

Programação Imperativa

1º Ano – LEI/LCC

Questões 1ª Parte

Listas Ligadas

Considere o seguinte tipo para representar listas ligadas de inteiros

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

1. Apresente uma definição não recursiva da função `int length (LInt)` que calcula o comprimento de uma lista ligada.
2. Apresente uma definição não recursiva da função `void freeL (LInt)` que liberta o espaço ocupado por uma lista.
3. Apresente uma definição não recursiva da função `void imprimeL (LInt)` que imprime no ecrã os elementos de uma lista (um por linha).
4. Apresente uma definição não recursiva da função `LInt reverseL (LInt)` que inverte uma lista (sem criar uma nova lista).
5. Apresente uma definição não recursiva da função `void insertOrd (LInt *, int)` que insere ordenadamente um elemento numa lista ordenada.
6. Apresente uma definição não recursiva da função `int removeOneOrd (LInt *, int)` que remove um elemento de uma lista ordenada. Retorna 1 caso o elemento a remover não exista, 0 no outro caso.
7. Defina uma função `merge (LInt *r, LInt a, LInt b)` que junta duas listas ordenadas (`a` e `b`) numa única lista ordenada (`r`).
8. Defina uma função `void splitMS (LInt l, int x, LInt *mx, LInt *Mx)` que, dada uma lista ligada `l` e um inteiro `x`, parte a lista em duas (retornando os endereços dos primeiros elementos da lista em `*mx` e `*Mx`): uma com os elementos de `l` menores do que `x` e a outra com os restantes. Note que esta função não deverá criar cópias dos elementos da lista.
9. Defina uma função `LList parteAmeio (LList *l)` que, parte uma lista não vazia `*l` a meio. Se `x` contiver os elementos `{1,2,3,4,5}`, após a invocação `y=parteAmeio(&x)` a lista `y` deverá conter os elementos `{1,2}` e a lista `x` os restantes `{3,4,5}`

10. Apresente uma definição não recursiva da função `int removeAll (LInt *, int)` que remove todas as ocorrências de um dado inteiro de uma lista, retornando o número de células removidas.
11. Apresente uma definição da função `int removeDups (LInt *)` que remove os valores repetidos de uma lista (deixando apenas a primeira ocorrência).
12. Apresente uma definição da função `int removeMaiorL (LInt *)` que remove (a primeira ocorrência) o maior elemento de uma lista não vazia, retornando o valor desse elemento.
13. Apresente uma definição não recursiva da função `void init (LInt *)` que remove o último elemento de uma lista não vazia (libertando o correspondente espaço).
14. Apresente uma definição não recursiva da função `void appendL (LInt *, int)` que acrescenta um elemento no fim da lista.
15. Apresente uma definição da função `void concatL (LInt *a, LInt b)` que acrescenta a lista `b` à lista `*a`.
16. Apresente uma definição da função `LInt cloneL (LInt)` que cria uma nova lista ligada com os elementos pela ordem em que aparecem na lista argumento.
17. Apresente uma definição não recursiva da função `LInt cloneRev (LInt)` que cria uma nova lista ligada com os elementos por ordem inversa.
 Por exemplo, se a lista `l` tiver 5 elementos com os valores `{1,2,3,4,5}` por esta ordem, a invocação `cloneRev(l)` deve corresponder a uma nova lista com os elementos `{5,4,3,2,1}` por esta ordem.
18. Defina uma função `int maximo (LInt l)` que calcula qual o maior valor armazenado numa lista não vazia.
19. Apresente uma definição iterativa da função `int take (int n, LInt *l)` que, dado um inteiro `n` e uma lista ligada de inteiros `l`, apaga de `l` todos os nodos para além do `n`-ésimo (libertando o respectivo espaço). Se a lista tiver `n` ou menos nodos, a função não altera a lista.
 A função deve retornar o comprimento final da lista.
20. Apresente uma definição iterativa da função `int drop (int n, LInt *l)` que, dado um inteiro `n` e uma lista ligada de inteiros `l`, apaga de `l` os `n` primeiros elementos da lista (libertando o respectivo espaço). Se a lista tiver `n` ou menos nodos, a função liberta a totalidade da lista.
 A função deve retornar o número de elementos removidos.
21. O tipo `LInt` pode ser usado ainda para implementar listas circulares. Defina uma função `LInt Nforward (LInt l, int N)` que, dada uma lista circular dá como resultado o endereço do elemento da lista que está `N` posições à frente.
22. Defina uma função `int listToArray (LInt l, int v[], int N)` que, dada uma lista `l`, preenche o array `v` com os elementos da lista.
 A função deverá preencher no máximo `N` elementos e retornar o número de elementos preenchidos.

