**First Question**

Fix the bugs.

public class MathUtils

{

public static double Average(int a, int b)

{

return (double) (a + b) / 2;

//return a + b / 2;

}

}

**Second Question**

Write a function that provides change directory (cd) function for an abstract file system.

Notes:

- Root path is '/'.

- Path separator is '/'.

- Parent directory is addressable as "..".

- Directory names consist only of English alphabet letters (A-Z and a-z).

For example, new Path("/a/b/c/d").Cd("../x").CurrentPath should return "/a/b/c/x".

using System;

public class Path

{

public string CurrentPath { get; private set; }

public Path(string path)

{

this.CurrentPath = path;

}

public Path Cd(string newPath)

{

// validating the input path.

if (Regex.IsMatch(newPath, @"^\.{2}/[A-z]+")) {

var NewDirectory = newPath.Substring (newPath.LastIndexOf ('/'));

if (string.Compare (CurrentPath, "/") == 0) {

// TODO:: as the empty case is not defined in the requirement,

// currently if current path is root directory, new directry will be changed to subdirectory to root

// Performing the string replacement and string concatination in the path.

string NewDirectoryPath = string.Format ("{0}{1}", CurrentPath, NewDirectory.Replace("/",string.Empty));

return new Path (NewDirectoryPath);

}

// return the input path a just the subdirectory to the root.

var returnValue = string.Format("{0}{1}",CurrentPath.Substring(0, CurrentPath.LastIndexOf('/')),NewDirectory);

return new Path (returnValue); ;

}

}

// return the input path a just the subdirectory to the root.

var returnValue = string.Format("{0}{1}",CurrentPath.Substring(0, CurrentPath.LastIndexOf('/')),NewDirectory);

return new Path (returnValue); ;

}

throw new ApplicationException ("Invalid path entered");

}

public static void Main(string[] args)

{

Path path = new Path("/a/b/c/d");

Console.WriteLine(path.Cd("../x").CurrentPath);

}

}

**Third Question**

Write a function that finds the zero-based index of the longest run in a string. A run is a consecutive sequence of the same character. If there is more than one run with the same length, return the index of the first one.

For example, IndexOfLongestRun("abbcccddddcccbba") should return 6 as the longest run is dddd and it first appears on index 6.

using System;

public class Run

{

public static int IndexOfLongestRun(string str)

{

// Validating string

if (string.IsNullOrEmpty (str.Trim())) {

return -1;

}

int currentLargest = 0;

int returnIndex =0;

MatchCollection result = Regex.Matches (str, "(\\w)\\1+");

foreach(Match item in result)

{

if (currentLargest < item.Length) {

returnIndex = item.Index;

currentLargest = item.Length;

}

}

return returnIndex;

}

public static void Main(string[] args)

{

Console.WriteLine(IndexOfLongestRun("abbcccddddcccbba"));

}

}

**Fourth Question**

Write a function that checks if a given binary tree is a valid binary search tree. A binary search tree (BST) is a binary tree where the value of each node is larger or equal to the values in all the nodes in that node's left subtree and is smaller than the values in all the nodes in that node's right subtree.

For example, for the following tree

- n1 (Value: 1, Left: null, Right: null)

- n2 (Value: 2, Left: n1, Right: n3)

- n3 (Value: 3, Left: null, Right: null)

call to IsValidBST(n2) should return true since a tree with root at n2 is a valid binary search tree. Explanation: Subtrees rooted at nodes n1 and n3 are valid binary search trees as they have no children. A tree rooted at node n2 is a valid binary search tree since its value (2) is larger or equal to the largest value in its left subtree (1, rooted at n1) and is smaller than the smallest value in its right subtree (3 rooted at n3).

using System;

public class Node

{

public int Value { get; set; }

public Node Left { get; set; }

public Node Right { get; set; }

public Node(int value, Node left, Node right)

{

Value = value;

Left = left;

Right = right;

}

}

public class BinarySearchTree

{

/// <summary>

/// Determines if the tree valid BS the specified root.

/// </summary>

/// <returns><c>true</c> if is valid BST; otherwise, <c>false</c>.</returns>

/// <param name="root">Root Node.</param>

public static bool IsValidBST(Node root)

{

if (root == null)

return(true);

if (root.Left!=null && MaxValue(root.Left) > root.Value)

return(false);

if (root.Right!=null && MinValue(root.Right) <= root.Value)

return(false);

if (!IsValidBST(root.Left) || !IsValidBST(root.Right))

return(false);

return(true);

}

/// <summary>

/// Finds the maximum value in the sub tree. This is a reccursive fuction

/// </summary>

/// <returns>The value.</returns>

/// <param name="node">Node.</param>

static int MaxValue (Node node)

{

int subNodeMaxValue =node.Right == null ? node.Value: MaxValue (node.Right);

int maxvalue = node.Value > subNodeMaxValue?node.Value : subNodeMaxValue ;

return maxvalue;

}

/// <summary>

/// Finds the minimums value in the sub tree. - This is a reccursive fuction

/// </summary>

/// <returns>Minimum value in the three</returns>

/// <param name="node">Node.</param>

static int MinValue (Node node)

{

int subNodeMinValue = node.Left == null ? node.Value: MinValue (node.Left);

int minValue = node.Value > subNodeMinValue?node.Value : subNodeMinValue ;

return minValue;

}

public static void Main(string[] args)

{

Node n1 = new Node(1, null, null);

Node n3 = new Node(3, null, null);

Node n2 = new Node(2, n1, n3);

Console.WriteLine(IsValidBST(n2));

}

}