

Algoritmos e Estruturas de Dados

Caros alunos, as vídeo aulas desta disciplina encontram-se no AVA (Ambiente Virtual de Aprendizagem).

Unidade 1

Introdução a Algoritmos e Estruturas de Dados

Introdução da Unidade

Olá aluno. Nesta unidade vamos conhecer mais sobre algoritmos e estruturas de dados. Vamos estudar um pouco das estruturas de dados mais conhecidas e que são amplamente utilizadas na solução de problemas computacionais e no desenvolvimento de aplicações das mais diversas, nesta unidade vamos nos concentrar nos *Arrays* e ao final já poderemos utilizar os recursos desta Estrutura para algumas aplicações bem bacanas.

No decorrer da disciplina abordaremos outras estruturas que são fundamentais para praticamente qualquer projeto de *software*.

O que vamos estudar nesta disciplina será de suma importância para desenvolver programas de qualidade e que sejam eficientes em termos de uso de processamento e de memória, ou seja, a ideia é produzir códigos ou programas que sejam executados mais rápido e que consumam menos memória.

Por se tratar de uma disciplina introdutória espero que você tenha apenas noções básicas de alguma linguagem de programação e que conheça o conceito de variáveis, métodos ou funções, estruturas de decisão do tipo *IF-THEN-ELSE*, e estruturas de repetição como os laços *FOR* e *WHILE*. Mas não se assuste, se você não conhece ou não lembra, terá a oportunidade de ser direcionado para isso no momento oportuno. Vou trabalhar os exercícios utilizando Java como linguagem, porém os conteúdos estudados são aplicáveis a qualquer outra linguagem que você queira utilizar.

Objetivos

- Conhecer os Fundamentos de Estruturas de Dados;
- Conhecer e utilizar a estrutura de um *Array* Unidimensional;
- Conhecer e utilizar a estrutura de um *Array* Multidimensional;
- Aplicar operações em *Arrays*.

Conteúdo programático

Aula 01 – Introdução às Estruturas de Dados

Aula 02 – *Arrays* Unidimensionais, Multidimensionais e Operações com *Arrays*



Você poderá também **assistir às videoaulas** em seu celular! Basta apontar a câmera para os **QRCodes** distribuídos neste conteúdo.

Pode ser necessário instalar um aplicativo de leitura QRcode no

Aula 1 Introdução às Estruturas de Dados



Videoaula Apresentação

Utilize o QRcode para assistir!

Videoaula Apresentação



Videoaula Mini Currículo

Utilize o QRcode para assistir!

Videoaula Mini Currículo



Por que estudamos Estruturas de Dados?

As estruturas de dados são formas de lidar e tratar dados de um programa e irão permitir que tenhamos códigos de melhor qualidade. Utilizar Estruturas adequadas irão proporcionar, portanto, códigos que sejam mais eficientes, tanto no aspecto de consumo de *hardware*, minimizando muitas vezes o consumo de processamento, quanto no aspecto de eficiência na execução dos programas.

Ao final desta disciplina espero que você consiga aplicar as estruturas corretas mais indicadas para situações computacionais que são comumente encontradas na implementação.

Será possível também utilizar as estruturas adequadas para resolver problemas mais complexos de algoritmos e programação sem ter que “reinventar a roda”, a maioria das linguagens de programação tem, senão todas, a maioria das estruturas de dados já implementadas e poderemos utilizá-las quase que nativamente.

As estruturas mais comuns são:

- Vetores (ou *Arrays*);

- Listas ou Filas e Pilhas (*Queues e Stacks*);
- Listas Encadeadas (*Linked Lists*);
- Listas Duplamente Encadeadas (*Double Linked Lists ou Doubly Linked Lists*);
- Conjuntos (*Set*);
- Tabelas Hash (*Hash Tables*);
- Árvores (*Trees*);
- Grafos (*Graphs*).

Conhecimentos prévios:

- Variáveis, Tipos de Dados, comandos de Entrada e Saída;
- Estruturas de Controle (*IF / CASE*);
- Estruturas de Repetição (*WHILE / FOR / DO WHILE*);
- Funções (métodos e passagem de parâmetros).

Tipos Primitivos

Para iniciarmos o nosso estudo sobre as estruturas de dados vamos relembrar alguns conceitos preliminares.

Os **tipos de dados primitivos (tipos primitivos)** são basicamente os tipos de dados que uma determinada linguagem pode processar de forma direta. Os tipos primitivos são muito similares entre a maioria das linguagens. Em Java, que é a linguagem que vamos utilizar de apoio para esta disciplina, temos basicamente os seguintes tipos primitivos de dados.

