Teste de Lógica & Linguagem de programação (C#)

**Instruções:**

* Preencha as respostas dos exercícios logo após o final de cada enunciado, respectivamente;
* Caso não saiba como responder, deixe em branco ou justifique qual foi a sua dificuldade, evite criar soluções que não consiga ter embasamento ou que não saiba explicar;
* Utilize sempre a linguagem de programação C# nos exercícios que assim exigir;
* Procure fazer “o básico bem feito”, não perca tempo com respostas mirabulosas, queremos avaliar apenas a sua forma de raciocínio, clareza de código e capacidade de resolver problemas.

**QUESTÕES**

# 1) Prática - Primo

Um número primo é aquele que é divisível apenas por **um** e por **ele mesmo**.

Obs: O número 1 não é primo.

**🡺 Escreva uma função otimizada que, dado um número inteiro positivo, retorne true se o número for primo ou false caso contrário, com o menor número de iterações possível.**

Imprima o resultado em tela da seguinte forma:

*“O número* ***num*** *é primo. Número de iterações necessárias:* ***count****”*

*ou*

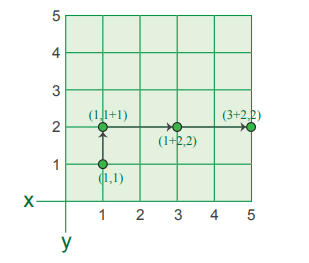
*“O número* ***num*** *não é primo. Número de iterações necessárias:* ***count****”*

# 2) Prática - Movimento do bot

Existe um bot localizado em um par de coordenadas inteiras, (x, y). Ele pode ser movido para um outro par de coordenadas. Embora o bot possa se mover quantas vezes quiser, ele só pode fazer os dois tipos de movimentos a seguir:

1. Da posição (x, y) para a posição (x + y, y).
2. Da posição (x, y) para a posição (x, x + y).

Por exemplo, se o bot começa em (1, 1), ele pode fazer a seguinte sequência de movimentos: (1, 1) → (1, 2) → (3, 2) → (5, 2). Observe que o movimento sempre será para cima ou para a direita.



**🡺 Escreva uma função que, dadas as coordenadas iniciais e finais, determine se o bot pode alcançar as coordenadas finais de acordo com as regras de movimento.**

Descrição da função:

A função deve retornar true se o bot puder atingir seu objetivo, caso contrário, retorne false.

A função tem o(s) seguinte(s) parâmetro(s):

x1: valor inteiro, coordenada x inicial

y1: valor inteiro, coordenada y inicial

x2: valor inteiro, coordenada x final

y2: valor inteiro, coordenada y final

# 3) Teoria - Classe e Objeto

Explique a diferença entre os conceitos de Classe e Objeto.

A Classe funciona como uma estrutura aonde se define as propriedades e peculiaridades de um dado objeto, podendo ser desde algo bem simples até estar atrelada a uma relação complexa.

O objeto se trata da classe instanciada em memória e com seus valores preenchidos ou então métodos, funções e assim por diante, aguardando sua invocação para ser utilizado.

# 4) Teoria - Orientação Objetos

Liste ao menos 4 conceitos de orientação objetos e explique cada um deles.

Herança, se trata do principio da POO onde classes podem herdar métodos, propriedades, funções e afins de outras classes, chamamos de classe base a qual pode ser herdada e de classe derivada a qual esta herdando no C# sendo representado pelo “:” a frente do nome da classe, vale ressaltar que o C# não permite múltipla herança de classe no entanto permite hierarquia através do conceito de herança transitiva. ( A classe **A** que herda de **B**, que por sua vez esta herdando de **C** ).

Polimorfismo, na minha opinião um dos princípios mais capiciosos do POO, é o principio que dita que um determinado método, função, pode ter várias formas (como sugere o nome) para ilustrar isso, podemos em uma interface criar métodos semelhantes que recebam parâmetros diferentes para se retornar um mesmo resultado, num exemplo prático, algumas vezes onde temos duas formas de identificar e obter um registro do banco de dados (como o padrão “Id” e “Guid”). Outra forma de se presenciar este principio ocorrendo é quando temos um método numa classe base que é um virtual, permitindo a alteração dele em tempo de execução, utilizado comumente para especificar filtros, obtenção de registros e/ou persistência no banco de dados seguindo alguma regra de negócio particular apenas a aquele cenário.

