



Sejam bem-vindos!

ESTRUTURA DE DADOS

Prof. Jarbas Araújo



Estrutura de dados

Para o profissional da área da computação, é importante conhecer a forma como os dados são armazenados, organizados e manipulados, pois essa atividade é uma das principais desenvolvidas pelos sistemas computacionais.

Nesse sentido, a estrutura de dados é um elemento que vem, principalmente na linguagem de programação C, para auxiliar na representação e na abstração de estruturas mais complexas, executando operações de armazenamento e busca de dados na memória, de maneira mais sofisticada e robusta.

Estrutura de dados

O QUE É UMA ESTRUTURA DE DADOS?

SINTAXE DA DECLARAÇÃO:

```
struct nome_da_estrutura {  
    tipo_do_campo1 nome_do_campo1;  
    tipo_do_campo2 nome_do_campo2;  
    tipo_do_campo3 nome_do_campo3;  
    ...  
};
```

Estrutura de dados

Serve para auxiliar o armazenamento, no computador, de dados que são vistos na vida real e, por isso, precisam de abstração.

É formada por elementos, membros ou campos.

Agrupar e manipular dados dentro de uma entidade identificada por meio de um único nome de variável.

Envolve o armazenamento de dados organizados na memória de forma mais sofisticada do que utilizando variáveis simples.

Permite executar operações e fazer manipulação de dados, indicando como os dados são representados e a forma possível de manipulá-los.

Faz referência aos dados utilizando índices que servem para otimizar a localização de um registro quando é feita uma consulta que envolve vários dados.

Estrutura de dados

A estrutura de dados

Um dado é algo normalmente quantificável, o qual não tem um significado relevante de maneira isolada. O dado é considerado como o fundamento da informação.

Conhecendo-se somente o dado, dificilmente é possível chegar a um entendimento sobre o assunto tratado, tomar decisões ou chegar a conclusões.

Já a informação consiste na ordenação e na organização dos dados, de maneira que se torne possível compreender o seu significado, ou entendê-la dentro de um contexto.

Em outras palavras, a informação é o fruto do processamento dos dados, depois que estes são analisados, interpretados de uma maneira pré-definida e qualificados.

Estrutura de dados

A eficiência de uma estrutura de dados envolve, normalmente, variáveis, como o tempo e o espaço. Se uma aplicação precisa manipular estruturas de dados de forma muito intensa e repetida, a velocidade com que essas manipulações são feitas será decisiva.

Da mesma forma, uma aplicação que precisa manipular uma grande quantidade de estruturas de dados, e tem um código gigantesco para fazer a representação dessas estruturas, não será eficiente (LAUREANO, 2008).

Estrutura de dados

Uma estrutura de dados é um elemento que permite executar operações e fazer a manipulação de dados. Ela serve para indicar como os dados são representados e como se torna possível a sua manipulação. Um exemplo muito comum para entender a estrutura de dados é a representação de um baralho de cartas:

- ☐ A representação das cartas pode ser feita por dois valores do tipo caractere, que são o naipe e o valor da carta.
- ☐ As operações que podem ser feitas com as cartas são:
 - a) embaralhar todas as cartas;
 - b) comprar uma carta do topo do baralho;
 - c) retirar uma carta de um lugar qualquer do meio do baralho;
 - d) inserir uma carta na base do baralho.

Estrutura de dados

A declaração de uma estrutura de dados normalmente é feita pela sua nomeação e pela indicação dos campos que farão parte dela, como o tipo e o nome das variáveis. Algumas observações importantes sobre a declaração da estrutura de dados são:

- ☐ A chave que fecha a declaração da estrutura de dados (struct) deve ser seguida de um ponto-e-vírgula.
- ☐ Não é possível que mais de um campo tenha o mesmo nome.

Estrutura de dados

Essa é a forma de declarar uma estrutura de dados na linguagem de programação C:

```
struct nome_da_estrutura {  
    tipo_do_campo1 nome_do_campo1;  
    tipo_do_campo2 nome_do_campo2;  
    tipo_do_campo3 nome_do_campo3;  
    ...  
};
```

Com base na declaração acima, este é um exemplo de declaração de estrutura de dados em C:

```
struct boletim (  
    char nome_do_aluno(50);  
    char nome_disciplina(50);  
    int qtd_faltas;  
    float nota1;  
    float nota2;  
    float nota_exame;  
);
```

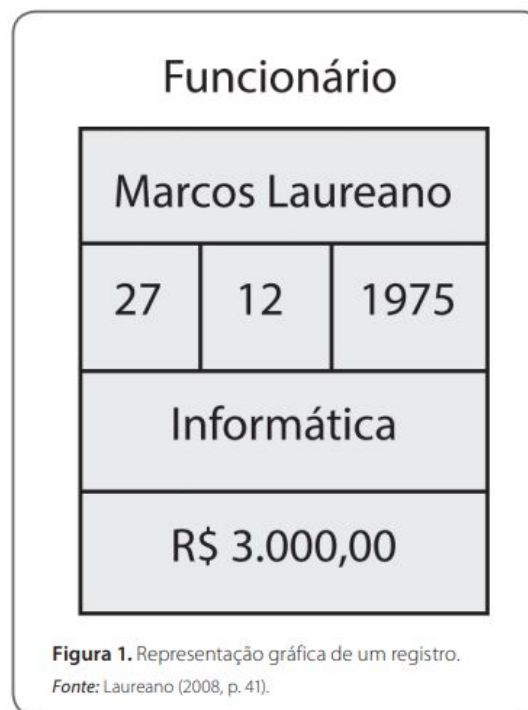
Estrutura de dados

Dados heterogêneos e homogêneos

Quando uma estrutura de dados armazena elementos de tipos diferentes, é dito que esta é uma estrutura de dados heterogênea. Normalmente, esse tipo de dado é chamado de registro, então, o registro é uma estrutura de dados que faz o agrupamento de dados de tipos diferentes entre si. Ele é formado por uma quantidade determinada de campos, os quais são itens individuais de dados e relacionados de forma lógica (LAUREANO, 2008)

Estrutura de dados

Os registros consistem em conjuntos de posições de memória, os quais são identificados pelo mesmo nome e são individualizados por meio de identificadores que se associam a cada conjunto de posições (Figura 1).



