// The Name Game takes user input, reverses it and prints it to screen.

// Taken from Bob Tabor C# video on MVA. Modified slightly by Trevor Heehs.

using System;

namespace BobMethods

{

class Methods

{

static void Main()

{

string begin = "WELCOME TO THE THE NAME GAME"; // Define vars.

string gameOver = "YOU ARE DONE CONGRATS";

Intro(begin); //Call Intro Method and print to console.

Console.Write("What's Your First Name? "); // Prompt user…

string firstName = Console.ReadLine(); // Assign input to var.

Console.Write("What's Your Last Name? ");

string LastName = Console.ReadLine();

Console.Write("In what city were you born in? ");

string cityName = Console.ReadLine();

string revFirst = ReverseString(firstName); // Reverse inputs and

string revLast = ReverseString(LastName); // assign to new var.

string revCity = ReverseString(cityName);

PrintResult(revFirst, revLast, revCity); // Call print method.

End(gameOver); // Call End method and printing outro.

}

private static void Intro(string b) // Intro method prints intro.

{

Console.WriteLine(b);

}

private static string ReverseString(string m)

// Reverse method accepts string as argument.

{

char[] charArr = m.ToCharArray(); // Create char Array.

Array.Reverse(charArr); // Reverse char Array.

return String.Concat(charArr);

//Returns flipped and joined Array.

}

private static void PrintResult(string f, string l, string c)

{

Console.WriteLine(String.Format($"{f} {l} {c}");

// Print Result Method returns result of flipped strings.

// added modern string interpolation for readability.

}

private static void End(string g) // End method prints outro.

{

Console.WriteLine(g);

}

}

}

// Due to not declaring a separate class for methods, all of these methods.

// are static members of the base class containing Main Method.

// For all numbers 1 to 100:

// If number is divisible by 5, print "Fizz"

// If number is divisible by 3, print "Buzz"

// If number is divisible by both 3 and 5, print "FizzBuzz"

// I added print "Pop" for all other numbers for continuity.

// I added "number: " using string interpolation for clarity.

// I placed program into a Method for execution.

// In alternate version I created the method in a seperate class.

// The program is a for loop with nested if else statement.

// Trevor Heehs

namespace FizzBang

{

class Program

{

static void Main()

{

FizzBuzz(); // Call FizzBuzz Method at execution.

void FizzBuzz() // FizzBuzz Method declaration.

{

for (int i = 1; i <= 100; i++) // For loop.

{

if (i % 3 == 0 && i % 5 == 0) // Nested if else.

{

Console.WriteLine($"{i}: FizzBuzz");

}

else if (i % 5 == 0)

{

Console.WriteLine($"{i}: Fizz");

}

else if (i % 3 == 0)

{

Console.WriteLine($"{i}: Buzz");

}

else

{

Console.WriteLine($"{i}: Pop");

}

}

}

}

}

}

// Same “FizzBuzz” program written as a Node.js console app in JS.

// Author Trevor Heehs 10/31/18 (Happy Haloween!).

FizzBuzz();

function FizzBuzz() {

for (let i = 1; i <= 100; i++) {

if (i % 5 == 0 && i % 3 == 0) {

console.log(`${i}: FizzBuzz`);

}

else if (i % 5 == 0) {

console.log(`${i}: Fizz`);

}

else if (i % 3 == 0) {

console.log(`${i}: Buzz`);

}

else {

console.log(`${i}: Pop`);

}

}

}

process.stdin.resume(); // Added to prevent immediate console close.

// Program creates UnitConverter Type used on Conversion Objects.

// Prints using string interpolation.

// Everything in program works using OOP principles, no static members.

// Our CS 101 teacher, Harold, asked how long our hair grows in Miles.

// Original code snippet from C# 4.0 in a Nutshell, Ch.1 p.14

// Reworked to convert inches (hair growth length) to miles by Trevor Heehs.

// 10/31/2018

namespace Haircut

{

class Program

{

static void Main(string[] args)

{

UnitConverter feetToInchesConverter = new UnitConverter(12);

// Instantiate UnitConverter object called feetToInches.

