# Решения на задачите от глава 04 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

**Име: Венцислав Василев Щерев**

Контакти: v.v.shterev@gmail.com

**Име: Весела Коцева**

Контакти: vs\_kotseva@yahoo.com

**Име: Павел Димитров Колев**

Контакти: paveldk@almaalter.com

**Име: Стилиян Марков**

Контакти: stelko88@abv.bg

# Шаблон за описание на задачите

|  |  |
| --- | --- |
| Задача 01.Брой срещания на число в дърво | |
| **Условие**  Да се напише програма, която намира броя на срещанията на дадено число в дадено дърво от числа. | |
| **Описание на входа**  Входа се състой от един единствен ред, в който се въвежда търсеното от нас число. | |
| **Описание на изхода**  Изхода се състой от един ред на който се изписва броя срещания не въведеното по горе число. | |
| **Анализ на задачата**  Задачате фактически се решава от метода с име CountOccurance, на който като параметтри му подаваме дърво от тип int и търсеното число.  В самия метот започваме като инициализираме опашка от TreeNode и във нея слагаме корена на дървото.  След това в while цикъл вадим поредния нод който е в опашката и слагаме в опашката всичките му деца. В същото време проверяваме всеки от нодовете дали не е равен на търсеното число,ако да – броим го и накрая връщаме броя като резултат. | |
| **Сорс код на самото дърво** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Trees  {  public class TreeNode<T>  {  private T value;  private bool hasParent;  private List<TreeNode<T>> children;  public TreeNode(T value)  {  if (value == null)  {  throw new ArgumentNullException(  "Cannot insert null value!");  }  this.value = value;  this.children = new List<TreeNode<T>>();  }  public T Value  {  get  {  return this.value;  }  set  {  this.value = value;  }  }  public int ChildrenCount  {  get  {  return this.children.Count;  }  }  public void AddChild(TreeNode<T> child)  {  if (child == null)  {  throw new ArgumentNullException(  "Cannot insert null value!");  }  if (child.hasParent)  {  throw new ArgumentException(  "The node already has a parent!");  }  child.hasParent = true;  this.children.Add(child);  }  public TreeNode<T> GetChild(int index)  {  return this.children[index];  }  }  public class Tree<T>  {  private TreeNode<T> root;  public Tree(T value)  {  if (value == null)  {  throw new ArgumentNullException(  "Cannot insert null value!");  }  this.root = new TreeNode<T>(value);  }  public Tree(T value, params Tree<T>[] children)  : this(value)  {  foreach (Tree<T> child in children)  {  this.root.AddChild(child.root);  }  }  public TreeNode<T> Root  {  get  {  return this.root;  }  }  private void TraverseDFS(TreeNode<T> root, string spaces)  {  if (this.root == null)  {  return;  }  Console.WriteLine(spaces + root.Value);  TreeNode<T> child = null;  for (int i = 0; i < root.ChildrenCount; i++)  {  child = root.GetChild(i);  TraverseDFS(child, spaces + " ");  }  }  public void TraverseDFS()  {  this.TraverseDFS(this.root, string.Empty);  }  public void TraverseBFS()  {  Queue<TreeNode<T>> queue = new Queue<TreeNode<T>>();  queue.Enqueue(this.root);  while (queue.Count > 0)  {  TreeNode<T> currentNode = queue.Dequeue();  Console.Write("{0} ", currentNode.Value);  for (int i = 0; i < currentNode.ChildrenCount; i++)  {  TreeNode<T> childNode = currentNode.GetChild(i);  queue.Enqueue(childNode);  }  }  }  public void TraverseDFSWithStack()  {  Stack<TreeNode<T>> stack = new Stack<TreeNode<T>>();  stack.Push(this.root);  while (stack.Count > 0)  {  TreeNode<T> currentNode = stack.Pop();  Console.Write("{0} ", currentNode.Value);  for (int i = 0; i < currentNode.ChildrenCount; i++)  {  TreeNode<T> childNode = currentNode.GetChild(i);  stack.Push(childNode);  }  }  }  }  } | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Trees;  namespace \_1\_\_\_Number\_Occurance  {  class Program  {  static int countOccurance = 0;  static void Main(string[] args)  {  Tree<int> tree =  new Tree<int>(7,  new Tree<int>(19,  new Tree<int>(1),  new Tree<int>(12),  new Tree<int>(12),  new Tree<int>(31)),  new Tree<int>(21),  new Tree<int>(14,  new Tree<int>(23),  new Tree<int>(6))  );  int num = int.Parse(Console.ReadLine());  CountOccurance(tree, num);  Console.WriteLine(countOccurance);  }  static void CountOccurance(Tree<int> tree, int num)  {  Stack<TreeNode<int>> stack = new Stack<TreeNode<int>>();  stack.Push(tree.Root);  while (stack.Count > 0)  {  TreeNode<int> currentNode = stack.Pop();  for (int i = 0; i < currentNode.ChildrenCount; i++)  {  TreeNode<int> childNode = currentNode.GetChild(i);  if (childNode.Value == num)  {  countOccurance++;  }  stack.Push(childNode);  }  }  }  }  } | |
