|  |
| --- |
| using System.Collections.Generic; |
|  | using MerchantGuideToGalaxy.Romans; |
|  |  |
|  | namespace MerchantGuideToGalaxy.Contexts |
|  | { |
|  | internal class Context |
|  | { |
|  | public Context() |
|  | { |
|  | Primitives = new Dictionary<string, RomanPrimitive>(); |
|  | Units = new Dictionary<string, double>(); |
|  | Questions = new List<string>(); |
|  | } |
|  |  |
|  | public string PrimUnit { get; set; } |
|  | public Dictionary<string, RomanPrimitive> Primitives { get; set; } |
|  | public Dictionary<string, double> Units { get; set; } |
|  | public List<string> Questions { get; set; } |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662/raw/cc1621d7fa32695090115b1d52f4714a54b9c0db/Context.cs)[**Context.cs**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662#file-context-cs) hosted with ❤ by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | using MerchantGuideToGalaxy.Contexts; |
|  |  |
|  | namespace MerchantGuideToGalaxy.Parsers |
|  | { |
|  | internal abstract class Parser |
|  | { |
|  | public Parser(Context ctx) |
|  | { |
|  | Context = ctx; |
|  | } |
|  |  |
|  | public Context Context { get; private set; } |
|  |  |
|  | public abstract bool Parse(string input); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662/raw/cc1621d7fa32695090115b1d52f4714a54b9c0db/Parser.cs)[**Parser.cs**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662#file-parser-cs) hosted with ❤ by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | using System; |
|  | using System.Linq; |
|  | using MerchantGuideToGalaxy.Contexts; |
|  | using MerchantGuideToGalaxy.Romans; |
|  |  |
|  | namespace MerchantGuideToGalaxy.Parsers |
|  | { |
|  | internal class PrimitiveParser : Parsers.Parser |
|  | { |
|  | public PrimitiveParser(Context ctx) |
|  | : base(ctx) |
|  | { |
|  | } |
|  |  |
|  | public override bool Parse(string input) |
|  | { |
|  | string[] lexers = input.Split(new[] {" is "}, StringSplitOptions.RemoveEmptyEntries); |
|  | if (lexers.Count() != 2) |
|  | return false; |
|  | if (lexers[1].Length > 1) |
|  | return false; |
|  | RomanPrimitive roman = RomanPrimitive.Parse(lexers[1][0]); |
|  | if (roman == null) |
|  | { |
|  | throw new Exception("syntex error."); |
|  | } |
|  | string name = lexers[0].Trim(); |
|  | Context.Primitives[name] = roman; |
|  | return true; |
|  | } |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662/raw/cc1621d7fa32695090115b1d52f4714a54b9c0db/PrimitiveParser.cs)[**PrimitiveParser.cs**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662#file-primitiveparser-cs) hosted with ❤ by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | using System; |
|  | using MerchantGuideToGalaxy.Contexts; |
|  |  |
|  | namespace MerchantGuideToGalaxy.Solvers |
|  | { |
|  | class PrimitiveSolver : Solver |
|  | { |
|  | public PrimitiveSolver(Context ctx) |
|  | : base(ctx) |
|  | { |
|  | } |
|  |  |
|  | public override bool Solve(string question, out string answer) |
|  | { |
|  | string qualifier = "how much is"; |
|  | if (!question.StartsWith(qualifier)) |
|  | { |
|  | answer = null; |
|  | return false; |
|  | } |
|  | string body = question.Substring(qualifier.Length + 1); |
|  | int value = Romans.Roman.Parse(body, Context.Primitives).Calculate(); |
|  | answer = value.ToString(); |
|  | Console.WriteLine(body + " is " + answer); |
|  | return true; |
