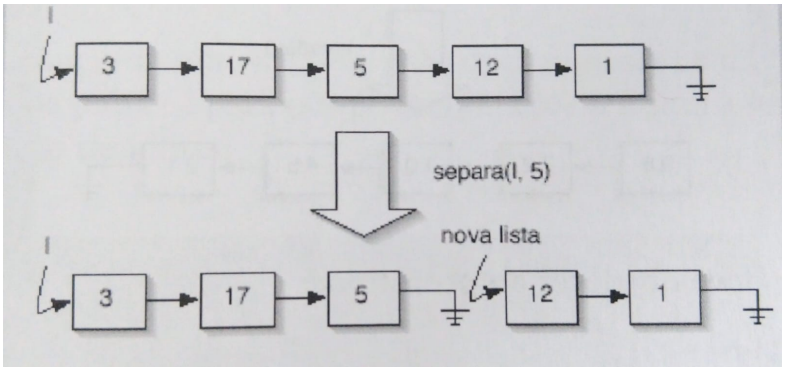
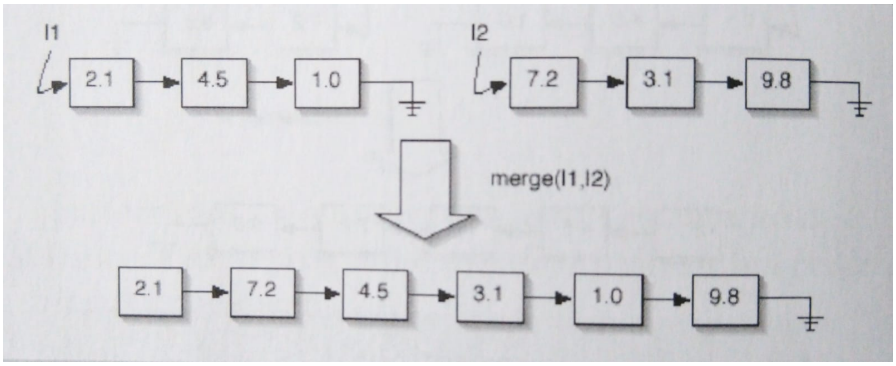
Lista 4
Listas encadeadas e Pilhas

1. Considere listas de valores inteiros e implemente uma função que receba como parâmetro uma lista encadeada e um valor inteiro n e divida a lista em duas, de forma à segunda lista começar no primeiro nó logo após a primeira ocorrência de n na lista original. A figura a seguir ilustra essa separação:



   A função deve retornar um ponteiro para a segunda subdivisão da lista original, enquanto `l` deve continuar apontando para o primeiro elemento da primeira subdivisão da lista.
   Essa função deve obedecer ao protótipo:
   `Lista* separa(Lista* l, int n);`

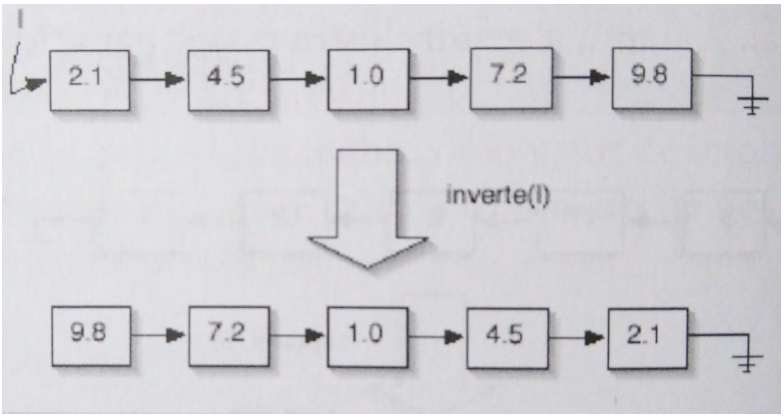
2. Implemente uma função que construa uma nova lista com a intercalação de nós de outras duas listas passadas como parâmetros. Essa função deve retornar a lista resultante, conforme ilustrado a seguir:



   Essa função deve obedecer ao protótipo:
   `Lista* merge(Lista* l1, Lista* l2);`

3. Implemente uma função que receba como parâmetro uma lista encadeada e inverta o encadeamento de seus nós, retornando a lista resultante. Após a execução dessa

função, cada nó da lista vai estar apontando para o nó que originalmente era seu antecessor, e o último nó da lista passará a ser o primeiro nós da lista invertida, conforma ilustrado a seguir:



Essa função deve obedecer ao protótipo:
`Lista* inverte(Lista* l);`

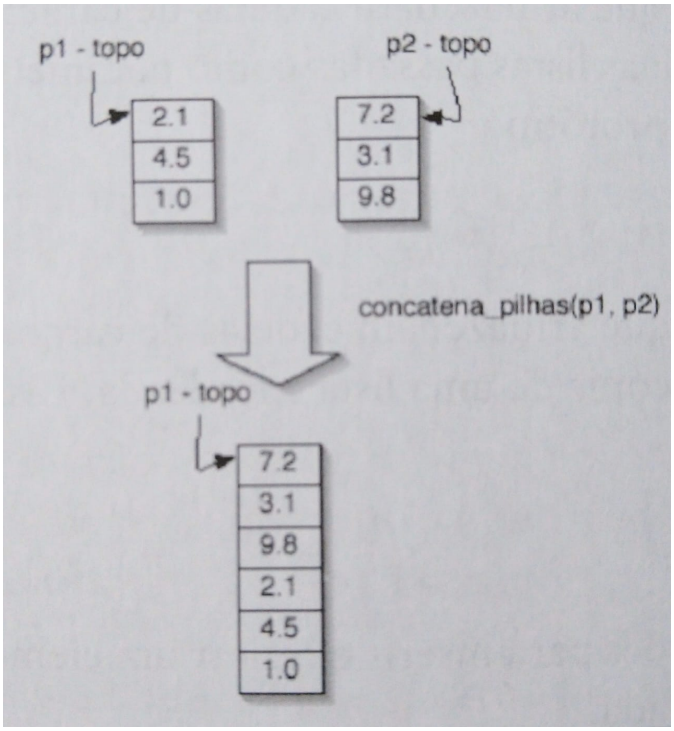
4. Faça um TAD de pilha com números reais com as funções básicas apresentadas na aula. Acrescente as funções a seguir no TAD `Pilha`.

5. Implemente uma função que receba uma pilha como parâmetro e retorne o valor armazenado em seu topo, restaurando o conteúdo da pilha. Essa função deverá obedecer ao protótipo:
   `float topo(Pilha *p);`

6. Implemente uma função que receba uma pilha como parâmetro e retorne como resultado uma cópia dessa pilha. Essa função deve obedecer ao protótipo:
   `Pilha* copiaPilha(Pilha *p);`
   Ao final da função, a pilha *p* recebida como parâmetro deve ter seu conteúdo original.
   Faça também numa versão recursiva desta função.

7. Implemente uma função que receba duas pilhas, *p1* e *p2*, e passe todo os elementos da pilha *p2* para o topo da pilha *p1*. A figura a seguir ilustra essa concatenação de pilhas:

   Ao final dessa função, a pilha *p2* vai estar vazia, e a pilha *p1* conterá todos os elementos das duas pilhas. Essa função deve obedecer ao protótipo:
   `void concatenaPilha(Pilha *p1, Pilha *p2);`
   Faça também numa versão recursiva desta função.


p1 - topo
2.1
4.5
1.0
p2 - topo
7.2
3.1
9.8
concatena_pilhas(p1, p2)
p1 - topo
7.2
3.1
9.8
2.1
4.5
1.0