

ANA CAROLINE VASCONCELLOS

AS AVENTURAS NA TERRA DO CÓDIGO

UM LIVRO DE PROGRAMAÇÃO PARA INICIANTEs

1ª EDIÇÃO

Neste livro, os leitores embarcam em uma jornada emocionante por uma terra mágica e desconhecida - a Terra do Código.

Eu sou Ana Caroline e através de exemplos práticos e exercícios, os leitores aprendem a codificar e desenvolvem habilidades essenciais de programação e assim aprender com mais facilidade uma linguagem de programação. Junte-se a nós em uma jornada emocionante e descubra os segredos da Terra do Código!

Se você é novo no mundo da programação, pode se sentir um pouco perdido ou intimidado no início. No entanto, não se preocupe - este livro é projetado para ser acessível a todos, independentemente do nível de conhecimento técnico. Você será guiado passo a passo através dos conceitos fundamentais da programação, como lógica, estruturas de dados e linguagens de programação, e assim tendo uma base fundamental inicial.

Você descobrirá que a programação pode ser divertida e envolvente, especialmente quando você tem a Terra do Código como seu destino final.

Então, prepare-se para embarcar nesta jornada emocionante. A Terra do Código está esperando por você!

Agradecimentos

Gostaria de expressar minha profunda gratidão a todas as pessoas que tornaram possível a criação deste livro. Primeiramente, agradeço à Deus e a minha família e amigos por seu apoio incondicional e por me incentivarem a seguir em frente, mesmo nos momentos em que parecia impossível.

Agradeço também aos professores e mentores em programação, que compartilharam seu conhecimento e experiência comigo e me inspiraram a desenvolver este projeto.

Não posso deixar de agradecer aos autores de livros e materiais didáticos que me inspiraram e forneceram referências importantes para este livro.

Por fim, gostaria de agradecer a todos que se dedicaram à leitura deste livro, e espero que ele possa ser útil para muitas pessoas que desejam aprender a programar. A todos os leitores, o meu muito obrigado!

Parte 1: Bem-vindo à Terra do Código!

Capítulo 1: O convite para a aventura

Capítulo 2: Preparando a mochila de programador

Capítulo 3: conhecendo a história da programação, os números binários e as portas lógicas

Capítulo 4: Desbravando o ambiente de desenvolvimento

Parte 2: Aprendendo a linguagem da Terra do Código

Capítulo 5: As caixinhas variáveis:

Capítulo 5.1: Desbravando os tipos de variáveis

Capítulo 5.2: A jornada da declaração de variáveis

Capítulo 5.3: A magia da manipulação de variáveis

Capítulo 6: A magia dos operadores:

Capítulo 6.1: Desvendando os operadores aritméticos

Capítulo 6.2: Enfrentando os operadores lógicos

Capítulo 6.3: A saga dos operadores de atribuição

Capítulo 6: A lógica das estruturas de controle:

Capítulo 7.1: Desvendando o Labirinto estruturas de Controle de fluxo

Capítulo 7.2: Explorando a Terra do Código com Laços de Repetição

Capítulo 7.3: Tomando decisões na Terra do Código com Estruturas Condicionais

Capítulo 8: Desafios à frente!

Parte 3: Descobrindo os segredos da Terra do Código

Capítulo 9: A jornada para aprender algoritmos

Capítulo 10: As estruturas de dados que habitam a Terra do Código:

Capítulo 10.1: A tribo dos Arrays

Capítulo 10.2: A lista dos Listas

Capítulo 10.3: O reino das Pilhas

Capítulo 10.4: A Jornada pela Fila Perfeita

Capítulo 10.5: A floresta das Árvores

Capítulo 10.6: O labirinto dos Grafos

Capítulo 10.7: O planalto das Matrizes

Capítulo 11: Desvendando os mistérios da Terra do Código

Parte 4: O desafio final

Capítulo 12: A hora de escolher um caminho

Parte 1

Bem-vindo à Terra do Código!

Capítulo 1: O convite para a aventura



Bem-vindo à Terra do Código, um lugar onde o desconhecido é bem conhecido! Meu nome é Devprodigio, e serei seu guia nesta terra que pode parecer desconhecida para você. Mas não se preocupe, estou aqui para ajudá-lo a aprender a programar e a dominar as habilidades necessárias para criar programas incríveis.

Antes de mergulharmos nas linguagens de programação específicas, é importante entender os conceitos fundamentais de lógica, estruturas de dados e algoritmos. A lógica é a capacidade de resolver problemas de programação, e a habilidade de desmembrar um problema em partes menores e elaborar uma solução coerente é a essência da programação.