Para referenciar um único campo dentro de um registro, é preciso utilizar essa sintaxe:

```
nome_da_estrutura[índice].nome_do_campo;
```

Estrutura de dados

Por sua vez, uma estrutura de dados que armazena somente um tipo de dado é conhecida, então, como dados homogêneos. Os elementos que compõem uma estrutura de dados homogênea correspondem às posições de memória, as quais são identificadas por meio de um mesmo nome, individualizadas por índices que são todos dos mesmos tipos de dados.

Os dados homogêneos são representados pelos vetores, ou estruturas de dados unidimensionais, e pelas matrizes, que são estruturas de dados bidimensionais (LAUREANO, 2008).

Estrutura de dados

Vetores

Os vetores são estruturas de dados unidimensionais lineares e, por isso, precisam de um único índice para fazer o endereçamento e para percorrer toda a estrutura. Um vetor é uma estrutura de dados utilizada para fazer o armazenamento de uma lista de valores do mesmo tipo, ou seja, em uma mesma variável, vários valores de mesmo tipo são armazenados.

Quando um vetor é definido, é declarada uma quantidade fixa de posições que ele deve ter, ou seja, um vetor é um elemento que será dividido em várias posições que serão entendidas e reconhecidas pelo computador.

Estrutura de dados

Uma estrutura de dados do tipo vetor tem as seguintes características (DEITEL; DEITEL, 2011):

- É alocado de forma estática, ou seja, no momento da sua declaração, deve-se conhecer o tamanho ou a quantidade de posições que ele terá.
- É uma estrutura de dados homogênea, formada por elementos com o mesmo tipo de dado.
- Cada posição do vetor contém somente um valor.
- Os tamanhos dos valores de cada posição, por serem do mesmo tipo, são iguais.
- Faz a alocação dos dados de forma sequencial (Figura 2).

A sintaxe para a definição do vetor ao lado seria:

```
tipo_de_dado  
nome_do_vetor[quantidade_de_posicoes_do_vetor]
```

Em C:

```
float nota[5];
```

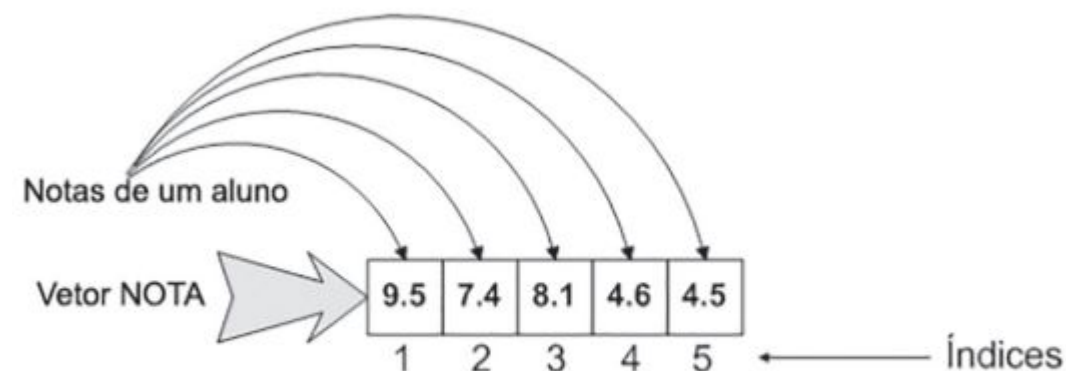


Figura 2. Representação gráfica de um vetor.

Fonte: Laureano (2008, p. 27).

Estrutura de dados

O funcionamento de um vetor consiste em determinar a localização de todos os seus elementos, partindo do endereço do primeiro elemento, o que se torna possível porque todos estes ficam dispostos lado a lado e cada elemento tem um tamanho fixo. É importante lembrar que a primeira posição de um vetor será sempre conhecida por 0 e nunca por 1.

```
nota[0] = 9.5;  
nota[1] = 7.4;  
nota[2] = 8.1;  
nota[3] = 4.6;  
nota[4] = 4.5;
```

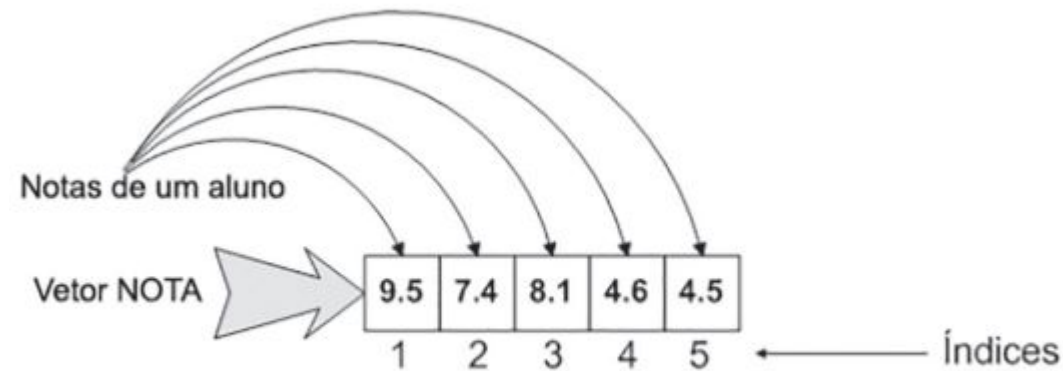


Figura 2. Representação gráfica de um vetor.

