# BCC202 – Estruturas de Dados I (2022-02)

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professor: **Pedro Silva** (`www.decom.ufop.br/`)
Estagiário docente: ——

## Trabalho Prático II (TP II) - 10 pontos, peso 1.

**- Data de entrega: 19/02/2023 até 23:55. O que vale é o horário do run.codes, e não do *seu*, ou do *meu* relógio!!!**

**- Clareza, identação e comentários no código também vão valer pontos. Por isso, escolha cuidadosamente o nome das variáveis e torne o código o mais legível possível.**

**- O padrão de entrada e saída deve ser respeitado exatamente como determinado no enunciado. Parte da correção é automática, não respeitar as instruções enunciadas pode acarretar em perda de pontos.**

**- Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas**.

**- A avaliação considerará o tempo de execução e o percentual de respostas corretas.**

**- Eventualmente serão realizadas entrevistas sobre os estudos dirigidos para complementar a avaliação;**

**- O trabalho é em grupo de até 3 (três) pessoas.**

**- Entregar um relatório.**

**- Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.**

**- Códigos cuja autoria não seja do aluno, com alto nível de similaridade em relação a outros trabalhos, ou que não puder ser explicado, acarretará na perda da nota.**

**- Códigos ou funções prontas específicas de algoritmos para solução dos problemas elencados não são aceitos**

**- Não serão considerados algoritmos parcialmente implementados.**

**- Procedimento para a entrega:**.

1. Submissão: via **run.codes**.
2. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
3. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
4. A solução deve ser devidamente modularizada e separar a especificação da implementação em arquivos *.h* e *.c* sempre que cabível.
5. Os arquivos a serem entregues, incluindo aquele que contém *main()*, devem ser compactados (*.zip*), sendo o arquivo resultante submetido via **run.codes**.
6. Você deve submeter os arquivos *.h*, *.c* e o *.pdf* (relatório) na raiz do arquivo *.zip*. Use os nomes dos arquivos *.h* e *.c* exatamente como pedido.
7. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.

***- Bom trabalho!***

## Ordenação de objetos móveis com base na trajetória

A análise exploratória de dados permite identificar informações relevantes em diversos tipos de dados. No contexto de objetos móveis, as trajetórias são um conjunto de pontos das posições ocupadas pelo objeto durante o seu movimento. Tais dados são utilizados para modelar fenômenos em diferentes domínios e aplicações, como estudos da dinâmica de automóveis em uma cidade, análise do movimento de animais, detecção de movimento em multidões, rastreio de objetos em operações militares e outros.

Uma trajetória pode ser representada com base no plano cartesiano, em que os pontos são dados em função de suas coordenadas $(x, y)$, como mostra a Figura 1A. No gráfico, a trajetória do objeto é formada pela sequência de pontos $A(x_A, y_A)$, $B(x_B, y_B)$ e $C(x_C, y_C)$. Note que entre esses pontos existem variações tanto no eixo $x$ quanto no eixo $y$. A partir deles é possível obter informações cruciais para análise de trajetórias como o deslocamento (D) e a distância percorrida (DP) pelo objeto móvel.

