**Listas encadeadas**

Em uma lista encadeada , para cada novo elemento inserido na estrutura, alocamos um espaço de memória para armazená-lo. Dessa forma, o espaço total ocupado na memória é proporcional ao número de elementos da lista. No entanto, não podemos garantir que os elementos armazenados na memória ocuparão um espaço contíguo, e por isso não temos acesso aos elementos da lista diretamente.

Para percorrer e ter acesso aos elementos da lista devemos guardar o seu encadeamento. Sendo assim a estrutura consiste em uma sequência encadeada de elementos, chamados de nós da lista. Cada nó da lista tem a sua informação, podendo ser um valor inteiro, real, caractere ou até uma sequência de caracteres, e um ponteiro para o próximo elemento da lista.

A imagem a seguir representa este conceito, onde do primeiro elemento, temos acesso ao segundo, e assim por diante. Sendo que o último elemento armazenado um ponteiro inválido, com valor NULL, indicando o fim da lista.

https://www.ufsm.br/app/uploads/sites/791/2020/09/listapilhafila1.png

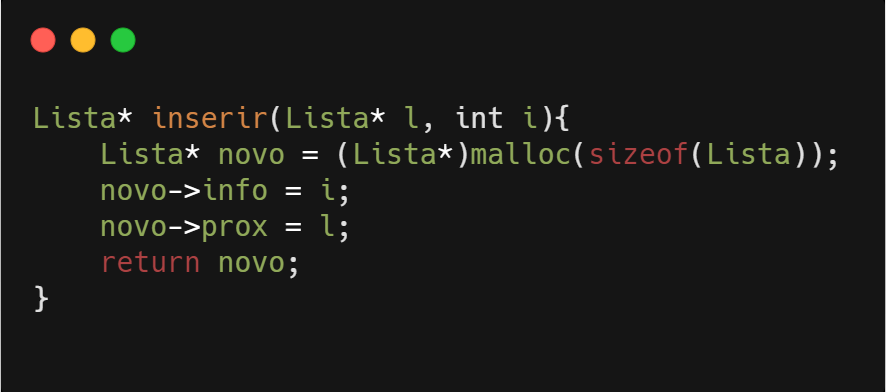
O código abaixo exemplifica a implementação de uma lista encadeada na linguagem C para armazenar valores inteiros.



Devemos notar que essa estrutura é auto-referenciada, pois o campo “***struct lista\* prox;****”* é um ponteiro para para uma estrutura do mesmo tipo. Além disso, uma boa estratégia e definir o tipo “***Lista***“ como sinônimo de ”***struct lista***“ conforme demonstra o trecho de código acima. Com isso, o tipo “***Lista***“ representa um nó da lista, e a estrutura de lista encadeada será representada por um ponteiro para o primeiro elemento da lista (tipo ***Lista\****). Sabendo a definição de lista , agora podemos começar a implementar funções necessárias para manipular essa estrutura.

