IF61C—Fundamentos de Programação 1

Lista de exercícios 13 - Funções (passagem de parâmetros por referência)

1. (*) Escreva uma função `void testaReferencia(float vet[], float *proxMedia, float *media)`. Ela recebe um vetor de floats e retorna (por referência): (a) o valor presente no vetor que esteja mais próximo da média dos elementos e (b) a média dos elementos do vetor.

2. (*) Adapte o exercício anterior para o protótipo `float testaReferencia(float vet[], float *proxMedia)`. Note que, neste caso, a média é retornada pela função!!

3. (*) Escreva a função `int conta_ocorrencias(char *str, char letra)`, a qual recebe por parâmetros uma letra e uma string. A função deve retornar o número de ocorrências da letra na string. Faça duas versões para o programa: uma usando índices e outra usando ponteiros para manipular o vetor. Pergunta: caso o protótipo da função acima fosse `int conta_ocorrencias(char str[], char letra)`, seria necessário fazer alguma alteração no código?

4. (**) Usando ponteiros para percorrer vetores, escreva uma função que copia em dest a substring de src com índice inicial `begin` e final `end`. Exemplo: Para `src="IF61C"`, `begin=2` e `end=4`, dest recebe “61C”. Protótipo: `void GetSubString(char dest[], char src[],int begin,int end);`. Lembre-se que vetores sempre são passados por referência.

5. (**) Usando ponteiros para percorrer a matriz, faça uma função `void encontraMaior(int n, int m[][NCOL]), int *k, int *lin, int* col)` . Ela recebe como entrada um inteiro `n`, uma matriz inteira `A` de dimensão `n x n` e devolve três inteiros: `k`, `lin` e `col`. O inteiro `k` é um maior elemento de `A` e é igual a `A[lin,col]`. Obs.: Se o elemento máximo ocorrer mais de uma vez, indique em lin e col qualquer uma das possíveis posições.

6. (***) Crie a função: `void busca(int vet[], int tam, int chave, int posicoes[], int *n)`. Você deve devolver em `posicoes[]` as posições de `vet` que possuem a `chave`, e devolver em `*n` o número de ocorrências da chave. OBS: Observe que, na chamada desta função, o vetor `posicoes` deve ter espaço suficiente (por exemplo, `tam`) para guardar todas as possíveis ocorrências da chave.

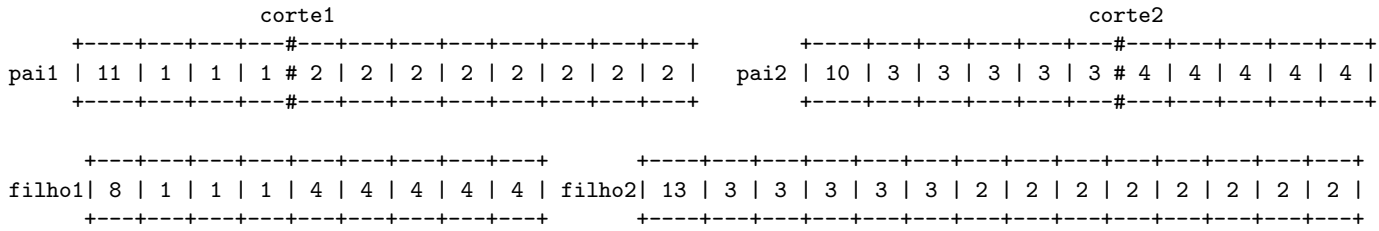
7. (***) Cebolinha é um personagem de história em quadrinhos que quando falava, trocava o ”R”pelo ”L”. A função acima deve gerar em dest uma versão do texto contido em src com todos ”R”e ”RR”trocados por ”L”, exceto no caso em que o ”R”ocorre no final de uma palavra. Protótipo: `void CebolinhaString(char dest[], char src[]);`

Complementares

1. Usando ponteiros para percorrer vetores, faça uma função que atribui a `dest` uma cópia de src tirando os espaços em branco do início e do final. Exemplo: Para `"Teste ok!  "` obtemos `"Teste ok!"`. Protótipo: `void TrimString(char dest[], char src[]);`

2. Usando ponteiros para percorrer vetores, escreva uma função que recebe uma string e um inteiro n, e retorna a n-ésima palavra (contagem começando do zero). Palavras são seqüências de caracteres delimitadas por caracteres. Protótipo: `char * palavra(char *entrada, int n)`

3. Um algoritmo genético é um procedimento computacional de busca inspirado no processo biológico de evolução que otimiza a solução de um problema. O problema é modelado por: uma população de indivíduos que representam possíveis soluções; uma função que avalia a qualidade da solução representada por cada indivíduo da população e um conjunto de operações genéticas. O procedimento consiste em aplicar os operadores genéticos sobre a população, gerando novos indivíduos, e selecionar os mais apropriados para constituirem a nova população. Esse processo é repetido até que uma solução adequada seja obtida. Dentre os operadores genéticos um importante é o de *crossover* de dois indivíduos. Esse operador corta em duas partes as seqüências de genes de dois indivíduos pais (pai_1 e pai_2) e gera dois novos indivíduos filhos ($filho_1$ e $filho_2$), onde $filho_1$ é dado pela concatenação da primeira parte dos genes de pai_1 com a segunda parte dos genes de pai_2, enquanto $filho_2$ é dado pela contatenação da primeira parte de pai_2 com a segunda parte de pai_1. O diagrama abaixo exemplifica a operação de crossover em indivíduos representados por vetores de inteiros onde a primeira posição contém o tamanho do vetor.



Escreva uma função em C que execute a operação de crossover descrita acima **apenas para o filho1**. A função deve receber cinco parâmetros: um vetor representando o primeiro pai, a posição de corte no primeiro pai, um vetor representando o segundo pai, a posição do corte no segundo pai, e um vetor que receberá o novo indivíduo. No exemplo apresentado a chamada do procedimento seria:

```
corte1 = 3; corte2 = 5;

crossover(pai1, corte1, pai2, corte2, filho1);
```

Note que os vetores devem iniciar na posição zero e essa posição é usada para armazenar o tamanho do vetor. No caso do exemplo, pai1[0]=11, pai2[0]=10, filho1[0]=8 e filho2[0]=13. O cabeçalho da função é `void crossover(int *pai1, int corte1, int *pai2, int corte2, int*filho1);`