Fonte: Laureano (2008, p. 27).

Estrutura de dados

Matrizes

As matrizes são estruturas de dados bidimensionais, os quais precisam de dois índices para fazer o endereçamento e para percorrer toda a estrutura: um que servirá para referenciar a linha e outro que servirá para referenciar a coluna da matriz (TENENBAUM; LANGSAM; AUGENSTEIN, 1995).

Assim como os vetores, as matrizes têm algumas características:

- são alocadas de forma estática, ou seja, no momento da sua declaração, deve-se conhecer o tamanho que elas terão;
- são uma estrutura de dados homogênea, a qual é formada por elementos com o mesmo tipo de dados;
- cada posição da matriz contém somente um valor;
- os tamanhos dos valores de cada posição, por serem do mesmo tipo, são iguais;
- fazem a alocação dos dados de forma sequencial (Figura 3)

Diagrama de uma matriz 3x6. As linhas são rotacionadas verticalmente e as colunas horizontalmente. A matriz contém as seguintes letras:

| | | | | | | | |
|----------|---|---|---|---|---|---|-----------|
| | 1 | 2 | 3 | 4 | 5 | 6 | ← Colunas |
| 1 | M | A | R | C | O | S | |
| 2 | N | A | S | S | E | R | |
| 3 | D | O | N | A | L | D | |
| ↑ Linhas | | | | | | | |

Figura 3. Representação gráfica de uma matriz.

Fonte: Laureano (2008, p. 31).

Estrutura de dados

A sintaxe para a definição da matriz acima seria:

```
tipo_de_dado  
nome_da_matriz[quantidade_de_linhas][quantidade_de_colunas]
```

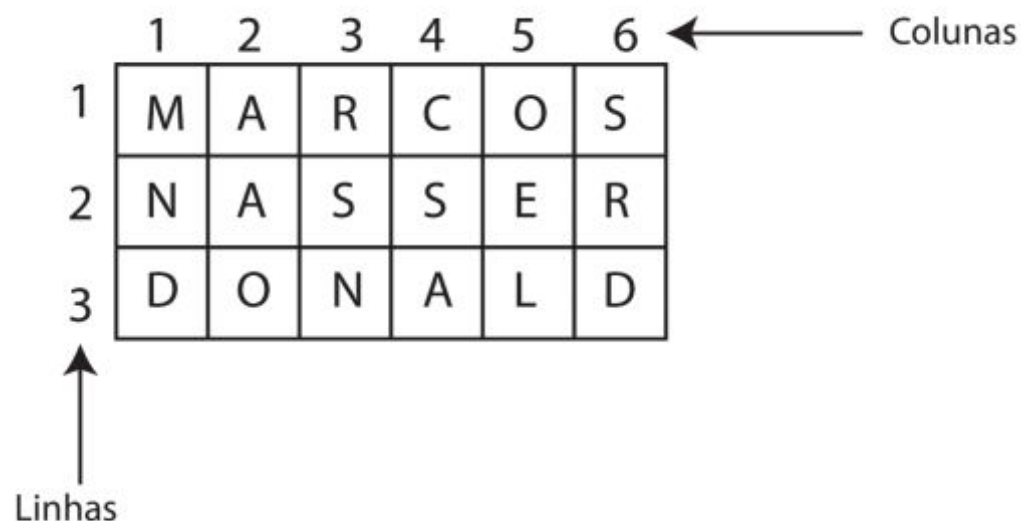
Em C:

```
char letras[3][6];
```

Estrutura de dados

A alocação de valores para os elementos da matriz pode ser feita linha por linha ou coluna por coluna. É importante lembrar, da mesma forma que os vetores, que a primeira posição de cada dimensão da matriz será sempre conhecida por 0 e nunca por 1.

```
letras[0][0] = "M";  
letras[0][1] = "A";  
letras[0][2] = "R";  
letras[0][3] = "C";  
letras[0][4] = "O";  
letras[0][5] = "S";  
letras[1][0] = "N";  
letras[1][1] = "A";  
letras[1][2] = "S";  
letras[1][3] = "S";  
letras[1][4] = "E";  
letras[1][5] = "R";  
letras[2][0] = "D";  
letras[2][1] = "O";  
letras[2][2] = "N";  
letras[2][3] = "A";  
letras[2][4] = "L";  
letras[2][5] = "D";
```



| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | M | A | R | C | O | S |
| 2 | N | A | S | S | E | R |
| 3 | D | O | N | A | L | D |

Figura 3. Representação gráfica de uma matriz.

Fonte: Laureano (2008, p. 31).

Estrutura de dados

Ponteiros

Um ponteiro é um tipo de dado que se diferencia de uma variável, no sentido de que a variável faz uma referência direta a um valor. Já o ponteiro faz uma referência indireta a um valor. Quando se faz a modificação do valor de um ponteiro, está sendo modificado o valor da variável para a qual o ponteiro está apontando. Em outras palavras, o ponteiro é um tipo de dado que permite referenciar a posição de um objeto na memória, mas também o próprio objeto (DEITEL; DEITEL, 2011).

Quando se trabalha com ponteiros, o operador unário `*` é utilizado para acessar o conteúdo da variável para o qual o ponteiro está apontando, enquanto o operador unário `&` é utilizado para acessar o endereço da variável para o qual o ponteiro está apontando.