23. Defina uma função `LInt arrayToList (int v[], int N)` que constrói uma lista com os elementos de um array, pela mesma ordem em que aparecem no array..
24. Defina uma função `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]`.
25. Defina uma função `void remreps (LInt l)` que, dada uma lista ordenada de inteiros, elimina dessa lista todos os valores repetidos assegurando que o espaço de memória correspondente aos nós removidos é correctamente libertado.
26. 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.
27. Defina uma função `LInt parte (LInt l)` que parte uma lista `l` em duas: na lista `l` ficam apenas os elementos das posições ímpares; na lista resultante ficam os restantes elementos.
Assim, se a lista `x` tiver os elementos `{1,2,3,4,5,6}` a chamada `y = parte (x)`, coloca na lista `y` os elementos `{2,4,6}` ficando em `x` apenas os elementos `{1,3,5}`

Árvores Binárias

Considere o seguinte tipo para representar árvores binárias de inteiros

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

28. Apresente uma definição da função `int altura (ABin)` que calcula a altura de uma árvore binária.
29. Defina uma função `ABin cloneAB (ABin)` que cria uma cópia de uma árvore.
30. Defina uma função `void mirror (ABin *)` que inverte uma árvore (sem criar uma nova árvore).
31. Defina a função `void inorder (ABin , LInt *)` que cria uma lista ligada de inteiros a partir de uma travessia *inorder* de uma árvore binária.
32. Defina a função `void preorder (ABin , LInt *)` que cria uma lista ligada de inteiros a partir de uma travessia *preorder* de uma árvore binária.
33. Defina a função `void posorder (ABin , LInt *)` que cria uma lista ligada de inteiros a partir de uma travessia *posorder* de uma árvore binária.

34. Apresente uma definição da função `int depth (ABin a, int x)` que calcula o nível (menor) a que um elemento está numa árvore binária (-1 caso não exista).
35. Defina uma função `int freeAB (ABin a)` que liberta o espaço ocupado por uma árvore binária, retornando o número de nodos libertados.
36. Defina uma função `int pruneAB (ABin *a, int l)` que remove (libertando o espaço respectivo) todos os elementos da árvore `*a` que estão a uma profundidade superior a `l`, retornando o número de elementos removidos.
Assuma que a profundidade da raiz da árvore é 1, e por isso a invocação `pruneAB(&a,0)` corresponde a remover todos os elementos da árvore `a`.
37. Defina uma função `int iguaisAB (ABin a, ABin b)` que testa se duas árvores são iguais (têm os mesmos elementos e a mesma forma).
38. Defina uma função `LInt nivell (ABin a, int n)` que, dada uma árvore binária, constrói uma lista com os valores dos elementos que estão armazenados na árvore ao nível `n` (assuma que a raiz da árvore está ao nível 1).
39. Defina uma função `int nivelV (ABin a, int n, int v[])` que preenche o vector `v` com os elementos de `a` que se encontram no nível `n`.
Considere que a raiz da árvore se encontra no nível 1.
A função deverá retornar o número de posições preenchidas do array.
40. Defina uma função `int dumpABin (ABin a, int v[], int N)` que dada uma árvore `a`, preenche o array `v` com os elementos da árvore segundo uma travessia inorder. A função deverá preencher no máximo `N` elementos e retornar o número de elementos preenchidos.
41. Defina uma função `ABin somasAcA (ABin a)` que, dada uma árvore de inteiros, calcula a árvore das somas acumuladas dessa árvore.
A árvore calculada deve ter a mesma forma da árvore recebida como argumento e em cada nodo deve conter a soma dos elementos da sub-árvore que aí se inicia.
42. Apresente uma definição da função `int contaFolhas (ABin a)` que dada uma árvore binária de inteiros, conta quantos dos seus nodos são folhas, i.e., que não têm nenhum descendente.
43. Defina uma função `ABin cloneMirror (ABin a)` que cria uma árvore nova, com o resultado de inverter a árvore (efeito de espelho).
44. Apresente uma definição não recursiva da função `int addOrd (ABin *a, int x)` que adiciona um elemento a uma árvore binária de procura. A função deverá retornar 1 se o elemento a inserir já existir na árvore ou 0 no outro caso.
45. Apresente uma definição não recursiva da função `int lookupAB (ABin a, int x)` que testa se um elemento pertence a uma árvore binária de procura.
46. Apresente uma definição da função `int depthOrd (ABin a, int x)` que calcula o nível a que um elemento está numa árvore binária de procura (-1 caso não exista).

47. Apresente uma definição não recursiva da função `int maiorAB (ABin)` que calcula o maior elemento de uma árvore binária de procura não vazia.
48. Defina uma função `void removeMaiorA (ABin *)` que remove o maior elemento de uma árvore binária de procura.
49. Apresente uma definição da função `int quantosMaiores (ABin a, int x)` que, dada uma árvore binária de procura de inteiros e um inteiro, conta quantos elementos da árvore são maiores que o inteiro dado.
50. Apresente uma definição da função `void listToBTree (LInt l, ABin *a)` que constrói uma árvore binária de procura a partir de uma lista ligada ordenada.
51. Apresente uma definição da função `int deProcura (ABin a)` que testa se uma árvore é de procura.