|  | } |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662/raw/cc1621d7fa32695090115b1d52f4714a54b9c0db/PrimitiveSolver.cs)[**PrimitiveSolver.cs**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662#file-primitivesolver-cs) hosted with ❤ by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using MerchantGuideToGalaxy.Contexts; |
|  | using MerchantGuideToGalaxy.Parsers; |
|  | using MerchantGuideToGalaxy.Solvers; |
|  |  |
|  | namespace MerchantGuideToGalaxy |
|  | { |
|  | internal class Problem |
|  | { |
|  | public static void Process(List<string> input) |
|  | { |
|  | Parsers.Parser[] parsers; |
|  | Solver[] solvers; |
|  | Context ctx = Initialize(out parsers, out solvers); |
|  |  |
|  | ParseValues(input, parsers); |
|  | ProcessQuestion(ctx, solvers); |
|  | Console.ReadKey(); |
|  | } |
|  |  |
|  | private static Context Initialize(out Parsers.Parser[] parsers, out Solver[] solvers) |
|  | { |
|  | var ctx = new Context(); |
|  |  |
|  | parsers = new Parsers.Parser[] |
|  | { |
|  | new PrimitiveParser(ctx), |
|  | new UnitParser(ctx), |
|  | new QuestionParser(ctx) |
|  | }; |
|  |  |
|  | solvers = new Solver[] |
|  | { |
|  | new PrimitiveSolver(ctx), |
|  | new UnitSolver(ctx) |
|  | }; |
|  | return ctx; |
|  | } |
|  |  |
|  | private static void ParseValues(List<string> input, IEnumerable<Parsers.Parser> parsers) |
|  | { |
|  | input |
|  | .ForEach(cmd => parsers.ToList() |
|  | .ForEach(parser => |
|  | { |
|  | try |
|  | { |
|  | parser.Parse(cmd); |
|  | } |
|  | catch |
|  | { |
|  | } |
|  | })); |
|  | } |
|  |  |
|  | private static void ProcessQuestion(Context ctx, IEnumerable<Solver> solvers) |
|  | { |
|  | ctx.Questions.ForEach(cmd => |
|  | { |
|  | var solved = false; |
|  | solvers.ToList().ForEach(solver => |
|  | { |
|  | string answer = string.Empty; |
|  | try |
|  | { |
|  | if (solver.Solve(cmd, out answer)) |
|  | { |
|  | solved = true; |
|  | } |
|  | } |
|  | catch |
|  | { |
|  | } |
|  | }); |
|  | if (!solved) |
|  | { |
|  | Console.WriteLine("I have no idea what you are talking about"); |
|  | } |
|  | }); |
|  | } |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662/raw/cc1621d7fa32695090115b1d52f4714a54b9c0db/Problem.cs)[**Problem.cs**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662#file-problem-cs) hosted with ❤ by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | using System; |
|  | using System.Collections.Generic; |
|  | using System.Runtime.CompilerServices; |
|  |  |
|  | namespace MerchantGuideToGalaxy |
|  | { |
|  | internal class Program |
|  | { |
|  | private static void Main() |
|  | { |
|  | var lines = new List<string>(); |
|  | string line = string.Empty; |
|  |  |
|  | do |
|  | { |
|  | line = Console.ReadLine(); |
|  | if (!string.IsNullOrEmpty(line)) |
|  | lines.Add(line); |
|  | } while (!string.IsNullOrEmpty(line)); |
|  |  |
|  | Problem.Process(lines); |
|  | } |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662/raw/cc1621d7fa32695090115b1d52f4714a54b9c0db/Program.cs)[**Program.cs**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662#file-program-cs) hosted with ❤ by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | using MerchantGuideToGalaxy.Contexts; |
|  |  |
|  | namespace MerchantGuideToGalaxy.Parsers |
|  | { |
|  | class QuestionParser : Parsers.Parser |
|  | { |
|  | public QuestionParser(Context ctx) |
|  | : base(ctx) |
|  | { |
|  | } |
|  |  |
|  | public override bool Parse(string input) |
|  | { |
|  | if (!input.EndsWith("?")) |
|  | return false; |
|  | Context.Questions.Add(input.Replace("?", "").Trim()); |
|  | return true; |