PILHAS

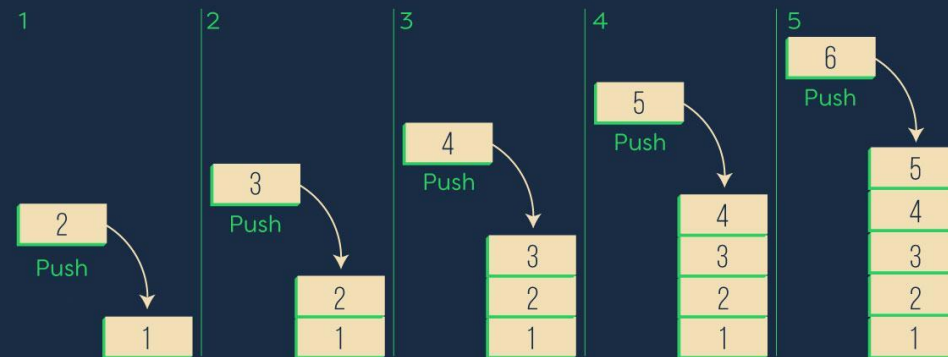
Pilha é uma estrutura de dados que admite remoção de elementos e inserção de novos objetos. Sempre que houver uma remoção, o elemento removido é o que está na estrutura há menos tempo.

Vários *softwares* utilizados no dia a dia contam com estrutura de dados, porém nem sempre as pessoas se dão conta. Bons exemplos são os *softwares* aplicativos, como editores de texto, editores de planilhas, etc. A estrutura de pilhas funciona como se fosse uma pilha de objetos mesmo, ou seja, você insere e retira elementos sempre pelo topo.

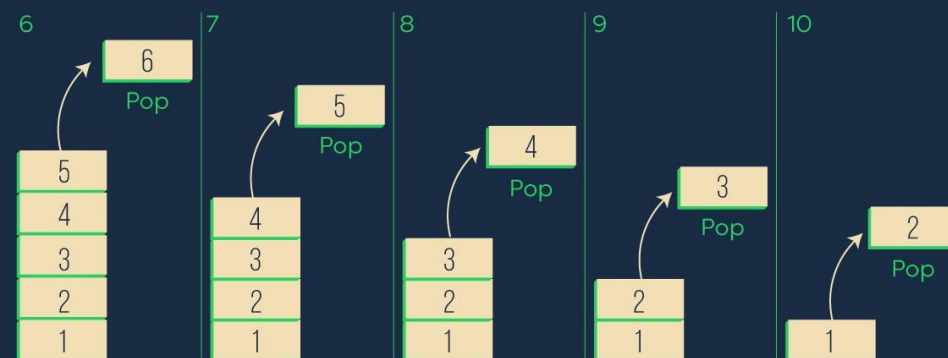
PILHAS

Pilhas são estruturas de dados do tipo last in, first out (LIFO), ou seja, o último elemento a ser inserido na estrutura, será o primeiro a ser retirado da estrutura. Podemos fazer uma comparação com uma pilha de pratos, em que, se quisermos adicionar um prato na pilha, devemos colocá-lo topo, e, para pegar um prato da pilha, retiramos o do topo. Dessa forma, temos que retirar o prato do topo para ter acesso ao próximo prato. Portanto, essa manipulação é feita apenas por uma das extremidades da lista, pelo topo. Para processar/acessar o penúltimo item da estrutura, deve-se remover o último item. Observe a representação de pilhas na Figura 1.

OS ELEMENTOS VÃO SENDO INSERIDOS UM A UM COM O COMANDO PUSH.



E RETIRADOS COM O COMANDO POP.

