

## Lista Ligada de Produtos

O objetivo deste EP é gerenciar uma Lista Ligada de Produtos de uma loja que deseja acessar seus produtos pelo seu valor total (de forma crescente, isto é, do menor para o maior valor total, sendo o valor total igual ao valor unitário vezes a quantidade do respectivo produto).

Para isto, você deverá gerenciar uma estrutura que contém uma lista ligada de produtos não circular e com nó cabeça. Essa lista conterá todos os produtos. Na representação em memória de cada produto haverá um ponteiro para o próximo produto da lista (cujo valor total será maior ou igual ao do produto atual).

Dentre as operações previstas para esta lista estão (em **negrito** estão destacadas as funções que deverão ser implementadas por você neste EP):

- criação de uma lista de produtos;
- busca por um produto;
- consulta à quantidade de produtos diferentes na lista (tamanho);
- consulta pelo valor unitário de um produto.
- **inserção de um novo produto;**
- **remoção de itens de um produto;**
- **alteração do valor unitário de um produto;**

Para este EP, você deverá implementar um conjunto de funções de gerenciamento de listas utilizando principalmente o conceito de **listas ligadas ordenadas não circulares e com nó cabeça**.

A seguir são apresentadas as estruturas de dados envolvidas nesta implementação e como elas serão gerenciadas.

A estrutura básica será o *REGISTRO*, que contém cinco campos: *id* (identificador inteiro do produto), *tipo* (número inteiro com o tipo do produto), *quantidade* (número inteiro com a quantidade do produto), *valorUnitario* (número inteiro com o valor unitário do produto em centavos de reais), *proxProd* (ponteiro para o endereço do registro posterior ao atual, isto é, aquele que possui valor total maior ou igual ao do produto atual).

```
typedef struct aux {  
    int id;  
    int tipo;  
    int quantidade;  
    int valorUnitario;  
    struct aux* proxProd;  
} REGISTRO, * PONT;
```

REGISTRO	
id	tipo
quantidade	valorUnitario
proxProd	

A estrutura *LISTADEPRODUTOS* possui apenas um campo: *cabeca* é um ponteiro para elementos do tipo *REGISTRO* e corresponde ao ponteiro para o nó-cabeça da lista (este será criado na inicialização da lista e nunca deverá ser apagado). Esta lista de elementos será ordenada e não será circular.

```
typedef struct {
    PONT cabeca;
} LISTADEPRODUTOS, * PLISTA;
```

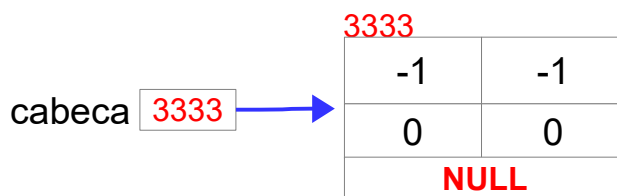
## LISTA DE PRODUTOS

*cabeca*

A função *criarLista* é responsável por criar uma nova lista, incluindo a criação do nó-cabeça cujo endereço será armazenado no campo *cabeca* da lista.

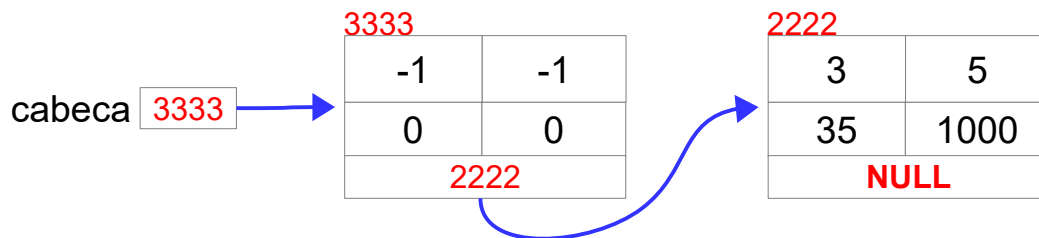
```
PLISTA criarLista(){
    PLISTA res = (PLISTA) malloc(sizeof(LISTADEPRODUTOS));
    res->cabeca = (PONT) malloc(sizeof(REGISTRO));
    res->cabeca->id=-1;
    res->cabeca->tipo=-1;
    res->cabeca->quantidade=0;
    res->cabeca->valorUnitario=0;
    res->cabeca->proxProd=NULL;
    return res;
}
```

Exemplo de lista recém-criada:

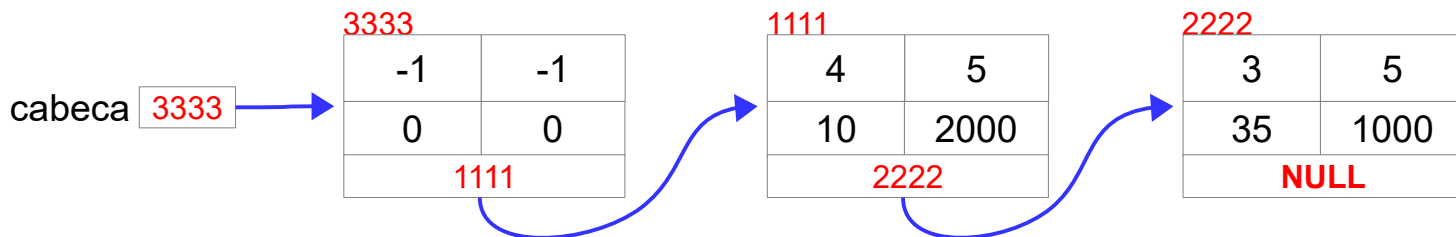


Ao se inserir um primeiro produto na estrutura, este deverá ser incluído como próximo elemento do nó-cabeça (utilizando o ponteiro *proxProd*).

