случки: {"Вече се чувствам добре.", "Успях да се променя.", "Той направи чудо.", "Не мога да повярвам, но вече се чувствам страхотно.", "Опитайте и вие. Аз съм много доволна."}.  -     Първо име на автор: {"Диана", "Петя", "Стела", "Елена", "Катя"}.  -     Второ име на автор: {"Иванова", "Петрова", "Кирова"}.  -     Градове: {"София", "Пловдив", "Варна", "Русе", "Бургас"}.  Тогава програма би могла да изведе следното случайно-генерирано рекламно съобщение:   |  | | --- | | Постоянно ползвам този продукт. Опитайте и вие. Аз съм доволна. -– Елена Петрова, Варна | | |
| **Описание на входа**  Вход от конзолата липсва ,входните данни се въвеждат в 5 масива съответно за Хвалебствени фрази,Хвалебствени случки,Първо име на автора,Второ име на автора и градове | |
| **Описание на изхода**  На изхода се изписва произволно генерирано съобщение с по един елемент от 5-те масива. | |
| **Анализ на задачата**  **Създават се 5 масива от тип string след което се запълват с произволен брои елементи**  **Създават се рандом генериращи числа за всеки масив чрез класа** Random;  **Така за всеки масив се връща прозиволно число от 0-ла до големината на броят елементи в него(така няма опасност да се генерира номер на елемент който не присъства в масива)** | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace RandomSentencesTask11  {  class RandomSentencesTask11  {  static void Main(string[] args)  {  string[] phrase = { "Great product", "Perfect product", "I use this product all the time", "The best product ever" };  string[] thingsHappened = { "I feel good now", "I've changed myself", "He've done a miracle", "You should try too" };  string[] firstName = { "Alex", "Dimiter", "Ivan", "Georgi", "Petko" };  string[] lastName = { "Djenkov", "Petrov", "Nenov" };  string[] city = { "Sofia", "Plovdiv", "Elhovo" };  Random rndGenerator= new Random();  int rndPhrase = rndGenerator.Next(0, phrase.Length);  int rndHappened = rndGenerator.Next(0, thingsHappened.Length);  int rndFirstName = rndGenerator.Next(0, firstName.Length);  int rndlastName = rndGenerator.Next(0, lastName.Length);  int rndCity = rndGenerator.Next(0, city.Length);  Console.WriteLine("{0}.{1}--{2} {3},{4}", phrase[rndPhrase], thingsHappened[rndHappened], firstName[rndFirstName], lastName[rndlastName],city[rndCity]);  }  }  } | |
| **Тестове**  Във входа са дадени масиви от стрингове и се тества дали дава произволни резултати изхода | |
| **Вход** | **Изход** |
| string[] phrase = { "Great product", "Perfect product", "I use this product all the time", "The best product ever" };  string[] thingsHappened = { "I feel good now", "I've changed myself", "He've done a miracle", "You should try too" };  string[] firstName = { "Alex", "Dimiter", "Ivan", "Georgi", "Petko" };  string[] lastName = { "Djenkov", "Petrov", "Nenov" };  string[] city = { "Sofia", "Plovdiv", "Elhovo" }; | I use this product all the time.I feel good now--Petko Nenov,Plovdiv |
| **Вход** | **Изход** |
| string[] phrase = { "Great product", "Perfect product", "I use this product all the time", "The best product ever" };  string[] thingsHappened = { "I feel good now", "I've changed myself", "He've done a miracle", "You should try too" };  string[] firstName = { "Alex", "Dimiter", "Ivan", "Georgi", "Petko" };  string[] lastName = { "Djenkov", "Petrov", "Nenov" };  string[] city = { "Sofia", "Plovdiv", "Elhovo" }; | Great product.He've done a miracle--Dimiter Nenov,Sofia |
| **Вход** | **Изход** |
| string[] phrase = { "one", "two"};  string[] thingsHappened = { "one", "two"}; string[] firstName = { "one", "two"}; string[] lastName = { "one", "two"}; string[] city = { "one", "two"}; | One.two—one one,two |