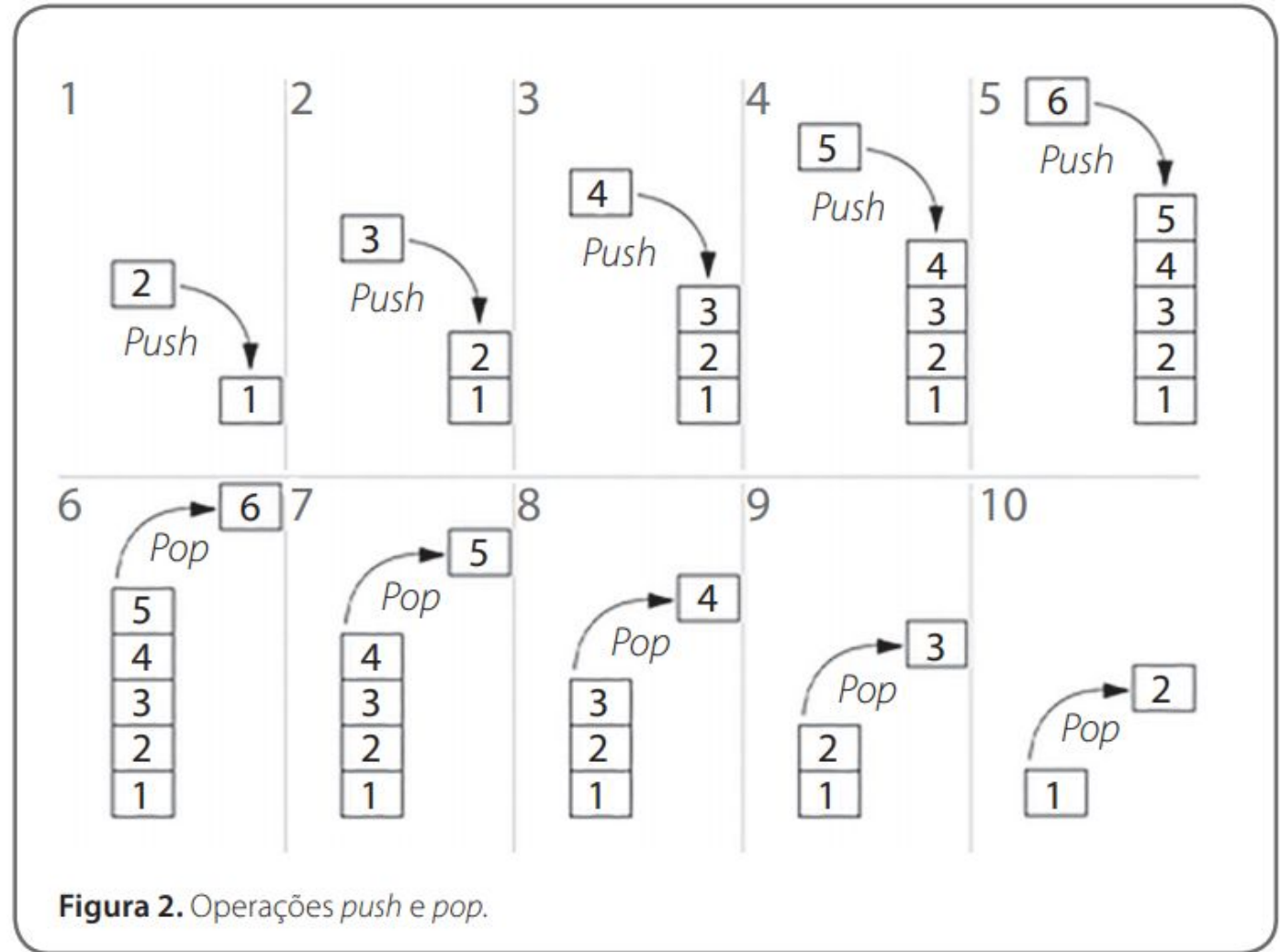


ASSIM SÃO EMPILHADOS E DESEMPILHADOS SEMPRE PELO TOPO DA PILHA.

PILHAS

Para manipular os dados em uma pilha estática, você pode usar as seguintes operações:

criação (pull);
inserção (push)
remoção (pop);
acessa elemento (top).



Filas

Filas são estruturas de dados que armazenam os elementos de forma sequencial (linear). As inserções e retiradas dos elementos são feitas em extremidades diferentes. Existe acesso às duas extremidades: começo, onde é feita a retirada, e término, onde é feita a inserção

FILAS

Segue o critério **FIFO – First in, first out**. Ou seja, o primeiro elemento inserido será o primeiro retirado: inserção por uma extremidade e exclusão pela outra.

INSERE ELEMENTOS NA FILA



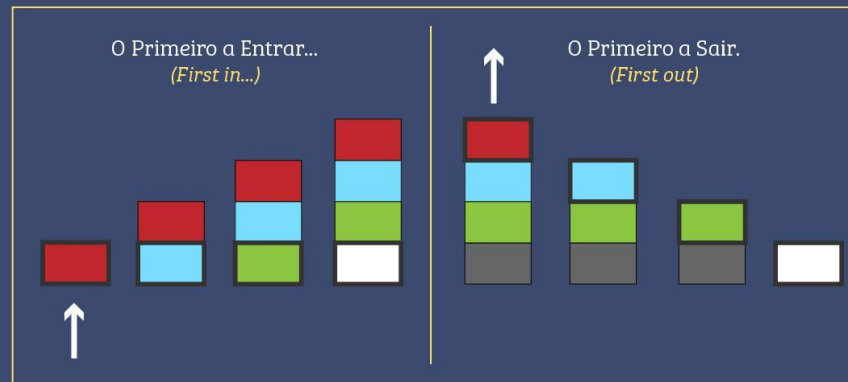
REMOVE ELEMENTOS NA FILA



Ou, ainda:

O Primeiro a Entrar...
(First in...)

O Primeiro a Sair.
(First out)



Filas

Estrutura de dados é o modo particular de armazenamento e organização de dados no computador, para que possam ser usados eficientemente, facilitando sua busca e modificação

A fila pode ser representada por duas formas, como descrito a seguir.

- Estática: caracteriza-se por utilizar um vetor (estrutura estática) para representar a fila.
As operações básicas suportadas em uma fila estática são:
inserção;
remoção;
consulta;
listagem.
- Dinâmica: caracteriza-se por utilizar uma lista encadeada (estrutura dinâmica) para representar a fila.
A fila dinâmica é sempre implementada com dois ponteiros, um ponteiro no início da fila e o outro ponteiro no final da fila.
As operações básicas suportadas em uma fila dinâmica são as mesmas da fila estática.

Ordenação de dados - Métodos simples

No ambiente computacional, a ordenação desempenha um papel fundamental, uma vez que diversas atividades necessitam apresentar dados classificados, como, por exemplo, listas de chamada e rankings de classificação.

Cada método de ordenação fornece uma solução diferente e que pode ser adequada para determinado tipo de tarefa, conforme a distribuição e a quantidade de dados a ser classificada.

Conhecer os métodos de ordenação simples é importante para compreender um dos principais fundamentos da estrutura de dados.



Ordenação de dados - Métodos simples

Há três técnicas de ordenação simples que pertencem ao universo da estrutura de dados: Bubblesort, Selectionsort e Insertionsort. Analistas e programadores devem conhecer o funcionamento de cada uma delas para implementar a mais adequada de acordo com a distribuição e o volume de dados existentes, buscando desenvolver rotinas tanto ágeis como eficazes.

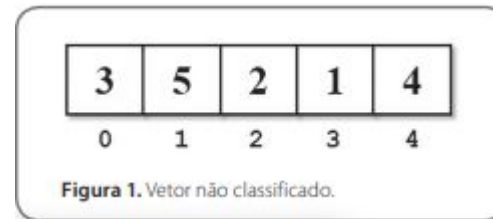
Ordenação de dados - Métodos simples

Método Bubblesort

Bubblesort, ou simplesmente ordenação bolha, é um método de ordenação simples de vetores, cujo nome se origina de uma comparação com a flutuação de bolhas em um recipiente com água, em que cada bolha procura seu próprio nível conforme o seu tamanho (SCHILDT, 1997).

As comparações são realizadas entre todos os elementos do vetor. Cada elemento, a partir da sua posição atual, será comparado com os elementos das posições subsequentes. Se o elemento atual for maior que os próximos elementos, haverá troca de posições, podendo haver várias trocas a cada iteração (CELES; CERQUEIRA; RANGEL, 2004).

