BCC202 – Estruturas de Dados I (2020-01)

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professor: **Pedro Silva** (www.decom.ufop.br/)
Estagiário docente: **Geovani Martins** (geovani.martins@aluno.ufop.edu.br)

Trabalho Prático 2 (TP2) - 10 pontos, peso 1.

- Data de entrega: 15/05/2022 até 23:55. O que vale é o horário do run.codes, e não do *seu*, ou do *meu* relógio!!!

- Clareza, identação e comentários no código também vão valer pontos. Por isso, escolha cuidadosamente o nome das variáveis e torne o código o mais legível possível.

- O padrão de entrada e saída deve ser respeitado exatamente como determinado no enunciado. Parte da correção é automática, não respeitar as instruções enunciadas pode acarretar em perda de pontos.

- Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.

- A avaliação considerará o tempo de execução e o percentual de respostas corretas.

- Eventualmente serão realizadas entrevistas sobre os estudos dirigidos para complementar a avaliação;

- O trabalho é em grupo de até 3 (três) pessoas.

- Entregar um relatório.

- Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.

- 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.

- Códigos ou funções prontas específicas de algoritmos para solução dos problemas elencados não são aceitos

- Não serão considerados algoritmos parcialmente implementados.

- 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. Você deve submeter os arquivos *.h*, *.c* e o *.pdf* (relatório) na raiz do arquivo *.zip*. Use os nomes dos arquivos *.h* e *.c* exatamente como pedido.
7. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.

- Bom trabalho!

Descrição do Problema

A Secretaria Municipal de Saúde de Ouro Preto/MG precisa alocar médicos nas Unidades de Pronto Atendimento (UPA 24h) de modo a otimizar a oferta de atendimento para a população em um determinado dia da semana. Seu programa deve ajudar os gestores nessa tomada de decisão, listando as UPAs em ordem de prioridade.

É importante salientar que os gestores avaliam a quantidade de pacientes em espera nas unidades, considerando a classificação da gravidade definida durante a triagem. A classificação da gravidade se baseia no Protocolo de Manchester (adaptado) como mostra a Figura 1. Os gestores devem priorizar as UPAs com maior número de pacientes em espera para Emergência, Urgência e Sem Urgência, respectivamente. Caso duas unidades tenha o mesmo número de pacientes para Emergência, deve-se considerar a demanda subsequente e assim sucessivamente. Caso os quantitativos entre duas UPAs sejam exatamente iguais, os gestores optam por aquela com menor número de médicos de plantão. Se persistir o empate, as UPAs devem ser listadas em ordem alfabética.

Figura 1: Protocolo de Manchester (adaptado): Pulseiras coloridas sinalizam o nível de gravidade de cada caso.

Imposições e comentários gerais

Neste trabalho, as seguintes regras devem ser seguidas:

- Seu programa não pode ter *memory leaks*, ou seja, toda memória alocada pelo seu código deve ser corretamente liberada antes do final da execução. (Dica: utilize a ferramenta *valgrind* para se certificar de que seu código libera toda a memória alocada)
- Um grande número de *Warnings* ocasionará a redução na nota final.

O que deve ser entregue

- Código fonte do programa em C (bem identado e comentado).
- A documentação do trabalho (relatório - **formato:** PDF) deve conter:
  - **Implementação:** descrição sobre a implementação do programa. Não faça "print screens" de telas. Ao contrário, procure resumir ao máximo a documentação, fazendo referência ao que julgar mais relevante. É importante, no entanto, que seja descrito o funcionamento das principais funções e procedimentos utilizados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado. Muito importante: os códigos utilizados na implementação devem ser inseridos na documentação.
  - **Impressões gerais**: descreva o seu processo de implementação deste trabalho. Aponte coisas que gostou bem como aquelas que o desagradou. Avalie o que o motivou, conhecimentos que adquiriu, entre outros.

- **Análise**: deve ser feita uma análise dos resultados obtidos com este trabalho.
- **Conclusão**: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.

Como deve ser feita a entrega

Verifique se seu programa compila e executa na linha de comando antes de efetuar a entrega. Quando o resultado for correto, entregue via *run.codes* até a 15/05/2022 até 23:55 um arquivo **.ZIP** com o nome e sobrenome do aluno. Esse arquivo deve conter: (i) os arquivos *.c* e *.h* utilizados na implementação, e (ii) o relatório em **PDF**. Exemplo de um nome do arquivo a ser entregue: *pedro-silva.zip*.

Detalhes da implementação

O código-fonte deve ser modularizado corretamente em três arquivos: *principal.c*, *ordenacao.h* e *ordenacao.c*. O arquivo *principal.c* deve apenas invocar as funções e procedimentos definidos no arquivo *ordenacao.h*. A separação das operações em funções e procedimentos está a cargo do aluno, porém, não deve haver acúmulo de operações dentro de uma mesma função/procedimento.

Além disto, as informações de cada UPA devem ser armazenadas em um tipo abstrato de dados criado especificamente para isso. O conjunto de informações de todas as UPAS deve ser armazenado em um vetor alocado dinamicamente (e posteriormente desalocado) para cada caso de teste.

O aluno pode escolher qual algoritmo de ordenação será utilizado para abordar o problema, entretanto, o limite de tempo para solução de cada caso de teste é de apenas um segundo. Utilize suas habilidades de programação e de análise de algoritmos para desenvolver um algoritmo correto e rápido!

Entrada

A entrada é dada pelo número de UPAs a serem avaliadas, seguido pela lista dos nomes das UPAs (sem acentos ou caracteres especiais), com as quantidades de pacientes em espera para Emergência, Urgência e Sem Urgência, além do total de médicos plantonistas.

Entrada no Terminal

```
5
UPANorte 2 5 2 3
UPASul 2 4 0 1
UPALeste 3 7 2 1
UPAOeste 3 7 2 2
UPACentral 3 7 2 2
```

Saída

A saída deve ser a lista de UPAs em ordem de prioridade com seus respectivos quantitativos de pacientes em Emergência, Urgência e Sem Urgência e o total de médicos plantonistas.

Saída esperada no Terminal

```
UPALeste 3 7 2 1
UPACentral 3 7 2 2
UPAOeste 3 7 2 2
UPANorte 2 5 2 3
UPASul 2 4 0 1
```

Diretivas de Compilação

```
$ gcc -c ordenacao.c -Wall
$ gcc -c principal.c -Wall
$ gcc ordenacao.o principal.o -o exe
```