T03 – Vetor de struct e Busca Binária

Os dados dos clientes de uma clínica estão registrados em um arquivo texto no seguinte formato (Fig. 1): *nome*: *altura peso*

Leo Silva: 1.70 70
Eva Maria: 1.70 75
Ana: 1.80 40
Turing: 1.80 50
Luiza Loren: 1.80 75
Lovelace: 1.90 60
Sato: 2.0 60
Duda: 2.0 65
von Neumann: 2.0 80

Fig. 1 Arquivo texto

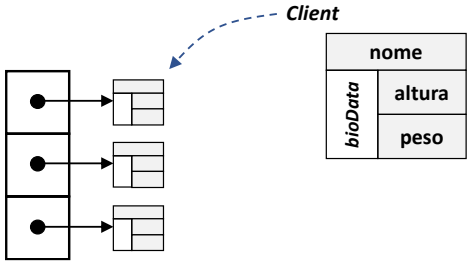

Fig. 2 Vetor de ponteiros para *struct*

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - media)^2}$$

Fig. 3 Desvio padrão

Considere que os dados neste arquivo estão ordenados crescentemente por altura e por peso. Considere também que os nomes nunca têm mais do que 80 caracteres. Na definição de estruturas, use obrigatoriamente typedef.

[1] A partir do arquivo texto, crie um vetor de ponteiros para estruturas contendo os dados de cada cliente, conforme ilustra a Fig.2. Você deve alocar dinamicamente a memória do tamanho exato. Para tanto, escreva obrigatoriamente a função int countLines(char* fName) que conta as linhas do arquivo texto. Para ler e montar o vetor escreva obrigatoriamente uma função chamada lerMontarVetor que recebe o nome do arquivo e o número de linhas e retorna o vetor de ponteiros. E, para imprimir o vetor, escreva obrigatoriamente uma função chamada imprimeVetor. Por fim, calcule a média de pesos e o desvio padrão (fórmula na Fig. 3), escrevendo as funções mean e standardDeviation.

[2] Escreva uma função que faça uma **busca binária** de maneira que, dada uma altura, retorna o índice (do vetor) que corresponde à pessoa de **maior** peso com a altura especificada[1]. Por exemplo, para 1.80 a função retorna o registro relativo a *Luiza Loren*, no caso da Fig. 1. Faça testes adequados. Por exemplo, teste também a situação que retorna o último elemento (*e.g.*: para 2.00 a função retorna o índice 8, no caso da Fig.1). Nos testes, imprima mensagens adequadas (*i.e.*, informativas, que não sejam apenas a exibição do índice). Faça também um teste no qual a busca não encontra a informação. Use obrigatoriamente uma função auxiliar de comparação, chamada comparaAltura, que tenha como argumento um ponteiro para *Client* (e.g., Client * b). E, em qualquer comparação durante a busca envolvendo a altura, use sempre esta mesma função.

Na *main*, faça os testes adequados de arquivo e memória. Faça também, no momento adequado, o teste para o caso do arquivo existir e estar vazio. Feche os arquivos e libere a memória nos momentos adequados (para liberar a memória escreva obrigatoriamente uma função auxiliar chamada freeMem). Exemplos de *output* nas Figs 4 e 5.

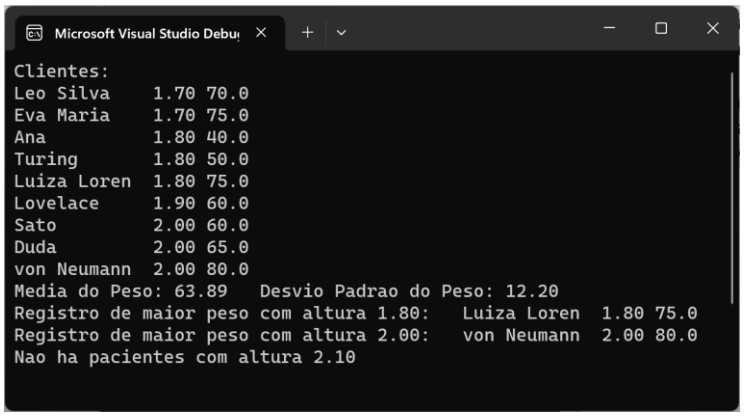

Fig.4 Arquivo existente e com dados

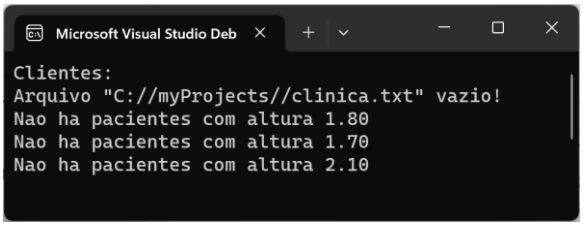

Fig.5 Arquivo existente, porém vazio

ATENÇÃO: Use obrigatoriamente um exemplo completamente diferente do exemplo acima (tanto em valores como em número de linhas). Deposite 3 arquivos no EAD: o programa completo (**.c**), o arquivo texto (**.txt**) e o arquivo PDF com os *outputs* (**.pdf**).

[1] Para este tipo de busca, dentro do algoritmo de busca binária, você deve fazer uma busca sequencial.