Considere como exemplo um vetor com cinco posições, em que os elementos estão fora de ordem classificatória. As posições desse vetor são identificadas de 0 a 4, conforme ilustra a Figura 1.



Ordenação de dados - Métodos simples



Figura 2. Vetor após a primeira iteração.

```
vetor [0] x vetor[1] (3 > 5 ? não faz nada)
vetor [0] x vetor[2] (3 > 2 ? troca o 2 com o 3)
vetor [0] x vetor[3] (2 > 1 ? troca o 1 com o 2)
vetor [0] x vetor[4] (1 > 4 ? não faz nada)
```



Figura 3. Vetor após a segunda iteração.



Figura 4. Vetor após a terceira iteração.

```
vetor [2] x vetor[3] (5 > 3 ? troca o 5 com o 3)
vetor [2] x vetor[4] (3 > 4 ? não faz nada)
```


Ordenação de dados - Métodos simples

O algoritmo desse método de ordenação está apresentado a seguir. Ele recebe um vetor de números inteiros e o total de elementos desse vetor, implementa dois laços de repetição e utiliza uma variável temporária para auxiliar as trocas durante as iterações (CELES; CERQUEIRA; RANGEL, 2004).

A função imprimir facilita a visualização do vetor antes e depois da ordenação.

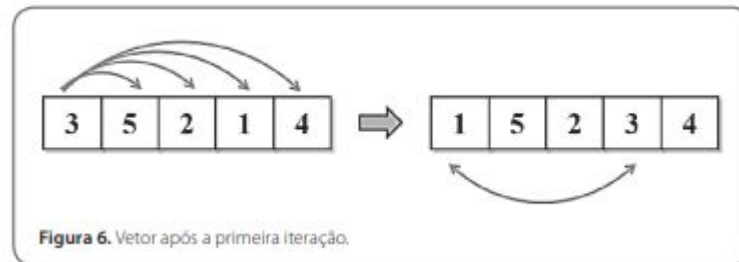
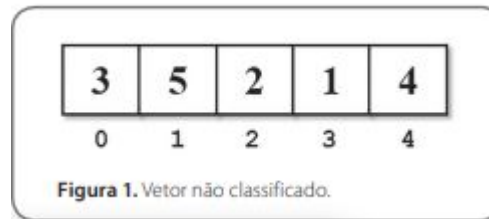
Ordenação de dados - Métodos simples

```
1 #include <stdio.h>
2 #define TAM 5
3
4 void bolha (int *vetor, int total);
5
6 main() {
7     int vetor[TAM]= {3, 5, 2, 1, 4};
8
9     imprimir(vetor, TAM);
10    bolha(vetor, TAM);
11    imprimir(vetor, TAM);
12 }
13 void bolha (int *vetor, int total) {
14     int i, j, tmp;
15
16     for (i= 0; i < total; i++) {
17         for (j= i+1; j < total; j++) {
18             if (vetor[i] > vetor[j]) {
19                 tmp= vetor[i];
20                 vetor[i]= vetor[j];
21                 vetor[j]= tmp;
22             }
23         }
24     }
25 }
26
27 void imprimir(int *vetor, int total) {
28     int j;
29     for (j= 0; j < total; j++) {
30         printf("%d ", vetor[j]);
31     }
32     printf("\n");
33 }
```

Ordenação de dados - Métodos simples

Método Selectionsort

Selectionsort, também conhecido como ordenação por seleção, é outro método de ordenação simples de vetores. Assim como o Bubblesort, possui dois laços de repetição para realizar a comparação de todos os elementos, porém, efetua apenas uma troca a cada fase de iterações (CELES; CERQUEIRA; RANGEL, 2004).



```
menor => 3 (posição 0)
vetor [0] x vetor[1] (5 < 3 ? não faz nada)
vetor [0] x vetor[2] (2 < 3 ? menor => 2)
vetor [0] x vetor[3] (1 < 2 ? menor => 1)
vetor [0] x vetor[4] (4 < 1 ? não faz nada)
```

Ordenação de dados - Métodos simples

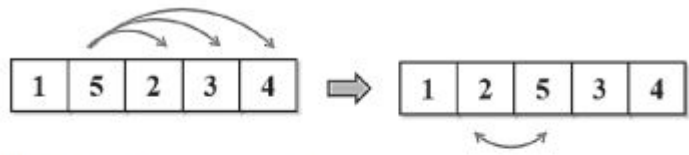


Figura 7. Vetor após a segunda iteração.

```
menor => 5 (posição 1)
vetor [1] x vetor[2] (2 < 5 ? menor => 2)
vetor [1] x vetor[3] (3 < 2 ? não faz nada)
vetor [1] x vetor[4] (4 < 1 ? não faz nada)
```



Figura 8. Vetor após a terceira iteração.

```
menor => 5 (posição 2)
vetor [2] x vetor[3] (3 < 5 ? menor => 3)
vetor [2] x vetor[4] (4 < 3 ? não faz nada)
```



Figura 9. Vetor após a última iteração.

```
menor => 5 (posição 3)
vetor [3] x vetor[4] (4 < 5 ? menor => 4)
```

Ordenação de dados - Métodos simples

Método Insertionsort

Insertionsort, ou ordenação por inserção, é o terceiro e último método de ordenação simples de vetores. Esse método realiza a comparação a partir do segundo elemento do vetor com os elementos das posições anteriores, para inseri-lo na posição correta até aquele momento (FORBELLONE, 2005).

No entanto, quando um vetor se encontra em ordem pré-classificada ou até classificada, sua eficiência é bem superior, proporcionando bons resultados em termos de desempenho, pois realiza poucas comparações e quase nenhuma troca, dependendo da distribuição dos dados (MATTOS; LORENZI; CARVALHO, 2007).

