Lista de Exercícios 4 – Variáveis Dinâmicas

Esta lista prevê a utilização de estruturas dinâmicas de dados, cujos tipos estruturados precisam ser sempre definidos antes que a subprograma possa ser utilizado, pois, tanto na passagem de parâmetros quanto na declaração de variáveis o nome do tipo (que normalmente representa um nó da lista) precisa ser previamente conhecido. Então, sempre defina um novo tipo.

`aloca(` *`nome-do-tipo`* `)` – esta **função**, que recebe o nome de um tipo, solicita ao sistema operacional que aloque uma área de memória de tamanho suficiente para guardar um valor daquele tipo e retorna o endereço inicial desta área de memória, que sempre deve ser atribuído a uma variável ponteiro. Caso não haja memória, a função retorna o valor constante especial `NULO`, que representa um endereço nulo, inexistente.

`desaloca(` *`variável-ponteiro`* `)` – este **procedimento** recebe uma variável do tipo `ponteiro` que deve ter o endereço de uma variável dinâmica que não é mais desejada, solicitando ao sistema operacional que torne novamente disponível (libere) o espaço por ela ocupado. Isto não modifica nem o valor guardado na *`variável-ponteiro`* e nem o conteúdo da variável dinâmica por ela apontada, apenas marca o espaço de memória como novamente disponível para novas utilizações. Porém, caso o endereço não seja de um espaço de memória alocado (ou seja, o de uma variável dinâmica), o procedimento falha e o seu programa sofre uma parada abrupta, ou seja, é um erro tentar usar `desaloca()` com um ponteiro que tem lixo.

↑ – operador de derreferência de uma variável ponteiro para a variável dinâmica cujo endereço a variável ponteiro armazena.

Exemplo: se `p` é uma variável do tipo `ponteiro para` *`M`*, sendo *`M`* um nome de tipo existente, então `p`↑ é uma variável (dinâmica) do tipo *`M`*, e como tal pode ser utilizada.

Consultas

1. Faça o algoritmo de uma função lógica para retornar se uma lista encadeada simples de inteiros, cujo ponteiro inicial é passado como parâmetro, está vazia.

2. Faça o algoritmo de uma função inteira **[iterativa|recursiva]** para retornar o tamanho de uma lista encadeada simples de reais, cujo ponteiro inicial é passado como parâmetro.

3. Faça o algoritmo de uma função lógica **[iterativa|recursiva]** para dizer se um determinado elemento x real aparece na lista encadeada simples.

4. Faça o algoritmo de uma função inteira **[iterativa|recursiva]** para dizer quantas vezes um determinado elemento x real aparece na lista encadeada simples.

5. Faça o algoritmo de uma função *string* **[iterativa]** que retorna a *string* que encontra-se na posição central de uma lista simplesmente encadeada. Retorne a *string* vazia no caso da lista estar vazia. Se a lista tem uma quantidade par de elementos, retorne o elemento da posição inteira imediatamente inferior à metade.

6. Faça o algoritmo de uma função *string* **[iterativa|recursiva]** que retorna o elemento que encontra-se na posição p de uma lista simplesmente encadeada. Retorne a *string* vazia no caso da posição não existir.

7. Faça o algoritmo de uma função real **[iterativa]** para retornar a média aritmética simples dos valores inteiros de uma lista encadeada simples. Retorne zero no caso da lista estar vazia.

8. Faça o algoritmo de uma função *string* **[iterativa|recursiva]** para retornar o maior valor *string* existente em uma lista encadeada simples. Retorne a *string* vazia em caso da lista estar vazia.

9. Faça o algoritmo de uma função lógica **[iterativa|recursiva]** para dizer se um determinado elemento x aparece junto de um elemento y, nesta ordem, na lista encadeada simples de reais.

10. Faça o algoritmo de uma função lógica **[iterativa|recursiva]** para dizer se dois elementos x e y aparecem, nesta ordem, mas não necessariamente juntos, na lista encadeada simples de reais.

11. Faça o algoritmo de uma função lógica **[iterativa|recursiva]** para dizer se um determinado elemento x real aparece na lista encadeada simples, que pressupõe-se que esteja ordenada de forma ascendente.

12. Faça o algoritmo de uma função lógica **[iterativa|recursiva]** que diz se duas listas simplesmente encadeadas de *strings* são exatamente iguais, ou seja, se têm exatamente os mesmos valores nas mesmas posições.

13. Faça o algoritmo de uma função lógica **[iterativa]** que diz se todos os elementos de uma lista simplesmente encadeada encontram-se também em outra lista simplesmente encadeada.