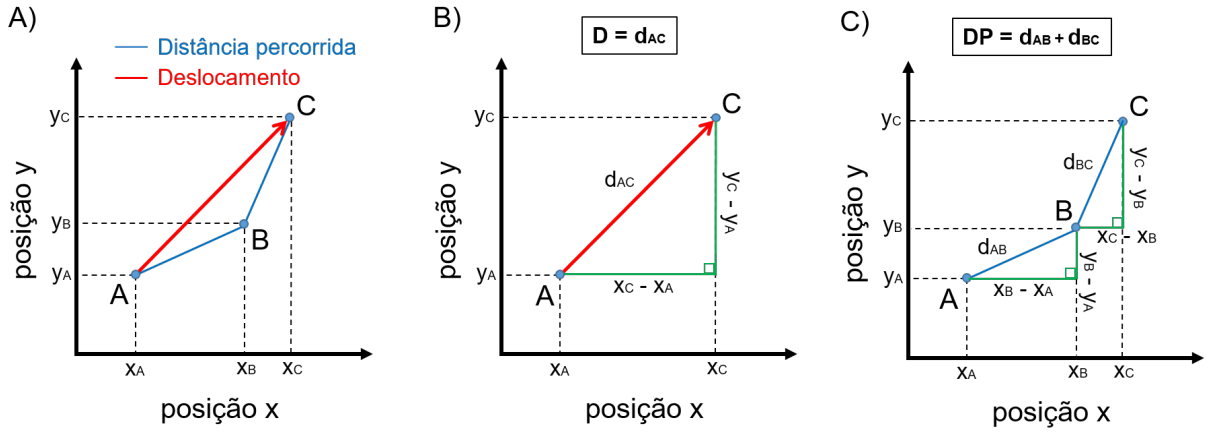


Figura 1: Trajetória no plano cartesiano. A) Representação dos pontos de trajetória; B) cálculo do deslocamento; C) cálculo da distância percorrida.

O deslocamento é uma grandeza vetorial que pode ser calculada a partir da diferença entre a posição final e a posição inicial. Note que um objeto que se move e volta para a mesma posição de onde partiu terá deslocamento nulo. A Figura 1A mostra em vermelho o vetor deslocamento que conecta os pontos A e C. Por outro lado, a distância percorrida é uma grandeza escalar, que corresponde a soma das distâncias que o objeto percorre até chegar à posição final, ou seja, equivale a medida associada à trajetória realmente descrita pelo objeto. Nesse caso, um objeto que se move e volta para a mesma posição de onde partiu terá distância percorrida diferente de zero. A Figura 1A mostra em azul os segmentos de trajetória entre os pontos A-B e B-C.

As Figuras 1B e 1C mostram como pode ser obtido o módulo do vetor deslocamento e o comprimento de cada segmento entre os pontos de trajetória. Como já se sabe as componentes $x$ e $y$, então pode-se obter essas medidas a partir do teorema de Pitágoras, ou ainda a partir do cálculo da distância entre dois pontos. Por exemplo, o módulo do deslocamento na Figura 1B pode ser calculado como segue:

$$D = d_{AC} = \sqrt{(x_C - x_A)^2 + (y_C - y_A)^2}$$

A distância percorrida na Figura 1C para esse caso em que a trajetória possui apenas três pontos seria igual:

$$D = d_{AB} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

$$D = d_{BC} = \sqrt{(x_C - x_B)^2 + (y_C - y_B)^2}$$

sendo $DP = d_{AB} + d_{BC}$. Note que se houvessem mais pontos de trajetória, então a distância percorrida incluiria em seu somatório as distâncias entre os novos pontos.

A Figura 2 mostra um caso particular da análise de trajetórias. Observe que as duas trajetórias têm o mesmo ponto inicial (A) e ponto final (B). Elas apresentam também o mesmo deslocamento (vermelho), porém o objeto que realiza a trajetória 1 (azul) percorre uma distância menor que o objeto que realiza a trajetória 2 (verde).

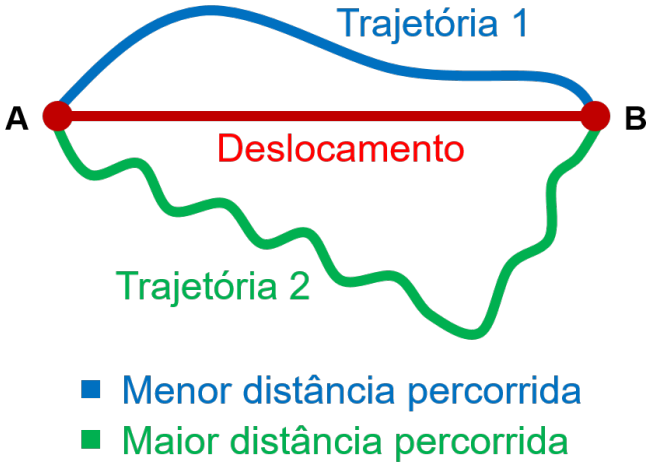


Figura 2: Trajetórias com o mesmo deslocamento e distâncias percorridas diferentes.

