**First Question**

Fix the bugs.

public class MathUtils

{

public static double Average(int a, int b)

{

return a + b / 2;

}

}

**Answer**:

public class MathUtils

{

/// <summary>

/// Find the average of two numbers

/// </summary>

/// <param name="a">First number.</param>

/// <param name="b">Second Number.</param>

public static double Average(int a, int b)

{

return (double) (a + b) / 2; //here double cast operator has //precedence over division, so (a+b) first converted to type double and //divide by 2 results in a double value

}

}

**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)

{

throw new NotImplementedException("Waiting to be implemented.");

}

public static void Main(string[] args)

{

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

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

}

}

**Answer:**

public class Path

{

public string CurrentPath { get; private set; }

public Path(string path)

{

this.CurrentPath = path;

}

/// <summary>

/// Change the directory to the new path specified

/// </summary>

/// <param name="newPath">New path.</param>

public Path Cd(string newPath)

{

Path retVal = null;

try

{

if (Regex.IsMatch(newPath, @"^\.{2}/[a-z]+")) //Check for valid path {

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

if (string.Compare (CurrentPath, "/") == 0) //checking root directory

{

//if CurrentPath is root directory, add the new directory name after root

string retPath = CurrentPath+DestPath.Replace("/",string.Empty);

retVal= new Path (retPath);

}

//Take the parent path from current path

var CurrentParentPath = CurrentPath.Substring (0, CurrentPath.LastIndexOf ('/'));

var returnPath= CurrentParentPath+DestPath;

retVal= new Path (returnPath);

}

else

{

Console.WriteLine("Invalid Path Entered");

}

}

catch (ApplicationException e)

{

Console.WriteLine("Exception: {0}", e.Message);

}

return retVal;

}

public static void Main (string[] args)

{

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

Path path = new Path("/");

Path newPath= path.Cd("../xyz");

if (newPath != null)

{

Console.WriteLine (newPath.CurrentPath);

Console.WriteLine(newPath.CurrentPath);

Console.WriteLine(newPath).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)

{

throw new NotImplementedException("Waiting to be implemented.");

}

public static void Main(string[] args)

{

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

}

}

**Answer**

public class Run

{

/// <summary>

/// To find the Index the of longest run in the given string

/// </summary>

/// <returns>Index of longest run.</returns>

/// <param name="str">String.</param>

public static int IndexOfLongestRun(string str)

{

int longestIndex = -1;

try

{

if(!string.IsNullOrEmpty(str))

{

Regex reg = new Regex(@"(\w)\1+"); //Expression for finding one or more run

MatchCollection matches = reg.Matches(str);

int longest = 0;

//Traverse through the collection of runs and find the longestIndex

foreach (Match match in matches)

{

if (longest < match.Length)

{

longest = match.Length;

longestIndex = match.Index;

}

}

}

else

{

Console.WriteLine("Error: Empty string");

}

return longestIndex;

}

catch (ApplicationException e)

{

Console.WriteLine("Exception: {0}", e.Message);

return longestIndex;

}

}

public static void Main (string[] args)

{

Console.WriteLine(Run.IndexOfLongestRun("abbcccddddcccbbadddd"));

Console.ReadLine ();

}

}

**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).

**Answer**

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 a tree is a valid BST or not.

/// </summary>

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

/// <param name="root">Root node of the tree.</param>

public static bool IsValidBST(Node root)

{

try

{

if (root == null)

return(true);

//check if the root node's value is less than or equal to left //child node's biggest value

if (root.Left != null)

{

int leftChildMaxVal = SubNodeMaxValue (root.Left);

if (root.Value < leftChildMaxVal)

return(false);

}

//check if the root node's value is greater than or equal to right //child node's smallest value

if (root.Right != null)

{

int rightChildMinVal = SubNodeMinValue (root.Right);

if (root.Value > rightChildMinVal)

return(false);

}

//Recursively checking the validity

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

return(false);

return(true);

}

catch (ApplicationException e)

{

Console.WriteLine("Exception: {0}", e.Message);

return(false);

}

}

/// <summary>

/// To find the biggest value of sub node

/// </summary>

/// <param name="node">Parent node of subtree</param>

/// <returns>biggest value of subtree</returns>

static int SubNodeMaxValue (Node node)

{

int childMaxVal = 0;

int maxvalue = 0;

if (node.Right == null)

{

childMaxVal = node.Value;

}

else

{

childMaxVal = SubNodeMaxValue (node.Right);

}

if (node.Value > childMaxVal)

{

maxvalue = node.Value;

}

else

{

maxvalue = childMaxVal;

}

return maxvalue;

}

/// <summary>

/// To find the smallest value of sub tree

/// </summary>

/// <param name="node">parent node of subtree</param>

/// <returns>smallest value of subtree</returns>

static int SubNodeMinValue (Node node)

{

int childMinValue = 0;

int minValue = 0;

if (node.Left == null)

{

childMinValue = node.Value;

}

else

{

childMinValue = SubNodeMinValue (node.Left);

}

if (node.Value < childMinValue)

{

minValue = node.Value;

}

else

{

minValue = childMinValue;

}

return minValue;

}

public static void Main (string[] args)

{

Node n4 = new Node (2, null, null);

Node n5 = new Node (10, null, null);

Node n6 = new Node (15, null, null);

Node n7 = new Node (25, null, null);

Node n1 = new Node (3, n4, n5);

Node n3 = new Node (20, n6, n7);

Node n2 = new Node (10, n1, n3);

Console.WriteLine(BinarySearchTree.IsValidBST(n2));

Console.ReadLine ();

}

}