Entrega do trabalho pelo CANVAS : até 28/09/2022
Valor: 5 pontos

Orientações para o trabalho:

- Todas as funções devem ser NÃO-DESTRUTIVAS, ou seja, ao final da função os dados recebidos como parâmetros devem estar intactos e na mesma ordem em que foram recebidos.
- Você deve entregar um arquivo **em formato PDF** contendo o enunciado e a solução de cada exercício. Faça uma capa adequada para um trabalho acadêmico.
- Você deve entregar 15 exercícios: 9 à sua escolha mais os 6 exercícios marcados com *).

1 – Crie a função ***public List<T> CopiaList<T>(List<T> origem)*** que copia todos os dados da lista de origem e retorna a nova lista criada. *Atenção: para esse exercício não será permitido usar os métodos Clone() e CopyTo().*

2 – Crie a função ***public List<T> CopiaQueueParaList<T>(Queue<T> origem)***, que copia todos os dados do fila de origem e retorna uma lista. *Atenção: para esse exercício não será permitido usar os métodos Clone() e CopyTo().*

3 – Crie a função ***public List<T> CopiaParteList<T>(List<T> origem, int inicio, int fim)***, que retorna uma lista contendo todos os elementos dentro do intervalo determinado pelos parâmetros inicio e fim. Não é permitido usar o método CopyTo() do ArrayList.
Atenção:

- *Se fim ultrapassar o tamanho da lista, copie até o último elemento.*
- *Se inicio for maior que o tamanho da lista, retorne uma lista vazia.*
- *Se inicio > fim, copie os dados em ordem invertida.*

4 – Crie o procedimento ***public void ApagaList<T>(List<T> origem, int inicio, int fim)***, o qual apaga todos os elementos no intervalo determinado pelos parâmetros inicio e fim. *Atenção: não é permitido usar o método RemoveRange(). Apenas Remove() e RemoveAt().*

Exemplo: ApagaList(AL, 7, 10) => apagar os elementos entre as posições 7 e 10.

5 – Crie o procedimento ***public void ApagaList2<T>(List<T> origem, int inicio, int qtde)***. Esse procedimento deve remover exatamente n elementos da lista (*onde n é igual ao valor informado através do parâmetro qtde*), começando a partir da posição indicada pelo parâmetro inicio.

- O elemento na posição inicio é o primeiro a ser removido.
- Em seguida, os próximos elementos consecutivos também devem ser apagados, até completar quantidade.
- Caso não existam elementos suficientes após inicio, remova todos os que houver até o fim da lista.

6 – Crie a função ***public List<T> ConcatenaList<T>(List<T> l1, List<T> l2)***, que retorna uma nova lista contendo todos os elementos de l1 seguidos de todos os elementos de l2.

Exemplo de uso:

```
List<int> A = new List<int>() { 19, 33, 2, 4 };
List<int> B = new List<int>() { 1, 2, 3, 4, 5 };
List<int> AmaisB; // Apenas a referência foi declarada.
                  // A nova lista será criada dentro da função.
AmaisB = ConcatenaList(A, B);
```

Resultado:

A = [19, 33, 2, 4]
B = [1, 2, 3, 4, 5]
AmaisB = [19, 33, 2, 4, 1, 2, 3, 4, 5]

* 7 – Crie o procedimento ***public void ConcatenaList2<T>(List<T> l1, List<T> l2, List<T> destino)***. Esse procedimento deve **copiar todos os elementos de l1 e l2 para a lista destino**. Observação: a lista destino já estará instanciada antes da chamada da função, uma vez que um procedimento não pode retornar valores.

Exemplo de uso:

```
List<int> A = new List<int>() { 19, 33, 2, 4 };
List<int> B = new List<int>() { 1, 2, 3, 4, 5 };
List<int> AmaisB = new List<int>(); // Lista já está instanciada.
       // Não será necessário instanciá-la dentro do procedimento.
ConcatenaList2(A, B, AmaisB);
```

Resultado:

A = [19, 33, 2, 4]
B = [1, 2, 3, 4, 5]
AmaisB = [19, 33, 2, 4, 1, 2, 3, 4, 5]

Além de entregar o código, entregue uma explicação das diferenças entre a função **ConcatenaList<T>** e o procedimento **ConcatenaList2<T>**.

