Programação Estruturada

2000/2001

Exercício I

Implemente uma função com o protótipo

```
void comuns(int a[ ], int m, int b[ ], int n);
```

onde `a[ ]` e `b[ ]` contêm respectivemte `m` e `n` elementos distintos ordenados por ordem crescente e sem repetição. A função deve imprimir por ordem crescente os elementos comuns aos 2 vectores. Por exemplo, para os vectores `[1,3,4,7,8]` e `[2,3,4,6,8,9]`, deve ser impresso `3 4 8`

Exercício II

Existem 2 vectores `proibidos[ ]` e `gratis[ ]` globais de "strings" que representam respectivamente *prefixos* de números de telefones de utilização proibida e grátis. Um exemplo:

```
char *proibidos[ ] = {"23", "2677"};
char *gratis[ ] = {"0800", "11"};
```

Implemente uma função com o protótipo `int f(char *tel)` que retorna respectivamente 1, 0 ou -1 conforme respectivamente o telefone `tel` é proibido (algum `proibidos[i]` é prefixo dele), grátis (algum `gratis[i]` é prefixo dele), ou nem grátis nem proibido. Para o exemplo dado, temos que `f("234455666")` retorna 1.
Recorda-se que, dadas duas palavras p e s, diz-se que p é prefixo de s se existe uma palavra y (possivelmente ε) tal que $s = py$.

Este função deverá utilizar outra com o protótipo `int prefixo(char *p, char *s)` que retorna 1 se `p` é prefixo de `s` e 0 caso contrário. Considera-se que uma palavra é prefixo de si própria.

Exercício III

Considere a seguinte codificação de um texto t ("string" contendo apenas letras maiúsculas) utilizando uma chave k (vector de inteiros compreendidos entre 0 e 25): ao código da primeira letra de t soma-se o primeiro inteiro do vector k, ajustando, se necessário o resultado de modo a se obter uma letra maiúscula. Procede-se de modo idêntico para as restantes letras de t. Sempre que a chave k chege ao fim, passa a ser utilizada de novo a partir do início. Seja, por exemplo

$$t = \texttt{"VAMOSATACAR"},\ k = \texttt{[7,2,1]}$$

Obtemos a cifra `CCNVUBACDHT`; Note por exemplo que se, a partir da letra `V`, avançarmos 7 posições para a frente de forma "rotativa", obtemos a letra `C`:

$$\texttt{V} \rightarrow \texttt{W} \rightarrow \texttt{X} \rightarrow \texttt{Y} \rightarrow \texttt{Z} \rightarrow \texttt{A} \rightarrow \texttt{B} \rightarrow \texttt{C}$$

1. Implemente em linguagem C uma função com o protótipo

   ```
   char *cifra(char *pal, int k[], int nk);
   ```

   que dá como resultado a codificação da palavra `x` com a chave `k` de comprimento `nk`. Para o exemplo dado: `cifra("VAMOSATACAR",k,3)` = `"CCNVUBACDHT"`.

2. Implemente em linguagem C uma função com o protótipo

   ```
   char *decifra(char *pal, int k[], int nk);
   ```

   que dá como resultado a descodificação da palavra `x` com a chave `k` de comprimento `nk`. Para o exemplo dado: `decifra("CCNVUBACDHT",k,3)` = `"VAMOSATACAR"`.

Exercício IV

O tipo de uma árvore binária de pesquisa é caracterizada da forma seguinte

```
typedef struct no{
  char *nome;
  struct no *esquerda;
  struct no *direita;
} *ARVORE;
```

Como sabe, numa árvore binária de pesquisa em que não ocorrem elementos repetidos, todos os elementos da sub-árvore esquerda de um nó são menores e os da sub-árvore direita maiores do que o elemento contido nesse nó.
As implementações seguintes devem ser recursivas e não devem utilizar qualquer variável global (ou `static`). Suponha que em nenhuma das árvores mencionadas ocorrem elementos repetidos.

1. Implemente uma função com o protótipo `int conta(ARVORE a);` que imprime o número total de caracteres contidos em todos os nomes associados aos nós de `a`.

2. Implemente uma função com o protótipo `void comuns(ARVORE a, ARVORE b);` que imprime os nomes comuns às árvores `a` e `b`.
   Método (Obrigatório) Percorrer por uma ordem apropriada a árvore `a`, procurando cada nome encontrado na árvore `b`. A sua implementação deve incluir uma função que implemente (eficientemente) a pesquisa de uma palavra numa árvore binária de pesquisa.