Quadro 1 – Tipos Primitivos da Linguagem Java

Tipo primitivo	Descrição	Exemplo de Valores
<i>boolean</i>	Valores booleanos	<i>true false</i>
<i>char</i>	1 caractere Unicode de 16 <i>bits</i>	uma letra, um número ou um símbolo da tabela ASCII
<i>byte</i>	um inteiro de 8 <i>bits</i>	um número inteiro entre -128 e 127
<i>short</i>	um inteiro de 16 <i>bits</i>	um número inteiro entre -32768 e 32767

<i>int</i>	um inteiro de 32 <i>bits</i>	um número inteiro entre - 2.147.483.648 e 2.147.483,647
<i>long</i>	um inteiro de 64 <i>bits</i>	um número inteiro entre -2^{63} e $2^{63}-1$
<i>float</i>	número com ponto flutuante de 32 <i>bits</i>	números com casas decimais
<i>double</i>	número com ponto flutuante de 64 <i>bits</i>	números com casas decimais

Fonte: elaborado pelo próprio autor (2020) (adaptado de DEITEL, 2010)

Indicação de Leitura

Caso você tenha dificuldade com os tipos elementares ou primitivos, reveja os conteúdos das disciplinas de Lógica de Programação e Algoritmos ou Linguagem de Programação, onde estes conceitos foram trabalhados de forma mais detalhada.

Variáveis e Constantes

Estamos diante de um dos conceitos e conteúdos mais importantes no desenvolvimento de algoritmos. As variáveis e as constantes são comumente utilizadas na maioria dos algoritmos desenvolvidos, desde os mais simples aos mais complexos. Sua utilização é importante na entrada de dados, no processamento e na saída de informações exibidas pelos sistemas ao usuário.

Este conteúdo faz parte do conhecimento prévio para iniciar os estudos nesta disciplina. Portanto vamos apenas revisar este conteúdo.

Variáveis são locais na memória, criados a partir do código do desenvolvedor para trabalhar com valores flutuantes durante a execução do sistema. As variáveis podem ser utilizadas pelo algoritmo. Por exemplo: em um algoritmo que se deseja armazenar o nome do usuário ou de uma pessoa para ser utilizado posteriormente, pode-se criar uma variável do tipo de dado caractere para armazenar o nome. Dentro de cada linguagem de programação tem-se os tipos de dados específicos. As constantes são valores fixos colocados direto no algoritmo, ou seja, são valores diretos. Quando se sabe que o valor percentual da comissão de um vendedor é 5%, esse valor pode ser colocado em forma de constante no código e não de uma variável.

Tipos de Dados

Os tipos de dados primitivos como já abordado, depende da linguagem de programação, porém se considerarmos a lógica de programação em si, temos os tipos:

- **Inteiros:** valores numéricos inteiros positivos e negativos incluindo o 0 (zero);
- **Reais:** valores numéricos reais positivos e negativos incluindo o 0 (zero) e qualquer valor fracionário;
- **Caracteres:** valores alfanuméricos incluindo símbolos especiais e números, porém, todos se comportando como valores textuais;
- **Lógicos:** valores booleanos que podem armazenar somente dois tipos de resultados e informações lógicas, sendo “verdadeiro” e “falso”.

Indicação de Leitura

Para melhor compreensão desta aula você poderá recorrer ao livro dos Deitel, disponível na biblioteca virtual. O capítulo 7, especificamente, aborda o tema *Arrays* que será apresentado nesta aula.

DEITEL, Harvey M.; DEITEL, Paul J. **Java**: como programar. 8. ed. São Paulo: Pearson, 2010.

Vetores / Arrays Indicação de Vídeo

Indicação de Vídeo

Veja este vídeo sobre *Arrays*. No vídeo é apresentado a estrutura de *Array* similarmente ao apresentado na aula:

Disponível em: <<https://www.youtube.com/watch?v=YdSycMcvY0&t=324s>>. Acesso em: 27 nov. 2020.

Para esta disciplina iremos utilizar as palavras Vetor e *Array* como sinônimos. A literatura, às vezes, utiliza uma ou outra terminologia e a implementação em algumas linguagens de programação podem, da mesma forma, divergir, mas basicamente é a estrutura que abriga um conjunto de dados de forma ordenada.

Um *Array* é o termo em inglês para referenciar um Vetor ou uma Matriz. Se for um *Array* “unidimensional” ele é equivalente a um Vetor, se temos um *Array* “bidimensional” ele é equivalente a uma Matriz.

Figura 1 - Um Array (ou Vetor) de 9 posições

0	1	2	3	4	5	6	7	8

Fonte: elaborado pelo próprio autor (2020)

Figura 2 - Um Array (ou Matriz) com [4,8] posições (4 linhas x 8 colunas)

	0	1	2	3	4	5	6	7
0								
1								
2								
3								

Fonte: elaborado pelo próprio autor (2020)

Perceba que em um Array temos o primeiro elemento referenciado pela posição 0 (ZERO).

[ATENÇÃO] é bastante comum no início o programador se confundir e desconsiderar a posição zero, referente ao primeiro elemento de um Array.