Inserções

14. Faça o algoritmo de uma função lógica para inserir um elemento x *string* no início de uma lista simplesmente encadeada.

15. Faça o algoritmo de uma função lógica **[iterativa|recursiva]** para inserir um elemento x *string* no fim da lista simplesmente encadeada.

16. Faça o algoritmo de uma função lógica **[iterativa|recursiva]** para inserir um elemento x inteiro na posição ordenada de uma lista simplesmente encadeada que está ordenada.

17. Faça o algoritmo de uma função lógica **[iterativa|recursiva]** para inserir um elemento x *string* na posição ordenada de uma lista simplesmente encadeada que está ordenada, desde que o mesmo ainda não exista na lista.

18. Faça o algoritmo de uma função lógica **[iterativa|recursiva]** para inserir um elemento x real numa posição p de uma lista **[simplesmente|duplamente]** encadeada. Um retorno `VERDADEIRO` significa que o elemento x foi inserido exatamente na posição p, e `FALSO` em caso contrário.

Remoções

19. Faça o algoritmo de uma função lógica para remover o primeiro elemento de uma lista simplesmente encadeada de *strings*, retornando `VERDADEIRO` se o elemento foi removido com sucesso.

20. Faça o algoritmo de uma função lógica **[iterativa|recursiva]** para remover um elemento da última posição de uma lista simplesmente encadeada de *strings*, retornando `VERDADEIRO` se o elemento foi removido com sucesso.

21. Faça o algoritmo de uma função lógica **[iterativa|recursiva]** para remover um elemento da posição p de uma lista simplesmente encadeada de *strings*, retornando `VERDADEIRO` se o elemento foi removido com sucesso.

22. Faça o algoritmo de uma função lógica **[iterativa|recursiva]** para remover a primeira ocorrência do elemento x de uma lista **[simplesmente|duplamente]** encadeada de *strings*, retornando `VERDADEIRO` se o elemento foi removido com sucesso.

23. Faça o algoritmo de uma função lógica **[iterativa|recursiva]** para remover todos os elementos com valores iguais ou maiores que x de uma lista simplesmente encadeada de *strings*, retornando `VERDADEIRO` se pelo menos um elemento foi removido com sucesso.

24. Faça o algoritmo de uma função inteira **[iterativa|recursiva]** para remover todos os elementos duplicados que aparecem em uma lista simplesmente encadeada de reais ordenada de forma ascendente, retornando a quantidade de elementos (nós) efetivamente retirados.

Alterações

25. Faça o algoritmo de um procedimento **[iterativa|recursiva]** que faça com que o último elemento de uma lista simplesmente encadeada de reais, recebida por parâmetro, vá para a primeira posição da mesma.

26. Faça o algoritmo de uma função lógica **[iterativa|recursiva]** para juntar dois elementos consecutivos de uma lista simplesmente encadeada de reais, das posições p e $p+1$, somando seus valores, em um único nó. Caso alguma das posição não exista, nada deve ser feito. A função deve retornar `VERDADEIRO` apenas no caso da junção ter sido feita.

Criações

27. Faça o algoritmo de uma função ponteiro de duplicação de uma lista simplesmente encadeada de reais, passada por parâmetro, retornando o ponteiro para a lista-cópia, ou `NULO` em caso de erro.

28. Faça o algoritmo de um função ponteiro que faça um *merge* de duas listas **[simplesmente|duplamente]** encadeadas ordenadas de *strings*, criando e retornando o ponteiro da nova lista que continue ordenada, ou `NULO` em caso de erro.

29. Dada uma lista simplesmente encadeada de inteiros, faça o algoritmo de um procedimento **[iterativa|recursiva]** para criar e devolver outras duas que contenham, respectivamente, somente os números pares e somente os números ímpares, mantendo a ordem relativa original dos elementos nas novas listas criadas.

30. Faça o algoritmo de uma função ponteiro **[iterativa|recursiva]** que realize a soma de dois polinômios passados como parâmetro. Os polinômios são representados por listas encadeadas simples, nas quais os nós contêm os valores do *coeficiente* e do *expoente* de cada termo do polinômio, ambos reais, sendo que os termos nulos (de coeficiente zero) **não** devem aparecer. Por exemplo, o polinômio constante igual a 0 (zero) deve ser representado por uma lista *nula* (uma lista na qual seu ponteiro aponta para NULO). Os polinômios originais não devem sofrer modificações. Considere que os nós das listas que representam os polinômios estão ordenados de forma decrescente pelo valor do expoente. A função deve retornar o ponteiro para o início do polinômio resultante da soma dos dois polinômios passados como parâmetros. Faça as devidas definições de tipos previamente à declaração da função.