| **Тестове**  Тъй като ресултата зависи от стойностите които се намират в самото дърво, затова щедам условни стоиности които се намират в дървото: 7 19 1 12 12 31 21 14 23 6  Интересни случаи може да са следните   * При вход число което не се съдържа в дървото се очаква 0 | |
| **Вход** | **Изход** |
| 7 | 1 |
| **Вход** | **Изход** |
| 33 | 0 |
| **Вход** | **Изход** |
| 31 | 1 |
| **Вход** | **Изход** |
| 12 | 2 |
|  | |
| Задача 02.Брой нодове с к на брой деца | |
| **Условие**  Да се напише програма, която извежда корените на онези поддървета на дадено дърво, които имат точно k на брой върха, където k e дадено естествено число. | |
| **Описание на входа**  Входа се състой от един единствен ред, в който се въвежда търсения от нас брой. | |
| **Описание на изхода**  Изхода се състой от един ред на който се изписва броя нодове с к деца. | |
| **Анализ на задачата**  Задачате фактически се решава от метода с име CountNodes, на който като параметтри му подаваме дърво от тип int и търсения от нас брой.  В самия метот започваме като инициализираме опашка от TreeNode и във нея слагаме корена на дървото.  След това в while цикъл вадим поредния нод който е в опашката и слагаме в опашката всичките му деца. В същото време проверяваме всеки от нодовете дали неговите даца отговарят на търсения брой, ако да увеличаваме брояча и накрая го връщаме като резултат. | |
| **Сорс код на самото дърво** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Trees  {  public class TreeNode<T>  {  private T value;  private bool hasParent;  private List<TreeNode<T>> children;  public TreeNode(T value)  {  if (value == null)  {  throw new ArgumentNullException(  "Cannot insert null value!");  }  this.value = value;  this.children = new List<TreeNode<T>>();  }  public T Value  {  get  {  return this.value;  }  set  {  this.value = value;  }  }  public int ChildrenCount  {  get  {  return this.children.Count;  }  }  public void AddChild(TreeNode<T> child)  {  if (child == null)  {  throw new ArgumentNullException(  "Cannot insert null value!");  }  if (child.hasParent)  {  throw new ArgumentException(  "The node already has a parent!");  }  child.hasParent = true;  this.children.Add(child);  }  public TreeNode<T> GetChild(int index)  {  return this.children[index];  }  }  public class Tree<T>  {  private TreeNode<T> root;  public Tree(T value)  {  if (value == null)  {  throw new ArgumentNullException(  "Cannot insert null value!");  }  this.root = new TreeNode<T>(value);  }  public Tree(T value, params Tree<T>[] children)  : this(value)  {  foreach (Tree<T> child in children)  {  this.root.AddChild(child.root);  }  }  public TreeNode<T> Root  {  get  {  return this.root;  }  }  private void TraverseDFS(TreeNode<T> root, string spaces)  {  if (this.root == null)  {  return;  }  Console.WriteLine(spaces + root.Value);  TreeNode<T> child = null;  for (int i = 0; i < root.ChildrenCount; i++)  {  child = root.GetChild(i);  TraverseDFS(child, spaces + " ");  }  }  public void TraverseDFS()  {  this.TraverseDFS(this.root, string.Empty);  }  public void TraverseBFS()  {  Queue<TreeNode<T>> queue = new Queue<TreeNode<T>>();  queue.Enqueue(this.root);  while (queue.Count > 0)  {  TreeNode<T> currentNode = queue.Dequeue();  Console.Write("{0} ", currentNode.Value);  for (int i = 0; i < currentNode.ChildrenCount; i++)  {  TreeNode<T> childNode = currentNode.GetChild(i);  queue.Enqueue(childNode);  }  }  }  public void TraverseDFSWithStack()  {  