Programação Imperativa
1º Ano – LEI/LCC

Exame Especial – Duração: 2 horas

15 de Setembro de 2007

Parte I

Esta parte do exame representa 12 valores da cotação total. Cada uma das 6 alíneas está cotada em 2 valores. **A obtenção de uma classificação abaixo de 8 valores nesta parte implica a reprovação no exame.**

1. Defina a função `int isNumber(char *)` que recebe uma *string* (terminada pelo caracter `\0`) e verifica se forma um número válido, isto é, só contém algarismos (de 0 a 9), no máximo um '.', podendo começar por um sinal '+' ou '-'.

2. Defina uma função `int maiorLinha (char [])` que recebe como argumento o nome de um ficheiro e retorna o comprimento da linha mais comprida (se o ficheiro estiver vazio ou não existir, a função deve retornar um valor negativo).

3. Defina uma função `void transposta (int a [N][N])` que recebe uma matriz quadrada e que a transforma na sua transposta. Faça-o sem usar outra matriz auxiliar intermédia.

4. Considere a seguinte função que inverte uma lista ligada de inteiros.

```
typedef struct slista {
              int valor;
              struct slista *seg;
             } *Lista;

Lista reverse (Lista l) {
         Lista r, aux;
         r = NULL;

         while (l != NULL) {
                  aux = l ; l = l->seg;
                  aux-> seg = r; r = aux;
         }
         return r;
}
```

   Uma lista diz-se *palíndrome* se é igual à sua inversa. Um aluno de Programação Imperativa resolveu usar a função acima para testar se uma lista é ou não palíndrome. O código apresentado foi:

```
int ePalindrome (Lista l){
    Lista b;
    resultado = 1;

    b = reverse (l);
    while (b!=NULL) {
        if (b->valor != l->valor) resultado = 0;
        b = b->seg; l = l->seg;
        }
```

```
return (resultado);
}
```

(a) Ao testar esta função constatou que em listas com mais do que um elemento a função dava sempre valor falso. Além disso em certas ocasiões, obtinha um `Segmentation Fault`. Explique o porquê de tais factos.

(b) Apresente uma definição da função em causa (usando ou não a função reverse)

(c) Apresente uma versão recursiva da função `reverse()`

Parte II

1. Considere uma lista ligada dinâmica contendo informação sobre as eleições que se realizaram num determinado país. Há três candidatos sujeitos a votação: A, B e C. Cada célula da lista contém informação sobre a votação obtida por cada candidato, em cada distrito do país.

```
typedef struct sVotos{
  char idistrito[N];
  int vA;
  int vB;
  int vC;
  struct sVotos *distseg;
} *ListaVotos;
```

(a) Defina uma função que calcule quantos distritos já foram apurados.

(b) Defina uma função que receba a lista de votações por distrito e um candidato e indique em quantos distritos ganhou esse candidato.

```
int quantos(ListaVotos l, char c)
{...}
```

(c) Defina uma função que calcule quem ganhou as eleições, apresentando o total de votos conseguidos.

```
char ganhou (ListaVotos l)
{...}
```

(d) Defina uma função que insira ordenadamente, na lista de votos apurados por distrito ordenada por ordem alfabética de nome de distrito, o número de votos obtidos por um dado candidatos num dado distrito (se já existe, acrescenta a informação do candidato, senão acrescenta a célula com o distrito e o candidato).

```
ListaVotos insere (int votos, char candidato, char nomeDist[N], ListaVotos l)
{...}
```