UnitConverter milesToFeetConverter = new UnitConverter(5280);

UnitConverter inchesToMilesConverter = new UnitConverter(63360);

// Create inchesToMiles object for Hair Length in Miles.

Console.WriteLine($"1 foot is:

{feetToInchesConverter.Convert(1)} inches");

Console.WriteLine($"1 mile is:

{milesToFeetConverter.Convert(1)} feet");

Console.WriteLine($"1 mile is: {feetToInchesConverter.Convert(milesToFeetConverter.Convert(1))} inches");

// Inner method called first, then outer method.

Console.WriteLine($"1 inch of hair is: {inchesToMilesConverter.ConvertMiles(1).ToString("E10")} miles.");

// Printed using exponential form.

}

}

// Created UnitConverter Type to handle all conversions.

class UnitConverter

{

private int ratio; // Field.

public UnitConverter (int unitRatio) // Constructor for Type.

{

ratio = unitRatio;

}

public int Convert (int unit) // Convert Method declaration.

{

return unit \* ratio;

}

public decimal ConvertMiles(int numberOfInches)

{

decimal result = (decimal)numberOfInches / (decimal)ratio;

// Inches to Miles required division not multiplication, so

// declared new method.

// Cast to decimal required for printing as Exponential.

return result;

}

}

}

/\* Coding problem from Eloquent Javascript 3rd Edition by Marijn Haverbeke. Solution not provided.

Adapted to C# by Trevor Heehs.

"Write a program that creates a string that represents an 8×8 grid, using newline characters to separate lines.

At each position of the grid there is either a space or a "#" character. The characters should form a chessboard.

Passing this string to console.log should show something like this..."

# # # #

# # # #

# # # #

# # # #

# # # #

# # # #

# # # #

# # # #

\*/

namespace Chessboard

{

class Program

{

static void Main(string[] args)

{

const int MAX\_SIZE = 8; // Variables declared globally to allow access inside multi-level nested structure.

string board = "";

string black = "#";

string white = " ";

for (int lineCount = 0; lineCount < MAX\_SIZE; lineCount++)

// Outer loop creates multiple lines: board.

{

for (int spaceCount = 0; spaceCount < MAX\_SIZE; spaceCount++)

// Inner loop creates single line.

{

if (lineCount % 2 == 0)

// If line number is even: runs the nested statement and print line.

if (spaceCount % 2 ==0) // Nested if-else determines even/odd space number and prints accordingly.

{

board += white;

}

else

{

board += black;

}

else

// Else: runs if line number is odd and offsets the same line effect.

{

if (spaceCount % 2 == 0) // If line number is odd nested if-else prints line and offsets values.

{

board += black;

}

else

{

board += white;

}

}

}

board += "\n"; // After inner loop prints a line and inner loop terminates, newline is printed and outer loop repeats.

}

Console.WriteLine(board); // After 8 lines are printed, loop exits and board is printed.

}

}

}

/\* Simple coding problem from Eloquent Javascript, 3rd Edition by Marijn Haverbeke. Solution not provided. Adapted to C# by Trevor Heehs.

"Write a loop that makes seven calls to console.log to output the following triangle..."

#

##

###

####

#####

######

#######

\*/

const int MAX\_LENGTH = 7;

string triangle = "";

for (int i = 0; i <= MAX\_LENGTH; i++)

{

while (triangle.Length < i)

// This while statement sets limit on number of printed characters.

{

triangle += "#";

Console.WriteLine(triangle);

}

}

}

}

}

/\* C# Program to create a Hangman Game from:

https://www.sanfoundry.com/csharp-programs-gaming-hangman/

Two flaws in program I changed:

1) Program does not recognize capital letters. Fixed with ToString().

2) Program throws System.FormatException when multiple characters are entered as input.

Fixed by including try/catch block.

3) This caused char playerGuess to be out of scope inside try/catch block.

