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 6

- **Data de entrega: Até 02 de junho 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!

Ordenando nomes e idades

A turma 23.1 de Nutrição da UFOP está conduzindo um estudo sobre os alunos da UFOP. Esse estudo envolve coletar o nome e a idade das pessoas. Para saber se uma pessoa já participou ou não da pesquisa, a turma de Nutrição optou por ordenar o nome de todos os participantes por ordem alfabética seguida pela idade.

Como você é muito proativo, se voluntariou para ajudar nessa tarefa. Dado o seu vasto conhecimento de métodos de ordenação, você optou por implementar o algoritmo *Merge Sort* para ajudá-las. Logo, o seu objetivo é utilizar o algoritmo *Merge Sort* (especificamente a versão recursiva) para ordenar um vetor de pessoas contendo os nomes e idades. O vetor deve ser ordenado pelo **nome** obedecendo a ordem lexicográfica. Caso dois ou mais nomes sejam iguais, a ordenação deve ser feita com base nas idades em ordem crescente.

Considerações

O código-fonte deve ser modularizado corretamente conforme os arquivos de protótipo fornecidos. Você deve criar um TAD `Pessoa` que tem uma função responsável por ordenar o vetor e tem os campos nome (*char* com até 50 caracteres) e idade (inteiro). O TAD **precisa** ser alocado e desalocado dinamicamente. Crie uma função recursiva para efetuar a ordenação e uma função para realizar a intercalação.

- 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 contém um único inteiro *n*, o qual indica o número de casos de teste. Cada caso de teste contém um número inteiro *p*, que indica quantos elementos o vetor de pessoas possui. Em seguida, serão inseridos o nome e a idade, separados por um espaço em branco.

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

Entrada	**Saída**
2 3 bárbara 21 jose 13 fred 51 4 loreto 90 jade 34 filipe 37 jade 23	Teste 1: bárbara 21 fred 51 jose 13 Teste 2: filipe 37 jade 23 jade 34 loreto 90

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`.