**Função de inserir elementos**

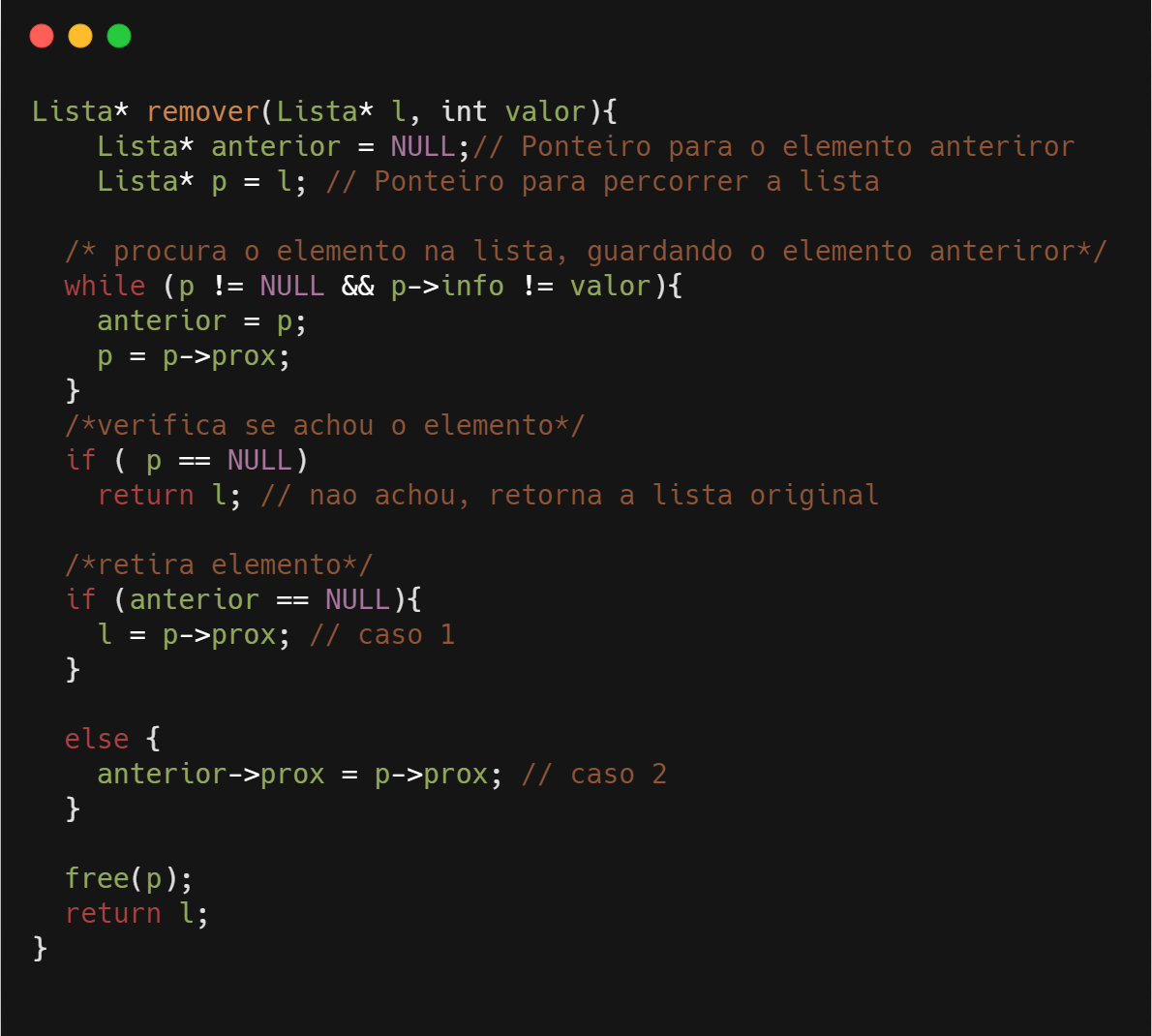
Para cada novo elemento inserido na lista devemos alocar dinamicamente a memória para armazená-lo, e posteriormente fazer o encadeamento na lista. Uma possível implementação dessa função é exemplificada a seguir. Deve-se observar que a inserção ocorre sempre no início da lista neste exemplo. A função tem como parâmetros um tipo *Lista\**, que representa um ponteiro para o primeiro elemento de uma lista e o valor da informação que queremos armazenar.



Note que após alocar a memória para o novo elemento, devemos atualizar o ponteiro que representa a lista. Nesse caso, o novo nó aponta (ou seja,tem como próximo elemento) para o elemento que era o primeiro da lista. E então a função retorna a nova lista, representada pelo ponteiro para o novo primeiro elemento.

**Função para remover um elemento**

Se o elemento que queremos remover for o primeiro, devemos fazer o novo valor da lista passar a ser o ponteiro para o segundo elemento, e então liberamos o espaço alocado para o elemento que iremos retirar. Se o elemento a ser removido estiver no meio da lista, devemos fazer com que o elemento anterior passe a apontar para o seguinte a ele, e então poderemos liberar o espaço de memória (***free(p)***).

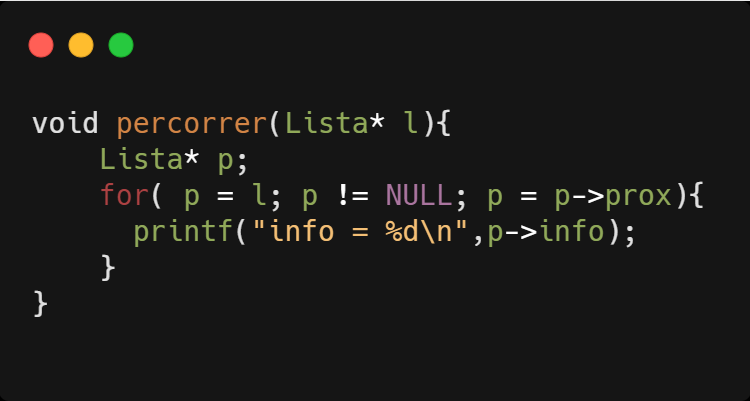


A função percorre a lista em busca do elemento com o valor enviado como parâmetro, caso o ponteiro ***p***tenha como valor final***NULL***, significa que toda a lista foi percorrida e não foi encontrado o elemento. Caso ele seja diferente de ***NULL***, o elemento foi encontrado e temos dois possíveis casos já citados anteriormente:

* Caso 1 – o ponteiro para o elemento anterior na lista é igual a ***NULL***, significando que o elemento encontrado é o primeiro da lista, então fazemos com que o ponteiro que representa a lista passe a ser o próximo (ou seja, o segundo).
* Caso 2 – o elemento está no meio da lista, então fazemos com que o ponteiro do elemento anterior passe a apontar para seguinte em relação ao que queremos remover.

**Função para imprimir uma lista**

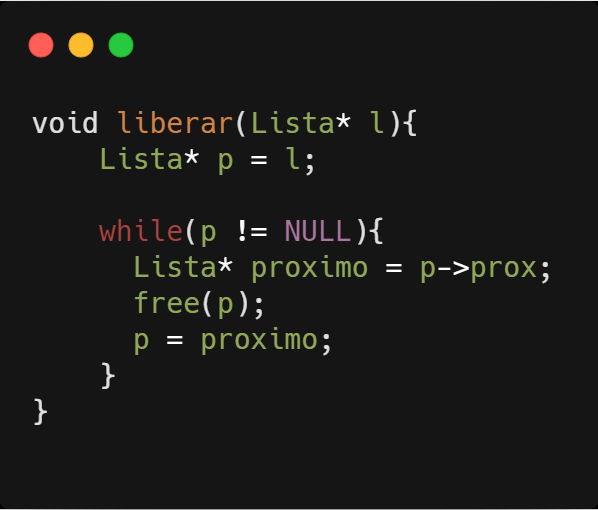
Para percorrer uma lista utilizamos uma variável auxiliar do tipo ponteiro, para armazenar o endereço de cada elemento. A imagem abaixo mostra uma função que percorre todos os elementos e imprime o seu valor.



Com estas funções básicas de manipulação de listas demonstradas acima podemos utilizar seus conceitos e melhorá-las,e assim criar funções de busca, inserções ordenadas, entre outras possibilidades.

**Função para liberar a lista**

Completando o conjunto de funções básicas de manipulação de listas, temos a função para destruir todos os elementos da lista e liberar a memória ocupada.



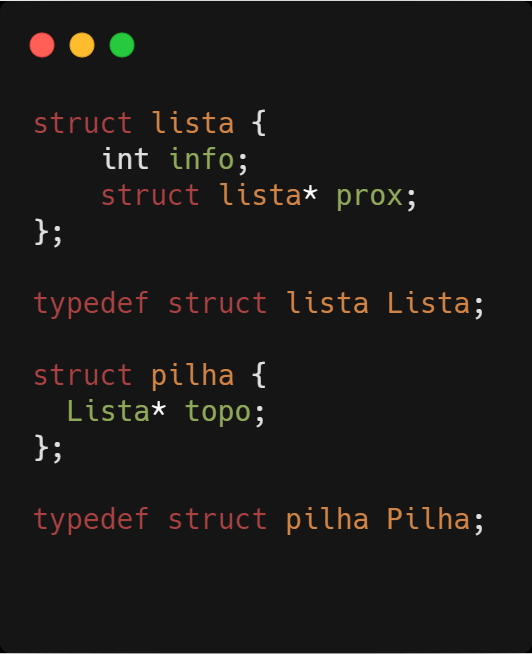
É importante notar que sempre guardamos a referência para o próximo elemento da lista antes de liberar o atual, pois perderíamos o encadeamento da lista e não teríamos mais acesso aos elementos, e acabaríamos tentando acessar um espaço de memória que não está reservado mais ao nosso uso.