|  |  |
| --- | --- |
| **Задача 12.ShuntingYard** | |
| **Условие** Напишете програма, която изчислява стойността на даден числен израз, зададен като стринг. Численият израз се състои от:   * реални числа, например 5, 18.33, 3.14159, 12.6; * аритметични оператори: +, -, \*, / (със стандартните им приоритети); * математически функции: ln(x), sqrt(x), pow(x,y); * скоби за промяна на приоритета на операциите: ( и ).   Обърнете внимание, че числовите изрази имат приоритет, например изразът -1 + 2 + 3 \* 4 - 0.5 = (-1) + 2 + (3 \* 4) - 0.5 = 12.5. | |
| **Описание на входа**  Входа е валиден числен израз състоящ се от:   * реални числа, например 5, 18.33, 3.14159, 12.6; * аритметични оператори: +, -, \*, / (със стандартните им приоритети); * математически функции: ln(x), sqrt(x), pow(x,y); * скоби за промяна на приоритета на операциите: ( и ). | |
| **Описание на изхода**  Изходът се извежда на един ред и гласи резултата от изчислението на въведения израз. | |
| **Анализ на задачата**  Класът ShuntingYardAlgorithm се използва за преобразуване не израза към Обратен Полски Запис(<http://en.wikipedia.org/wiki/Reverse_Polish_notation>), чрез извикване на метода Transform().  Той извиква методите Initialize(), ProcesTokens(), AppendAllPendingTokens() и метода ProduceResult, който връща израза в ОПЗ.  Initialize() разделя входа, чрез String().Split(‘ ’), на токени и ги запазва в списъка listOfInputTokens и извиква конструкторите на другите полета на класа.  ProcesTokens() извиква методи които оценяват дали токена е литерал, скоби, разделител или оператор и извиква съответния метод, който да обработи тази токен според типът му. Типът на токените се разпознава от методи използващи регулярни или директно сравнение на чарове. Обработката на токените, които са оператори се извършва чрез метод който определя техния преоритет – CalculatePrecedence();  AppendAllPendingTokens() добавя изчакващите токени във списъка със изходни токени.  ProduceResult() слепва всички изходни токени от outputTokens във стринг, който връща.  Класът CalCulateRNP се използва за пресмятане на израза, след като той е бил пренаписан в Обратен Полски Запис.  Методът CalculateNotation пресмята стойността на израза на няколко стъпки. Първо инициализира стек, който пази числата израза. После итерира по израза, като бута в стека всяко число, а като стигне оператор се определя броя на операндите за този оператор - GetNumberOfOperators. Този брой операнди се вадят от стека и се извършва изяисляване на стойността на този прост съставен израз – CurrentCalculation(), като резултата се пъха във стека и се продължава със следващата итерация. След последната итерация стека съдържа един единствен елемент, който е резултата на израза. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Text.RegularExpressions;  namespace ShuntingYard  {  class ShuntingYardTest  {  static void Main(string[] args)  {  string input = Console.ReadLine();  var algorithm = new ShuntinYardAlgorithm();  Console.WriteLine(algorithm.Transform(input));  var calculation = new CalCulateRNP();  string expression = algorithm.Transform(input);  Console.WriteLine(calculation.CalculateNotation(expression));  }  }  class ShuntinYardAlgorithm  {  private List<string> listOfInputTokens;  private Stack<string> pendingTokens;  private List<string> outputTokens;  public string Transform(string expression)  {  Initialize(expression);  ProcesTokens();  AppendAllPendingTokes();  return ProduceResult();  }  private