Seu programa deve receber um conjunto de trajetórias e extrair duas informações: deslocamento e distância percorrida. A saída deve listar os objetos móveis em **ordem decrescente** da distância percorrida, ou seja, aqueles objetos com maior distância percorrida no conjunto de dados devem vir primeiro. Caso dois ou mais objetos apresentem a mesma distância percorrida, então você deve ordená-los de forma **crescente** com base no deslocamento. Se persistir o empate, então você deve assegurar que esses objetos estejam listados em **ordem crescente** do identificador/nome do objeto móvel.

## Imposições e comentários gerais

Neste trabalho, as seguintes regras devem ser seguidas:

- Seu programa não pode ter *memory leaks*, ou seja, toda memória alocada pelo seu código deve ser corretamente liberada antes do final da execução. (Dica: utilize a ferramenta *valgrind* para se certificar de que seu código libera toda a memória alocada)
- Um grande número de *Warnings* ocasionará a redução na nota final.

### O que deve ser entregue

- Código fonte do programa em C (**bem identado e comentado**).
- Documentação do trabalho (relatório[1]). A documentação deve conter:
  1. **Implementação:** descrição sobre a implementação do programa. Não faça "*print screens*" de telas. Ao contrário, procure resumir ao máximo a documentação, fazendo referência ao que julgar mais relevante. É importante, no entanto, que seja descrito o funcionamento das principais funções e procedimentos utilizados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado. Muito importante: os códigos utilizados na implementação devem ser inseridos na documentação.
  2. **Impressões gerais**: descreva o seu processo de implementação deste trabalho. Aponte coisas que gostou bem como aquelas que o desagradou. Avalie o que o motivou, conhecimentos que adquiriu, entre outros.
  3. **Análise**: deve ser feita uma análise dos resultados obtidos com este trabalho.
  4. **Conclusão**: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
  5. **Formato:** PDF ou HTML.

[1]Exemplo de relatório: `https://www.overleaf.com/latex/templates/modelo-relatorio/vprmcsdgmcgd`.

### Como deve ser feita a entrega

Verifique se seu programa compila e executa na linha de comando antes de efetuar a entrega. Quando o resultado for correto, entregue via *run.codes* até a 19/02/2023 até 23:55 um arquivo **.ZIP** com o nome e sobrenome do aluno. Esse arquivo deve conter: (i) os arquivos *.c* e *.h* utilizados na implementação, (ii) instruções de como compilar e executar o programa no terminal, e (iii) o relatório em **PDF**.

## Detalhes da implementação

Para atingir o seu objetivo, você deverá construir um Tipo Abstrato de Dados (TAD) `Ponto` como representação de um ponto. O TAD deverá implementar, pelo menos, as seguintes operações:

1. `alocaPontos`: aloca um vetor de TAD `Ponto`.
2. `desalocaPontos`: desaloca um vetor de TAD `Ponto`.
3. `calcularDistancia`: calcula a distância percorrida considerando os pontos que estão em um vetor de TAD `Ponto`.
4. `calcularDeslocamento`: calcula o deslocamento considerando os pontos que estão em um vetor de TAD `Ponto`.
5. `ordena`: ordena um vetor de trajetórias, onde cada trajetória é um vetor de TAD *Ponto*.
6. `imprime`: imprime o vetor de TAD `Ponto` ordenado.

Alocação de um ou mais TADs `Ponto` fica a critério do aluno.

O TAD deve ser implementado utilizando a separação interface no *.h* e implementação *.c* discutida em sala, bem como as convenções de tradução. Caso a operação possa dar errado, devem ser definidos retornos com erro, tratados no corpo principal. A alocação da TAD **necessariamente deve ser feita de forma dinâmica**.

O código-fonte deve ser modularizado corretamente em três arquivos: *tp.c*, *odenacao.h* e *ordenacao.c*. O arquivo *tp.c* deve apenas invocar e tratar as respostas das funções e procedimentos definidos no arquivo *ordenacao.h*. A separação das operações em funções e procedimentos está a cargo do aluno, porém, **não deve haver acúmulo** de operações dentro de uma mesma função/procedimento.