|  | } |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662/raw/cc1621d7fa32695090115b1d52f4714a54b9c0db/QuestionParser.cs)[**QuestionParser.cs**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662#file-questionparser-cs) hosted with ❤ by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  |  |
|  | namespace MerchantGuideToGalaxy.Romans |
|  | { |
|  | class Roman |
|  | { |
|  | private readonly List<RomanPrimitive> \_primitives; |
|  |  |
|  | private bool \_dirty = true; |
|  | private int \_valueCache; |
|  |  |
|  | private Roman() |
|  | { |
|  | \_primitives = new List<RomanPrimitive>(); |
|  | } |
|  |  |
|  | public int Calculate() |
|  | { |
|  | if (\_dirty) |
|  | { |
|  | \_valueCache = 0; |
|  | int result = 0; |
|  | int length = \_primitives.Count(); |
|  | for (int i = 0; i < length; i++) |
|  | { |
|  | RomanPrimitive current = \_primitives[i]; |
|  | result += current.OctValue; |
|  |  |
|  | if (i == length - 1) |
|  | return result; |
|  |  |
|  | RomanPrimitive next = \_primitives[i + 1]; |
|  | if (current.OctValue < next.OctValue) |
|  | { |
|  | result = next.OctValue - result; |
|  | i++; |
|  | } |
|  | else if (current.OctValue == next.OctValue) |
|  | { |
|  | if (!current.AllowRepeat) |
|  | { |
|  | throw new Exception(string.Format("{0} can't be repeated", current.Symbol)); |
|  | } |
|  | int count = 2; |
|  | for (int j = i + 2; j < length; j++) |
|  | { |
|  | if (\_primitives[j].Symbol != current.Symbol) |
|  | break; |
|  | count++; |
|  | result += current.OctValue; |
|  | i++; |
|  | if (count > 3) |
|  | { |
|  | throw new Exception(string.Format("{0} can't be repeated more than 3 times", |
|  | current.Symbol)); |
|  | } |
|  | } |
|  | } |
|  | } |
|  | \_valueCache = result; |
|  | } |
|  |  |
|  | return \_valueCache; |
|  | } |
|  |  |
|  | public static Roman Parse(string str, Dictionary<string, RomanPrimitive> map) |
|  | { |
|  | string[] left = str.Split(' '); |
|  | var number = new StringBuilder(); |
|  | left.ToList().ForEach(l => number.Append(map[l].Symbol)); |
|  | Roman roman = Parse(number.ToString()); |
|  | return roman; |
|  | } |
|  |  |
|  | public static Roman Parse(string str) |
|  | { |
|  | var roman = new Roman(); |
|  | char[] chars = str.ToCharArray(); |
|  | chars.ToList().ForEach(c=>roman.\_primitives.Add(RomanPrimitive.Parse(c))); |
|  | return roman; |
|  | } |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662/raw/cc1621d7fa32695090115b1d52f4714a54b9c0db/Roman.cs)[**Roman.cs**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662#file-roman-cs) hosted with ❤ by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  |  |
|  | namespace MerchantGuideToGalaxy.Romans |
|  | { |
|  | internal class RomanPrimitive |
|  | { |
|  | private static readonly ILookup<char, RomanPrimitive> Primitives = new List<RomanPrimitive> |
|  | { |
|  | new RomanPrimitive('I', 1, true, true), |
|  | new RomanPrimitive('V', 5), |
|  | new RomanPrimitive('X', 10, true, true), |
|  | new RomanPrimitive('L', 50), |
|  | new RomanPrimitive('C', 100, true, true), |
|  | new RomanPrimitive('D', 500), |
|  | new RomanPrimitive('M', 1000, true), |
|  | }.ToLookup(\_ => \_.Symbol); |
|  |  |
|  | public RomanPrimitive(char symbol, int octValue, bool allowRepeat = false, bool allowSubtract = false) |
|  | { |
|  | Symbol = symbol; |
|  | OctValue = octValue; |
|  | AllowRepeat = allowRepeat; |
|  | AllowSubtract = allowSubtract; |
|  | } |
|  |  |
|  | public int OctValue { get; private set; } |
|  | public char Symbol { get; private set; } |
|  | public bool AllowRepeat { get; private set; } |
|  | public bool AllowSubtract { get; private set; } |