As estruturas de dados são utilizadas para organizar informações de forma eficiente. Ao escolher a estrutura de dados correta, você pode garantir que seu programa seja escalável e de alto desempenho.

Os algoritmos são sequências bem definidas de instruções que resolvem um determinado problema. Algoritmos eficientes são essenciais para a criação de programas robustos e de alta performance.

Para se tornar um bom programador, é importante ter um conjunto de habilidades que vão além do conhecimento de uma linguagem de programação específica. Algumas dessas habilidades incluem: lógica e resolução de problemas, conhecimento de matemática básica, capacidade de trabalhar em equipe, compreensão de algoritmos e compreensão de estruturas de dados.

Agora, vamos dar um exemplo simples de como aplicar esses conceitos na prática. Digamos que você queira criar um programa que calcule a média de notas de um aluno. Para fazer isso, você precisaria seguir alguns passos lógicos como:

- Pedir as notas do aluno
- Armazenar as notas em uma estrutura de dados
- Calcular a soma das notas
- Calcular a média das notas e exibir o resultado para o usuário.

Para implementar isso em uma linguagem de programação, você precisaria entender os conceitos de variáveis, estruturas de dados (no caso, um array) e a lógica de soma e divisão. Você também precisaria entender como estruturar o seu código em algoritmos, seguindo os passos lógicos acima.

Com o tempo, à medida que você aprende mais sobre programação, poderá aplicar esses conceitos em problemas mais complexos e desafiadores. A Terra do Código está cheia de aventuras emocionantes, e estou animado para guiá-lo em sua jornada para se tornar um programador habilidoso. Então, vamos começar!

Capítulo 2: Preparando a mochila do programador

Antes de embarcarmos na Terra do Código, é importante entender a história da programação. Desde o desenvolvimento de formas avançadas de cálculo e organização de dados há muitos séculos, até a proposta de Charles Babbage em 1822 de uma máquina diferencial que seria programada por cartões perfurados. Embora essa máquina nunca tenha sido concluída durante a vida de Babbage, ela foi um precursor importante para o desenvolvimento da programação moderna.

Na década de 1930, John von Neumann propôs um novo modelo para a programação de computadores, que se tornou o modelo padrão para a programação moderna, usando um processo chamado "execução sequencial". Durante a Segunda Guerra Mundial, as potências aliadas usaram máquinas de criptografia para decifrar as comunicações dos inimigos, que foram programadas usando um sistema de fiação manual, o que tornou o processo de programação demorado e propenso a erros.

Foi só na década de 1950 que a programação se tornou mais eficiente e acessível com a introdução de linguagens de programação, como o FORTRAN e o COBOL. Desde então, a programação tem evoluído continuamente, com o surgimento de novas linguagens e tecnologias, sendo uma habilidade essencial em muitas áreas da vida, desde a construção de sites e aplicativos até a robótica e a inteligência artificial.

Para embarcar na aventura da Terra do Código, é importante estar preparado e entender a importância da programação em nossa sociedade digital. A programação permite a criação de novas tecnologias, melhoria da eficiência das empresas e organizações, crescimento da economia e também é uma habilidade valiosa para a carreira, permitindo desenvolver novas habilidades e adquirir conhecimentos. Além disso, a programação ensina a pensar de forma lógica e sistemática, uma habilidade importante para resolver problemas em muitas áreas da vida.

Por isso, é fundamental preparar a mochila para a programação com livros, recursos on-line, ferramentas de programação e muita dedicação. Agora que entendemos a história e importância da programação, vamos explorar a Terra do Código!

Capítulo 3: Os números binários e as portas lógicas

Os números binários

Os números binários são um sistema numérico que utiliza apenas dois dígitos, 0 e 1, para representar valores. Esse sistema é fundamental para a programação de computadores, pois a base do funcionamento dos computadores é a manipulação de números binários.

Os números binários são compostos por uma sequência de dígitos binários, cada um deles representando uma potência de 2, começando com a potência 0. Por exemplo, o número binário 1011 representa o valor decimal (base 10) de 11, calculado como:

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 0 + 2 + 1 = 11$$

Ou seja, cada dígito binário indica se a potência correspondente deve ser considerada no cálculo (1) ou não (0).

Os números binários são usados em computadores porque a eletrônica dos computadores trabalha com sinais elétricos que podem estar em dois estados diferentes: ligado ou desligado. Esses estados são representados como 0 e 1, respectivamente, o que significa que a lógica de um computador é construída em torno da manipulação de números binários.

Por exemplo, quando um programa é executado em um computador, as instruções são representadas em forma binária e manipuladas pelo processador. O processador executa operações lógicas e aritméticas nos números binários para realizar as tarefas necessárias. Todos os dados no computador, como arquivos e imagens, também são armazenados como números binários em disco ou em memória.