**Pilhas**

Uma das estrutura de dados mais simples é a pilha. Sua ideia fundamental é que todo acesso a seus elementos seja feito a partir do topo. Assim quando um elemento é inserido na pilha, ele passa a ser o elemento do topo e só temos acesso a ele. Logo, os elementos da pilha só podem ser retirados na ordem inversa a ordem que foram inseridos : o primeiro que sai e o último que entrou (LIFO – Last in ,first out). Uma pilha pode ser implementada utilizando um vetor, se tivermos a informação do número máximo de elementos que iremos armazenar, ou utilizando uma lista encadeada quando não sabemos quantos elementos iremos armazenar.

**Implementação de pilha com lista**

Nesse caso os elementos são armazenados em uma lista encadeada, e a pilha pode ser representada simplesmente por um ponteiro para o primeiro nó da lista.



Para criar uma pilha, é necessário primeiro alocar a memória referente a estrutura da pilha e depois inicializamos a lista como vazia.

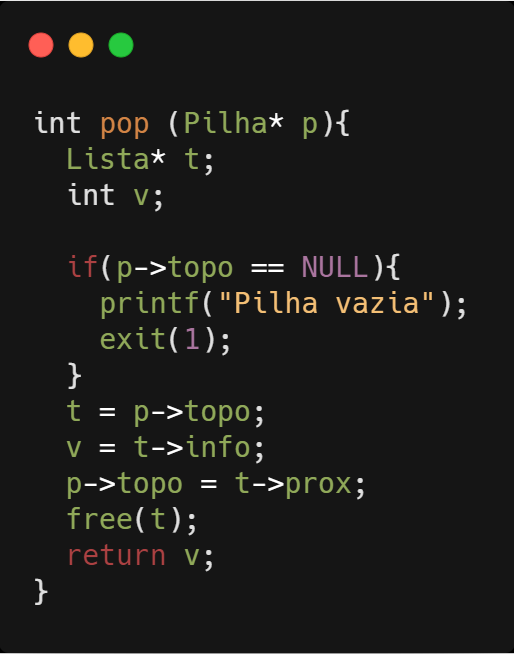


Ao manipular uma pilha temos duas operações básicas que devem ser implementadas, a de empilhar um novo elemento no topo (push) e a de desempilhar um elemento (pop). Sabendo que o primeiro elemento da lista representa o topo da pilha. A cada nova inserção esse elemento vai para o início da lista, e consequentemente, o elemento durante a operação de remoção também é o primeiro da lista. Segue abaixo exemplo das funções de push e pop.

***Função push***



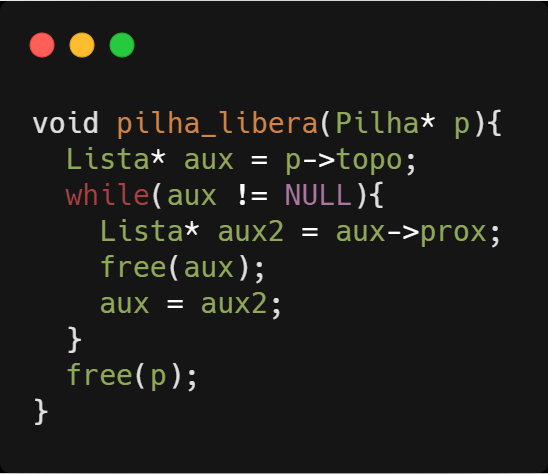
***Função pop***



Note que a função retorna o valor do elemento que foi retirado.

**Função para liberar pilha**

Para liberar a memória alocada para uma estrutura do tipo pilha, temos que liberar primeiro todos os elementos da lista.



**Filas**

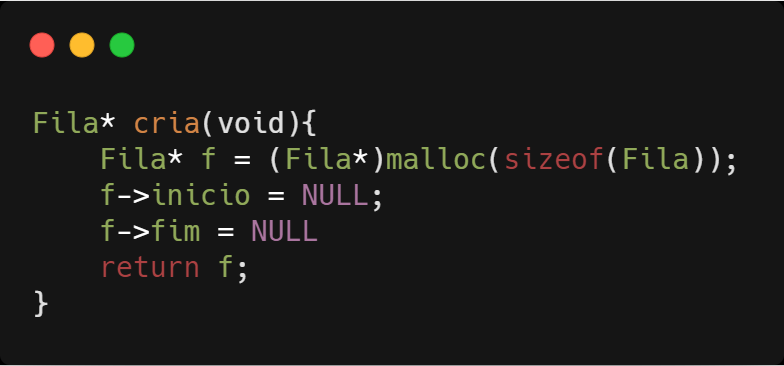
Na estrutura de fila, o acesso aos elementos também segue regras. Sua ideia fundamental é que só podemos inserir elementos no final da fila e retirar do início da fila (FIFO – First in, first out). Assim como a pilha, a fila pode ser implementada com vetores ou com listas encadeadas, dependendo apenas se soubermos a quantidade máxima de elementos que cabem nesta fila.