Fixed by declaring playerGuess in global scope as empty char ' '.

Trevor Heehs

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Hangman

{

class Program

{

static void Main()

{

Console.WriteLine("Welcome to Hangman!!!!!!!!!!"); // Welcome message.

string[] listWords = new string[10];

// Declare and initilize an array of 10 strings.

listWords[0] = "sheep";

// This array could be initilized more tersely.

listWords[1] = "goat";

listWords[2] = "computer";

listWords[3] = "america";

listWords[4] = "watermelon";

listWords[5] = "icecream";

listWords[6] = "jasmine";

listWords[7] = "pineapple";

listWords[8] = "orange";

listWords[9] = "mango";

Random randGen = new Random();

// Instantiates Random object for use.

var randomNumber = randGen.Next(0, 9);

// Selects a random number between 0 and 9 and assigns to variable.

string mysteryWord = listWords[randomNumber];

// Uses random number to choose word from listWords Array and assign to variable.

char[] guess = new char[mysteryWord.Length];

// Assigns letter count of word to char Array.

Console.Write("Please enter your guess: "); // Prompt user input

for (int p = 0; p < mysteryWord.Length; p++)

// Loops through mysteryWord and prints \* for all chars in array.

{

guess[p] = '\*';

}

char playerGuess = ' ';

// Moved playerGuess declaration into global scope and initialized to empty char.

while (true)

{

// Added try/catch block to hanlde FormatException thrown when user enters multiple characters.

try

{

playerGuess = char.Parse(Console.ReadLine().ToLower());

}

catch (System.FormatException) // Catches exception here.

{

Console.WriteLine("Only one character input!!!"); // And handles it!

}

for (int j = 0; j < mysteryWord.Length; j++)

// Loops through mystery word and matches user guess.

{

if (playerGuess == mysteryWord[j])

// If user guess = any char in mystrery word array...

guess[j] = playerGuess;

// Assign correct user guess to char array mysteryWord...

}

Console.WriteLine(guess);

// Print result of guesses and repeat loop until all characters are guessed.

//No conclusion statement: You won! }

}

}

}

/\*

Here are three versions of the Fibonacci Sequence. One using iteration.

The other using method recursion. Inspired by the book: If Hemingway wrote JavaScript by Angus Croll.

Source code is from Prakash Tripathi

https://www.c-sharpcorner.com/UploadFile/19b1bd/calculate-fibonacci-series-in-various-ways-using-C-Sharp/

Very few modifications were made to source code as these tended to break the program.

Added while loop to explore implementation.

Trevor Heehs

\*/

using System;

namespace FibonacciSeq

{

class Program

// All (3) Fibonacci methods are declared statically and called by class from Main.

// Newline escape characters used for output presentation.

{

static void Main(string[] args)

{

Console.WriteLine("The Fibonacci For Loop: \n");

FibFor.DoTheForFib(25); // Call to For Loop Method.

Console.WriteLine("\n");

Console.WriteLine("The Fibonacci While Loop: \n");

FibWhile.DoTheWhileFib(25); // Call to While Loop Method.

Console.WriteLine("\n");

Console.WriteLine("The Fibonacci Sequence Recursive Method: \n");

FibRec.DoTheRecursiveFib(25); // Call to Recursive Method.

Console.WriteLine("\n");

}

}

}

// This is the cleanest example in my opinion and the easiest to read.

namespace FibonacciSeq

{

class FibFor

{

public static void DoTheForFib(int lengthOfFib)

{

// Declare 3 starting variables.

// Start the sequence at a(0) and b(1).

// c is initialized to default(0) to hold the sum and a and b.

int a = 0, b = 1, c;

Console.Write($"{a} {b}"); // Print 0 and 1.

// The loop's i must be initialized at 2 due to 0 and 1 being set outside the loop.

// If initialized at zero sequence will exceed lengthOfFib by 2.

for (int i = 2; i < lengthOfFib; i++)

{

c = a + b; // Set variable c to the sum of a and b.

