==========

SSC0603 – ED1 – Prof. Fernando Osório – ICMC / USP 2021-2
Prof. Fernando Osório – Estrutura de Dados 1 (Eng.Comp.)
Run.Codes: Trabalho Prático - TP-01 Versão TP-01 v.01
Trabalho: SORT - Ordenação com Listas Encadeadas

==========

TP01 - SORT (Merge Sort)

** Definição Geral do Trabalho: TADs LDES (simples), pode ser feito com TAD LDED (dupla) e incluir TADs de Pilha e Fila e Deque (em "C" ou "C++")
(*) Aceito em Python mas tem que usar LISTAS ENCADEADAS DINÂMICAS na Ling. Python 3 com rotinas usando comandos como "self.data", "self.ref", "self.next"

Este trabalho é relacionado ao uso de uma **lista encadeada dinâmica** (simples ou duplamente encadeada) e que irá ser usada para armazenar listas de valores que devem ser unificados em uma lista única ordenada (algoritmo baseado no "merge sort" – discutido em aula – semana 4).

O programa irá receber duas ou mais listas de dados numéricos, unificar estas listas de dados gerando uma lista final ordenada como saída. A lista de dados numéricos serão valores do tipo ponto flutuante (com ou sem casas após o ponto), separados por espaço e terminando por um valor numérico Zero (0 ou 0.0). Exemplo: 1 2 3 4 5 0

O programa inicialmente irá ler um "cabeçalho" composto por 2 entradas iniciais. O primeiro indicando o tipo do arquivo de entrada (valor inteiro = 1 ou 2 ou 3) e seguido de um segundo valor inteiro indicando o total de listas a serem unificadas (valor inteiro entre 2 ou 4). A leitura é feita como se estivesse lendo o teclado ("scanf"). Segue abaixo um primeiro exemplo dos dados de **entrada** do programa:

```
1
2
1 3 5 7 9 0
2 4 6 8 10 0
```

A saída obtida para os dados acima é: (unificação ORDENADA das 2 listas – "merge sort")

```
1 2 3 4 5 6 7 8 9 10
```

A saída é exibida na tela ("printf"). E o valor zero não é incluído na lista, só marca o fim de lista.

Este exemplo de entrada acima é do tipo 1 (valor da primeira linha de entrada). Os dados do tipo 1 contém listas de dados que já estão ordenadas, sendo necessário apenas a unificação das duas listas (merge). Dados do tipo 2 (valor da primeira linha da entrada) contém listas de dados que não foram ainda ordenadas. E tipo 3 são listas não ordenadas, mas que devem ser exibidas na ordem inversa (decrescente) no final do processo de ordenação e unificação. Nestes casos (tipo 2 e 3) deve ser usada uma **inserção ordenada** na lista de dados.

Tipos de entradas: 1 => Os dados das listas já vêm ordenados
2 => Os dados das listas NÃO vêm ordenados (usar inserção ordenada crescente)
3 => Os dados das listas NÃO vêm ordenados, ordenação inversa (ordenação decrescente)

O dado da segunda linha de entrada do exemplo acima indica quantas listas de dados teremos no arquivo. Neste caso, o valor 2 indica que temos duas listas de dados a serem unificadas. As listas são uma sequência de valores numéricos, terminados por um valor Zero (0 ou 0.0). Uma lista de valores pode ter até 1024 valores separados por espaços. E podemos ter 2 ou 4 listas a serem unificadas. Apesar destas limitações de tamanhos máximos serem previamente definidos, implemente este trabalho usando listas dinâmicas (não use vetores estáticos!).

NÃO devem ser usados comandos de SORT sobre todos os dados das listas, a ordenação deve ser realizada na inserção dos dados (baseado no método tipo Insertion Sort, inserção ordenada) e na unificação das listas (baseado no método de Merge Sort). O Merge Sort deve ser implementado por você – sem usar bibliotecas prontas para a unificação das listas através do método de Merge Sort. É possível, por exemplo, o uso de estruturas de dados do tipo pilha para a realização do Merge e para a Inversão da Ordenação, e de listas, filas, pilhas ou deques encadeadas para armazenar e manipular os dados.

O trabalho TP-01 deverá ser implementado usando os conceitos de modularidade e abstração vistos em aula, podendo ser feito a partir dos TADs disponibilizados aos alunos (TAD LDES, TAD LDED – incluindo o uso dos TADs Pilha, Fila e Deque construídos a partir destes TADs básicos), ou usando as estruturas básicas da linguagem Python 3 para manipulação de "linked lists".

** Descrição "Genérica" dos Dados de entrada:

<tipo de lista: 1=ordenadas, 2=não ordenadas, 3=não ordenadas e saída inversa (decrescente)>

<quantidade de listas: 2 ou 4>

<lista1-valor1> <lista1-valor2> <lista1-valor3> <lista1-valor3> ... <lista1-valorN> 0

<lista2-valor1> <lista2-valor2> <lista2-valor3> ... <lista1-valorM> 0

Os dados de saída representam uma única lista que unifica as demais listas, com todos dados ordenados em ordem crescente, ou, decrescente se a opção for tipo de lista 3.

** Exemplos de Dados:

Entrada:

```
1
2
1 3 5 7 9 0
2 4 6 8 10 0
```

Saída:

```
1 2 3 4 5 6 7 8 9 10
```

Entrada:

```
1
4
1.1 1.4 1.7 0.0
1.2 1.5 1.8 0.0
1.3 1.6 1.9 2.0 0.0
2.123 2.234 2.345 2.456 0.0
```

Saída:

```
1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4
```

NOTA IMPORTANTE: Nro. inteiros são exibidos SEM casas após o ponto. Nros. com casas decimais são exibidos com **apenas 1 casa decimal**

Entrada:

```
2
2
9 3 7 5 1 0
6 4 8 2 10 0
```

Saída:

```
1 2 3 4 5 6 7 8 9 10
```

Entrada:

```
3
2
9 3 7 5 1 0
6 4 8 2 10 0
```

Saída:

```
10 9 8 7 6 5 4 3 2 1
```

```
===========
 F.Osorio
 Set. 2021
===========
```