Em resumo, os números binários são a base fundamental da lógica de programação e funcionamento dos computadores. Entender como eles funcionam é crucial para qualquer pessoa interessada em programação ou no funcionamento interno dos computadores.

Para representar os números binários, é comum utilizar uma tabela conhecida como "Tabela Verdade". Essa tabela lista todas as possíveis combinações de valores para um conjunto de entradas e suas saídas correspondentes. Na tabela verdade para números

binários, as entradas são todas as possíveis combinações de 0 e 1 para cada dígito binário e as saídas são os valores decimais correspondentes a cada combinação.

Por exemplo, a tabela verdade para números binários de 4 dígitos seria:

Dígito 4	Dígito 3	Dígito 2	Dígito 1	Decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Cada linha da tabela mostra uma combinação de dígitos binários e seu valor decimal correspondente. A tabela verdade é útil para a programação de computadores, pois permite que o programador entenda como os números binários são convertidos em valores decimais e vice-versa.

Além disso, a tabela verdade é essencial para o desenvolvimento de circuitos lógicos que operam em valores binários. Esses circuitos são usados em muitas aplicações, como processadores de computador, controladores de tráfego e sistemas de segurança. Através da combinação de diferentes operações lógicas, os circuitos podem realizar tarefas complexas, como realizar cálculos e tomadas de decisões.

Para representar caracteres em forma de números binários, é usada uma tabela conhecida como "Tabela ASCII" (American Standard Code for Information Interchange), composta por 128 caracteres. Cada caractere possui um valor decimal correspondente que pode ser representado em binário. Por exemplo, a letra "A" maiúscula é representada pelo valor decimal 65, que em binário é 01000001. Já a letra "a" minúscula tem o valor decimal 97 e pode ser representada em binário como 01100001. Todos os caracteres da tabela ASCII possuem um valor decimal correspondente, que pode ser convertido em binário com a ajuda da tabela de conversão decimal-binário.

A tabela ASCII é amplamente utilizada na programação para representar caracteres em sistemas de computador, incluindo textos em arquivos, entrada de texto em programas e comunicação de texto entre diferentes sistemas.

Bin	Oct	Dec	Hex	Abrev	Notação com circunflexo	Código escape	Nome
0000 0000	0	0	0	NULL	^@	\0	Nulo (inglês Null)
0000 0001	1	1	1	SOH	^A		Início de cabeçalho (inglês Start of Header)
0000 0010	2	2	2	STX	^B		Início de texto (inglês Start of Text)
0000 0011	3	3	3	ETX	^C		Fim de texto (inglês End of Text)
0000 0100	4	4	4	EOT	^D		Fim de transmissão (inglês End of Transmission)
0000 0101	5	5	5	ENQ	^E		Consulta; inquirição (inglês Enquiry)
0000 0110	6	6	6	ACK	^F		Confirmação (inglês Acknowledge)
0000 0111	7	7	7	BEL	^G	\a	Campainha; sinal sonoro (inglês Bell)
0000 1000	10	8	8	BS	^H	\b	Espaço atrás; retorno de 1 caractere (inglês Back-space)
0000 1001	11	9	9	HT	^I	\t	Tabulação horizontal (inglês Horizontal Tabulation)
0000 1010	12	10	0A	LF	^J	\n	Alimentação de linha; mudança de linha; nova linha (inglês Line Feed)
0000 1011	13	11	0B	VT	^K	\v	Tabulação vertical (inglês Vertical Tabulation)
0000 1100	14	12	0C	FF	^L	\f	Alimentação de formulário (inglês Form Feed)
0000 1101	15	13	0D	CR	^M	\r	Retorno do carro; retorno ao início da linha (inglês Carriage Return)
0000 1110	16	14	0E	SO	^N		Mover para fora; deslocamento para fora (inglês Shift Out)
0000 1111	17	15	0F	SI	^O		Mover para dentro; deslocamento para dentro (inglês Shift In)
0001 0000	20	16	10	DLE	^P		escape do linque de dados; escape de conexão (inglês Data-Link Escape)
0001 0001	21	17	11	DC1	^Q		Controle de dispositivo 1 (inglês Device Control 1)
0001 0010	22	18	12	DC2	^R		Controle de dispositivo 2 (inglês Device Control 2)
0001 0011	23	19	13	DC3	^S		Controle de dispositivo 3 (inglês Device Control 3)
0001 0100	24	20	14	DC4	^T		Controle de dispositivo 4 (inglês Device Control 4)
0001 0101	25	21	15	NAK	^U		Confirmação negativa (inglês Negative-Acknowledge)

