BCC202 – Estruturas de Dados I

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

MÉTODOS DE ORDENAÇÃO

- **Submissão via *Moodle***.

- **Data e hora de entrega disponíveis no *Moodle*.**

- Procedimento para a entrega:.

1. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
2. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
3. A solução deve ser devidamente modularizada e separar a especificação da implementação em arquivos *.h* e *.c* sempre que cabível.
4. Os arquivos a serem entregues, incluindo aquele que contém *main()*, devem ser compactados (*.zip*), sendo o arquivo resultante submetido via ***Moodle***.
5. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.
6. Siga atentamente quanto ao formato da entrada e saída de seu programa, exemplificados no enunciado.
7. Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.
8. A avaliação considerará o tempo de execução e o percentual de respostas corretas.
9. Eventualmente, serão realizadas entrevistas sobre os estudos dirigidos para complementar a avaliação.
10. Considere que os dados serão fornecidos pela entrada padrão. Não utilize abertura de arquivos pelo seu programa. Se necessário, utilize o redirecionamento de entrada.
11. Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.
12. 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.
13. Códigos ou funções prontas específicos de algoritmos para solução dos problemas elencados não são aceitos.
14. Não serão considerados algoritmos parcialmente implementados.

- Bom trabalho!

Ajudando tia Joana

Tia Joana é uma respeitada professora e tem vários alunos. Em sua última aula, ela prometeu que iria sortear um aluno para ganhar um bônus especial na nota final: ela colocou n pedaços de papel numerados de 1 a n em um saquinho e sorteou um determinado número k; o aluno premiado foi o k-ésimo aluno na lista de chamada contando do último até o primeiro (note que k começa em 1 e não 0).

O problema é que a Tia Joana esqueceu o diário de classe, então ela não tem como saber qual número corresponde a qual aluno. Ela sabe os nomes de todos os alunos, e que os números deles, de 1 até n, são atribuídos de acordo com a ordem alfabética, mas os alunos dela estão muito ansiosos e querem logo saber quem foi o vencedor.

Dados os nomes dos alunos da Tia Joana e o número sorteado, determine o nome do aluno que deve receber o bônus.

Considerações

O código-fonte deve ser modularizado corretamente conforme os arquivos de protótipo fornecidos. O problema deve ser resolvido pela implementação do *Shell sort*. O conjunto de dados **deve ser** representado por um **vetor alocado dinamicamente**.

A função `compare` precisa ser implementada para realizar a comparação entre duas strings e retornar -1 se a primeira string for menor, 0 (zero) se ambas forem iguais, 1 caso contrário. Esta função deve ser invocada pelo algoritmo de ordenação implementado. Cada caso de teste deve ser resolvido em até 1 segundo.

- 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 contém dois inteiros *n* e *k* separados por um espaço em branco. Cada uma das *n* linhas seguintes contém uma cadeia de caracteres de tamanho mínimo 1 e máximo 20 representando os nomes dos alunos. Os nomes são compostos apenas por letras de 'a' a 'z'.

Seu programa deve imprimir uma única linha, contendo o nome do aluno que deve receber o bônus.

Entrada	Saída
5 3 maria joao carlos vanessa jose	jose

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`.