BCC202 – Estruturas de Dados I (2023-01)

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

AULA PRÁTICA 3

- **Data de entrega: Até 28 de maio às 23:59:59.**

- 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!

Arranjos de tamanho k

O supermercado ITAH te contratou para criar um programa que gera todos os arranjos de um tamanho k dado um conjunto de n elementos. A única restrição é que $k > 0$ e $k \leq n$.

O seu gerente pediu que você implemente uma **função recursiva** em C que imprime todos os possíveis arranjos de n elementos tomados k a k. Cada arranjo deve ser impresso em uma linha. A função deve receber, entre outros parâmetros: o vetor de elementos (v), o número de elementos (n) e o valor k.

Considerações

O código-fonte deve ser modularizado corretamente conforme os arquivos de protótipo fornecidos. Você deve criar um TAD `Arranjo`. O TAD deve ser alocado e desalocado dinamicamente.

- 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.

Especificação da Entrada e da saída

A entrada é composto por somente dois valores. O primeiro valor corresponde ao número de elementos do vetor (n). O segundo valor corresponde ao valor de k. Vale ressaltar que os valores do vetor variam de 1 até n.

A saída é a impressão de cada arranjo tomado de k a k.

Entrada	Saída
5 2	1 2 1 3 1 4 1 5 2 1 2 3 2 4 2 5 3 1 3 2 3 4 3 5 4 1 4 2 4 3 4 5 5 1 5 2 5 3 5 4

Entrada	Saída
3 3	1 2 3 1 3 2 2 1 3 2 3 1 3 1 2 3 2 1

Diretivas de Compilação

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