# Ponteiros

Em programação, um ponteiro ou apontador é um tipo de dado de uma linguagem de programação cujo valor se refere diretamente a um outro valor alocado em outra área da memória, através de seu endereço.

# Listas Encadeadas

**São:** Estruturas de dados implementadas por meio de ponteiros. Podemos compará-las com vetores, que são conjuntos de variáveis, mas diferente destes, as listas são conjuntos de nós (dado + ponteiros).

**Devem:** Armazenar elementos já ordenados.

**Vantagens:** Quando não há uma previsão do quanto de espaço iremos precisar.

**Desvantagens:** Diferente de um vetor, nós não temos acesso direto aos elementos da lista, para percorrer os elementos da lista devemos guardar seu "encadeamento", ou seja, seus ponteiros.

**Funcionamento:** Existem nós, os quais contêm: o conteúdo de cada elemento e um ponteiro que aponta para o próximo nó, cada nó é representado por uma struct. O último elemento da lista aponta para o elemento null, pois não há mais elementos.


**Remoção no início:** O ponteiro que inicialmente apontava para o primeiro nó, deve passar a apontar para o segundo nó, e depois, devemos remover o espaço alocado no lugar do elemento removido.

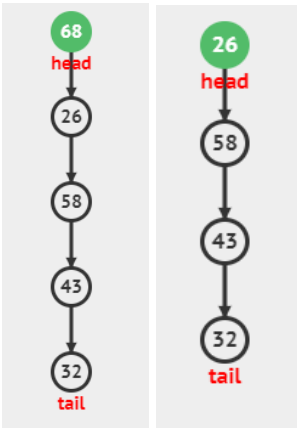
**Remoção no meio:** O ponteiro do nó anterior ao nó removido deve passar a apontar para o nó posterior ao nó removido.

# Pilhas

São: Estruturas de dados semelhantes a uma pilha de livros, nas quais a inserção de novos elementos sempre ocorre no topo. Podem ser representadas por vetores ou por listas encadeadas.

Funcionamento: Devemos imaginar uma pilha de objetos, o primeiro elemento a entrar será o último a sair. De forma semelhante, podemos imaginar que o último elemento a entrar será o primeiro a sair. Utilizamos duas expressões para descrever seu funcionamento, *push* para empilhar e *pop* para desempilhar.

**Push** **Pop**




## Pilhas Estáticas

Quando sabemos a quantidade de dados que iremos usar, o ideal é usar uma pilha estática, através do uso de vetores.

## Pilhas Dinâmicas

Quando não há previsão do quanto de memória iremos usar, utilizamos uma pilha dinâmica, através do uso de uma lista encadeada.

# Filas

Filas são estruturas de dados que funcionam de maneira análoga a filas de pessoas: o primeiro elemento a entrar é o primeiro elemento a sair. Em computação, uma aplicação muito útil para filas é no funcionamento de impressoras.

**Funcionamento:** A implementação deve ser como a de uma lista encadeada, porém devemos ter dois ponteiros, um apontando para o início da fila e outro apontando para o final.

# Árvores

São as estruturas de dados mais adequadas quando se trata da organização de dados hierarquizados. A forma mais natural de manipulá-las é através de funções recursivas, as quais são funções que chamam a si mesmas. Um exemplo computacional que utiliza a estrutura árvore é o diretório de arquivos de um sistema operacional.

**Funcionamento:** As árvores são compostas por nós, os quais se dividem entre pais e filhos, os nós que não possuem filhos são chamados de nó-folha. Em uma árvore binária, cada nó tem no máximo dois filhos. Em uma árvore binária de busca, os filhos que estão à direita de cada sub-árvore são maiores que o nó pai, e os que estão à direita são menores. A implementação de uma árvore é representada por um ponteiro que aponta para o nó raiz. Além disso, ao liberarmos a memória alocada pela árvore, é necessário que liberamos as subárvores antes de liberarmos o nó raíz, para que o acesso a elas não seja perdido.

# Algoritmos de Busca

## Busca Linear

Consiste em um algoritmo de busca que percorre toda a estrutura. Devido a isso, é o algoritmo mais simples e intuitivo, porém é o algoritmo mais “pesado”, com complexidade de O(n). Seu desempenho pode ser melhorado caso a estrutura seja ordenada, porém a sua complexidade ainda será a mesma.

## Busca Binária

*A estrutura precisa estar ordenada*.

Consiste em um algoritmo de busca no qual a estrutura é “dividida” em duas partes. O valor procurado é comparado ao valor do meio, caso seja menor, a segunda parte da estrutura é “descartada” e busca passa para a primeira parte. Caso seja maior, ocorre o contrário. O algoritmo vai dividindo a estrutura até encontrar o valor buscado. É um algoritmo de maior eficiência que o algoritmo de busca linear, tendo complexidade de log(n). Devemos lembrar também que a implementação iterativa é mais eficiente que a implementação recursiva, apesar de ser menos intuitiva.