Figura 3 – Erro comum de programação



Erro comum de programação 7.1

Um índice deve ser um valor `int` ou um valor de um tipo que pode ser promovido para `int` — ou seja, `byte`, `short` ou `char`, mas não `long`; caso contrário, ocorrerá um erro de compilação.

Fonte: Deitel (2010)

Vetores de tipos primitivos

A primeira estrutura que vamos trabalhar são os vetores com tipos primitivos de dados. Esta Estrutura permite que tenhamos conjuntos de dados de um mesmo tipo agrupados no Array.

Figura 4 - Conjuntos de dados agrupados no *Array*

The diagram illustrates an array structure. On the left, the text 'Nome do array (c)' has an arrow pointing to the first element 'c[0]'. Below it, the text 'Índice (ou subscripto) do elemento no array c' has an arrow pointing to the index '11' in the last element 'c[11]'. The array elements are listed in a column, each followed by its value in a yellow box. The values are: -45, 6, 0, 72, 1543, -89, 0, 62, -3, 1, 6453, and 78.

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Fonte: Deitel (2010)

Veja alguns exemplos:

Suponha que na sua aplicação (programa) você precise armazenar “marcas de veículos”, poderíamos criar muitas variáveis (ou definir diversas constantes), por exemplo:

Videoaula 1

A primeira videoaula abordará os assuntos introdutórios e a codificação das primeiras classes de trabalho com estrutura de dados. Neste vídeo iremos explorar os conceitos citados para que a estrutura de Vetor/*Array* possa ser compreendida pelo aluno.

Classe desenvolvida: ExemploVetores



Videoaula 1

Utilize o QRcode para assistir!

Neste vídeo iremos explorar os conceitos citados para que a estrutura de Vetor/Array possa ser compreendida pelo aluno.



Ou ainda que você precisasse armazenar a pontuação dos 10 melhores jogadores em um jogo qualquer:

Videoaula 2

Nesta aula, será apresentado um algoritmo trabalhando uma ideia central de Melhores Jogadores. Exemplo de código: **Classe desenvolvida: código da classe MelhoresJogadores**



Videoaula 2

Utilize o QRcode para assistir!

Nesta aula, será apresentado um algoritmo trabalhando uma ideia central de Melhores Jogadores. Exemplo de código: Classe desenvolvida: << código da classe MelhoresJogadores>>



Muito bem, apesar dos códigos apresentados passarem pelo compilador e, se executados, apresentarem as saídas na saída padrão (console por meio do comando `System.out.print()`), imagine que você precise incluir uma nova marca de veículo no primeiro caso ou ainda alterar o placar de algum jogador ou selecionar os “TOP 3 de jogadores”. No primeiro caso teríamos que efetivamente “reescrever” o código inserindo uma nova variável (ou constante) para a nova marca. No segundo caso poderia ser resolvido com a implementação de um método (função) que verificasse todas as variáveis e selecionasse as 3 maiores, armazenando em variáveis auxiliares, e isso pode tornar nossa aplicação extremamente confusa.

Para contornar este tipo de situação, onde temos conjuntos dinâmicos de dados ou informações, as Estruturas de Dados permitem que tenhamos maior controle destas coleções e sua organização ou estruturação facilita e simplifica a utilização deles dentro dos trechos de códigos.

Como já mencionado, esta organização irá permitir que utilizemos funções previamente definidas para, por exemplo, inserir, procurar, ordenar, excluir elementos de forma a otimizar a execução dos programas.

As linguagens de programação podem utilizar formas diferentes de criar ou instanciar uma estrutura do tipo Vetor ou *Array*.

Retomando o exemplo para registro dos melhores resultados obtidos em determinado jogo poderemos, ao invés de criar 10 variáveis distintas como fizemos anteriormente, criar um único vetor que irá armazenar os placares dos jogadores.

(nos códigos usar fonte monoespaçada) `int recorde[]; //`

declaração de um vetor em Java `recorde = new int[10]; //aloca`

dez posições de memória para

`// armazenamento no vetor`

Ou podemos simplificar e declarar o tamanho do vetor de uma só vez.