Console.Write($" {c}"); // Print c to sequence...

a = b; // Bind variable a to the value of b.

b = c; // Bind variable b to value of c.

// Repeat loop [argument 'lengthOfFib'] number of times.

}

}

}

}

namespace FibonacciSeq

{

// While loop version operates similar to for loop.

// Requires the declaration of additional counter variable.

// Counter is initialized at 2 due to 0 and 1 outside loop.

// Not as easy to read as for loop version.

class FibWhile

{

public static void DoTheWhileFib(int length)

{

int a = 0, b = 1, c, counter = 2;

Console.Write($"{a} {b}");

while (counter < length)

{

c = a + b;

Console.Write($" {c}");

a = b;

b = c;

counter++;

}

}

}

}

namespace FibonacciSeq

{

// The recursive method is unnecessarily complicated.

// For implementation I'd stick to the much simpler for loop.

class FibRec

{

// Create the method called during execution.

// In the body of the called method make called to worker method and pass arguments.

public static void DoTheRecursiveFib(int len)

{

FibRecursiveWork(0, 1, 1, len);

}

// Create the work method...this is where the magic happens.

// Arguments are provided from calling method and the values are worked by recursive method

// declared inside method body.

// If statement ends recursive loop when false.

private static void FibRecursiveWork(int a, int b, int counter, int len)

{

if (counter <= len)

{

// Counter ++ causes infinite loop.

Console.Write($"{a} ");

FibRecursiveWork(b, a + b, counter + 1, len);

}

}

}

}

/\*

This program is from a pluralsight course with Jesse Liberty:

https://www.pluralsight.com/courses/csharp-from-scratch

I modified the object initialization syntax for single line declaration.

I included a foreach loop instead of for loop to iterate over List<T>.

The program goes over key OOP topics. Inheritance, encapsulation, polymorphism.

Trevor Heehs

\*/

using System;

using System.Collections.Generic;

namespace LibertyOOP

{

class Program

{

static void Main(string[] args)

{

// Instantiates 3 new Employee objects of type worker/manager and passes args.

Employee bob = new Worker("Bob", 35.00);

Employee joe = new Manager("Joe", true);

Employee sally = new Worker("Sally", 27.50);

// Creates a List of type Employeee containing 3 employee objects.

// I altered by initializing on one line as opposed to multiples lines.

List<Employee> employees = new List<Employee> { joe, bob, sally };

/\* Original code included for loop. I created a foreach loop.

for (int i = 0; i < employees.Count; i++)

{

employees[i].TakeVacation();

Console.WriteLine(employees[i]);

}

\*/

foreach (var employee in employees)

{

employee.TakeVacation();

Console.WriteLine($"{employee}");

}

}

}

// Using Employee class, define the Employee type...

public class Employee

{

// Property with getter and setter.

public string Name { get; set; }

// Public field?

public double vacationDays;

// Virtual Method declaration. No body provided.

// Method will be inherited by child class and provided with a body then.

public virtual void TakeVacation() { }

// Employee Constructor (constructors always have same name as type/class).

// This "instance" constructor creates a new Employee object.

public Employee(string name)

{

Name = name; // Gets and Sets Employee name and assigns to the Name property.

}

public override string ToString()

// Define how employee should be presented when ToString is called.

{

return $"[Employee: Name = {Name}]";

}

}

// The Worker class inherits from base (Employee) class

public class Worker : Employee // Inherits from base (Employee) class

{

public double HourlyWage { get; set; } // Property

// Instance Constructor creates a new worker object.

// Inherits "name" from base (Employee) class.

// Assigns wage to HourlyWage property. This property is unique to worker class.

public Worker (string name, double wage)

: base(name)

{

HourlyWage = wage;

}

// Child Worker class overrides method inherited from Employee class.

public override void TakeVacation()

{

vacationDays += 10;

}

// Define how Worker object should be presented when ToString is called.

public override string ToString()

{

return $"[Worker Name: {Name} HourlyWage: {HourlyWage} Vacation: {vacationDays}]";

}

}

// Child Manager class inherits from base (Employee) class.

public class Manager : Employee

{

public bool CompanyCar { get; set; } // Property

// Manager constructor, creates manager object.