Abstração, diz respeito ao principio que pode ser entendido que cada classe, função deve saber apenas aquilo que é necessário, num exemplo prático temos as Interfaces, que dizem para as classes derivadas o que elas devem fazer mas o “como” fazer fica a encargo delas, em outras palavras a interface define o que é necessário ter e quem executa eh a classe criando essa noção de abstração no código. Acredito que dentro do Pattern conhecido como Factory se tem um tanto deste principio melhor trabalhado, onde a função que é a factory vai ter cada um dos métodos que realiza uma etapa necessária para obtenção de um dado resultado, no entanto a facotry é “ignorante” ao que estes métodos estão realizando, assim como os métodos são “ignorantes” a factory, cada um realizando apenas aquilo que lhe interessa.

Encapsulamento, diz respeito ao principio de organização onde se restringe a “visibilidade” e/ou “alterações” dos métodos, propriedades por outras classes, as Keywords pertinentes são, public, private, protected, internal, protected internal, cada um permitindo uma maneira de acesso pelo código e a quem esta instanciando ou herdando a classe, um dos exemplos comuns é o usado em conjunto ao pattern factory, normalmente a função que é a factory é publica permitindo que se chame de qualquer lugar onde estiver instanciando a classe, mas seus métodos internos muitas vezes são privados permitindo acesso apenas dentro daquela classe indiferente de quem a esta instanciando.

# 5) Teoria - Sobrecarga de métodos

O que é sobrecarga em programação e quando é utilizado? Exemplifique.

Sobrecarga é o princípio do Polimorfismo aplicado, seria sobrescrever, sobrecarregar um método, função, normalmente só métodos que são virtual podem ter a sobrecarga aplicada e é utilizado quando se tem em mente que alguns métodos podem ter aplicações diferentes dependendo do caso ou da regra de negócio.

# 6) Teoria - Exceções

O que são exceções e em qual ordem são executadas? Exemplifique, justificando a sua resposta.

Exceções são erros disparados pelo código capturadas logo no momento em que ocorrem e colocadas no topo da pilha de chamada (callstack) para serem invocadas pelo “catch”. Elas também podem ser disparadas por comandos de código no C#. Podemos invocar a classe da exceção através de um try, catch via código.

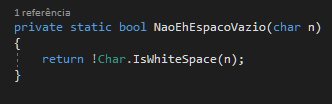
As exceções quando bem administradas fazem parte de um Pattern/Forma de “codar” eficiente, pois evitamos que quebre a aplicação de forma descomedida ou incompreensível, normalmente sempre se tem em mente onde possa ter uma exceção ocorrendo e caso ocorra, personalizar esta para dar mais informações aos desenvolvedores sobre o porque ela ocorre e possíveis soluções.

# 7) Teoria - SOLID

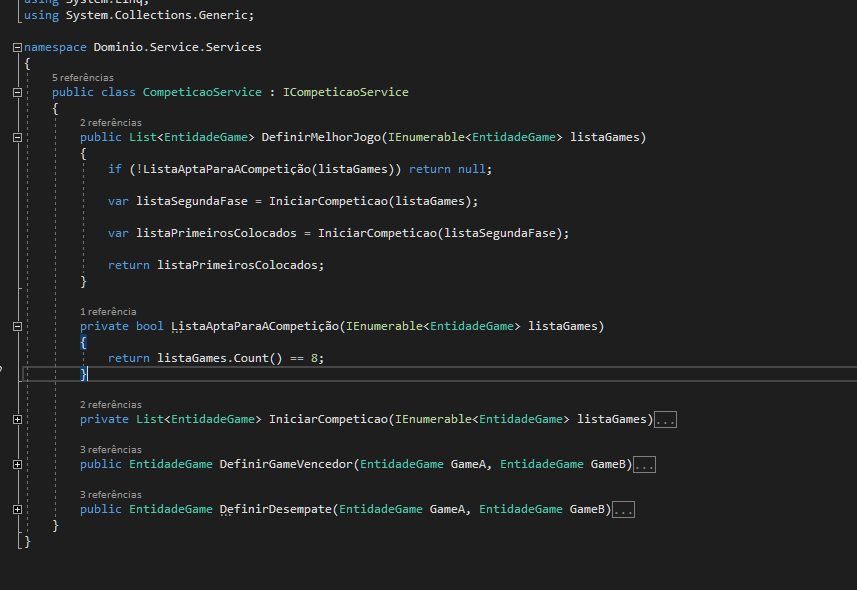
Explique o conceito de SOLID e para que é usado. Cite dois e explique seus princípios com um exemplo prático.

SOLID é uma sigla para uma série de 05 princípios de desenvolvimento “apanhados” e sintetizados pelo Robert C. Martin (Uncle Bob) visando uma escrita de código mais fácil de prestar manutenção, mais eficiente e funcional e infinitamente mais organizada.