Stack<TreeNode<T>> stack = new Stack<TreeNode<T>>();  stack.Push(this.root);  while (stack.Count > 0)  {  TreeNode<T> currentNode = stack.Pop();  Console.Write("{0} ", currentNode.Value);  for (int i = 0; i < currentNode.ChildrenCount; i++)  {  TreeNode<T> childNode = currentNode.GetChild(i);  stack.Push(childNode);  }  }  }  }  } | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Trees;  namespace \_1\_\_\_Number\_Occurance  {  class Program  {  static int countOccurance = 0;  static void Main(string[] args)  {  Tree<int> tree =  new Tree<int>(7,  new Tree<int>(19,  new Tree<int>(1),  new Tree<int>(12),  new Tree<int>(12),  new Tree<int>(31)),  new Tree<int>(21),  new Tree<int>(14,  new Tree<int>(23),  new Tree<int>(6))  );  int num = int.Parse(Console.ReadLine());  CountOccurance(tree, num);  Console.WriteLine(countOccurance);  }  static void CountNodes(Tree<int> tree, int num)  {  Stack<TreeNode<int>> stack = new Stack<TreeNode<int>>();  stack.Push(tree.Root);  while (stack.Count > 0)  {  TreeNode<int> currentNode = stack.Pop();  for (int i = 0; i < currentNode.ChildrenCount; i++)  {  TreeNode<int> childNode = currentNode.GetChild(i);  if (childNode.ChildrenCount == num)  {  countNodes++;  }  stack.Push(childNode);  }  }  }  }  } | |
| **Тестове**  Интересни случаи може да са следните   * При вход k което не се съдържа в дървото се очаква 0 | |
| **Вход** | **Изход** |
| 2 | 1 |
| **Вход** | **Изход** |
| 3 | 0 |
| **Вход** | **Изход** |
| 4 | 1 |
| **Вход** | **Изход** |
| 1 | 0 |
|  | |
| Задача 03.Брой на листа и брой на вътрешни върхове | |
| **Условие**  Да се напише програма, която намира броя на листата и броя на вътрешните върхове на дадено дърво. | |
| **Описание на входа**  Задачата не се нуждае от вход | |
| **Описание на изхода**  Изходът се състои от два реда. Единя за броя на листата, другия за броя на вътрешните върхове. | |
| **Анализ на задачата**  Задачате фактически се решава от метода с име CountLeafsAndInner, на който като параметтри му подаваме дърво от тип int.  В самия метот започваме като инициализираме опашка от TreeNode и във нея слагаме корена на дървото.  След това в while цикъл вадим поредния нод който е в опашката и слагаме в опашката всичките му деца. В същото време проверяваме всеки сегашен нод, ако той има децаувеличаваме брояча countInnerLeafs и ако няма деца - countLeafs.  countInnerLeafs и countLeafs са статични променливи и не се налага да ги връщаме като резултат. | |
| **Сорс код на самото дърво** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Trees  {  public class TreeNode<T>  {  private T value;  private bool hasParent;  private List<TreeNode<T>> children;  public TreeNode(T value)  {  if (value == null)  {  throw new ArgumentNullException(  "Cannot insert null value!");  }  this.value = value;  this.children = new List<TreeNode<T>>();  }  public T Value  {  get  {  return this.value;  }  set  {  this.value = value;  }  }  public int ChildrenCount  {  get  {  return this.children.Count;  }  }  public void AddChild(TreeNode<T> child)  {  if (child == null)  {  throw new ArgumentNullException(  "Cannot insert null value!");  }  if (child.hasParent)  {  throw new ArgumentException(  "The node already has a parent!");  }  child.hasParent = true;  this.children.Add(child);  }  public TreeNode<T> GetChild(int index)  {  return this.children[index];  }  }  public class Tree<T>  {  private TreeNode<T> root;  public Tree(T value)  {  if (value == null)  {  throw new ArgumentNullException(  "Cannot insert null value!");  }  this.root = new TreeNode<T>(value);  }  public Tree(T value, params Tree<T>[] children)  : this(value)  {  foreach (Tree<T> child in children)  {  this.root.AddChild(child.root);  }  }  public TreeNode<T> Root  {  get  {  return this.root;  }  }  private void TraverseDFS(TreeNode<T> root, string spaces)  {  if (this.root == null)  {  return;  }  Console.WriteLine(spaces + root.Value);  TreeNode<T> child = null;  for (int i = 0; i < root.ChildrenCount; i++)  {  child = root.GetChild(i);  TraverseDFS(child, spaces + " ");  }  }  public void TraverseDFS()  {  this.TraverseDFS(this.root, string.Empty);  }  public void TraverseBFS()  {  Queue<TreeNode<T>> queue = new Queue<TreeNode<T>>();  queue.Enqueue(this.root);  while (queue.Count > 0)  {  TreeNode<T> currentNode = queue.Dequeue();  Console.Write("{0} ", currentNode.Value);  for (int i = 0; i < currentNode.ChildrenCount; i++)  {  TreeNode<T> childNode = currentNode.GetChild(i);  queue.Enqueue(childNode);  }  }  }  public void TraverseDFSWithStack()  {  Stack<TreeNode<T>> stack = new Stack<TreeNode<T>>();  stack.Push(this.root);  while (stack.Count > 0)  {  TreeNode<T> currentNode = stack.Pop();  Console.Write("{0} ", currentNode.Value);  for (int i = 0; i < currentNode.ChildrenCount; i++)  {  TreeNode<T> childNode = currentNode.GetChild(i);  stack.Push(childNode);  }  }  }  }  } | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Trees;  namespace \_1\_\_\_Number\_Occurance  {  class Program  {  static int countOccurance = 0;  static void Main(string[] args)  {  Tree<int> tree =  new Tree<int>(7,  new Tree<int>(19,  new Tree<int>(1),  new Tree<int>(12),  new Tree<int>(12),  new Tree<int>(31)),  new Tree<int>(21),  new Tree<int>(14,  new Tree<int>(23),  new Tree<int>(6))  );  int num = int.Parse(Console.ReadLine());  CountOccurance(tree, num);  Console.WriteLine(countOccurance);  }  static void CountLeafsAndInner(Tree<int> tree)  {  Stack<TreeNode<int>> stack = new Stack<TreeNode<int>>();  stack.Push(tree.Root);  while (stack.Count > 0)  {  TreeNode<int> currentNode = stack.Pop();  if (currentNode.ChildrenCount > 0)  {  countInnerLeafs++;  for (int i = 0; i < currentNode.ChildrenCount; i++)  {  TreeNode<int> childNode = currentNode.GetChild(i);  stack.Push(childNode);  }  }  else countLeafs++;    }  }  }  } | |
| **Тестове** | |
| **Вход** | **Изход** |
|  | 7  3 |
| Задача 04.Сумата от всяко ниво на бинарно дърво | |
| **Условие**  Напишете програма, която по дадено двоично дърво от числа намира сумата на върховете от всяко едно ниво на дървото. | |
| **Описание на входа**  Тази програме не се нуждае от вход | |
| **Описание на изхода**  Изхода се състой от един ред на който се изписва сумата от върховете на даденото дърво | |
| **Анализ на задачата**  Задачате фактически се решава от метода с име SumTreeNodes на класа BinaryTree, на който като параметтри му подаваме корена на двойчното дърво от тип int.  Метода първоначално проверява дали модадения му нод не е null, ако е спира изпълнението на метода.  След което метода вика рекурсивно само севе си с параметри лявото и съответно дясното дете на текущия нод.  Сумата се натрупва в променливата sumTreeNodes, която е статична променлива на класа BinaryTree. | |
| **Сорс код на самото дърво** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace \_4\_\_\_Sum\_of\_leafs  {  public class BinaryTreeNode<T>  {  private T value;  private bool hasParent;  private BinaryTreeNode<T> leftChild;  private BinaryTreeNode<T> rightChild;  public BinaryTreeNode(T value,  BinaryTreeNode<T> leftChild,  BinaryTreeNode<T> rightChild)  {  if (value == null)  {  throw new ArgumentNullException(  "Cannot insert null value!");  }  this.value = value;  this.LeftChild = leftChild;  this.RightChild = rightChild;  }  public BinaryTreeNode(T value)  : this(value, null, null)  {  }  public T Value  {  get  {  return this.value;  }  set  {  this.value = value;  }  }  public BinaryTreeNode<T> LeftChild  {  get  {  return this.leftChild;  }  set  {  if (value == null)  {  return;  }  if (value.hasParent)  {  throw new ArgumentException(  "The node already has a parent!");  }  value.hasParent = true;  this.leftChild = value;  }  }  public BinaryTreeNode<T> RightChild  {  get  {  return this.rightChild;  }  set  {  if (value == null)  {  return;  }  if (value.hasParent)  {  throw new ArgumentException(  "The node already has a parent!");  }  value.hasParent = true;  this.rightChild = value;  }  }  }  public class BinaryTree<T>  {  private BinaryTreeNode<T> root;  public static int sumTreeNodes = 0;  public BinaryTree(T