CPSC 5031 - Homework 7

Name: David Nguyen

|  |
| --- |
| Description |
| Being Greedy Selected algorithm: Change-making problem – be able to input different kinds of coins |

|  |
| --- |
| Summary after completed the homework |
| Even this is the simple algorithm to work. However, I think this couple a base tool that can be used and implemented into a cashier system. The system can simply build and help cashiers quickly get result of how many coins the buyer can get back a change.   * Algorithm developed in C# * No add on support packages needed |

|  |
| --- |
| Assumption to run this algorithm or tool |
| 1. The algorithm can be run base on any currency (user has to modify the coin names and value of each coin before run it) 2. There is no limit number of changes amount (0 cent to more than 100 cents) 3. Tool only runs on Windows OS |

|  |
| --- |
| Source code |
| https://github.com/davednguyen/cpsc5031\_hw7.git |
| Branch |
| Develop |
| Main code project name: CPSCHomework7 |
| Test codes project name: MainTestProject |
| Homework write-up location: <https://github.com/davednguyen/cpsc5031_hw7/tree/develop/writeup> |
| (easy view) main codes: <https://github.com/davednguyen/cpsc5031_hw7/tree/develop/writeup> |
| (easy view) test codes: <https://github.com/davednguyen/cpsc5031_hw7/tree/develop/writeup> |

|  |  |  |
| --- | --- | --- |
| Total test cases: 10 | | |
| Test framework: MS Test | | |
| Language: C# | | |
| Test case name | Expected test result | Actual test result |
| HappyPath\_FullSetOfCoins\_with\_real\_change\_amount | Pass | passed |
| FullSetOfCoins\_Change\_amount\_is\_0 | Pass | passed |
| FullSetOfCoins\_Change\_amount\_is\_negative\_number | Pass | passed |
| FullSetOfCoins\_Change\_amount\_is\_101 | Pass | passed |
| FullSetOfCoins\_is\_null\_Change\_amount\_is\_50 | Pass | passed |
| SetOfCoins\_is\_null\_And\_CoinNames\_is\_null\_Change\_amount\_is\_50 | Pass | Passed |
| SetOfCoins\_is\_null\_And\_CoinNames\_is\_null\_Change\_amount\_is\_negative | Pass | Passed |
| SetOfCoins\_is\_null\_And\_CoinNames\_is\_null\_Change\_amount\_is\_0 | Pass | Passed |
| SetOfCoins\_with\_missing\_coins\_And\_CoinNames\_is\_full\_Change\_amount\_is\_75 | Pass | Passed |
| SetOfCoins\_And\_CoinNameswith\_with\_missing\_coinsName\_Change\_amount\_is\_75 | Pass | Passed |
|  | | |

|  |
| --- |
| Coin list (US currency) |
|  |
| Coin names (US currency) |
|  |

|  |
| --- |
| Change-making algorithm codes |
| namespace CPSCHomework7  {  public class Program  {  static void Main(string[] args)  {  var coins = new Dictionary<string, int>()  {  { "Penny", 1 },  { "Dime", 10 },  { "Nickel", 5},  { "Quarter", 25},  { "Half Dollar", 50}  };  Dictionary<string, int> coins\_short\_list = new Dictionary<string, int>()  {  { "Penny", 1 },  { "Dime", 10 }  };  var names = new string[] { "Penny", "Dime", "Nickel", "Quarter", "Half Dollar"};  //for(int i = 0; i < 100; i++)  //{  // Console.WriteLine(CalculateChange(coins, names, i));  //}  //Console.WriteLine(CalculateChange(null, names, 50));  Console.WriteLine(CalculateChange(coins\_short\_list, names, 50));  }  /// <summary>  /// Main function to calculate changes based on coin  /// </summary>  /// <param name="coins">list of coins</param>  /// <param name="names">list of coin names</param>  /// <param name="change">required changes</param>  /// <returns></returns>  public static string CalculateChange(Dictionary<string, int> coins, string[]names, int change)  {  var calculateChange = GetChanges(coins, change);  var total = TotalCoins(calculateChange, names);  var display = DisplayCoins(calculateChange, names, change);  return "Total coin: "+ total +" " + display;  }  /// <summary>  /// Main function to calculate changes based on coin - for testing purposes  /// </summary>  /// <param name="coins">list of coins</param>  /// <param name="names">list of coin names</param>  /// <param name="change">required changes</param>  /// <returns></returns>  public string CalculateChanges(Dictionary<string, int> coins, string[] names, int change)  {  return CalculateChange(coins, names, change);  }  /// <summary>  /// function to provide list of coins and change needed  /// </summary>  /// <param name="coins">list of available coins</param>  /// <param name="change">required change</param>  /// <returns></returns>  public static Dictionary<string, int> GetChanges(Dictionary<string, int> coins, int change)  {  if (change != 0 && coins != null && coins.Count > 0)  {  //save possilbe change calculation  var possibleChangeResults = new Dictionary<string, int>();  //order the coin values decending values  coins.OrderByDescending((value) => value.Value);  //search through list of coins find possible changes amount  foreach (var coin in coins.OrderByDescending((value) => value.Value))  {  //update total coin needed for a change  int totalCoin = 0;  while (change >= coin.Value)  {  change = change - coin.Value;  totalCoin++;  }  possibleChangeResults.Add(coin.Key, totalCoin);  }  return possibleChangeResults;  }  else  {  return null;  }  }  /// <summary>  /// list for total coins needed for a change  /// </summary>  /// <param name="coins">list of coins</param>  /// <param name="coinName"> list of coin names</param>  /// <returns>total number of coins based on each type</returns>  public static int TotalCoins(Dictionary<string, int> coins, string[] coinName)  {  if(coins != null && coinName != null)  {  int total = 0;  for (int i = 0; i < coinName.Length; i++)  {  string name = coinName[i];  if (coins.ContainsKey(name))  {  total = total + coins[name];  }  }  return total;  }  else  {  return 0;  }  }  /// <summary>  /// Display how many coins need of each type.  /// </summary>  /// <param name="coins"> list of coins needed</param>  /// <param name="coinName"> list of coin names</param>  /// <param name="change">change amount required</param>  /// <returns>String - display each coin type needed</returns>  public static string DisplayCoins(Dictionary<string, int> coins, string[] coinName, int change)  {  if(coins != null && coins.Count > 0 && change > 0 && coinName != null)  {  string totalCoin = "Change amount: " + change;  for (int i = 0; i < coinName.Length; i++)  {  string name = coinName[i];  if (coins.ContainsKey(name))  {  totalCoin = totalCoin + " coin name: " + name + ":" + coins[name];  }  }  return totalCoin;  }  else  {  return null;  }  }  }  } |