string ProduceResult()  {  string result = "";  foreach (string current in outputTokens)  {  if (result.Length > 0)  {  result += " ";  }  result += current;  }  return result;  }  private void AppendAllPendingTokes()  {  while (ThereArePendingTokens())  {  AppendPendingToken();  }  }  private void AppendPendingToken()  {  if (ThereArePendingTokens())  {  AppendToken(pendingTokens.Pop());  }  }  private bool ThereArePendingTokens()  {  return pendingTokens.Count > 0;  }  private void ProcesTokens()  {  foreach (string current in listOfInputTokens)  {  if (IsLiteral(current))  {  HandleLiteral(current);  }  else if (IsOpenParentheses(current))  {  HandleOpenParentheses();  }  else if (IsClosedParentheses(current))  {  HandleClosedParentheses();  }  else if (IsParameterSeparator(current))  {  HandleParameterSeparator();  }  else  {  HandleOperator(current);  }  }  }  private void HandleParameterSeparator()  {  BalanceToOpenParentheses();  }  private bool IsParameterSeparator(string current)  {  return current.Equals(",");  }  private void HandleClosedParentheses()  {  BalanceToOpenParentheses();  DropOpenParentheses();  ConditionallyAddFunctionNameToOutput();  }  private void ConditionallyAddFunctionNameToOutput()  {  if (PendingTokenIsFunctionName())  {  AppendToken(BildFunctionName());  }  }  private string BildFunctionName()  {  return pendingTokens.Pop() + "()";  }  private bool PendingTokenIsFunctionName()  {  return ThereArePendingTokens() && IsFunctionName(pendingTokens.Peek());  }  private void DropOpenParentheses()  {  pendingTokens.Pop();  }  private void BalanceToOpenParentheses()  {  while (LastPendingTokenIsNotOpenParentheses())  {  AppendPendingToken();  }  }  private bool LastPendingTokenIsNotOpenParentheses()  {  if (ThereArePendingTokens())  {  return !pendingTokens.Peek().Equals("(");  }  else  {  return false;  }  }  private bool IsClosedParentheses(string current)  {  return current.Equals(")");  }  private void HandleOpenParentheses()  {  if (LastOperatorIsFunctionName())  {  MoveLastOutputTokenToPendingTokens();  }  AddToPendingTokens("(");  }  private void AddToPendingTokens(string token)  {  pendingTokens.Push(token);  }  private void MoveLastOutputTokenToPendingTokens()  {  AddLastOutputTokenToPending();  DropLastOutputToken();  }  private void AddLastOutputTokenToPending()  {  AddToPendingTokens(LastOutputToken());  }  private string LastOutputToken()  {  return outputTokens[CalculateLastIndexOfOutputTokens()];  }  private void DropLastOutputToken()  {  outputTokens.RemoveAt(CalculateLastIndexOfOutputTokens());  }  private int CalculateLastIndexOfOutputTokens()  {  int lastIndex = outputTokens.Count - 1;  return lastIndex;  }  private bool LastOperatorIsFunctionName()  {  return ThereAreOutputTokens() && IsFunctionName(LastOutputToken());  }  private bool ThereAreOutputTokens()  {  return outputTokens.Count > 0;  }  private bool IsFunctionName(string tokens)  {  return Regex.IsMatch(tokens, "[a-z]") && !Regex.IsMatch(tokens, @"\W");  }  private bool IsOpenParentheses(string current)  {  return current.Equals("(");  }  private void HandleOperator(string current)  {  while (PendingTokenExecutedBeforeCurrent(current))  {  AppendPendingToken();  }  AddToPendingTokens(current);  }  private bool PendingTokenExecutedBeforeCurrent(string current)  {  if (ThereArePendingTokens() && !pendingTokens.Peek().Equals("="))  {  int pendingPrecedence = CalculatePrecedence(pendingTokens.Peek());  int currentPrecedence = CalculatePrecedence(current);  return pendingPrecedence >= currentPrecedence;  }  