Lista de exercícios
Estruturas lineares
Prof. Diogo S. Martins

1. Adicione no TAD `LinkedList` uma função que recebe duas listas encadeadas (`LinkedList`) como argumento e imprima os elementos da primeira lista cujas posições estão armazenadas na segunda lista. A função deve ter a seguinte assinatura: `void list_printElements(LinkedList* l1, LinkedList* l2)`. Por exemplo, se a primeira lista contém os elementos `30, 40, 10, 80, 5` e a segunda lista contém os elementos `1, 3, 4`, a função deve imprimir os elementos `40, 80, 5`.

2. Adicione no TAD `LinkedList` uma função que recebe como argumento duas listas encadeadas e retorna true se as listas são iguais e false caso contrário. A função deve ter a seguinte assinatura: `bool list_equals(LinkedList* l1, LinkedList* l2)`. Por exemplo, se a primeira lista contém os elementos `1, 2, 3, 4, 5` e a segunda lista contém os elementos `1, 2, 3, 4, 5`, a função deve retornar `true`. Se a primeira lista contém os elementos `1, 2, 3, 4, 5` e a segunda lista contém os elementos `1, 2, 3, 4, 6`, a função deve retornar `false`.

3. Adicione no TAD `LinkedList` uma função que recebe como argumento uma lista encadeada e retorne como resultado o código 1 caso a lista esteja ordenada em ordem crescente, 2 caso esteja ordenada em ordem descrescente e 0 caso não esteja ordenada. A função deve ter a seguinte assinatura: `int list_isSorted(LinkedList* l)`. Por exemplo, se a lista encadeada contém os elementos na ordem `1, 2, 3, 4, 5`, a função deve retornar 1. Se a lista contém os elementos na ordem `5, 4, 3, 2, 1`, a função deve retornar 2. Se a lista contém os elementos na ordem `1, 3, 2, 4, 5`, a função deve retornar 0.

4. Adicione ao TAD `LinkedList` uma função que remova todos os elementos duplicados de uma lista encadeada. A função deve ter a seguinte assinatura: `void list_removeDuplicates(LinkedList* l)`. Por exemplo, se a lista encadeada contém os elementos `1, 2, 3, 2, 4, 5, 3`, a lista resultante deve conter os elementos `1, 2, 3, 4, 5`.

5. Adicione no TAD `LinkedList` uma função que recebe uma lista encadeada como argumento e imprima os elementos da lista na ordem inversa. A função deve ter a seguinte assinatura: `void list_printReverse(LinkedList* l)`. Por exemplo, se a lista encadeada contém os elementos `1, 2, 3, 4, 5`, a função deve imprimir os elementos na ordem `5, 4, 3, 2, 1`. Não é permitido permitido alocar dinamicamente memória adicional para resolver o problema.

6. Escreva um código-cliente que contenha uma função que recebe como argumento uma lista encadeada (`LinkedList`) e retorne como resultado outra lista encadeada contendo os mesmos elementos na ordem reversa. A função deve ter a seguinte assinatura: `LinkedList* reverse(LinkedList* l)`. Por exemplo, se a lista encadeada contém os elementos na ordem `1, 2, 3, 4, 5`, a lista retornada deve conter os elementos na ordem `5, 4, 3, 2, 1`. Não é permitido violar o encapsulamento da lista recebida como argumento, todas as manipulações devem ocorrer via operações da interface da lista. Além disso, terminada a função, a lista recebida como argumento deve ter os mesmos elementos que tinha inicialmente, na mesma ordem.

7. Escreva uma função que recebe uma pilha (`Stack`) como argumento e retorne uma cópia da pilha. A função deve ter a seguinte assinatura: `Stack* copy(Stack* s)`. Por exemplo, se a pilha contém os elementos `top -> 1, 2, 3, 4, 5`, a pilha retornada deve conter os mesmos elementos. A função deve estar em um código-cliente da pilha. Obviamente, não é permitido violar o encapsulamento da pilha recebida como argumento, todas as manipulações devem ocorrer via operações da interface da pilha. Além disso, terminada a função, a pilha recebida como argumento deve ter os mesmos elementos que tinha inicialmente, na mesma ordem.

8. Construa uma função que receba como argumento uma pilha (`Stack`) e retorne como resultado uma fila (`Queue`) com os elementos da pilha na ordem inversa. A função deve ter a seguinte assinatura: `Queue* reverse(Stack* s)`. Por exemplo, se na pilha os elementos estão na ordem `top-> 1, 2, 3, 4, 5`, na fila os elementos devem estar na ordem `front -> 5, 4, 3, 2, 1`. A função deve estar em um código-cliente. Não é permitido violar o encapsulamento nem da pilha nem da fila. Todas as manipulações devem ocorrer via operações das respectivas interfaces. Além disso, terminada a função, a pilha recebida como argumento deve ter os mesmos elementos que tinha inicialmente, na mesma ordem.

9. Escreva uma função que receba como argumento duas filas (`Queue`) e retorne como resultado `true`, caso as filas sejam iguais, e `false` caso contrário. As filas são iguais se contém os mesmos elementos, na mesma ordem. Não é permitido violar o encapsulamento das filas recebidas como argumento, todas as manipulações devem ocorrer via operações da interface da fila. Além disso, terminada a função, as filas recebidas como argumento devem ter os mesmos elementos que tinham inicialmente. A função deve ter a seguinte assinatura: `bool equals(Queue* q1, Queue* q2)`.

10. Escreva um programa que recebe como argumento uma string contendo uma expressão aritmética, contendo parênteses, colchetes e chaves, e verifica se a expressão está corretamente balanceada. A função deve ter a seguinte assinatura: `bool isBalanced(char* expr)`. Por exemplo, a expressão `[2 * (3 + 4)]` está corretamente balanceada, enquanto a expressão `[2 * (3 + 4)` não está.