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

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

**AULA PRÁTICA 05**

---

- **Data de entrega: Até 05 de fevereiro às 23:59:59.**

**- 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. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.
7. Siga atentamente quanto ao formato da entrada e saída de seu programa, exemplificados no enunciado.
8. Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.
9. A avaliação considerará o tempo de execução e o percentual de respostas corretas.
10. Eventualmente, serão realizadas entrevistas sobre os estudos dirigidos para complementar a avaliação.
11. Considere que os dados serão fornecidos pela entrada padrão. Não utilize abertura de arquivos pelo seu programa. Se necessário, utilize o redirecionamento de entrada.
12. Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.
13. 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.
14. Códigos ou funções prontas específicos de algoritmos para solução dos problemas elencados não são aceitos.
15. Não serão considerados algoritmos parcialmente implementados.

***- Bom trabalho!***

## O jogo do Bowser

Bowser está propondo um jogo onde uma fila é montada de acordo com a chegada dos jogadores. Contudo, querendo favorecer alguns jogadores, o Bowser inverte a fila ao seu bel prazer. Para isso, ele teve a seguinte ideia:

- O programa deverá ler os jogadores do game do Bowser e colocá-los na fila.
- O programa deverá inverter essa fila.

Como ele não sabe como inverter essa fila, ele o "contratou". É um trabalho de vida ou morte. Como você sabe computação e conhece a implementação de listas, pilhas e filas, você propôs fazer a inversão da fila usando uma pilha.

### Considerações

- Não altere o nome dos arquivos.
- O arquivo `.zip` deve conter na sua raiz somente os arquivos-fonte.
- Há vários casos de teste. Você terá acesso (entrada e saída) de casos específicos para realizar os seus testes localmente.

O código-fonte deve ser modularizado corretamente conforme os arquivos de protótipo fornecidos. Cada um dos jogadores deve ser armazenado em uma **lista** implementada por *ponteiros*. A criação da fila invertida de jogadores deve ser realizada com base na manipulação da fila utilizando uma pilha. Para implementar tanto a fila quanto a pilha você deve utilizar a implementação da lista. As funções *ListaInicia*, *ListaInsereFinal*, *ListaInsereInicio*, *ListaRetiraPrimeiro*, *ListaEhVazia* e *ListaLibera* são as mesmas vistas na aula teórica com modificações mínimas para tratar strings. As funções de lista devem ser implementadas pelo aluno. As funções de pilha e fila devem ser implementadas utilizando as funções da lista.

## Especificação da Entrada e da saída

Você deve ler na primeira linha a quantidade *n* de jogadores. Nas próximas *n* linhas estão os nomes dos jogadores (um em cada linha). O nome dos jogadores tem até 30 caracteres. A ordem com que os nomes aparecem é também a ordem de inserção na fila.

Seu programa deverá exibir a fila na ordem invertida. **Não vale somente imprimir a fila de trás para frente, você precisa inverter a lista.**

| Entrada | Saída |
|---|---|
| 7<br>Jones<br>Pedro<br>Carlos<br>Lucas<br>Juca<br>Valdineia<br>Jovander | Jovander<br>Valdineia<br>Juca<br>Lucas<br>Carlos<br>Pedro<br>Jones |

| Entrada | Saída |
|---|---|
| 5 Jones<br>Carlos<br>Lucas<br>Pedro<br>Daiane | Daiane<br>Pedro<br>Lucas<br>Carlos<br>Jones |

## Diretivas de Compilação

```
$ gcc -c lista.c -Wall
$ gcc -c pilha.c -Wall
$ gcc -c fila.c -Wall
$ gcc -c pratica.c -Wall
$ gcc lista.o pilha.o fila.o pratica.o -o exe
```

## Avaliação de *leaks* de memória

Uma forma de avaliar se não há *leaks* de memória é usando a ferramenta valgrind. Um exemplo de uso é:

```
gcc -g -o exe *.c -Wall; valgrind --leak-check=yes -s ./exe < casoteste.in
```

Espera-se uma saída com o fim semelhante a:

```
==38409== ERROR SUMMARY: 0 errors from 0 contexts (suppressed:  0 from 0)
```

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