BCC202 – Estruturas de Dados I (2023-02)

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

AULA PRÁTICA 04

- **Data de entrega: Até 27 de outubro às 23:55.**

- Procedimento para a entrega:.

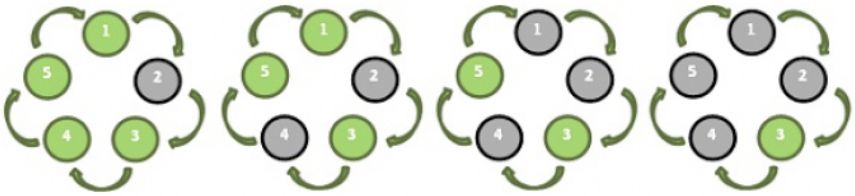
1. Submissão: via ***Moodle***.
2. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
3. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
4. A solução deve ser devidamente modularizada e separar 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 ***Moodle***.
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 problema de Josephus

O problema de Josephus é assim conhecido por causa da lenda de Flavius Josephus, um historiador judeu que viveu no século 1. Segundo o relato de Josephus do cerco de Yodfat, ele e seus companheiros (40 soldados) foram presos em uma caverna, cuja saída foi bloqueada pelos romanos. Eles decidiram entregar uma das pessoas em troca de não serem capturados. Foi decidido que iriam formar um círculo e começar a eliminar candidatos saltando de três em três em sentido horário. Quem ficasse por último seria entregue aos romanos.

Segue um exemplo com 5 candidatos e um salto igual a 2. Inicialmente, salta-se o candidato número 1 e elimina-se o candidato número dois. Depois, salta-se o candidato 3 e elimina-se o candidato 4. Na sequência, salta-se o candidato 5 e elimina-se o candidato 1. Na rodada final, salta-se o candidato 3 e elimina-se o candidato 5. Neste exemplo o candidato que restará após as eliminações é o 3.



Considerações

O código-fonte deve conter uma função recursiva para efetuar o cálculo desejado e deve ser modularizado corretamente conforme os arquivos de protótipo fornecidos. Especialmente nesta aula prática, **não** será permitido criar outras funções ou procedimentos. Também **não** será permitido criar variáveis adicionais no *main*.

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

Especificação da Entrada e da saída

Em cada caso de teste de entrada haverá um par de números inteiros positivos n e k. O número n representa a quantidade de pessoas no círculo, numeradas de 1 até n. O número k representa o tamanho do salto de um candidato até o próximo candidato que será eliminado.

Para cada caso de teste de entrada será apresentada uma linha de saída informando que foi o último candidato.

Entrada	**Saída**
`1234 233`	`25`

Diretivas de Compilação

```
$ gcc -c josephus.c -Wall
$ gcc -c pratica.c -Wall
$ gcc josephus.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`.