`int recorde[] = new int[10];`

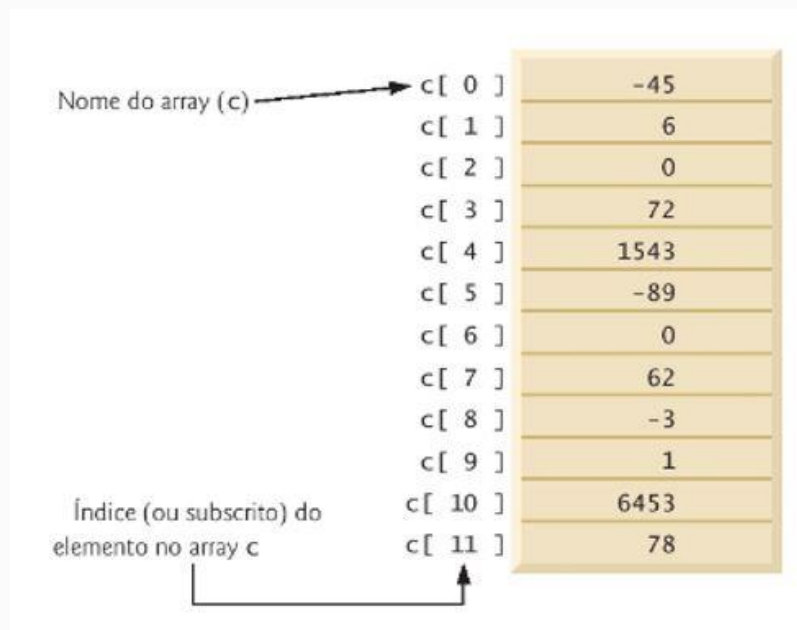
Este comando é equivalente a alocar 10 variáveis diferentes, do tipo inteiro para armazenar valores (similar ao que fizemos na abertura desta aula).

Graficamente, a representação gráfica do vetor seria algo parecido com:

vetor inteiro chamado recorde : <code>int recorde[] = new int[10]</code>									
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Ou

Figura 5 - Representação gráfica do vetor



Fonte: Deitel (2010)

Seguindo este mesmo princípio podemos instanciar outros vetores dos diversos tipos primitivos, veja alguns exemplos:

```
char letras[] = new char[100]; // cria e aloca 100 posições para o tipo char
```

```
double salarios[] = new double[20]; // cria array do tipo double e aloca 20 posições
```

Podemos também definir o conteúdo de cada elemento do *array* em tempo de criação:

```
// cria array já definindo o valor de cada elemento.
```

```
String marcas[] = {"Ford", "Fiat", "GM", "VW", "Honda", "Audi"};
```

Curiosidades

O Java possui dois tipos de dados que são divididos por valor (tipos primitivos) e por referência (tipos por referência).

Os tipos por referência, são classes que especificam os tipos de objeto: *Strings*, *Arrays* Primitivos e Objetos.

Uma variável do tipo primitivo pode armazenar exatamente um valor de seu tipo declarado por vez, quando outro valor for atribuído a essa variável, seu valor inicial será substituído.

As variáveis de instância de tipo primitivo são inicializadas por padrão, as variáveis dos tipos *byte*, *char*, *short*, *int*, *long*, *float* e *double* são inicializadas como 0, e as variáveis do tipo *boolean* são inicializadas como *false*. Esses tipos podem especificar seu próprio valor inicial para uma variável do tipo primitivo, atribuindo a variável um valor na sua declaração.

O Java fornece dois tipos primitivos para armazenar números de ponto flutuante na memória, o tipo *float* e *double*.

A diferença entre eles é que as variáveis *double* podem armazenar números com maior magnitude e mais detalhes, ou seja, armazena mais dígitos à direita do ponto de fração decimal, do que as variáveis *float*. As variáveis do tipo *float* representam números de ponto flutuante de precisão simples e podem representar até 7 dígitos.

As variáveis do tipo *double* representam números de ponto flutuante de precisão dupla, onde precisam duas vezes a quantidade de memória das variáveis *float* fornecendo 15 dígitos, sendo o dobro da precisão de variáveis *float*. Os valores do tipo *double* são conhecidos como literais de ponto flutuante. Para números de ponto flutuante precisos, o Java fornece a classe `BigDecimal` (pacote `java.math`).

Fonte: disponível em: <<https://www.devmedia.com.br/tipos-de-dados-por-valor-e-porreferencia-em-java/25293>>. Acesso em: 14 dez. 2020.

Vamos voltar ao exemplo do nosso placar de jogo e implementar um código em Java que armazene os valores no nosso *Array*.

Videoaula 3

Nesta aula, será apresentado um algoritmo trabalhando o desenvolvimento da classe `Placar.java`. Exemplo de código: **Classe desenvolvida: << código da classe Placar>>**



Videoaula 3

Utilize o QRcode para assistir!

Nesta aula, será apresentado um algoritmo trabalhando o desenvolvimento da classe `Placar.java`. Exemplo de código: Classe desenvolvida: << código da classe Placar>>



Agora que já temos os conceitos básicos de como funciona um *Array*, vamos construir uma pequena aplicação que nos permita:

- Construir um objeto da classe de `Placar` e definir o tamanho do *array* durante a criação;
- Adicionar elementos no *array* na próxima posição vazia OU em uma posição qualquer;
- Retornar uma posição qualquer do *array*;
- Mostrar o tamanho do *array*.

Vamos praticar implementando um *array* que armazene as temperaturas máximas durante 1 semana, cada posição do *array* deverá conter a máxima registrada.