0001 0110	26	22	16	SYN	^V		Estado ocioso síncrono; espera síncrona (inglês Synchronous Idle)
0001 0111	27	23	17	ETB	^W		Bloco de fim de transmissão (inglês End of Transmission Block)
0001 1000	30	24	18	CAN	^X		Cancelar (inglês Cancel)
0001 1001	31	25	19	EM	^Y		Fim de mídia; fim do meio (inglês End of Medium)
0001 1010	32	26	1A	SUB	^Z		Substituir (inglês Substitute)
0001 1011	33	27	1B	ESC	^[Escapar (inglês Escape)
0001 1100	34	28	1C	FS	^\		Separador de arquivos (inglês File Separator)
0001 1101	35	29	1D	GS	^]	\e	Separador de grupos (inglês Group Separator)
0001 1110	36	30	1E	RS	^^		Separador de registros (inglês Record Separator)
0001 1111	37	31	1F	US	^_		Separador de unidades (inglês Unit Separator)
0111 1111	177	127	7F	DEL	^?		Deletar (inglês Delete)

Sinais Gráficos (imprimíveis)

Bin	Oct	Dec	Hex	Sinal
0010 0000	40	32	20	(espaço)
0010 0001	41	33	21	!
0010 0010	42	34	22	"
0010 0011	43	35	23	#
0010 0100	44	36	24	\$
0010 0101	45	37	25	%
0010 0110	46	38	26	&
0010 0111	47	39	27	
0010 1000	50	40	28	(
0010 1001	51	41	29)
0010 1010	52	42	2A	*
0010 1011	53	43	2B	=+
0010 1100	54	44	2C	,
0010 1101	55	45	2D	-
0010 1110	56	46	2E	.
0010 1111	57	47	2F	/
0011 0000	60	48	30	0
0011 0001	61	49	31	1
0011 0010	62	50	32	2
0011 0011	63	51	33	3
0011 0100	64	52	34	4

0011 0101	65	53	35	5
0011 0110	66	54	36	6
0011 0111	67	55	37	7
0011 1000	70	56	38	8
0011 1001	71	57	39	9
0011 1010	72	58	3A	:
0011 1011	73	59	3B	;
0011 1100	74	60	3C	<
0011 1101	75	61	3D	=
0011 1110	76	62	3E	>
0011 1111	77	63	3F	?

Sinais Gráficos (imprimíveis)

Bin	Oct	Dec	Hex	Sinal
0100 0000	100	64	40	@
0100 0001	101	65	41	A
0100 0010	102	66	42	B
0100 0011	103	67	43	C
0100 0100	104	68	44	D
0100 0101	105	69	45	E
0100 0110	106	70	46	F
0100 0111	107	71	47	G
0100 1000	110	72	48	H

0100 1001	111	73	49	I
0100 1010	112	74	4A	J
0100 1011	113	75	4B	K
0100 1100	114	76	4C	L
0100 1101	115	77	4D	M
0100 1110	116	78	4E	N
0100 1111	117	79	4F	O
0101 0000	120	80	50	P
0101 0001	121	81	51	Q
0101 0010	122	82	52	R
0101 0011	123	83	53	S
0101 0100	124	84	54	T
0101 0101	125	85	55	U
0101 0110	126	86	56	V
0101 0111	127	87	57	W
0101 1000	130	88	58	X
0101 1001	131	89	59	Y
0101 1010	132	90	5A	Z
0101 1011	133	91	5B	[
0101 1100	134	92	5C	\
0101 1101	135	93	5D]
0101 1110	136	94	5E	^
0101 1111	137	95	5F	_

0110 0011	143	99	63	c
0110 0100	144	100	64	d
0110 0101	145	101	65	e
0110 0110	146	102	66	f
0110 0111	147	103	67	g
0110 1000	150	104	68	h
0110 1001	151	105	69	i
0110 1010	152	106	6A	j
0110 1011	153	107	6B	k
0110 1100	154	108	6C	l
0110 1101	155	109	6D	m
0110 1110	156	110	6E	n
0110 1111	157	111	6F	o
0111 0000	160	112	70	p
0111 0001	161	113	71	q
0111 0010	162	114	72	r
0111 0011	163	115	73	s
0111 0100	164	116	74	t
0111 0101	165	117	75	u
0111 0110	166	118	76	v
0111 0111	167	119	77	w
0111 1000	170	120	78	x
0111 1001	171	121	79	y
0111 1010	172	122	7A	z
0111 1011	173	123	7B	{
0111 1100	174	124	7C	
0111 1101	175	125	7D	}
0111 1110	176	126	7E	~

Sinais Gráficos (imprimíveis)

Bin	Oct	Dec	Hex	Sinal
0110 0000	140	96	60	`
0110 0001	141	97	61	a
0110 0010	142	98	62	b

Apesar disso, a tabela ASCII tem algumas limitações, como o fato de que só pode representar caracteres em inglês e não suporta caracteres de outros idiomas ou símbolos especiais. Por essa razão, outras tabelas de caracteres foram desenvolvidas, como a Unicode, que é capaz de representar uma ampla variedade de caracteres de diferentes idiomas e scripts, incluindo emojis e símbolos especiais.