// Inherits "name" from base (Employee) class

public Manager(string name, bool hasCar)

:base(name)

{

CompanyCar = hasCar;

}

// Overrides vactation method in a different manner than for Employee

public override void TakeVacation()

{

vacationDays += 15;

}

// Define how Manager should be presented when ToString is called.

public override string ToString()

{

return $"[Manager Name: {Name} HasCar: {CompanyCar} Vacation: {vacationDays}]";

}

}

}

/\* This is my first contribution to open source.

\* The published code can be found at:

\* https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/strings/

\* Written by: Trevor Heehs

\* Simple programming showing two ways to print format strings.

\* One using String.Format method, the other using string interpolation.

\*/

using System;

namespace StringCompare

{

class Program

{

static void Main()

{

// Used Named Value.Tuple (new in C# 7.1) for variable declaration.

// Print using String interpolation.

var jh = (firstName: "Jupiter", lastName: "Hammon", born: 1711, published: 1761);

Console.WriteLine($"{jh.firstName} {jh.lastName} was an African American poet born in {jh.born}.");

Console.WriteLine($"He was first published in {jh.published} at the age of {jh.published - jh.born}.");

Console.WriteLine($"He'd be over {Math.Round((2018d - jh.born) / 100d) \* 100d} years old today.");

// Output:

// Jupiter Hammon was an African American poet born in 1711.

// He was first published in 1761 at the age of 50.

// He'd be over 300 years old today.

// </SnippetStringInterpolation>

System.Console.WriteLine();

// Print using `Format.String` method.

var pw = (firstName: "Phillis", lastName: "Wheatley", born: 1753, published: 1773);

Console.WriteLine("{0} {1} was an African American poet born in {2}.", pw.firstName, pw.lastName, pw.born);

Console.WriteLine("She was first published in {0} at the age of {1}.", pw.published, pw.published - pw.born);

Console.WriteLine("She'd be over {0} years old today.", Math.Round((2018d - pw.born) / 100d) \* 100d);

// Output:

// Phillis Wheatley was an African American poet born in 1753.

// She was first published in 1773 at the age of 20.

// She'd be over 300 years old today.

// </SnippetStringFormat>

}

}

}

// Here’s the associated content published with above code.

A format string is a string whose contents are determined dynamically at runtime. Format strings are created by embedding \*interpolated expressions\* or placeholders inside of braces within a string. Everything inside the braces (`{...}`) will be resolved to a value and output as a formatted string at runtime. There are two methods to create format strings: string interpolation and composite formatting.

### String Interpolation

Available in C# 6.0 and later, [\*interpolated strings\*](../../language-reference/tokens/interpolated.md) are identified by the `$` special character and include interpolated expressions in braces. If you are new to string interpolation, see the [String interpolation - C# interactive tutorial](../../tutorials/intro-to-csharp/interpolated-strings.yml) for a quick overview.

Use string interpolation to improve the readability and maintainability of your code. String interpolation achieves the same results as the `String.Format` method, but improves ease of use and inline clarity.

[!code-csharp[csProgGuideFormatStrings](~/samples/snippets/csharp/programming-guide/strings/Strings\_1.cs#StringInterpolation)]

### Composite Formatting

The <xref:System.String.Format%2A?displayProperty=nameWithType> utilizes placeholders in braces to create a format string. This example results in similar output to the string interpolation method used above.

[!code-csharp[csProgGuideFormatStrings](~/samples/snippets/csharp/programming-guide/strings/Strings\_1.cs#StringFormat)]

For more information on formatting .NET types see [Formatting Types in .NET](../../../standard/base-types/formatting-types.md).

/\* This program implements three different methods to reverse strings.

\* The RevString1 method uses no built-in methods with a foreach loop.

\* The RevString2 method uses no built-in methods with a for loop.

\* The RevString3 method uses the built-in Array.Reverse method.