Desafio: utilizando 2 vetores (*arrays*), verificar a temperatura máxima e mínima registrada durante 1 semana. Lembre-se que a função de registro deverá fazer a validação (temperatura mínima não poderá ser maior que a máxima do dia e vice-versa). Utilizar a classe *Random* para gerar temperaturas aleatórias (que sejam válidas, algo entre -2 graus até +45 graus). Mostre o dia com a maior temperatura máxima e com a menor mínima.

Agora que já tivemos uma breve noção de como funciona um *Array*, já é possível imaginar o quanto importante esta estrutura é para o programador. Imagine que você precisa fazer um *select* em um banco de dados ou recuperar uma coleção de dados (ou informações) a partir de uma consulta a bancos não relacionais.

Podemos ter *arrays* de objeto, onde cada objeto do meu vetor (*array*) é na verdade uma outra estrutura de dados em forma de um registro de clientes do banco de dados, por exemplo.

Há uma infinidade de soluções que dependem de estruturas simples como os *Arrays* para dar mais agilidade ao código, consumir menos tempo de processamento e de memória, por exemplo. Nas disciplinas de análise de algoritmos estes conceitos serão fundamentais.

Agora que conseguimos compreender a estrutura básica dos *Arrays*, verificaremos as operações mais simples que geralmente são aplicadas a esta estrutura.

Aula 2 - Arrays Unidimensionais, Multidimensionais e Operações com Arrays

Conceitos iniciais de Algoritmos e Estruturas de Dados

Vamos iniciar nossa segunda aula falando um pouco sobre a definição de algoritmos e as principais ações envolvidas em sua definição.

A palavra algoritmo dentro da lógica de programação tem seu significado associado com a resolução de problemas ou aos passos para chegar em sua resolução. Por definição, um algoritmo é um conjunto determinado de instruções que, quando seguidas, desempenham uma tarefa particular, e dependem da representação e da estrutura de dados. É um pouco complicado entender no início, porém detalhando a definição ela significa que um algoritmo é um conjunto de ações ou tarefas seguidas para resolver um problema proposto. Como o algoritmo será desenvolvido depende das estruturas de dados aplicadas.

Em uma outra definição, “algoritmo são regras formais para a obtenção de um resultado ou da solução de um problema, englobando fórmulas de expressões aritméticas” (MANZANO, 1997, p.6). Existem várias definições de outros autores da área de lógica de programação similares a esta de Manzano, no entanto, em todas elas temos ações envolvidas. São essas ações é que vão dar vida ao algoritmo criado. As ações são de:

- Entradas;
- Processamento;
- Saídas;
- Escopo;
- Eficiência.

Por exemplo, as entradas são os comandos de entrada de dados solicitando informações que o programa precisa para ser iniciado. Logo após, essas informações adentrarem nas variáveis do algoritmo e elas serão processadas. O processamento basicamente está ligado a cálculos ou rotinas de código que não são visíveis ao usuário. Dentro de um algoritmo básico essa linha tem a característica de conter o sinal de atribuição.

Sinal de atribuição em Java = (o caractere “=” recebe o nome de atribuição, ou seja ele atribui valor a uma variável).

Na sequência, as Saídas são informações que o programa, após a entrada e o processamento, exibe na interface para o usuário. Geralmente corresponde a um valor solicitado ou uma pesquisa realizada. Imagine que você está fazendo uma busca de uma pessoa no sistema, a entrada é o nome dessa pessoa, o processamento são as rotinas desempenhadas pelo sistema

para fazer a busca dessa pessoa. Já a saída é o próprio sistema informar se a pessoa foi encontrada resultando nos dados dessa pessoa na tela, ou se a pessoa não for encontrada, uma mensagem dizendo que o sistema não encontrou o nome pesquisado.

Os fatores como escopo e eficiência dizem respeito à regra do algoritmo e a seu funcionamento. O algoritmo desenvolvido tem que ter um escopo, ou seja, início e fim. Isso faz parte das regras de um algoritmo. Quanto à sua eficiência, é simplesmente avaliar se ele está cumprindo o seu propósito. Literalmente, se o algoritmo faz o que foi projetado para fazer.

Indicação de Leitura

Para melhor compreensão deste tópico, você pode estudar o livro do Manzano. O capítulo 1, especificamente, apresenta as definições básicas, diferenciação de nomenclaturas e formas de representação gráfica.

MANZANO, Jose Augusto N. G. Estudo Dirigido: algoritmos. São Paulo: Érica, 1997. 265 p.

Os Algoritmos são responsáveis pela construção de todos os sistemas, desde os mais básicos até os mais complexos. Entender o seu funcionamento é fundamental na criação de um sistema computacional.