Com o objetivo de codificar mais caracteres, o código ASCII foi estendido para 8 bits, ou seja, um byte. Esse código é conhecido como "código ASCII estendido" e atribui valores de 0 a 255 para letras maiúsculas e minúsculas, números, marcas de pontuação e outros símbolos. No caso do código iso-latin1, ele também inclui caracteres acentuados. No entanto, é importante notar que esse código não é único e depende da plataforma utilizada.

Vamos a um exemplo simples para simples para transformar uma palavra em binário.
A palavra "AMOR".

1. Procure na tabela ASCII os valores decimais correspondentes a cada letra. Você pode encontrar esses valores em uma tabela ASCII online ou no próprio programa que estiver usando. No caso da palavra "AMOR", os valores decimais correspondentes a cada letra são:

A: 65 M: 77 O: 79 R: 82

2. Em seguida, converta cada valor decimal em binário usando a tabela de conversão decimal-binário que mencionamos anteriormente. Por exemplo, para converter o valor decimal 65 em binário, você pode dividir sucessivamente por 2 e anotar o resto da divisão, começando pelo último resto até chegar ao quociente 0:

$65 / 2 = 32$, resto 1

$32 / 2 = 16$, resto 0

$16 / 2 = 8$, resto 0

$8 / 2 = 4$, resto 0

$4 / 2 = 2$, resto 0

$2 / 2 = 1$, resto 0

$1 / 2 = 0$, resto 1

Então, o valor decimal 65 em binário é 01000001. Repita esse processo para cada valor decimal correspondente a cada letra.

3. Por fim, junte os valores binários para formar a representação em binário da palavra "AMOR". Nesse caso, temos:

A: 01000001 M: 01001101 O: 01001111 R: 01010010

Então, a representação em binário da palavra "AMOR" é 01000001 01001101 01001111 01010010.

Agora deixo um desafio para você. Tente converter seu nome em binário.

Portas lógicas

Imagine que você é um programador de computadores e precisa criar uma aplicação que responda a diferentes entradas de usuários. Por exemplo, quando um usuário clica em um botão, sua aplicação precisa fazer algo em resposta. Para que isso aconteça, você precisa criar uma lógica que decida o que fazer com base na entrada do usuário. É aí que as portas lógicas entram em cena.

As portas lógicas são blocos de construção fundamentais da eletrônica digital e da programação. Elas são responsáveis por realizar operações lógicas entre sinais digitais, ou seja, tomar decisões com base em inputs binários (1 ou 0). Essas portas podem ser usadas para criar circuitos digitais complexos, como microprocessadores, memórias e outros sistemas digitais.

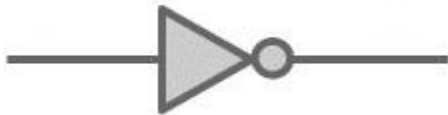
Vamos aprender sobre as principais portas lógicas:

Porta NOT

É uma porta lógica básica que possui apenas uma entrada e uma saída. A saída da porta NOT é o inverso da entrada, ou seja, se a entrada for 0 (baixo), a saída será 1 (alto), e se a entrada for 1 (alto), a saída será 0 (baixo).

A porta NOT é representada por um círculo com uma seta na entrada e outra seta na saída, indicando que a saída é o inverso da entrada. A porta NOT também pode ser representada por um símbolo lógico que pode ser utilizado em diagramas e circuitos digitais.

A porta NOT é frequentemente usada em circuitos digitais para inverter o valor de um sinal lógico. Ela também é usada como um bloco básico para construir outras portas lógicas, como a porta AND, a porta OR e a porta NAND. Além disso, a porta NOT é usada em programação para negar uma expressão booleana, ou seja, transformar o valor true em false e o valor false em true. Em programação, a operação de "negação" lógica pode ser implementada usando o operador "!" (exclamação) em muitas linguagens de programação.

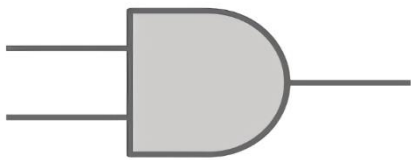


Porta AND

É uma porta lógica básica que possui duas entradas e uma saída. A saída da porta AND é 1 (alto) somente se ambas as entradas forem 1 (alto), caso contrário, a saída será 0 (baixo).