O limite de tempo para solução de cada caso de teste é de apenas **um segundo**. Além disso, o seu programa não pode ter *memory leaks*, ou seja, toda memória alocada pelo seu código deve ser corretamente liberada antes do final da execução. (Dica: utilize a ferramenta *valgrind* para se certificar de que seu código libera toda a memória alocada). *Warnings* ocasionará a redução na nota final. Assim sendo, utilize suas habilidades de programação e de análise de algoritmos para desenvolver um algoritmo correto e rápido!

### Entrada

A entrada é dada pelo número de objetos móveis $N$ e o total de pontos $T$ de cada trajetória, seguido pela lista de trajetórias, em que as posições são apresentadas em termos das coordenadas $x$ e $y$. Considere que as distâncias são dadas em metros (m) e o tempo é constante entre um ponto e outro. Além disso, admita que as posições foram fornecidas sequencialmente conforme o movimento do objeto. A Figura 3 mostra um exemplo de trajetória com 3 pontos e como eles serão especificados para entrada em seu programa.

| | PONTO A | | PONTO B | | PONTO C | |
|---|---|---|---|---|---|---|
| Nome | $X_A$ | $Y_A$ | $X_B$ | $Y_B$ | $X_C$ | $Y_C$ |
| OBJETO1 | 13 | 20 | 27 | 48 | 56 | 49 |

Figura 3: Especificação de trajetórias.

A linha em azul na Figura 3 especifica a trajetória do objeto móvel `OBJETO1`, sendo que os dois primeiros valores correspondem às coordenadas $x$ e $y$ do ponto A. Os valores seguintes correspondem às coordenadas $x$ e $y$ dos pontos B e C, respectivamente.

### Saída

A saída deve listar os objetos móveis, apresentando o identificador/nome, distância percorrida e deslocamento. A lista deve estar ordenada de forma **decrescente com base na distância percorrida** ou, em caso de empate, **em ordem crescente do deslocamento e do nome**, respectivamente. Observação: apresente os valores de deslocamento e distância percorrida com apenas 2 casas decimais.

### Exemplo de um caso de teste

Exemplo da saída esperada dada uma entrada:

| **Entrada** | **Saída** |
|---|---|
| 10 3<br>D000 1 1 0 1 0 1<br>R001 1 1 1 1 2 1<br>E002 0 0 1 1 1 0<br>O003 3 3 2 1 3 2<br>X004 2 1 0 2 1 1<br>P005 0 1 1 1 1 2<br>A006 0 3 3 3 0 3<br>V007 0 1 0 1 1 3<br>O008 1 3 2 2 1 0<br>E009 2 3 1 3 2 1 | A006 6.00 0.00<br>O003 3.65 1.00<br>X004 3.65 1.00<br>O008 3.65 3.00<br>E009 3.24 2.00<br>E002 2.41 1.00<br>V007 2.24 2.24<br>P005 2.00 1.41<br>D000 1.00 1.00<br>R001 1.00 1.00 |

A SAÍDA DA SUA IMPLEMENTAÇÃO DEVE SEGUIR EXATAMENTE A SAÍDA PROPOSTA.

### Diretivas de Compilação

As seguintes diretivas de compilação devem ser usadas (essas são as mesmas usadas no run.codes).

```
$ gcc -c ordenacao.c -Wall
$ gcc -c tp.c -Wall
$ gcc ordenacao.o tp.o -o exe
```

### Avaliação de *leaks* de memória

Uma forma de avaliar se não há *leaks* de memória é usando a ferramenta *valgrind*. O *valgrind* é um *framework* de instrumentação para análise dinâmica de um código e é muito útil para resolver dois problemas em seus programas: **vazamento de memória e acesso a posições inválidas de memória** (o que pode levar a *segmentation fault*). Um exemplo de uso é:

```
1 gcc -g -o exe ordenacao.c tp.c -Wall
2 valgrind --leak-check=full -s ./exe < casoteste.in
```

Espera-se uma saída com o fim semelhante a:

```
1 ==xxxxx== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Para instalar no Linux, basta usar: `sudo apt install valgrind`.