else  {  if (ThereArePendingTokens() && pendingTokens.Peek().Equals("="))  {  int pendingPrecedence = CalculatePrecedence(pendingTokens.Peek());  int currentPrecedence = CalculatePrecedence(current);  return pendingPrecedence > currentPrecedence;  }  }  return false;  }  private int CalculatePrecedence(string current)  {  int precedence = 80;  switch (current)  {  case "ln":  case "pow":  return 110;  case "\*":  case "/":  return 100;  case "+":  case "-":  return 90;  case "=":  return precedence - 1;  case "(":  case ")":  return 10;  default:  return 1;  }  }  private void Initialize(string expression)  {  pendingTokens = new Stack<string>();  listOfInputTokens = new List<string>();  listOfInputTokens.AddRange((expression ?? "").Split());  outputTokens = new List<string>();  }  private void HandleLiteral(string current)  {  AppendToken(current);  }  private void AppendToken(string token)  {  outputTokens.Add(token);  }  private bool IsLiteral(string token)  {  return Regex.IsMatch(token, @"(\w+|\d+)");  }  }  class CalCulateRNP  {  private Stack<double> numbers;  public double CalculateNotation(string expression)  {  double result = 0;  string[] tokens = expression.Split(' ');  numbers = new Stack<double>();  double currentNumber;  double[] numbersForCalculation = new double[2];  for (int i = 0; i < tokens.Length; i++)  {  if (IsNumber(tokens[i]))  {  currentNumber = double.Parse(tokens[i]);  numbers.Push(currentNumber);  }  else  {  int neededOperators = GetNumberOfOperators(tokens[i]);  for (int j = 0; j < neededOperators; j++)  {  numbersForCalculation[j] = numbers.Pop();  }  numbers.Push(CurrentCalculation(numbersForCalculation, tokens[i]));  }  }  result = numbers.Pop();  return result;  }  private double CurrentCalculation(double[] numbersForCalculation, string token)  {  switch (token)  {  case "+":  return numbersForCalculation[0] + numbersForCalculation[1];  case "-":  return numbersForCalculation[1] - numbersForCalculation[0];  case "/":  return numbersForCalculation[1] / numbersForCalculation[0];  case "\*":  return numbersForCalculation[0] \* numbersForCalculation[1];  case "ln()":  return Math.Log(numbersForCalculation[0], Math.E);  case "sqrt()":  return Math.Sqrt(numbersForCalculation[0]);  case "pow()":  if (numbersForCalculation[0] > 0)  {  return Math.Pow(numbersForCalculation[1], numbersForCalculation[0]);  }  else  {  return 1 / Math.Pow(numbersForCalculation[1], Math.Abs(numbersForCalculation[0]));  }  default:  return 0;  }  }  private int GetNumberOfOperators(string current)  {  switch (current)  {  case "+":  case "-":  case "/":  case "\*":  case "pow()":  return 2;  default: return 1;  }  }  private bool IsNumber(string token)  {  return Regex.IsMatch(token, @"\d");  }  }  } | |
| **Тестове**  Тестовете проверяват вход 0, прости аритметични операции, промянана на преоритета на операцията от скоби, използване на функции, както и функции и аретметияни оператори, вложени във други функции. | |
| **Вход** | **Изход** |
| 0 | 0 |
| **Вход** | **Изход** |
| 4 + 3 \* 5 | 19 |
| **Вход** | **Изход** |
| ( 4 + 3 ) \* 5 | 35 |
| **Вход** | **Изход** |
| 3.4 + 5.5 | 8.9 |
| **Вход** | **Изход** |
| sqrt ( sqrt ( 81 ) ) | 3 |
| **Вход** | **Изход** |
| pow ( 3 , 1 + 1 ) | 9 |
| **Вход** | **Изход** |
| ( 4 + ( 3.4 – 1.4 ) ) \* 5 | 30 |
| **Вход** | **Изход** |
| ( pow ( 2 , 2 ) + sqrt ( 4 ) ) \* 1.5 | 9 |
| **Вход** | **Изход** |
| sqrt ( pow ( sqrt ( 4 ) , 1 + 1 ) ) | 2 |
| **Вход** | **Изход** |
| pow ( 2 , 3.14 ) \* ( 3 - ( 3 \* sqrt ( 2 ) - 3.2 ) + 1.5 \* 0.3 ) | 21.2214523409317 |
|  |  |