No início, quando estamos estudando as características e conceitos, e aprendendo como os algoritmos funcionam, nos deparamos com algumas dificuldades para entender os comandos, manipulação das variáveis e as estruturas básicas dos algoritmos, porém é importante persistir, pois só a prática e a realização de muitos exercícios levará você a uma compreensão mais aprofundada sobre o assunto.

A disciplina de Algoritmos e Lógica de Programação é pré-requisito para a disciplina de Algoritmos e Estruturas de Dados, na qual as estruturas de dados estudadas são mais aprofundadas. Na disciplina inicial aprendemos os conceitos básicos e as estruturas básicas, como já mencionada nos conhecimentos prévios da Aula 1 (Estruturas de seleção e repetição, por exemplo).

No próximo tópico será abordado o conteúdo de *Arrays* Multidimensionais, dando ênfase na criação, manipulação e operações com *Arrays* por meio da linguagem de programação Java.

Arrays Multidimensionais

Esta estrutura requer um pouco mais de habilidade para ser trabalhada dentro de um algoritmo computacional. Como já abordado na Aula 1, desta unidade, o *Array* “unidimensional” é equivalente a um Vetor, se temos um *Array* “multidimensional” ele é equivalente a uma Matriz, ou seja, além da variação no número de colunas existe a variação no número de linhas.

O *Array* unidimensional terá somente o índice das colunas para ser controlado. O que faz o controle desse índice, ou seja, em qual coluna do vetor o conteúdo se encontra chamamos de “apontadores”. São características bem pontuais como pode ser observado nas Figuras abaixo:

Figura 1 - Um Array (ou Vetor) de 9 posições

0	1	2	3	4	5	6	7	8
				15				

Fonte: elaborado pelo próprio autor (2020)

Na Figura 1, o valor ou conteúdo 15 está no índice 4. Sabe-se disso internamente via código, por meio de um apontador. Para que o valor 15 fosse adicionado no índice 4 do *Array*, o apontador está posicionado exatamente neste índice. Observe o código a seguir, apresentado na Figura 2.

Figura 2 – Manipulação de um Array Unidimensional

```
59 public static int[] ordenaMergeSort(){
60
61     int[] vetOrd = new int[9];
62
63     vetOrd[4] = 15;
64
65     for (int i = 0; i <= 8; i++){
66
67         vetOrd[i] = i + 3;
68
69     }
70
71     return vetOrd;
72
73 }
```

Fonte: elaborado pelo próprio autor (2020) via *software* NetBeans

No código apresentado na Figura 2, observamos que na linha 63 o apontador está no índice 4, por isso o valor 15 é atribuído no vetor no índice 4. As linhas 65 a 69 contêm um laço com o apontador, variando em todos os índices do vetor para que ao final do laço ele esteja todo preenchido. A saída para este código seria a apresentação dos valores (**3 4 5 6 7 8 9 10 11**), lado a lado.

Videoaula 1

Agora, assista ao vídeo que aborda a criação de Arrays unidimensionais. No vídeo, você verá como criar um Array e manipular elementos de acordo com o tamanho do Array criado.



Videoaula 1

Utilize o QRcode para assistir!

Agora, assista ao vídeo que aborda a criação de Arrays unidimensionais. No vídeo, você verá como criar um Array e manipular elementos de acordo com o tamanho do Array criado.



O Array multidimensional terá além do índice das colunas para ser controlado, o índice das linhas. Isso mesmo, porque quando o apontador chegar na última coluna da primeira linha, ele deve ser zerado e ao mesmo tempo incrementado o apontador da linha. O que fará com que seja varrido a linha 2 do Array. Isso vale para leitura ou para saída dos elementos do Array. A Figura 3, a seguir, também apresenta o Array Multidimensional também conhecido como Matriz. O exemplo desta estrutura está exibido com 4 linhas e 8 colunas, portanto os índices das linhas vão de 0 a 3 e os índices das colunas vão de 0 a 7.

Figura 3 - Um Array (ou Matriz) com [4,8] posições (4 linhas x 8 colunas)

	0	1	2	3	4	5	6	7
0					15			
1								
2		22						
3								

Fonte: elaborado pelo próprio autor (2020)

Na Figura 3 temos outro cenário, pois agora o Array é Multidimensional. Existem dois valores inseridos no Array e para isso é necessário a informação completa do endereço, linha e coluna, para que o valor seja inserido. Na primeira linha coluna 5 temos o valor 15, porém como sabemos que os índices se iniciam a partir do zero (0), temos que nos atentar para isso. O endereço correto para que o valor 15 seja inserido no Array acima é 0 para linha e 4 para a

coluna, e na sequência, para que o valor 22 seja inserido no *Array* é 2 para linha e 1 para coluna. Observe o código a seguir, apresentado na Figura 4.

