Programação Imperativa 2021/2022 (CC1003), DCC/FCUP

Folha 7

7.1 Escreva um programa que lê uma sequência de valores inteiros positivos terminada por -1 e no final escreve toda a sequência sem repetidos.

7.2 Escreva um programa que lê carateres e contabiliza o número de vezes que ocorreu cada letra (A a Z) ignorando a distinção entre maiúsculas e minúsculas. No final deve imprimir uma tabela com as contagens:

```
A: 3
B: 2
C: 2
...
```

7.3 Escreva uma função `int calc(char str[])` que implementa uma mini-calculadora: a cadeia de carateres dada tem sempre carateres correspondente a algarismos decimais de `'0'` até `'9'` (pelo menos um), seguidos por um sinal de operação (`'+'`, `'-'`, `'*'`) seguido por mais carateres correspondente a algarismos decimais. A função deve calcular o valor da expressão e retornar o inteiro correspondente. Exemplos: `calc("15-3")` dá 12; `calc("2*3")` dá 6.

7.4 Defina uma função `void eliminar(char str[], char ch)` que elimina a primeira ocorrência de uma carater `ch` de uma cadeia de carateres. Exemplo: se `str = "ABBA"`, então depois de execcutar `eliminar(str, 'B')` devemos ter `str = "ABA"`. Tenha o cuidado de colocar corretamente o terminador `'\0'`.

7.5 Considere a seguinte função para inserir um valor num vetor de n elementos baseada no algoritmo usado na *ordenação por inserção*; assumimos que os valores do vetor estão por ordem ascendente à entrada da função; no final, o vetor terá mais um valor e mantém a ordem ascendente.

```
void inserir(int vec[], int n, int x) {
    int j = n-1;
    while(j>=0 && vec[j]>x) {
        vec[j+1] = vec[j];
        j--;
    }
    vec[j+1] = x;
}
```

Acrescente asserções à função para exprimir as seguintes condições de correção:

- Uma *pré-condição* à entrada da função: o vetor está ordenado, ou seja, $\mathrm{vec}[i] \leq \mathrm{vec}[i+1]$, para todos os índices $0 \leq i < n$.
- Uma *pós-condição* à saída da função: o vetor continua ordenado e o comprimentos aumentou de 1 unidade, ou seja, $\mathrm{vec}[i] \leq \mathrm{vec}[i+1]$, para todos os índices $0 \leq i \leq n$.

7.6 Implemente uma função `void ordenar(char str[])` que ordena os carateres numa cadeia pelos seus códigos. Por exemplo: se `str = "ALGORITMO"` então após execução devemos ter `str = "AGILMOORT"`.

7.7 Escreva uma função `int anagramas(char str1[], char str2[])` que determina se duas cadeias de carateres são *anagramas*, isto é, se se escrevem com os mesmos carateeres. O resultado deve ser 1 em caso afirmativo e 0 caso contrário. Por exemplo, ”deposit” e ”topside” são anagramas.

7.8 O conceito de anagrama pode ser aplicado mais geralmente a uma *frase* com várias palavras, ignorando os espaços, sinais de pontuação e a distinção entre maiúsculas e minúsculas. Por exemplo, a pergunta em Latim `"Quid est veritas?"` (*O que é a verdade?*) é um anagrama de `"Est vir qui adest"` (*É o homen que aqui está*). Modifique o programa do exercício anterior para testar anagramas neste sentido mais lato.
Sugestão: comece por escrever uma função auxiliar `void normalizar(char str[])` para “normalizar” uma cadeia de carateres eliminando todos os carateres não-letra e convertendo todas as letras em minúsculas.

7.9 O programa para *Quicksort* apresentado na aula teórica escolhe sempre o primeiro valor da sub-sequência para *pivot*. Isto causa que a complexidade no caso em que a sequência está ordenada seja quadrática.
Pretende-se melhorar esta escolha usando como *pivot* a mediana do primeiro, último e do ponto-médio da sub-sequência. Para tal, basta trocar a mediana com o valor no início da partição. Implemente esta modificação no programa do *Quicksort*.