Ordenação de dados - Métodos simples

| | | | | |
|---|---|---|---|---|
| 3 | 5 | 2 | 1 | 4 |
| 0 | 1 | 2 | 3 | 4 |

Figura 1. Vetor não classificado.



Figura 10. Vetor após a primeira iteração.

Numero => 5 (posição 1)
vetor [1] x vetor[0] (5 < 3 ? não faz nada)

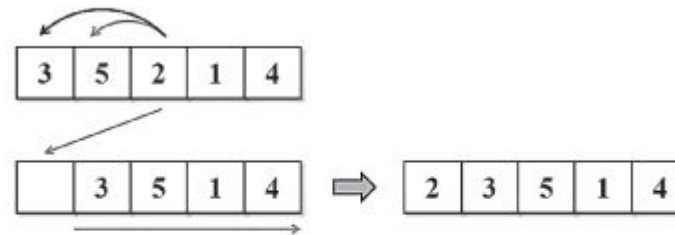
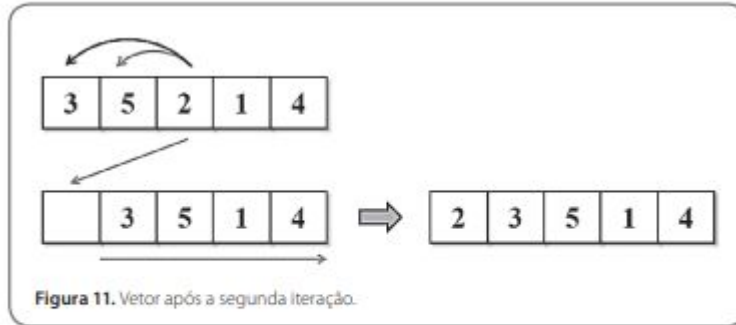


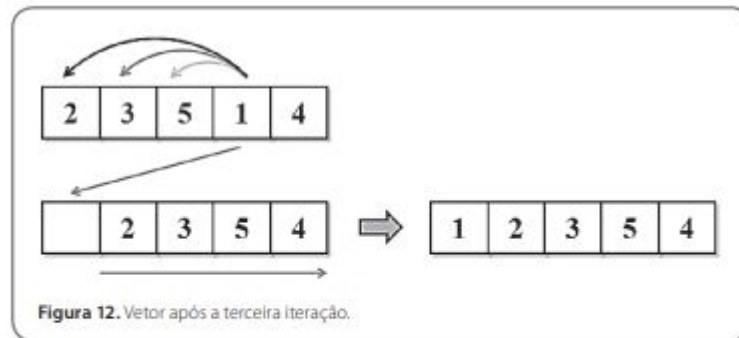
Figura 11. Vetor após a segunda iteração.

número => 2 (posição 2)
vetor [2] x vetor[0] (2 < 3 ? nova posição identificada)
vetor [2] x vetor[1] (2 < 5 ? essa comparação foi descartada)

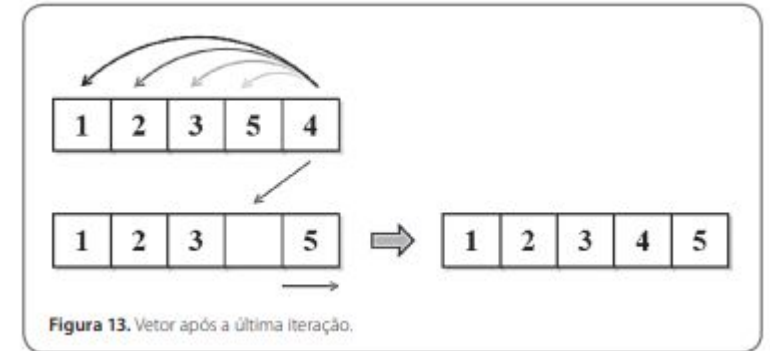
Ordenação de dados - Métodos simples



número => 2 (posição 2)
vetor [2] x vetor[0] (2 < 3 ? nova posição identificada)
vetor [2] x vetor[1] (2 < 5 ? essa comparação foi descartada)



número => 1 (posição 3)
vetor [3] x vetor[0] (1 < 2 ? nova posição identificada)
vetor [3] x vetor[1] (2 < 3 ? essa comparação foi descartada)
vetor [3] x vetor[2] (2 < 5 ? essa comparação foi descartada)



número => 4 (posição 4)
vetor [4] x vetor[0] (4 < 1 ? não faz nada)
vetor [4] x vetor[1] (4 < 2 ? não faz nada)
vetor [4] x vetor[2] (4 < 3 ? não faz nada)
vetor [4] x vetor[3] (4 < 5 ? nova posição identificada)

MODULARIZAÇÃO

À medida que um software vai aumentando em tamanho e complexidade, a sua **manutenibilidade** vai tornando-se mais difícil e dispendiosa. Dentre as várias técnicas que você pode utilizar para ajudar a resolver esse problema, destaca-se a modularização, que consiste na divisão do código em pequenos módulos, chamados de procedimentos e funções.

Quando dividimos um programa em pequenas partes, cada uma delas tem seu papel bem definido na execução. Como vantagens de dividir o programa em pequenas partes, podemos citar:

- Facilidade ao escrever o algoritmo: o programador pode focalizar pequenas partes do problema complicado e escrever a solução para essas partes, uma de cada vez, em vez de tentar resolver o problema como um todo e de uma só.
- O algoritmo fica mais fácil de ler: dividir o algoritmo em módulos permite que alguém que não seja o autor, entenda o algoritmo mais rapidamente por tentar entender os seus módulos separadamente, pois cada módulo é menor e mais simples do que o algoritmo monolítico correspondente, entender separadamente é mais fácil do que tentar entender como um todo e de uma só vez.