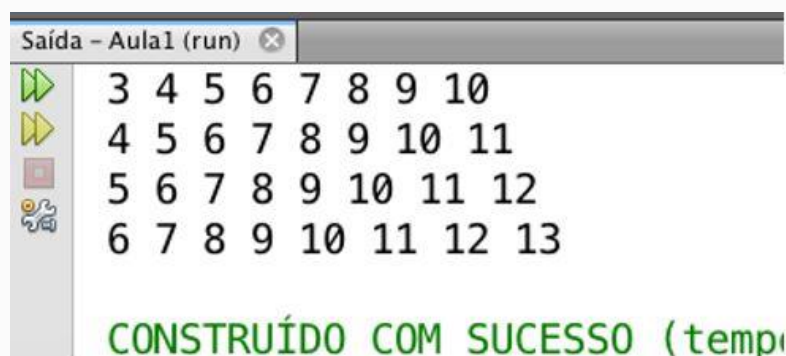
Figura 4 – Manipulação de um *Array* Multidimensional

```
35      matOrd[0][4] = 15;
36      matOrd[2][1] = 22;
37
38      int l, c = 0;
39
40      for (l = 0; l <= 3; l++){
41          for(c = 0; c <= 7; c++){
42              matOrd[l][c] = (c + l) + 3;
43          }
44      }
45
46      for (l = 0; l <= 3; l++){
47          for(c = 0; c <= 7; c++){
48              System.out.print(matOrd[l][c]+ " ");
49          }
50          System.out.println("");
51      }
```

Fonte: elaborada pelo próprio autor (2020) via *software* NetBeans

No código apresentado na Figura 4, observamos que na linha 35 os apontadores estão nos índices 0 e 4, sendo assim, o valor 15 é atribuído na matriz, Figura 3. Por sua vez, na linha 36 os apontadores estão nos índices 2 e 1, portanto, o valor 22 é atribuído também na matriz. A saída para este código pode ser observada em detalhes na execução do programa apresentada na Figura 5.

Figura 5 – Execução do código do *Array*



```
Saída - Aula1 (run)
3 4 5 6 7 8 9 10
4 5 6 7 8 9 10 11
5 6 7 8 9 10 11 12
6 7 8 9 10 11 12 13

CONSTRUÍDO COM SUCESSO (tempo
```

Fonte: elaborada pelo próprio autor (2020) via *software* NetBeans

Veja que os valores apresentados são aleatórios de acordo com o cálculo que é feito na linha 48 da Figura 4. Contudo, as informações estão sendo apresentadas separadas por linha. O Comando responsável por essa quebra de linha é o (`System.out.println(" ")`), apresentado na linha 50 da Figura 5.

Videoaula 2

Agora, assista ao vídeo que aborda a criação de Arrays Multidimensionais. No vídeo, você verá como criar um Array Multidimensional e a manipular elementos de acordo com o tamanho do Array criado, controlando por apontadores das linhas e colunas.



Videoaula 2

Utilize o QRcode para assistir!

No vídeo, você verá como criar um Array Multidimensional e a manipular elementos de acordo com o tamanho do Array criado, controlando por apontadores das linhas e colunas.



Espero, que até este ponto do conteúdo, tenha ficado claro algumas noções básicas sobre o desenvolvimento de Algoritmos com essa estrutura de dados. A partir de agora, veremos alguns exemplos práticos de operações com *Arrays*.

Operações com *Arrays*

É bem comum dentro dos *Arrays* trabalharmos com operações de vários tipos, entre elas se encontram as operações matemáticas, por exemplo. As 4 operações básicas da matemática podem ser utilizadas para praticar um pouco mais sobre a manipulação de elementos dentro de um *Array*. Partindo para um exemplo prático, dentro de um Algoritmo podemos criar 2 *Arrays* Unidimensionais do tipo Inteiro, de 10 elementos cada um. Na sequência, pedimos para que esses *Arrays* sejam preenchidos com informações aleatórias fornecidas pelo próprio usuário. Até aqui teríamos dois *Arrays* de inteiros, com 10 valores aleatórios preenchidos em cada um.

Para o nosso cenário, vamos pensar que o Algoritmo deverá gravar o produto dos dois *Arrays* em um terceiro *Array*. Então para ficar mais claro vamos ilustrar por meio de imagens. A Figura 8 mostra as estruturas A, B e C, respectivamente. A estrutura A e B já estão preenchidas aleatoriamente. O que acontece agora?

O primeiro elemento do *Array* C irá receber o produto (multiplicação) do primeiro elemento do *Array* A vezes o primeiro elemento do *Array* B, na sequência, após a incrementação do apontador, o segundo elemento do *Array* C irá receber o produto do segundo elemento do *Array* A vezes o segundo elemento do *Array* B, e assim, sucessivamente.