**Implementação de fila com lista**

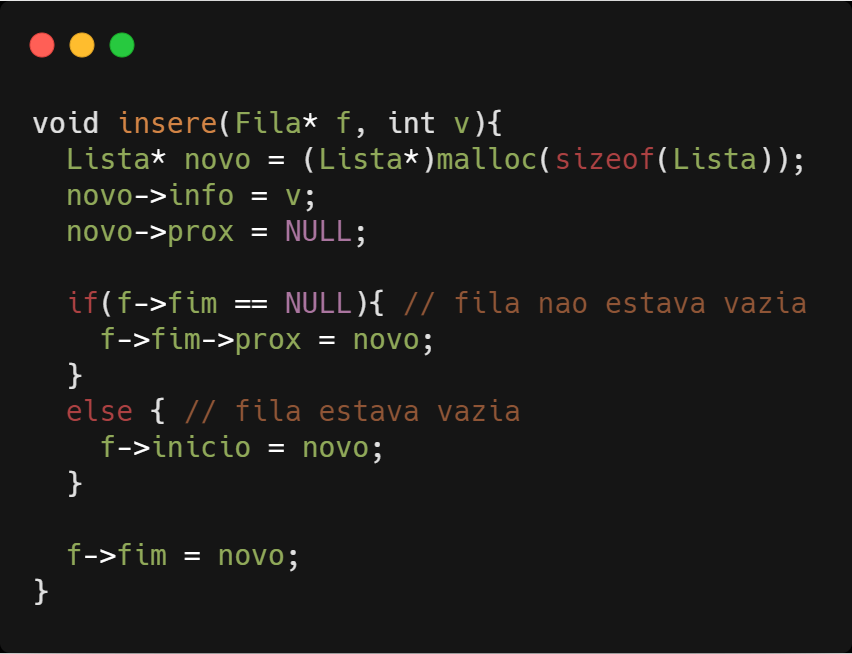
Assim como na pilha, os elementos são armazenados em uma lista encadeada, no entanto teremos que inserir e retirar os elementos das extremidades da lista. A fila será representada por dois ponteiros, um aponta para o início e outro para o fim da fila.



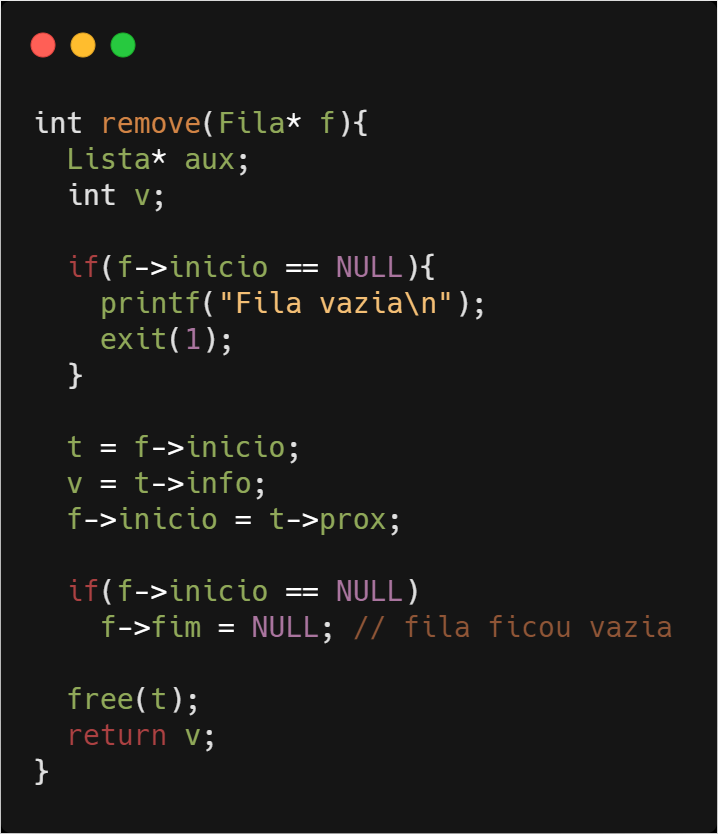
Inicialmente criamos a fila, alocando a memória para a estrutura fila e inicializamos a lista como vazia.