- **Abstração:** é possível entender o que um algoritmo faz por saber apenas o que seus módulos fazem, sem a necessidade de entender os detalhes internos dos módulos.
- **Economia de tempo e esforço:** necessitamos executar a mesma tarefa em vários locais em um mesmo algoritmo. A partir de um módulo, você pode chama-lo quantas vezes quiser e de qualquer parte do algoritmo, evitando reescrevê-lo mais de uma vez.
- **Reutilização:** após a criação de um módulo, é possível utilizá-lo em outros algoritmos sem que eles sejam modificados, da mesma forma que se usa os operadores leia e escreva.

MODULARIZAÇÃO

Mesmo sendo implementados de forma diferente, os conceitos da modularização continuam valendo para todas as linguagens de programação.

O que muda é a forma
de implementação

Algumas criam funções
e procedimentos

Outras criam somente métodos

MODULARIZAÇÃO

Criação e chamadas de funções e procedimentos Existem duas formas de definir um módulo: pela função ou pelo procedimento.

- ❑ Função: é um módulo que produz um único valor de saída. Pode ser vista como uma expressão que é avaliada para um único valor, sua saída, ou uma função.
- ❑ Procedimento: é um tipo de módulo usado para várias tarefas, não produzindo valores de saída.

Suporte

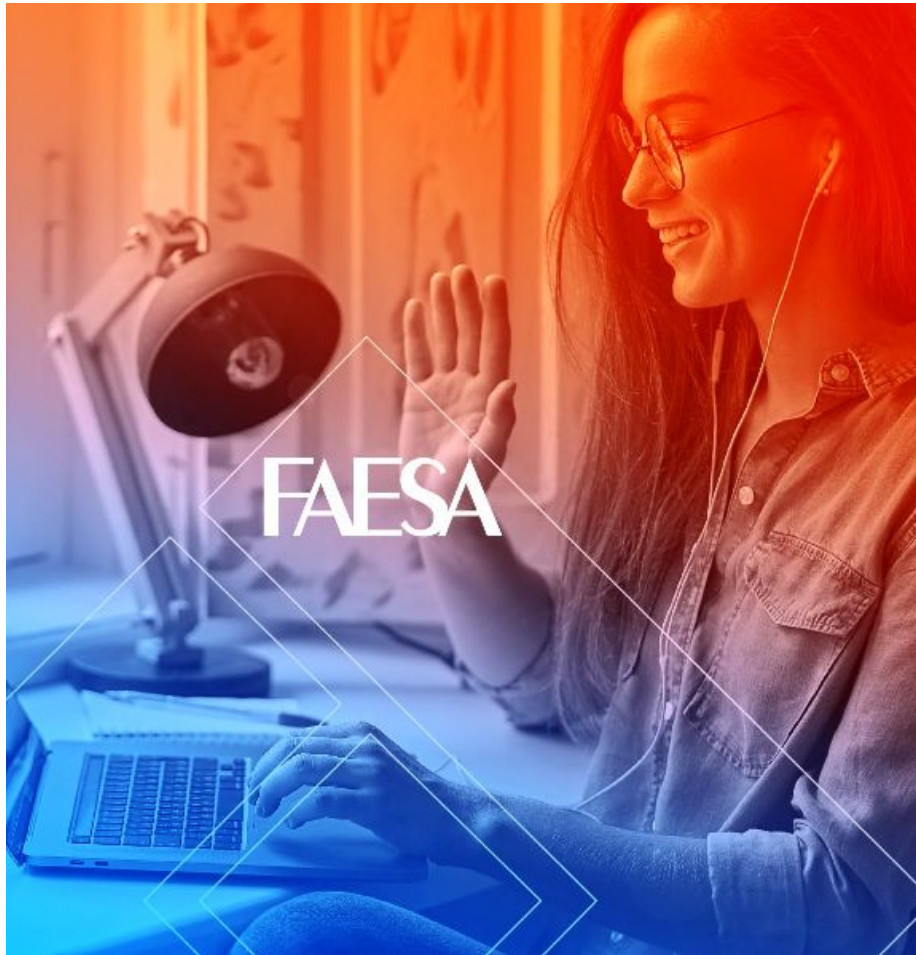
Atendimento de segunda a sexta-feira,
das 13h às 21h (exceto feriados).



Flávia
27 99278-7017

AVA FAESA -

<https://faesa.grupoa.education/plataforma>



Seja bem-vindo

 Usuário

 Senha



Entrar

[Esqueci o usuário](#) ou [Esqueci a senha](#)



AVA FAESA - Conteúdo da disciplina

1. Home da unidade de estudo.
2. Apresentação da unidade.
3. Desafio sobre o tema da unidade.
4. Infográfico que resume o conteúdo a ser estudado.
5. Conteúdo do livro – capítulos selecionados para estudo.
6. Dica do Professor.
7. Exercícios de autoestudo.
8. Aplicação prática dos conteúdos.
9. Indicação de novas leituras e outros recursos para aprofundamento.
10. Impressão (em papel ou em PDF) de toda a Unidade (exceto material multimídia).

The screenshot displays the FAESA EAD interface for the course 'Estrutura de dados' (Disciplina: Estrutura de Dados (10700010054_20231_01)). The interface is divided into two main sections: a sidebar menu on the left and a main content area on the right.

Sidebar Menu (Left):

- Conteúdo** (highlighted with an orange circle '1')
 - Apresentação
 - Desafio
 - Infográfico
 - Conteúdo do Livro
 - Dica do Professor
 - Exercícios
 - Na prática
 - Saiba mais

Main Content Area (Right):

- Estrutura de dados** (highlighted with an orange circle '10')
 - Disciplina: Estrutura de Dados (10700010054_20231_01)
 - Thumbnail image showing a laptop screen with C++ code and the word 'Est'.

At the bottom of the interface, there is a small text line: 'Para o profissional da área da computação é importante conhecer a'.



Obrigado! Ótimos estudos.

Jarbas Araujo

 @profjarbasaraujo