\* Source code: www.stackoverflow.com/questions/16962322/

\* Trevor Heehs

\*/

using System;

namespace ReverseString

{

class ReverseStringsThreeWays

{

static void Main()

{

// I’ve excluded a robust UI, focusing more on method logic.

Console.WriteLine(RevString1("Hello World"));

Console.WriteLine(RevString2("Hello Galaxy"));

Console.WriteLine(RevString3("Hello Universe"));

}

// RevString1 method takes input string and reverses from front to back.

static string RevString1(string s)

{

// Declare empty string to store reversed string.

string result = string.Empty;

// For “Hello” the loop will do the following...

// 1. Result = ‘H’ + string.Empty -> H

// 2. Result = ‘e’ + “H” -> eH

// 3. Result = ‘l’ + “eH” -> leH

// …continues until result = “olleH”

foreach (char c in s)

{

result = c + result;

}

return result;

}

// RevString2 method reverses the RevString1 approach and works back to front.

static string RevString2(string s)

{

string result = string.Empty;

// For “Hello” the loop will do the following.

// 1. Result = string.Empty + ‘o’ -> o

// 2. Result = ‘o’ + ‘l’ -> ol

// 3. Result = ‘ol’ + ‘l’ -> oll

// …continues until result = “olleH”

for (int i = s.Length - 1; i >= 0; i--)

{

result += s[i];

}

return result ;

}

// RevString3 method uses built-in methods, “the normal way”.

static string RevString3(string s)

{

// Convert string to char array.

char[] arr = s.ToCharArray();

// Use built in Array.Reverse method to flip the array.

Array.Reverse(arr);

// Return reversed array back to a new string.

return new string(arr);

}

}

}

/\* This program prints duplicate characters from an input string.

\* Many examples remove duplicate characters or detect them.

\* My goal was to store duplicates and print them.

\* Modified source code:

\* <https://www.csharpstar.com/csharp-program-print-first-non-repeated-character-from->string/.

\*/ Trevor Heehs.

using System;

namespace Dupes

{

public class Program

{

public static void Main()

{

// Call method and store result in variable.

string extract1 = ExtractDupes("Google");

// Print variable.

Console.WriteLine(extract1);

}

// Declare method with string input.

public static string ExtractDupes(string input)

{

string result = string.Empty;

// Outer loop loops through each index position in input string. (0, 1, 2..)

for (int i = 0; i < input.Length; i++)

{

// Inner loop compares each index position to a single index position

// of outer loop. 0(i) compares to 0,1,2,3..(j).

for (int j = 0; j < input.Length; j++)

{

// If value of char in outer loop (G) == value of char in inner loop.

// AND the value of those two index position ints are NOT equal…

// NOTE: && will short circuit and loop will continue if values !=.

if ((input[i] == input[j]) && i != j)

{

// Add the value of char from inner loop then

// BREAK out of inner loop and go back to outer loop.

// Writing [j] or [i] to result works, [j] is more readable to me.

result += input[j];

break;

}

}

}

// return string of duplicate chars...

return result;

}

}

/\* Simple program that highlights instance members vs static members.

\* Source C# 7.0 Pocket Reference, Albahari & Albahari, p.15

\* Added public Properties and made Fields private.

\*/ Trevor Heehs

using System;

namespace StaticTest

{

class Program

{

private static void Main()

{

// Create two instances of Panda.

Panda p1 = new Panda("Pan Dee");

Panda p2 = new Panda("Pan Dah");

// Access instance members from object instance.

Console.WriteLine(p1.Name); // Pan Dee

Console.WriteLine(p2.Name); // Pan Dah

// Access static members from the class.

Console.WriteLine(Panda.Population); // 2

}

}

public class Panda

{

// Instance Field

private string name;

// Static Field

private static int population;

// Instance Property

public string Name { get; set; }

// Static Property

public static int Population { get; set; }

// Panda constructor

public Panda(string n)

{

Name = n;

Population = Population + 1;

}

}

}