8 – Crie a função ***public List<T> Intersecao<T>(List<T> l1, List<T> l2)*** que retorna uma lista contendo os elementos **em comum** armazenados em l1 e l2, sem repetição.

9 – Crie a função ***public List<T> Uniao<T>(List<T> l1, List<T> l2)*** que retorna uma lista contendo os elementos armazenados em l1 e l2, sem repetição.

10 – Crie o procedimento ***public void ApagaRepetidos<T>(List<T> origem)*** que apaga todos os elementos repetidos da lista recebida como parâmetro, mantendo apenas a 1ª ocorrência de cada elemento.

11 – Crie a função ***public List<T> OcorrenciasDe<T>(List<T> origem, T elemento)*** que retorna uma lista contendo todas as posições que contem o elemento passado como parâmetro.

12 – Crie a função ***public int QtdeOcorrencias<T>(List<T> origem, T elemento)*** que retorna a quantidade de vezes que o elemento passado como parâmetro está armazenado na lista passada por parâmetro.

13 – Crie a função ***public Queue<T> CopiaQueue<T>(Queue<T> origem)*** que retorna uma cópia da fila passada por parâmetro. *Obs: Os elementos da Queue origem devem permanecer na mesma ordem original. Atenção: não é permitido usar os métodos Clone() e CopyTo() da classe Queue.*

14 – Crie a função ***public Stack<T> CopiaStack<T>(Stack<T> origem)*** que retorna uma cópia da pilha passada por parâmetro. Os elementos da pilha origem devem permanecer na mesma ordem original. *Atenção: não é permitido usar os métodos Clone() e CopyTo() da classe Stack.*

15 – Crie o procedimento ***public void VaiProFundo<T>(Stack<T> origem, T elemento)*** que empilha o elemento passado como parâmetro no fundo da pilha.

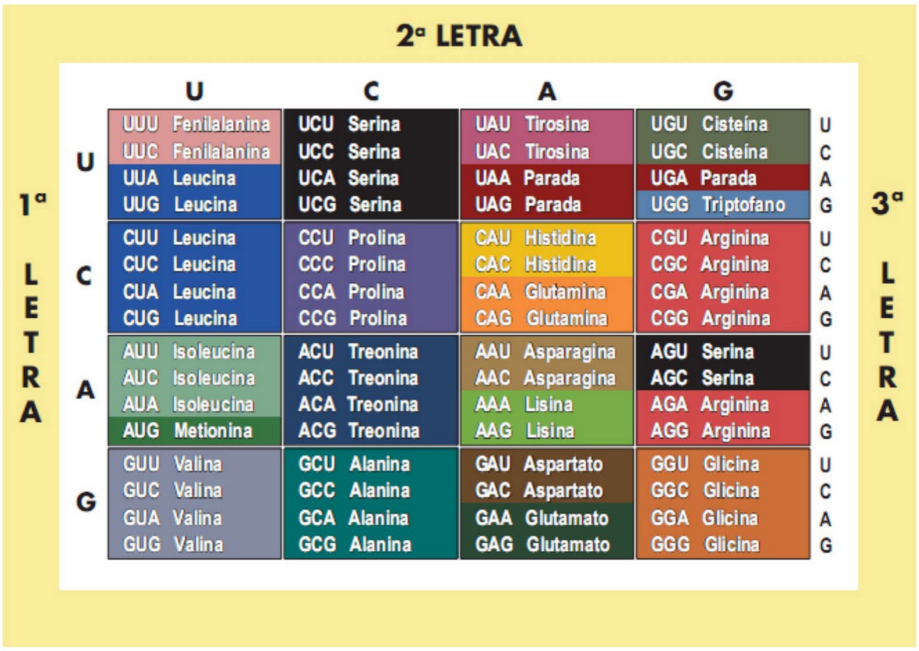
16 – Crie o procedimento ***public void InverteQueue<T>(Queue<T> Q)*** que inverte a ordem dos elementos da fila Q. *OBS1: utilize outras estruturas que julgar necessárias - vetor, List, Stack ou Queue. OBS2: nesse exercício você não deve utilizar o método* ***reverse****.*

17 – Crie o procedimento ***public void ApagaQueue<T>(Queue<T> origem, int inicio, int fim)*** que remove os elementos dentro do intervalo [inicio, fim].