A porta AND é representada por um símbolo lógico que pode ser utilizado em diagramas e circuitos digitais. Esse símbolo consiste em um círculo com a letra "E" dentro, que representa a operação "E" lógica, também conhecida como "conjunção".

A porta AND é frequentemente usada em circuitos digitais para implementar funções booleanas que exigem que duas ou mais condições sejam satisfeitas. Ela também é usada como um bloco básico para construir outras portas lógicas, como a porta OR e a porta NAND. Em programação, a operação "E" lógica pode ser implementada usando o operador "&&" (dois símbolos "e" comerciais) em muitas linguagens de programação.

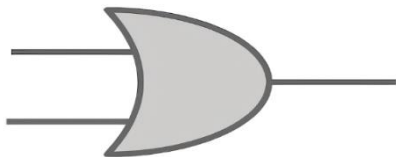


Porta OR

É uma porta lógica básica que possui duas entradas e uma saída. A saída da porta OR é 1 (alto) se pelo menos uma das entradas for 1 (alto), caso contrário, a saída será 0 (baixo).

A porta OR é representada por um símbolo lógico que pode ser utilizado em diagramas e circuitos digitais. Esse símbolo consiste em um círculo com a letra "OU" dentro, que representa a operação "OU" lógica, também conhecida como "disjunção".

A porta OR é frequentemente usada em circuitos digitais para implementar funções booleanas que exigem que pelo menos uma das condições seja satisfeita. Ela também é usada como um bloco básico para construir outras portas lógicas, como a porta AND e a porta NAND. Em programação, a operação "OU" lógica pode ser implementada usando o operador "||" (duplo pipe ou barra vertical) em muitas linguagens de programação.

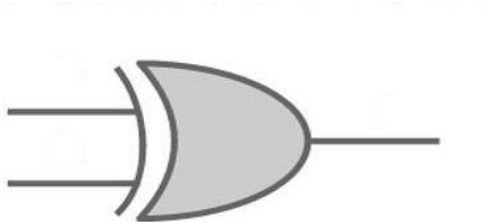


Porta XOR

É uma porta lógica básica que possui duas entradas e uma saída. A saída da porta XOR é 1 (alto) se as entradas forem diferentes entre si, caso contrário, a saída será 0 (baixo).

A porta XOR é representada por um símbolo lógico que pode ser utilizado em diagramas e circuitos digitais. Esse símbolo consiste em um círculo com o símbolo "+" dentro, que representa a operação "OU exclusivo" lógica, também conhecida como "disjunção exclusiva".

A porta XOR é frequentemente usada em circuitos digitais para implementar funções booleanas que exigem que as entradas sejam diferentes. Ela também é usada como um bloco básico para construir outras portas lógicas, como a porta XNOR. Em programação, a operação "OU exclusivo" lógica pode ser implementada usando o operador "^" (símbolo circunflexo) em muitas linguagens de programação.

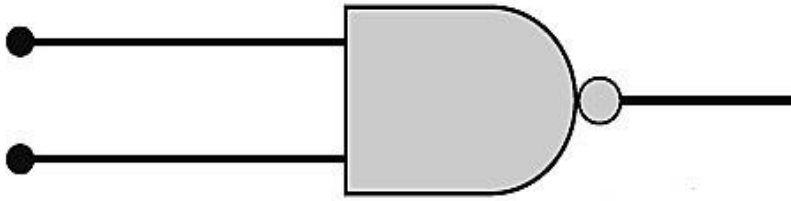


Porta NAND

Uma porta NAND é um tipo de porta lógica usada em circuitos digitais. Ela é uma das portas lógicas universais, o que significa que qualquer circuito digital pode ser construído usando apenas portas NAND.

A porta NAND (NOT-AND) é uma combinação das portas NOT (NOT) e AND (E). Ela possui duas entradas (A e B) e uma saída (Q). A saída Q será alta (1) apenas quando ambas as entradas A e B estiverem baixas (0). Caso contrário, a saída será baixa (0). Em outras palavras, a porta NAND produz o inverso da saída de uma porta AND.

A porta NAND é comumente usada em circuitos digitais para implementar funções booleanas complexas, como somadores, multiplicadores e divisores. Ela também é usada para controlar o fluxo de dados em circuitos de memória e processadores. Além disso, as portas NAND são frequentemente usadas para construir outras portas lógicas, como a porta NOT e a porta OR. Em programação, a operação "não e" é comumente representado como a combinação dos operadores lógicos NOT e AND: `!(a && b)` ou `!a || !b` em muitas linguagens de programação.