Figura 6 – Operações com *Arrays*

```
12      int[] a = new int[]{3,10,1,8,7,2,5,4,9,6};
13      int[] b = new int[]{1,9,7,5,2,8,10,6,4,3};
14      int[] c = new int[10];
15
16      int col = 0;
17
18      for (col = 0; col <= 9; col++){
19          c[col] = (a[col] * b[col]);
20      }
21
22  }
```

Fonte: elaborada pelo próprio autor (2020) via *software* NetBeans

Observe que o código exibido na Figura 6 é bem intuitivo e fácil de ser compreendido. As linhas 12 a 15 se encarregam da criação dos três *Arrays* do nosso cenário. Todos eles do tipo inteiro, pois os resultados propositadamente não serão números fracionários. A linha 16 cria o apontador para controlar os índices do *Array*.

Nas linhas 18 a 22, encontra-se o comando **for**. Comando pertencente as estruturas de laços de repetições que estão fazendo a exibição do *Array* C na tela para o usuário. A exibição dos elementos dos *Arrays* A e B, devidamente multiplicados, estão apresentados na Figura 7.

Figura 7 – Apresentação do resultado do exemplo prático



```
Saída - Aula1 (run) x
run:
3 90 7 40 14 16 50 24 36 18
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Fonte: elaborada pelo autor, 2020 por meio do *software* NetBeans

Videoaula 3

Agora, assista ao vídeo que aborda, na prática, um exemplo de operações com *Arrays*. Na aula será abordado os *Arrays* Unidimensionais para construção e apresentação dos resultados do algoritmo.



Videoaula 3

Utilize o QRcode para assistir!

Agora, assista ao vídeo que aborda, na prática, um exemplo de operações com Arrays!



Após o conteúdo desta Aula, você estará apto a trabalhar com a estrutura *Array*, explorando suas particularidades na busca de implementar algoritmos com novas formas de resolução de problemas utilizando o *Array*.

Indicação de Leitura

Para melhor compreensão do conteúdo abordado nesta aula, você poderá recorrer ao livro de FORBELLONE e EBERSPÄCHER, que está disponível na biblioteca virtual. O capítulo 4 – Estruturas de Dados, especificamente, aborda o tema de *Arrays* que foi abordado nesta aula.

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. **Lógica de programação**. 2. ed. São Paulo: Makron Books, 2000. 195 p.

Curiosidades

O livro de FORBELLONE e EBERSPÄCHER, traz um ótimo aprendizado aos que decidem estudá-lo. Não somente as estruturas unidimensionais e multidimensionais, mas em todo o conteúdo de algoritmos e lógica de programação. Realmente vale a pena estudá-lo com dedicação e trabalhar com muita motivação nos exercícios contidos no livro.

Reforçando, o capítulo 4, apresenta o conceito, a aplicação e a manipulação de vetores e matrizes. Além de abordar o conceito e manipulação de registros. Conteúdo também relevante para a disciplina de Algoritmos e Estrutura de Dados.

Este livro consta na seção de Bibliografia desta Unidade de Ensino. Na mesma vertente o livro dos DEITEL, “Java: como programar”, retrata com profundidade o universo do desenvolvimento por meio de algoritmos, apresentando as diversas estruturas com conceitos, características e exercícios práticos.

Em nossa Biblioteca Digital você vai encontrar estes e demais livros para aprofundar seus conceitos na disciplina de Algoritmos e Estrutura de Dados e outras disciplinas correlatas.

Link para Biblioteca Digital: Disponível em:

<<https://web.unifil.br/pergamum/biblioteca/index.php>>. Acesso em: 14 dez. 2020.

Encerramento

Chegamos ao fim desta Unidade de Ensino após caminhar por conteúdos importantes para o seu desenvolvimento acadêmico até aqui. A Unidade 1, apresentou uma introdução à Estrutura de Dados abordando o *Array* como recurso para a aplicação de seus conteúdos. Foi detalhado processos como a criação de um *Array*, a manipulação de seus elementos e também as operações comuns entre essas estruturas.

Após a assimilação dos conteúdos abordados nesta Unidade, você estará apto a prosseguir estudando as estruturas abordadas nas próximas Unidades de Ensino.

Tenha uma ótima leitura e não se esqueça de ler e estudar a Bibliografia e as aulas gravadas pelo professor.

Obrigado pela oportunidade de ser o mediador entre os conteúdos específicos abordados e o conhecimento adquirido por você por meio da didática e abordagem desta Unidade.

Referências

DEITEL, Harvey M.; DEITEL, Paul J. **Java: como programar**. 8. ed. São Paulo: Pearson, 2010.

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. **Lógica de programação**. 2. ed. São Paulo: Makron Books, 2000. 195 p.

MANZANO, Jose Augusto N. G. **Estudo Dirigido: ALGORITMOS**. São Paulo: Érica, 1997. 265 p.

Esperamos que este guia o tenha ajudado compreender a organização e o funcionamento de seu curso. Outras questões importantes relacionadas ao curso serão disponibilizadas pela coordenação.

Grande abraço e sucesso!