S – Single-Responsability Principle (Principio da responsabilidade única), um método e/ou classe deve conter apenas uma função. Auto explicativo, quando se elabora um método para realizar uma função o ideal é que ele faça apenas uma coisa, quando se projeta uma classe para uma determinada função, o ideal é que ela lide apenas com o “concept” daquela função.



O – Open-Closed Principle (Princípio do Aberto e fechado), do ponto de vista de concept de criação e planejamento, uma classe pode estar aberta a receber métodos e updates, mas deve estar fechada a executar coisas diferentes ou novas fora de seu concept/escopo de criação. Sintetizando, uma classe que trata apenas de realizar divisão, pode receber refacts, updates e incrementos referentes a problemática da divisão, mas não poderia por exemplo ter um método com a finalidade de retornar o resultado de uma Adição ou Subtração de forma separada.

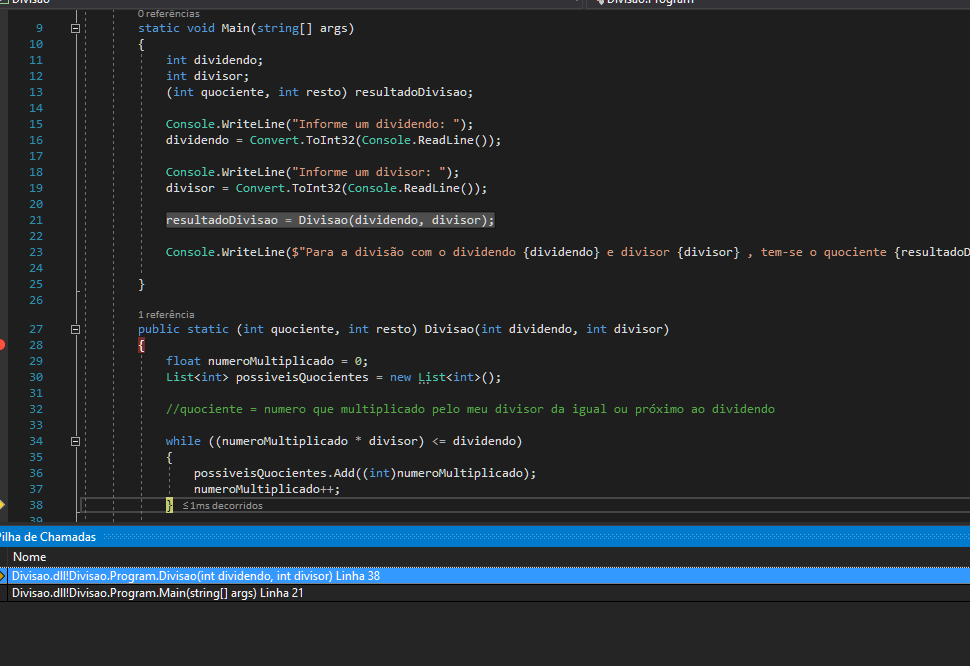


Na imagem acima por exemplo eu tenho uma classe chamada CompeticaoService que tem como finalidade abranger todas as etapas de uma competição entre jogos e definir um melhor jogo de acordo com os parâmetros estabelecidos pela regra de negócio, todos seus métodos são pertinentes a competição e ela está aberta a “refatoramento” ou criação de um novo método se necessário, tal como uma nova regra de negócio, mas ela apesar de lidar com uma lista de jogos não faria sentido ser responsável por criar um jogo, atualizar um jogo ou montar uma lista dos jogos para posteriormente iniciar a competição.

# 8) Teoria - Heap e Stack

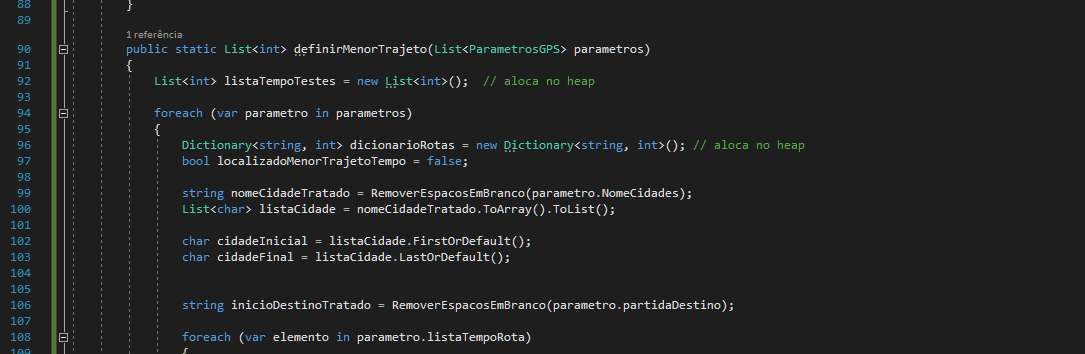
Qual a diferença entre heap memory e pilha? Justifique a sua resposta com um exemplo prático.