Esses são apenas alguns exemplos de como as portas lógicas podem ser aplicadas na programação. As portas lógicas podem ser combinadas para criar circuitos digitais mais complexos, que são utilizados em diversas áreas, como automação industrial, robótica, jogos eletrônicos, entre outras. O conhecimento sobre as portas lógicas é essencial para quem deseja atuar nesses campos e também, é importante para a compreensão básica da eletrônica digital e da programação de computadores.

Tabela verdade

Lá trás usamos a tabela verdade para os binários, mas não expliquei, pois queria dar toda a explicação desse capítulo para sim entendermos mais a tabela.

A tabela verdade é uma ferramenta utilizada para descrever o comportamento de uma porta lógica em relação aos seus inputs. Ela consiste em uma tabela com todas as possíveis combinações de inputs e os respectivos resultados da porta lógica.

As tabelas verdade são úteis para entender o comportamento de cada porta lógica e para combinar várias portas lógicas e criar circuitos mais complexos. Ao utilizar tabelas verdade para combinar portas lógicas, é possível criar um circuito digital que realiza uma operação específica, como um controle de acesso ou um sistema de votação.

A tabela verdade também é útil para verificar o funcionamento de um circuito digital. Ao comparar os resultados esperados com os resultados obtidos, é possível identificar se há algum erro ou problema no circuito.

A seguir, vamos ver na tabela verdade o comportamento de cada porta para compreendermos melhor:

Porta NOT:

Input	Saída
0	1

1	0
---	---

A porta NOT, também conhecida como inversor, inverte o valor lógico da entrada. Ela retorna 1 se a entrada for 0 e retorna 0 se a entrada for 1.

Porta AND:

Input 1	Input 2	Saída
0	0	0
0	1	0
1	0	0
1	1	1

A porta AND retorna 1 apenas se ambos os inputs forem 1. Caso contrário, ela retorna 0.

Porta OR:

Input 1	Input 2	Saída
0	0	0
0	1	1
1	0	1
1	1	1

A porta OR retorna 1 se pelo menos um dos inputs for 1. Ela retorna 0 apenas se ambos os inputs forem 0.

Porta XOR:

Input 1	Input 2	Saída
0	0	0
0	1	1
1	0	1
1	1	0

A porta XOR, ou porta OU exclusivo, retorna 1 apenas se um dos inputs for 1 e o outro for 0. Caso contrário, ela retorna 0.

Porta NAND:

Input 1	Input 2	Saída
0	0	1
0	1	1
1	0	1
1	1	0

A porta NAND, como mencionado anteriormente, é a negação da porta AND. Ela retorna 0 apenas se ambos os inputs forem 1. Caso contrário, ela retorna 1.

Em resumo, a tabela verdade é uma ferramenta fundamental na programação com portas lógicas. Ela ajuda a descrever o comportamento de cada porta lógica e a combinar várias portas para criar circuitos digitais mais complexos.

Capítulo 4: Desbravando o ambiente de desenvolvimento

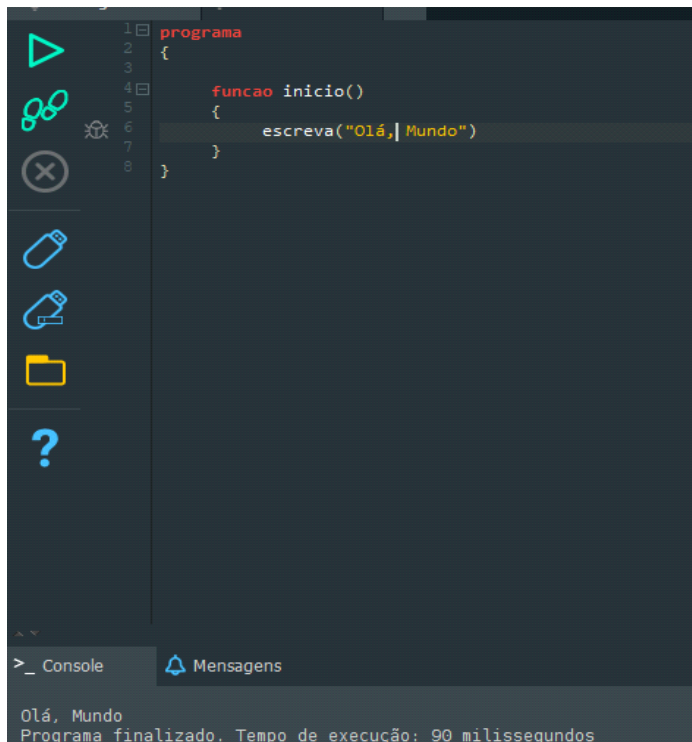
Agora que você preparou a mochila e conheceu um pouco mais sobre a programação, vamos desbravar(explorar) o ambiente de desenvolvimento da nossa terra.

