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 7

- **Data de entrega: Até 09 de julho 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!

Organizando pacotes

Uma empresa de entregas precisa organizar seus pacotes de acordo com seu peso. Cada pacote possui um código de identificação e um peso em quilogramas. O gerente da empresa solicitou que você implemente uma função, que recebe como parâmetro um array de pacotes e o tamanho do array.

Sua tarefa é implementar a função `ordenarPacotes`, que utiliza o algoritmo de ordenação *Quick Sort* para ordenar os pacotes em ordem crescente de peso. Se dois pacotes possuírem o mesmo peso, eles devem ser ordenados em ordem crescente de código. A função deve ordenar o array de pacotes com base nesses critérios de ordenação. Além disso, você deve criar uma função adicional chamada `imprimirPacotes`, que recebe o array ordenado de pacotes e seu tamanho como parâmetros e imprime na tela que pediu, o código e o peso de cada pacote, um por linha.

A ordenação deve ser feita pelo peso na ordem crescente e se estes forem iguais, usa-se o código do pacote para realizar a ordenação crescente.

Considerações

O código-fonte deve ser modularizado corretamente conforme os arquivos de protótipo fornecidos. Você deve criar um TAD `Pacote` que tem uma função responsável por ordenar o vetor e tem os campos nome de quem pediu (*char* com até 50 caracteres), peso (*float)* e código (inteiro). O TAD **precisa** 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 primeira linha da entrada um número inteiro *n*, que indica quantos pacotes o vetor possui. Em seguida, serão inseridos o nome, código e peso, separados por um espaço em branco.

Ao término da inserção de cada vetor de pessoas, será exibida uma saída do seguinte formato:

Entrada	**Saída**
7 barbara 21 10.1 jose 13 1.1 fred 51 20.0 loreto 90 4.3 jade 34 5.9 filipe 37 80.0 jade 23 21.1	jose 13 1.1 loreto 90 4.3 jade 34 5.9 barbara 21 10.1 fred 51 20.0 jade 23 21.1 filipe 37 80.0

Diretivas de Compilação

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