Stack é a memória “empilhavel” normalmente limitada aonde se aloca tipos brutos, referenciais de funções ou classes (ponteiros), ele tem por padrão a estrutura LIFO (last in first out) onde os inputs vão ser processados na ordem de último a ser posto o primeiro a ser realizado.



Na imagem acima na callstack estamos vendo que o programa esta organizando por ordem de chamada das funções, O main sendo a primeira a ser colada na pilha e na sequencia a função Divisão e para livrar esta pilha, ele vai resolver primeiramente a função Divisão ( ultima que entrou ) e posteriormente a Main, tendo resolvido todas o programa se encerra.

A heap é um bloco de memória genérico, normalmente gigantesco para alocar os mais diversos elementos que tendem a ser reutilizáveis ou vão ser necessários mais de uma vez no decorrer da execução de um programa, normalmente quem faz o gerenciamento dele em uma linguagem de baixo nível é o próprio desenvolvedor, em linguagem como o C# de mais alto nível, contamos com o GC.



# 9) Prática - Palíndromo

Palíndromo, do grego palin (novo) e dromo (percurso), é toda palavra ou frase que quando lida ao contrário, desconsiderando espaços e pontuações, possui o mesmo sentido. Ex.: “asa”, “ovo”, “A base do teto desaba”.

**🡺Escreva uma função que receba uma string como parâmetro e retorne true caso o valor dessa string seja um palíndromo ou false, caso contrário.** **Importante: Não utilizar funções, bibliotecas e métodos pré definidos para resolução do problema.**

# 10) Prática - Divisão

Sem utilizar a operação de divisão, **escreva um algoritmo que divida dois números inteiros (positivos) num e den. Esses dois números são lidos inicialmente**. Apresente o resultado da seguinte forma:

**“Para a divisão com o dividendo num e divisor den, tem-se o quociente quo e o resto rest”.**

Exemplos:

num = 4 den=2: quo=2 rest = 0

num = 10 den=3: quo=3 rest = 1

num = 7 den=15: quo=0 rest = 7

Diagram

Description automatically generated

**Obs: Utilizar while e verificar os valores de entrada.**

# 11) Prática - GPS

Você é um desenvolvedor responsável pela manutenção de um software GPS famoso no mercado. Você recebeu a tarefa de calcular a melhor rota possível entre duas cidades distintas. Seu programa receberá informações relacionadas às cidades e estradas que os conectam entre si. Além disso, este famoso software de GPS pode se comunicar com satélites para receber dados em tempo real sobre o tráfego fluxo das cidades. Portanto, seu programa precisará considerar algum tempo gasto durante o tráfego intenso, quando tentando deixar algumas cidades.

Para simplificar o desenvolvimento de software, considere o seguinte:

* cada cidade será nomeada com apenas 1 letra minúscula (de a a z);
* cada estrada liga duas cidades, e o tráfego pode ir em ambas as direções;
  + exemplo: em uma estrada que liga as cidades aez, o tráfego pode ir de a a z e vice-versa;
* para cada estrada, será fornecido um número inteiro t representando o tempo necessário para o veículo atual viajar toda a estrada, em horas;
* cidades nomeadas com uma vogal (a, e, i, o ou u), estão atualmente com tráfego intenso de saída - isso causará um atraso de 5 horas na saída da cidade

Dada uma lista de cidades, todas as estradas que as conectam, uma cidade de partida e, finalmente, a cidade de destino, seu programa precisa produzir o tempo mínimo necessário para a viagem.

**Entrada**

A entrada começa com uma linha contendo um inteiro T indicando o número de casos de teste. Para cada caso de teste, a entrada acontecerá da seguinte forma:

* uma linha contendo um inteiro C, indicando o número de cidades;
* uma linha contendo os nomes de todas as cidades, que são nomeadas com uma letra minúscula, separadas por um espaço;
* uma linha contendo um inteiro R, indicando o número de estradas;
* R linhas, cada uma contendo os seguintes dados, separados por um espaço:
  + uma letra minúscula representando uma cidade, em uma das pontas da estrada; uma letra minúscula representando a cidade do outro lado da estrada;
  + um número inteiro t representando o tempo, em horas, necessário para percorrer toda a estrada (independentemente de a direção do tráfego);
* Finalmente, a última linha de um caso de teste conterá duas letras minúsculas, separadas por um espaço, indicando uma cidade de partida e uma cidade de destino.