Tendo criado a fila, agora podemos inserir elementos. Lembrando que as inserções ocorrem sempre no fim da fila. Vamos a um exemplo de implementação desse procedimento.



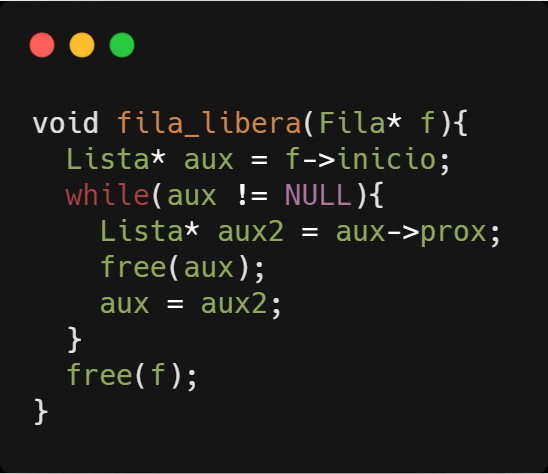
As remoções ocorrem sempre no início da fila, então devemos fazer com que o início da fila passe a ser o elemento seguinte.



Devemos ressaltar, que o primeiro **“if”**testa se a fila está vazia, entretanto no segundo **“if“**o ponteiro para o início recebe o próximo elemento da lista, e então caso ele seja **NULL**, significa que a fila ficou vazia após a remoção. Em seguida, atualizamos o ponteiro para o final da lista como vazio.

**Função para liberar fila**

As funções para liberar a memória alocada para ambas as estruturas pilhas e filas, são semelhantes, nos dois casos temos que liberar primeiro todos os elementos da lista.



Devo ressaltar que existem variações dessas estruturas, como as listas circulares, listas duplamente encadeada, filas circulares e filas duplamente encadeadas. Além das versões de pilha e fila implementadas com vetores.

Caso tenha dúvidas a respeito de structs, ponteiros e alocação dinâmica:

[https://www.ime.usp.br/~pf/algoritmos/aulas/stru.html](https://www.google.com/url?q=https%3A%2F%2Fwww.ime.usp.br%2F~pf%2Falgoritmos%2Faulas%2Fstru.html&sa=D&sntz=1&usg=AFQjCNEXYPNleJj4YaThR3gy5UR0BZB5fA)

[https://www.ime.usp.br/~pf/algoritmos/aulas/pont.html](https://www.google.com/url?q=https%3A%2F%2Fwww.ime.usp.br%2F~pf%2Falgoritmos%2Faulas%2Fpont.html&sa=D&sntz=1&usg=AFQjCNFxl78gEppyeGzchxa56V79RgPKuA)

[https://www.ime.usp.br/~pf/algoritmos/aulas/aloca.html](https://www.google.com/url?q=https%3A%2F%2Fwww.ime.usp.br%2F~pf%2Falgoritmos%2Faulas%2Faloca.html&sa=D&sntz=1&usg=AFQjCNHELm5_YSyc9u3x9U3VfPktexoDeA)

Referências :

[https://www.ime.usp.br/~pf/algoritmos/aulas/lista.html](https://www.google.com/url?q=https%3A%2F%2Fwww.ime.usp.br%2F~pf%2Falgoritmos%2Faulas%2Flista.html&sa=D&sntz=1&usg=AFQjCNEfo6YPKvqBpicLf8-V_sNTldWkpg)

[https://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html](https://www.google.com/url?q=https%3A%2F%2Fwww.ime.usp.br%2F~pf%2Falgoritmos%2Faulas%2Fpilha.html&sa=D&sntz=1&usg=AFQjCNEscCeCP7C3p8M8_NWx-x9q3VXSTA)

[https://www.ime.usp.br/~pf/algoritmos/aulas/fila.html](https://www.google.com/url?q=https%3A%2F%2Fwww.ime.usp.br%2F~pf%2Falgoritmos%2Faulas%2Ffila.html&sa=D&sntz=1&usg=AFQjCNH8OtHfR-iiod3RzAtMEos2rNNA9A)

CELES, W.; CERQUEIRA, R.; RANGEL, J. L. Introdução à Estrutura de Dados. São Paulo: Campus/Elsevier, 2004.