BCC202 – Estruturas de Dados I (2022-02)

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

AULA PRÁTICA 9

- **Data de entrega: Até 12 de março às 23:59:59.**

- Procedimento para a entrega:.

1. Submissão: via ***run.codes***.
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 ***run.codes***.
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!

A ordem de entrada em um hospital

A sua amiga Ana trabalha na recepção de um hospital de tratamento de câncer. Diariamente são distribuídas senhas para atendimento dos pacientes no dia seguinte. Para evitar problemas de prioridade, o hospital simplesmente fornece números aleatórios para cada pessoa.

A atividade da sua amiga Ana é ordenar as senhas em ordem crescente para definir o horário de atendimentos (o paciente com a menor senha tem o primeiro horário, o com maior, o último) e avisar os pacientes de acordo com a senha. As senhas são compostas por até M dígitos. Como você é um ótimo(a) amigo(a), oferece-se para ajudar a escrever um programa que, dada a lista de todas as senhas, você imprime todas elas na ordem crescente.

Há um problema no sistema do hospital e senhas repetidas são geradas e, independente do nome da pessoa, a ordem em que os números foram gerados, deve ser respeitada para a marcação dos horários de atendimento.

Considerações

O código-fonte deve ser modularizado corretamente conforme os arquivos de protótipo fornecidos. O problema deve ser resolvido pela implementação do *Radix Sort* e o *Counting Sort* para ordenar o array *v* pelo i-ésimo dígito.

Um vetor estático (sabe-se o tamanho máximo do vetor) deve ser criado e ordenado para armazenar os dados de um paciente (senha e nome) e resolver o problema. 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 entrada é composta de um único caso de teste. A primeira linha consiste de dois inteiros N e M, dizendo quantas senhas foram geradas e quantos dígitos tem a senha. As N linhas seguintes contêm cada uma, uma senha e o nome do paciente.

Seu programa deve imprimir o vetor ordenado em ordem **crescente** somente considerando a senha. **Você não deve usar o nome para a ordenação.**

Restrições do problema:

- $1 \leq N \leq 10000$
- $1 \leq M \leq 7$

Exemplo de entrada e saída:

Entrada	Saída
7 3 100 carlos 200 joao 200 carla 150 pedro 030 daiane 525 filipe 942 andrea	030 daiane 100 carlos 150 pedro 200 joao 200 carla 525 filipe 942 andrea

Entrada	Saída
4 5 01100 carlos 00100 carla 00100 joao 11500 pedro 00003 daiane	00003 daiane 00100 carla 00100 joao 01100 carlos 11500 pedro

Diretivas de Compilação

```
$ gcc -c ordena_linear.c -Wall
$ gcc -c pratica.c -Wall
$ gcc ordena_linear.o pratica.o -o exe -lm
```

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`.