18 – Crie o procedimento ***public void InverteStack<T>(Stack<T> S)*** que inverte a ordem dos elementos da pilha S. *OBS1: utilize outras estruturas que julgar necessárias - vetor, List, Stack ou Queue. OBS2: nesse exercício você não deve utilizar o método* ***reverse****.*

19 – Crie a função ***public Dictionary<TKey, TValue> ConcatenaDictionary<TKey, TValue>(Dictionary<TKey, TValue> d1, Dictionary<TKey, TValue> d2)***, que retorna um novo dicionário contendo todos os elementos de d1 e d2. Caso haja chaves duplicadas entre d1 e d2, mantenha o par chave/valor de d1.

* 20 – Um biólogo precisa de um programa que traduza uma trinca de nucleotídeos em seu aminoácido correspondente. Por exemplo, a trinca de aminoácidos ACG é traduzida como o aminoácido Treonina, e GCA em Alanina. Crie um programa em C# que use um **Dictionary<string, string>** para criar um dicionário do código genético. O usuário deve digitar uma trinca (chave) e seu programa deve mostrar o nome (valor) do aminoácido correspondente. Use a tabela abaixo para cadastrar todas as trincas/aminoácidos.



* 21 – Crie um dicionário com URL's e IP's dos websites abaixo e mais 5 à sua escolha. O seu dicionário deve ser implementado usando um **Dictionary<string, string>** e terá a URL como chave e o IP correspondente como valor (por exemplo, se digitarmos como chave a URL www.google.com, seu programa deve retornar o IP 74.125.234.81). O seu programa deve permitir que o usuário digite uma URL e deve imprimir o IP correspondente. Para descobrir o IP de um website, basta digitar **ping + URL do website** (exemplo: **ping www.google.com**).

www.google.com	www.yahoo.com	www.amazon.com	www.uol.com.br
www.pucminas.br	www.microsoft.com	research.microsoft.com	www.hotmail.com
www.gmail.com	www.x.com	www.facebook.com	www.cplusplus.com
www.youtube.com	www.brasil.gov.br	www.whitehouse.gov	www.nyt.com
www.capes.gov.br	www.wikipedia.com	www.answers.com	www.apple.com

* 22 – Faça um programa que use um **Dictionary<int, string>** para adicionar a matrícula (key) e nome (value) de vários alunos. Para encerrar o cadastramento de alunos, o usuário deve digitar uma matrícula negativa. Após o cadastro, seu programa deve permitir ao usuário pesquisar alunos através de sua matrícula. O usuário deve digitar um número negativo para interromper a pesquisa.

* 23 – Faça um programa que cadastre em um **Dictionary<string, List<string>>** alguns modelos de carros de montadoras nacionais, conforme a tabela abaixo (você deve fazer esse cadastro internamente, não o usuário – crie uma função para isso). Seu dicionário tem como chave o nome da montadora, e como valor um List<> contendo os modelos de carros da tabela abaixo. Após o cadastro, peça ao usuário que digite o nome de uma montadora. Você deve imprimir a quantidade de modelos e nome de cada um.

Fiat	Fiorino	Argo	Pulse	Cronos	Fastback	Strada	Mobi	Toro	Titano
Volkswagen	Taos	Voyage	Polo	Tera	Amarok	Nivus	Golf	Jetta	Tiguan
Hyundai	Creta	HB20	HB20S	Kona	Tucson				
GM	Onix	Tracker	S10	Spin	Montana				

* 24 – Faça um programa que monte a estrutura abaixo usando ***Dictionary<>*** ou ***SortedList<>***:

Chave (continente)	**Valor**
América	Chave (País) / Valor (População) Brasil – 202,656,784 México – 120,286,656 ... – ... Estados Unidos – 318,892,096
Ásia	Chave (País) / Valor (População) China – 1,355,692,544 ... – ... Japão – 127,103,392
...	

Você deve inserir pelo menos 3 continentes com no mínimo 6 países em cada um.

Gere um relatório com base no layout abaixo:

Continente: nome do continente
Nome do país *1* – População: população do país 1
Nome do país *2* – População: população do país 2
...
Nome do país *n* – População: população do país n
População total: 999999

No link http://www.indexmundi.com/map/?l=pt você encontrará uma tabela com as populações de diversos países. *Obs: você pode implementar todo o seu código dentro do Main(). O objetivo desse exercício é que você aprenda a manipular uma estrutura tão sofisticada quanto essa.*