|  |
| --- |
| Test codes |
| using Microsoft.VisualStudio.TestTools.UnitTesting;  using CPSCHomework7;  using System.Collections.Generic;  namespace MainTestProject  {  [TestClass]  public class Main  {  //global set of coins and names  Dictionary<string, int> coins = new Dictionary<string, int>()  {  { "Penny", 1 },  { "Dime", 10 },  { "Nickel", 5},  { "Quarter", 25},  { "Half Dollar", 50}  };  //global set of coins and names  Dictionary<string, int> coins\_short\_list = new Dictionary<string, int>()  {  { "Penny", 1 },  { "Dime", 10 }  };  string[] names = new string[] { "Penny", "Dime", "Nickel", "Quarter", "Half Dollar" };  string[] names\_shortlist = new string[] { "Penny", "Dime", "Quarter"};  [TestMethod]  public void HappyPath\_FullSetOfCoins\_with\_real\_change\_amount()  {  string expected = "Total coin: 8 Change amount: 99 coin name: Penny:4 coin name: Dime:2 coin name: Nickel:0 coin name: Quarter:1 coin name: Half Dollar:1";  Program program = new Program();  string test = program.CalculateChanges(coins, names, 99);  Assert.AreEqual(expected, test);  }  [TestMethod]  public void FullSetOfCoins\_Change\_amount\_is\_0()  {  string expected = "Total coin: 0 ";  Program program = new Program();  string test = program.CalculateChanges(coins, names, 0);  Assert.AreEqual(expected, test);  }  [TestMethod]  public void FullSetOfCoins\_Change\_amount\_is\_negative\_number()  {  string expected = "Total coin: 0 ";  Program program = new Program();  string test = program.CalculateChanges(coins, names, -1);  Assert.AreEqual(expected, test);  }  [TestMethod]  public void FullSetOfCoins\_Change\_amount\_is\_101()  {  string expected = "Total coin: 3 Change amount: 101 coin name: Penny:1 coin name: Dime:0 coin name: Nickel:0 coin name: Quarter:0 coin name: Half Dollar:2";  Program program = new Program();  string test = program.CalculateChanges(coins, names, 101);  Assert.AreEqual(expected, test);  }  [TestMethod]  public void FullSetOfCoins\_is\_null\_Change\_amount\_is\_50()  {  string expected = "Total coin: 0 ";  Program program = new Program();  string test = program.CalculateChanges(null, names, 50);  Assert.AreEqual(expected, test);  }  [TestMethod]  public void SetOfCoins\_is\_null\_And\_CoinNames\_is\_null\_Change\_amount\_is\_50()  {  string expected = "Total coin: 0 ";  Program program = new Program();  string test = program.CalculateChanges(null, null, 50);  Assert.AreEqual(expected, test);  }  [TestMethod]  public void SetOfCoins\_is\_null\_And\_CoinNames\_is\_null\_Change\_amount\_is\_negative()  {  string expected = "Total coin: 0 ";  Program program = new Program();  string test = program.CalculateChanges(null, null, -1);  Assert.AreEqual(expected, test);  }  [TestMethod]  public void SetOfCoins\_is\_null\_And\_CoinNames\_is\_null\_Change\_amount\_is\_0()  {  string expected = "Total coin: 0 ";  Program program = new Program();  string test = program.CalculateChanges(null, null, 0);  Assert.AreEqual(expected, test);  }  [TestMethod]  public void SetOfCoins\_with\_missing\_coins\_And\_CoinNames\_is\_full\_Change\_amount\_is\_75()  {  string expected = "Total coin: 12 Change amount: 75 coin name: Penny:5 coin name: Dime:7";  Program program = new Program();  string test = program.CalculateChanges(coins\_short\_list, names, 75);  Assert.AreEqual(expected, test);  }  [TestMethod]  public void SetOfCoins\_And\_CoinNameswith\_with\_missing\_coinsName\_Change\_amount\_is\_75()  {  string expected = "Total coin: 1 Change amount: 75 coin name: Penny:0 coin name: Dime:0 coin name: Quarter:1";  Program program = new Program();  string test = program.CalculateChanges(coins, names\_shortlist, 75);  Assert.AreEqual(expected, test);  }  }  } |