Algoritmos e Estruturas de Dados I
Lista de exercícios: listas, pilhas, etc
Prof. João B. Oliveira

Pilhas

1. Declare os tipos necessários para formar um lista encadeada onde cada nodo tem como informação um inteiro. Use esta estrutura para construir uma pilha com as operações abaixo (construa as operações nesta ordem e use as anteriores para testar a próxima operação):

   (a) push(int): insere um inteiro no topo da pilha;

   (b) pop(): retira e devolve o elemento que está no topo da pilha;

   (c) isEmpty(): retorna um booleano informando se a pilha está vazia ou não;

   (d) print(): imprime todos os elementos da pilha, começando pelo topo. A pilha não estará modificada depois da operação;

   (e) add(), sub(), mult(), div(): retiram dois operandos da pilha e colocam o resultado da operação de volta na pilha;

   (f) sin(), cos(), atan2(): retiram operandos da pilha e recolocam o resultado da operação. A função atan2() deve ser usada com dois operandos;

   (g) dup(): duplica o resultado no topo da pilha;

   (h) swap(): troca de ordem os dois elementos que estão no topo;

   (i) exist(int): devolve um booleano informando se o inteiro existe dentro da pilha. A pilha não pode estar modificada depois da operação e dentro do método exist() você só pode usar push(), pop() e isEmpty().

   (j) ordena(): devolve um booleano informando se a pilha tem seus elementos em ordem (crescente ou decrescente, você escolhe). A pilha não pode estar modificada depois da operação e dentro do método exist() você só pode usar push(), pop() e isEmpty().

2. Escreva um programa que recebe uma pilha P e um inteiro e retorna um booleano informando se o inteiro está contido na pilha. Para fazer isso você não pode usar outras pilhas para ajudar e deve deixar a pilha P exatamente do jeito que encontrou.

3. Escreva um programa que recebe uma pilha P e ordena a pilha, ou seja, mexe com os elementos para que eles fiquem em ordem crescente. Para fazer isso você não pode usar outras pilhas para ajudar e só pode usar push(), pop() e isEmpty() para alterar a pilha. Não mexa nela de nenhuma outra maneira.

4. Usando as pilhas definidas anteriormente, escreva um programa que recebe uma string S e determina se ela tem os caracteres (e) balanceados, desprezando quaisquer outros caracteres. Seu programa só pode avançar nos caracteres da string S e só pode processar cada caractere uma vez.

5. Usando pilhas, escreva um programa que recebe uma string S e determina se ela tem os caracteres (,), [,], {, }, $<$ e $>$ balanceados, desprezando quaisquer outros caracteres. Seu programa só pode avançar nos caracteres da string S e só pode processar cada caractere uma vez.

6. Usando pilhas, escreva um programa que recebe uma string S e determina se ela tem o formato $A^n B^n C^n$, ou seja, n letras A seguidas de n letras B e depois n letras C. Seu programa só pode avançar nos caracteres da string S e só pode processar cada caractere uma vez.

Listas

1. Declare os tipos necessários para formar um lista encadeada onde cada nodo tem como informação um inteiro. Escreva um método que recebe um valor *val* e um inteiro *pos* e coloque o valor *val* na posição *pos* da lista. Teste-o nas seguintes condições:

   (a) Inserir um elemento numa lista que está vazia. Funciona?

   (b) Inserir um elemento no primeiro lugar de uma lista já existente. Funciona?

   (c) Inserir um elemento no meio de uma lista. Funciona?

   (d) Inserir um elemento no último lugar de uma lista. Funciona?

   (e) Inserir um elemento que já está na lista. Funciona?

2. Escreva um método `void print()` que imprime todos os elementos da lista e use-o para verificar as operações anteriores.

3. Escreva um método `void insertSorted(int)` que coloca elementos na lista na posição correta para que eles fiquem sempre em ordem crescente.

4. Escreva um método `boolean exist(int)` que percorre uma lista e retorna um booleano informando se o inteiro está na lista.

5. Escreva um método `getpos(int)` que retira (e devolve) o elemento que está na posição fornecida.

6. Escreva um método `boolean del(int)` que percorre uma lista e deleta a primeira ocorrência do inteiro dado. O método retorna um booleano informando se houve deleção ou não.

7. Escreva um método `int min()` que percorre uma lista e retorna seu menor elemento.

8. Escreva um método `fill(int)` que cria uma lista contendo inteiros de 1 até o inteiro que foi dado.

9. Escreva um método `int getmin()` que percorre uma lista e retira seu menor elemento, retornando-o.

10. Escreva um método `boolean kill(int)` que percorre uma lista e deleta todas as ocorrências do inteiro dado.

11. Escreva um método `int total(int)` que percorre uma lista e conta o número de ocorrências do inteiro dado.

12. Escreva um método `unique(int)` que percorre uma lista e deixa apenas uma ocorrência do inteiro dado. Você não pode usar `total(int)` para ajudar nesta tarefa.

13. Escreva um método `unique()` que percorre uma lista e deixa apenas um de cada um dos elementos que estão na lista.

14. Escreva um método `ordena()` que altera a lista, deixando-a com os valores em ordem crescente.

15. Faça um método que imprime os valores de uma lista encadeada de trás para a frente ou seja, a partir do último elemento.

16. Depois de imprimir seus valores, a lista pode ser destruída. Faça um método que desaloque **todos** os elementos da lista, liberando a memória para outras aplicações.

17. Faça uma lista onde cada elemento contém um número inteiro e um nome. Escreva um método que recebe um número e procura na lista, imprimindo o nome da posição correspondente.

18. Altere o método anterior para que depois de procurar o nome ele também mova o nodo com este nome uma posição mais para a frente da lista. Assim, caso ele seja procurado mais vezes, ele deverá ser encontrado mais rapidamente.

19. Escreva um método que recebe duas listas e retorna um booleano informando se as listas são iguais. Ao final do método as duas listas não podem ter sido alteradas.

20. Crie duas listas encadeadas de inteiros, já ordenadas. Depois escreva um método que junta os nodos das duas listas em uma terceira lista também com os elementos em ordem. Por exemplo, se você tinha L1 = {3, 5, 8, 13, 19} e L2 = {1, 4, 6, 9, 15}, o resultado deve ser L3 = {1, 3, 4, 5, 6, 8, 9, 13, 15, 19}.

21. Crie duas listas de inteiros que não estão ordenadas. Depois escreva um método que crie uma terceira lista a partir destas duas, mas com os elementos em ordem. Por exemplo, se você tinha L1 = {13, 5, 3, 8, 19} e L2 = {9, 1, 4, 15, 6}, o resultado deve ser L3 = {1, 3, 4, 5, 6, 8, 9, 13, 15, 19}.

Comece declarando os tipos necessários, depois escreva um dos métodos e teste-o. Em seguida desenvolva o segundo método e faça seu teste, e depois faça o terceiro. Isto ajuda a isolar erros pois colocando-se um método de cada vez e testando pode-se saber exatamente onde deve estar um possível erro. Eventualmente, os dois primeiros métodos podem ser desenvolvidos ao mesmo tempo, pois a cada teste de inserção seria útil se imprimir os valores na lista, para ver se eles estão na ordem correta.