value, BinaryTree<T> leftChild,  BinaryTree<T> rightChild)  {  if (value == null)  {  throw new ArgumentNullException(  "Cannot insert null value!");  }  BinaryTreeNode<T> leftChildNode =  leftChild != null ? leftChild.root : null;  BinaryTreeNode<T> rightChildNode =  rightChild != null ? rightChild.root : null;  this.root = new BinaryTreeNode<T>(  value, leftChildNode, rightChildNode);  }  public BinaryTree(T value)  : this(value, null, null)  {  }  public BinaryTreeNode<T> Root  {  get  {  return this.root;  }  }  private void PrintInorder(BinaryTreeNode<T> root)  {  if (root == null)  {  return;  }  PrintInorder(root.LeftChild);  Console.Write(root.Value + " ");  PrintInorder(root.RightChild);  }  public void PrintInorder()  {  PrintInorder(this.root);  Console.WriteLine();  }  public void PrintSumTreeNodes()  {  SumTreeNodes(this.root);  Console.WriteLine(sumTreeNodes);  }  private void SumTreeNodes(BinaryTreeNode<T> root)  {  if (root == null)  {  return;  }  SumTreeNodes(root.LeftChild);  sumTreeNodes += int.Parse(root.Value.ToString());  SumTreeNodes(root.RightChild);  }  }  } | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Trees;  namespace \_4\_\_\_Sum\_of\_leafs  {  class Program  {  static void Main(string[] args)  {  BinaryTree<int> binaryTree =  new BinaryTree<int>(14,  new BinaryTree<int>(19,  new BinaryTree<int>(23),  new BinaryTree<int>(6,  new BinaryTree<int>(10),  new BinaryTree<int>(21))),  new BinaryTree<int>(15,  new BinaryTree<int>(3),  null));  binaryTree.PrintSumTreeNodes();  }  }  } | |
| **Тестове**  Тъй като резултата зависи от стойностите които се намират в самото дърво, затова ще дам условни стоиности които се намират в дървото: 14 19 23 6 10 21 15 3 | |
| **Вход** | **Изход** |
|  | 111 |
|  | |
| Задача 11.Брой срещания на число в дърво | |
| **Условие**  Напишете програма, която обхожда директорията C:\Windows\ и всичките и поддиректории рекурсивно и отпечатва всички файлове, който имат разширение \*.exe. | |
| **Описание на входа**  Тази задача не се нуждае от вход. | |
| **Описание на изхода**  Изхода се състой от достатъчен на брои редове, за да се изпечатат всички файлове завършващи с .exe. | |
| **Анализ на задачата**  Само по себе си задачата представлява обхождане на дърво, което в случея е постигнато с рекурсивно търсене в дълбочина(DFS).  Първото нещо което е добре да направим е да направим променлива регуларен израз в която да запазим формата на това което търсим а именно \*.exe.  Следващата стъпка е да стартираме рекурсивното търсене представено от метода с име TraverseDir, като параметър му подаваме пътя на началната директория под формата на DirectoryInfo. За да вземем всичките файлове в тази директория е нужно да ползваме метода на DirectoryInfo – GetFiles(). След като вече имаме лист от всички файлове ги проверяваме и печатаме тези от тях които пасват на регулярния израз. Това става с този метод: catchExe.IsMatch(file.FullName); ако има съвпадение го печатаме на конзолата.  В следващата стъпка подобно на предишната намираме всички под директории с метода GetDirectories(). За всяка една от директориите бускаме рекурсивно метода TraverseDir().  Цялото тяло на рекурсивния метод е сложено в try конструкция, защото по време на изпълнението на програмата се очаква да възникнат грешки който да бъдат игнорирани. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.IO;  using System.Text.RegularExpressions;  namespace \_1.TraverseDirectories  {  public static class TraverseDirs  {  public static Regex catchExe = new Regex("[.]\*\\.exe$");  public static void Main()  {  string directoryPath = "C:\\WINDOWS\\";  TraverseDir(new DirectoryInfo(directoryPath));  }  private static void TraverseDir(DirectoryInfo dir)  {  try  {  FileInfo[] files = dir.GetFiles();  foreach (FileInfo file in files)  {  if (catchExe.IsMatch(file.FullName))  {  Console.WriteLine(file.FullName);  }  }  DirectoryInfo[] children = dir.GetDirectories();  foreach (DirectoryInfo child in children)  {  TraverseDir(child);  }  }  catch (Exception)  {  }  }  }  } | |