|  |  |
|  | public static RomanPrimitive Parse(char symbol) |
|  | { |
|  | if (!Primitives.Contains(symbol)) |
|  | return null; |
|  | return Primitives[symbol].First(); |
|  | } |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662/raw/cc1621d7fa32695090115b1d52f4714a54b9c0db/RomanPrimitive.cs)[**RomanPrimitive.cs**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662#file-romanprimitive-cs) hosted with ❤ by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | using MerchantGuideToGalaxy.Contexts; |
|  |  |
|  | namespace MerchantGuideToGalaxy.Solvers |
|  | { |
|  | internal abstract class Solver |
|  | { |
|  | public Solver(Context ctx) |
|  | { |
|  | Context = ctx; |
|  | } |
|  |  |
|  | public Context Context { get; private set; } |
|  |  |
|  | public abstract bool Solve(string input, out string answer); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662/raw/cc1621d7fa32695090115b1d52f4714a54b9c0db/Solver.cs)[**Solver.cs**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662#file-solver-cs) hosted with ❤ by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | using System; |
|  | using System.Linq; |
|  | using MerchantGuideToGalaxy.Contexts; |
|  |  |
|  | namespace MerchantGuideToGalaxy.Parsers |
|  | { |
|  | class UnitParser : Parsers.Parser |
|  | { |
|  | public UnitParser(Context ctx) |
|  | : base(ctx) |
|  | { |
|  | } |
|  |  |
|  | public override bool Parse(string input) |
|  | { |
|  | string[] lexers = input.Split(new[] {" is "}, StringSplitOptions.RemoveEmptyEntries); |
|  | if (lexers.Count() != 2 || input.EndsWith("?")) |
|  | return false; |
|  |  |
|  | string[] left = lexers[0].Split(' '); |
|  |  |
|  | if (left.Length < 2) |
|  | return false; |
|  | int rValue = int.Parse(lexers[1].Split(' ')[0]); |
|  | string primUnit = lexers[1].Split(' ')[1]; |
|  | Context.PrimUnit = primUnit; |
|  |  |
|  |  |
|  | Romans.Roman roman = Romans.Roman.Parse(String.Join(" ", left.Take(left.Length - 1)), Context.Primitives); |
|  | int calculated = roman.Calculate(); |
|  |  |
|  | string unit = left.Last(); |
|  | double unitValue = rValue/(double) calculated; |
|  |  |
|  | Context.Units[unit] = unitValue; |
|  | return true; |
|  | } |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662/raw/cc1621d7fa32695090115b1d52f4714a54b9c0db/UnitParser.cs)[**UnitParser.cs**](https://gist.github.com/wiz-kid/10d5abd81f59a03c8662#file-unitparser-cs) hosted with ❤ by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | using System; |
|  | using System.Linq; |
|  | using MerchantGuideToGalaxy.Contexts; |
|  |  |
|  | namespace MerchantGuideToGalaxy.Solvers |
|  | { |
|  | internal class UnitSolver : Solver |
|  | { |
|  | public UnitSolver(Context ctx) |
|  | : base(ctx) |
|  | { |
|  | } |
|  |  |
|  | public override bool Solve(string question, out string answer) |
|  | { |
|  | string primUnit = Context.PrimUnit; |
|  | string qualifier = String.Format("how many {0} is", primUnit); |
|  | if (!question.StartsWith(qualifier)) |
|  | { |
|  | answer = null; |
|  | return false; |
|  | } |
|  | string body = question.Substring(qualifier.Length + 1); |
|  | string[] lexers = body.Split(' '); |
|  | string unit = lexers.Last().Trim(); |
|  | double unitValue = Context.Units[unit]; |
|  | int value = Romans.Roman.Parse(String.Join(" ", lexers.Take(lexers.Length - 1)), Context.Primitives).Calculate(); |
|  | answer = value\*unitValue + " " + primUnit; |
|  | Console.WriteLine(body + " is " + answer); |
|  | return true; |
|  | } |
|  | } |
|  | } |