Exemplo de fila após a inserção do elemento de id=3, tipo = 5, quantidade=35 e valor=1000



Exemplo da mesma lista após a inserção do elemento de id=4, tipo=5, quantidade=10 e valor=2000. Observe que o novo registro é inserido após o nó-cabeça e antes do registro cujo id é igual a 3, pois seu valor total é igual a 20.000, o que é inferior ao valor total do produto cujo id é igual a 3 (35.000).



## Funções que deverão ser implementadas no EP

*bool inserirNovoProduto(PLISTA l, int id, int tipo, int quantidade, int valor):* função que recebe o endereço de uma lista de produtos, o identificador do novo produto, seu tipo, sua quantidade e seu valor unitário e retorna um valor booleano.

Esta função deverá retornar *false* caso já exista um registro com esse identificador na lista ou caso ao menos um dos parâmetros de entrada não seja positivo (ou seja, só será possível inserir um novo produto caso não exista um produto com o mesmo id na lista e caso seu id, tipo, quantidade e valor unitário sejam maiores do que zero).

Caso seja possível inserir, a função deverá alocar memória de um novo registro para inserir os dados desse produto, preencher os campos referentes ao produto e inseri-lo na posição correta da lista, isto é, de acordo com o valor total do produto (que é dado pela multiplicação do seu valor unitário pela quantidade), acertar todos os ponteiros necessários e retornar *true*. Se já houver outros produtos na lista ordenada pelo valor total com o mesmo valor total do produto que será inserido, então este deverá ser inserido antes de todos os outros produtos que possuem valor total igual ao do novo produto.

*bool removerItensDeUmProduto(PLISTA l, int id, int quantidade):* função que recebe o endereço de uma lista de produtos, o identificador de um produto e uma quantidade e retorna um valor booleano.

Esta função deverá retornar *false* caso não haja na lista um produto com o identificador passado como parâmetro, caso a quantidade passada como parâmetro seja menor ou igual a zero ou se a quantidade passada como parâmetro for maior do que a quantidade do respectivo produto na lista.

Caso contrário, a quantidade do respectivo produto deve ser atualizada (seu valor atual deve ser subtraído pela quantidade passada como parâmetro), a lista ordenada deve ser reorganizada, caso seja necessário, e a função deverá retornar *true*. Observe que, se a quantidade total do produto, após a atualização, ficar igual a zero este produto deve ser excluído da lista e sua memória deve ser liberada, caso contrário, se o valor total do produto atual se tornar estritamente menor do que o valor total do elemento anterior, ele deverá mudar de posição na lista ligada ordenada (de forma que a lista continue ordenada).\*

*bool atualizarValorDoProduto(PLISTA l, int id, int valor):* esta função recebe como parâmetro o endereço de uma lista de produtos, o identificador de um produto e seu valor e deverá retornar *false* caso não existe um produto com o identificador passado como parâmetro ou se o valor passado como parâmetro for menor ou igual a zero. Caso contrário, deverá alterar o valor unitário do respectivo produto (atualizando o valor desse campo para o valor recebido como parâmetro), reorganizar a lista ordenada, caso seja necessário, e retornar *true*. Observe que, caso o valor total do produto atual se torne estritamente menor do que o valor total do produto anterior ou estritamente maior do que o valor total do próximo produto, ele deverá mudar de posição na lista ligada ordenada (de forma que a lista continue ordenada).\*

**\* Nos casos em que for necessário reposicionar o produto na lista ligada ordenada, se já houver outros produtos na lista com o mesmo valor total do produto atual, então este deverá ser posicionado antes de todos os outros produtos que possuem valor total igual ao do novo produto (isto deve ser feito apenas se for necessário reposicioná-lo de acordo com as regras presentes na remoção de itens e atualização de valor).**

## Informações gerais:

Os EPs desta disciplina são trabalhos individuais que devem ser submetidos pelos alunos via sistema TIDIA (ae4.tidia-ae.usp.br/) até às 23:55h (com margem de tolerância de 60 minutos).

Você receberá três arquivos para este EP:

- `listadeprodutos.h` que contém a definição das estruturas, os *includes* necessários e o cabeçalho/assinatura das funções. Você não deverá alterar esse arquivo.
- `listadeprodutos.c` que conterá a implementação das funções solicitadas (e funções adicionais, caso julgue necessário). Este arquivo já contém o esqueleto geral das funções e alguns códigos implementados.
- `usaListaDeProdutos.c` que contém alguns testes executados sobre as funções implementadas.

Você deverá submeter **apenas** o arquivo `listadeprodutos.c`, porém renomeie este arquivo para `seu_número_USP.c` (por exemplo, `3140792.c`) antes de submeter.

Não altere a assinatura de nenhuma das funções e não altere as funções originalmente implementadas (*exibirLog* e *criarLista*, etc).

Nenhuma das funções que você implementar deverá imprimir algo. Para *debugar* o programa você pode imprimir coisas, porém, na versão a ser entregue ao professor, suas funções não deverão imprimir nada (exceto pela função *exibirLog* que já imprime algumas informações).

Você poderá criar novas funções (auxiliares), mas não deve alterar o arquivo `listadeprodutos.h`. Adicionalmente, saiba que seu código será testado com uma versão diferente do arquivo `usaListaDeProdutos.c`. Suas funções serão testadas individualmente e em conjunto.

Todos os trabalhos passarão por um processo de verificação de plágios. **Em caso de plágio, todos os alunos envolvidos receberão nota zero.**