| **Тестове**  Интересни случаи може да са следните   * Индересно е да се види дали е съобразено че са възможни да възникнах грешки по време на изпълнението на задачата | |
| **Вход** | **Изход** |
|  | C:\WINDOWS\ Alcrmv.exe  …  C:\WINDOWS\ winsxs\x86\_wvmic.inf\_31bf3856ad36e35\_6.1.7601…\vmicsvc.exe |
|  | |
| Задача 12.Клас файл и клас папка. | |
| **Условие**  Дефинирайте класове File { string name, int size } и Folder { string name, File[] files, Folder[] childFolders }. Използвайки тези класове, постройте дърво, което съдържа всички файлове и директории на твърдия диск, като започнете от C:\Windows\. Напишете метод, който изчислява сумата от големините на файловете в дадено поддърво и програма, която тества този метод. За обхождането на директориите използвайте рекурсивно обхождане в дълбочина (DFS). | |
| **Описание на входа**  Програмата не се нуждае от вход | |
| **Описание на изхода**  Изхода представлява един ред на който е изписан размера на файловете в директорията на windows. | |
| **Анализ на задачата**  За да се реши тази задача трябва да се създадът два класа един за папка и еднин за фаил за да може като обходим всеки фаил в дадената директория да можем някъде да съхраним неговия размер.  Първоначално започваме като си създаваме една папка която рекурсивно я попълваме със всички фаилове и папки от директорията на windows(подобно на предияшната задача).  След като сме я инициализирали тази папка , размера на всички нейни деца(файлове). | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.IO;  namespace \_2.CalculateFiles  {  class File  {  public string name { get; set; }  public double size { get; set; }  public File(string \_name, double \_size)  {  this.name = \_name;  this.size = \_size;  }  }  class Folder  {  public string name;  public File[] files;  public Folder[] childFolders;  public Folder()  {  }  public Folder(string \_name, int filesSize, int childFoldersSize)  {  this.name = \_name;  files = new File[filesSize];  childFolders = new Folder[childFoldersSize];  }  public Folder(string \_name)  {  this.name = \_name;  }  public void SetFilesSize(int filesSize)  {  files = new File[filesSize];  }  public void SetFoldersSize(int childFoldersSize)  {  childFolders = new Folder[childFoldersSize];  }  }  class Program  {  static void Main(string[] args)  {  string directoryPath = "C:\\WINDOWS";  Folder C = GetFiles(directoryPath);  Console.WriteLine(GetFolderSize("C:\\WINDOWS"));  }  private static Folder GetFiles(string directoryPath)  {  Folder allFiles = new Folder();  Folder newFolder = allFiles;  TraverseDir(new DirectoryInfo(directoryPath), newFolder);  return newFolder;  }  private static double GetFolderSize(string directoryPath)  {  return TraverseSizeDir(new DirectoryInfo(directoryPath)) / (1024 \* 1024);  }  private static double TraverseSizeDir(DirectoryInfo dir)  {  double sum = 0;  try  {  DirectoryInfo[] children = dir.GetDirectories();  FileInfo[] files = dir.GetFiles();    foreach (FileInfo file in files)  {  sum += (file.Length);  }  foreach (DirectoryInfo child in children)  {  sum += TraverseSizeDir(child);  }  }  catch (Exception)  {  }  return sum;  }  private static void TraverseDir(DirectoryInfo dir, Folder newFolder)  {  try  {  DirectoryInfo[] children = dir.GetDirectories();  FileInfo[] files = dir.GetFiles();  newFolder.name = dir.FullName;  newFolder.SetFilesSize(files.Length);  newFolder.SetFoldersSize(children.Length);  for (int file = 0; file < files.Length; file++)  {  newFolder.files[file] = new File(files[file].Name, files[file].Length);  }  for (int child = 0; child < children.Length; child++)  {  Folder backtrack = newFolder;  newFolder = newFolder.childFolders[child];  newFolder.childFolders[child] = new Folder();  TraverseDir(children[child], newFolder);  newFolder = backtrack;  }  }  catch (Exception)  {  }  }  }  } | |
| **Тестове**  Интересни случаи може да са следните   * Индересно е да се види дали е съобразено че са възможни да възникнах грешки по време на изпълнението на задачата | |
| **Вход** | **Изход** |
|  | 16552.1825456619 |