Exame de Recurso de Programação Imperativa

LCC/MIEF/MIEI

21 de Junho de 2019

Parte A

Considere as seguintes definições de tipos:

```
typedef struct slist {
   int valor;
   struct slist *prox;
} *LInt;
```

```
typedef struct nodo {
   int valor;
   struct nodo *esq, *dir;
} *ABin;
```

1. Defina uma função `char charMaisfreq (char s[])` que determina qual o caracter mais frequente numa *string*. A função deverá retornar 0 no caso de `s` ser a *string* vazia.

2. Defina uma função `int maxCresc (int v[], int N)` que calcula o comprimento da maior sequência crescente de elementos consecutivos num vector `v` com `N` elementos. Por exemplo, se o vector contiver 10 elementos pela seguinte ordem: `1, 2, 3, 2, 1, 4, 10, 12, 5, 4`, a função deverá retornar 4, correspondendo ao tamanho da sequência `1, 4, 10, 12`.

3. Defina uma fução `LInt somasAcL (LInt l)` que, dada uma lista de inteiros, constrói uma nova lista de inteiros contendo as somas acumuladas da lista original (que deverá permanecer inalterada). Por exemplo, se a lista `l` tiver os valores `[1,2,3,4]` a lista contruída pela invocação de `somasAcL (l)` deverá conter os valores `[1,3,6,10]`.

4. Defina uma função `LInt rotateL (LInt l)` que coloca o primeiro elemento de uma lista no fim. Se a lista for vazia ou tiver apenas um elemento, a função não tem qualquer efeito prático (i.e., devolve a mesma lista que recebe como argumento). Note que a sua função não deve alocar nem libertar memória. Apenas re-organizar as células da lista.

5. Considere o seguinte tipo para representar a posição de um robot numa grelha.

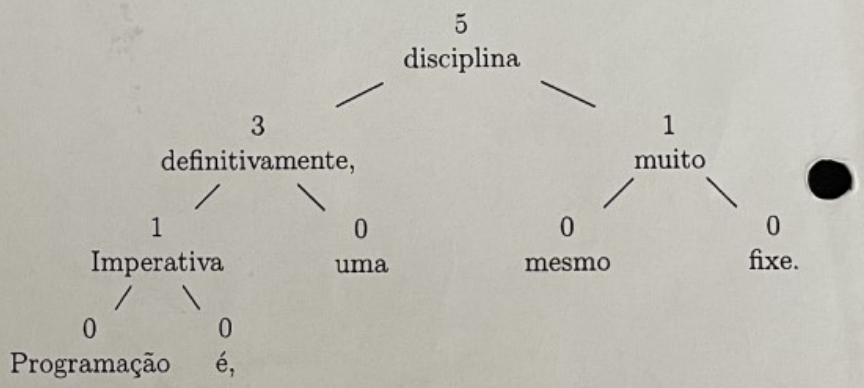
```
typedef struct posicao {
    int x, y;
} Posicao;
```

Defina a função `int vizinhos (Posicao p, Posicao pos[], int N)` que, dada uma posição e um array com `N` posições, calcula quantas dessas posições são adjacentes à posição dada.

Parte B

Uma *Rope* é uma árvore binária que representa um "texto", mas que possibilita um acesso eficiente às linhas intermédias desse mesmo texto. Cada nó da árvore irá armazenar uma linha do texto (uma *string*), sendo que na sub-árvore esquerda são armazenadas as linhas que lhe antecedem, e na sub-árvore direita as linhas que lhe sucedem. É ainda armazenada a informação do número de linhas contidas na sub-árvore esquerda, para permitir determinar a posição da linha armazenada no texto. Exemplificando, o texto apresentado abaixo no lado esquerdo poderia ser representado pela *Rope* do lado direito.

```
0 Programação
1 Imperativa
2 é,
3 definitivamente,
4 uma
5 disciplina
6 mesmo
7 muito
8 fixe.
```



Considere a seguinte estrutura de dados para representar *Rope*s em C:

```
typedef struct nodo {
  int nlesq;
  char* linha;
  struct nodo *esq, *dir;
} *Rope;
```

Defina as seguintes funções de manipulação de *Rope*s em C:

1. Defina a função `int nlinhas(Rope r)` que retorna o número de linhas do texto representado;
2. Defina a função `void print(Rope r)` que imprime no ecrã o texto representado na *Rope* (obs: assuma que as *string*s das linhas não incluem o terminador "`\n`").
3. Defina a função `char index(int l,int c,Rope r)` que retorna o caracter que se encontra na linha l e coluna c do texto (`\0` se estiver fora dos limites). Considere que os índices das linhas/caracteres começam em 0.
4. Defina a função `Rope concat(Rope r1,Rope r2)` que concatena o texto de duas *Rope*s.
5. Defina a função `int insereLinha(int l,char *str,Rope *r)` que acrescenta uma nova linha com texto *str* na posição l e retorna 0 em caso de sucesso (`l` é uma posição válida, i.e. *Rope* contém pelo menos l linhas).