**Resultado**

Um inteiro M indicando o tempo mínimo necessário para viajar da cidade de partida à cidade de destino.

**Exemplo**

Chart, diagram

Description automatically generated

**Entrada:**

3

4

z a b c

4

z a 1

z b 2

a c 2

b c 1

z c

4

z a b c

4

z a 1

z b 2

a c 2

b c 1

z a

4

z a b c

4

z a 1

z b 2

a c 2

b c 1

z b

**Saída:**

3

1

2

No exercício do GPS eu entendo que o código pode ser bastante otimizado e melhorado ainda e vejo que a lógica dele pode ser falha para cenários de teste fora dos estipulados aqui no documento, devido a necessidade na agilidade da entrega eu deixei como esta, mas se desejável, com mais 02 dias eu conseguiria alinhar ele para cobrir possivelmente uma gama maior de cenários de teste.

# 12) Prática - Binários

O sistema binário (ou de base 2) é um sistema de numeração posicional em que todas as quantidades são representadas com base em dois números, ou seja, zero e um (0 e 1). Um método fácil de converter números decimais em números binários equivalentes é utilizar o número decimal e dividir continuamente por 2 (dois) para dar um resultado e um resto de "1" ou "0" até o resultado final é igual a zero.

O resultado binário é obtido colocando todos os restantes em ordem, com o bit menos significativo (LSB) no topo e o bit mais significativo (MSB) no fundo.

Implementar uma função recursiva que, dado um inteiro sem sinal de no máximo 32 bits (0-4294967295) inserido como um parâmetro, calcule a representação binária desse número e retorne o valor binário como uma string.

**ENTRADA:**

Um número inteiro sem sinal entre 0 e 4294967295 (0 <= n <= 4294967295)

**SAÍDA:**

Uma string composta de zeros e uns, representando o código binário do número de entrada.

**Exemplo:**

Entrada: 16

Resultado: 10000

**OBS: NÃO UTILIZE AS FUNÇÕES DE LINGUAGEM, SE HOUVER, PARA DEVOLVER O CÓDIGO BINÁRIO DO VALOR DADO, O VALOR BINÁRIO DEVE SER CALCULADO.**

13) Lógica - Bandeira do Japão

Imagine a bandeira do Japão, que possui um círculo perfeito da cor vermelha em seu centro e encontra-se em perfeitas condições. Desconsiderando que ela esteja pendurada em um mastro, e então você faça um movimento girando-a num ângulo de 180º:

1. Qual será a imagem formada ao final desse movimento, que diferença a bandeira passou a ter ?
2. Para obter a imagem da resposta anterior, a rotação deve ser para qual lado ?
3. A bandeira estará “de lado”, “de pé” ou tombada em diagonal ?
4. Por que? Justifique todas as respostas dos itens anteriores.
5. Um Circulo.
6. Tanto faz o lado, sempre será um círculo.
7. “de lado”, estaria de “de pé” numa rotação de 90 graus e tombada em diagonal na rotação de 45/60 graus.
8. Devido as propriedades do circulo perfeito, sempre será um circulo não importando para onde esta virada.

14) Lógica – Matriz de 1.000 posições

Considere uma matriz de 1.000 posições (uma linha, mil colunas), preenchida aleatoriamente com 1.000 números (somente números) que não se repetem, de 1 a 1.000. Dado este cenário:

1. Qual é a melhor melhor ação a ser tomada, de partida, para se começar a trabalhar com esta matriz ?
2. Caso deseje-se encontrar o número 889, como deveria ser feita essa busca ?
3. Justifique as respostas dos itens anteriores.

1. Passar a matriz para uma lista com o “.ToList()”, do contrário ordenaria ela.

2. Passando para uma lista utilizaria o Linq e o .Where(), do contrário apenas faço o array[i].

3. Como falamos de C# me parece mais produtivo usar as bibliotecas dele que estão tornando mais prático o trabalho e fazendo os mesmos procedimentos que teríamos de fazer na mão. Mas do ponto de vista lógico, como só existe um número de cada sem repetição, ordenar eles vai alinhar todos em ordem crescente, como são números, ordenar eles em ordem crescente vai me dar o número 889 na posição correta (888 , pois a contagem começa em zero ).