Como você está começando a se aventurar no mundo da programação, é normal ficar um pouco perdido em relação a como começar e quais ferramentas utilizar. Para facilitar sua vida, neste guia vamos desbravar juntos um ambiente de desenvolvimento simples e acessível, para que você possa começar a programar sem dificuldades.

O ambiente de desenvolvimento que utilizaremos é o Portugol Studio, uma ferramenta gratuita e fácil de usar que utiliza uma pseudo-linguagem chamada Portugol, que utiliza comandos em português, tornando o aprendizado de lógica de programação mais acessível e intuitivo para iniciantes.

Para começar, você precisa baixar o instalador do Portugol Studio no site oficial do programa. Para isso, acesse <http://lite.acad.univali.br/portugol/> e clique no botão "Download" para baixar o instalador.

Assim que o download for concluído, execute o instalador e siga as instruções na tela para concluir a instalação do Portugol Studio em seu computador. É importante certificar-se de que a instalação foi concluída com sucesso antes de prosseguir.



Com o Portugol Studio instalado, abra o programa e crie um novo arquivo de código. Na tela principal do Portugol Studio, clique em "Arquivo" e selecione "Novo" para criar um novo arquivo (exemplo na imagem). Por padrão já vem assim, então aperte esse primeiro botão ao lado para executar, e pronto você acabou de executar um código e exibi-lo na tela. Dê um nome para o arquivo e salve-o em uma pasta de sua escolha.

Agora você está pronto para começar a escrever seu código em Portugol! No Portugol Studio, você pode utilizar os comandos de declaração de variáveis, atribuição de valores, entrada e saída, e estruturas de controle de fluxo que foram mencionados no texto anterior. Utilize esses comandos para criar programas simples e experimente diferentes maneiras de utilizá-los para aprimorar suas habilidades de programação.

É importante lembrar que, assim como em qualquer outra habilidade, a prática é fundamental para aprimorar suas habilidades de programação. Por isso, não tenha medo de experimentar e testar diferentes ideias e soluções em seus programas. Com o tempo, você irá se tornar mais confiante em sua capacidade de programar e poderá experimentar outras ferramentas e linguagens de programação para se aventurar ainda mais na terra do código.

Parte 2:

Aprendendo a linguagem da Terra do Código

Capítulo 5: As caixinhas variáveis

As variáveis são elementos importantes na programação, pois permitem que os programas armazenem e manipulem valores. Uma variável é uma caixa na memória do computador que pode armazenar um valor específico.

Capítulo 5.1: Desbravando os tipos de variáveis

Juntos, seguimos pela Terra do Código até chegarmos ao local onde vivem as caixinhas variáveis. Mas você sabe por que elas são chamadas assim? É porque, as variáveis são como pequenas caixinhas que nos permitem armazenar diferentes tipos de valores. Elas são uma ferramenta fundamental na programação, pois permitem criar programas mais dinâmicos e flexíveis.

Existem diferentes tipos de variáveis que podem ser usados para armazenar cada tipo de valor. Alguns dos tipos de variáveis mais comuns são:

- Números inteiros (int): usados para armazenar números inteiros, positivos ou negativos, sem casa decimal.
- Números de ponto flutuante (float): usados para armazenar números com casa decimal, positivos ou negativos.
- Booleanos (bool): usados para armazenar valores verdadeiros ou falsos.
- Cadeias de caracteres (string): usadas para armazenar texto ou sequências de caracteres.
- Listas (list): usadas para armazenar coleções de valores, como uma lista de números ou uma lista de strings.

Capítulo 5.2: A jornada da declaração de variáveis

Antes de utilizar uma variável em um programa, é necessário declará-la. A declaração de uma variável informa ao programa qual o tipo de valor que será armazenado e qual será o nome da variável.

Para declarar uma variável, utiliza-se a sintaxe:

```
tipo nomeDaVariavel;
```

Por exemplo, para declarar uma variável inteira chamada "idade", usamos o seguinte código:

```
inteiro idade;
```

Depois de declarar a variável, é possível atribuir um valor a ela:

```
3  
4     idade = 25;  
5
```

Também é possível fazer a declaração e atribuição de valor em uma única linha:

```
6  
7     inteiro idade = 25;  
8
```

Ao declarar variáveis, é importante escolher nomes significativos que possam ser facilmente identificados no programa. Além disso, é necessário escolher um tipo de variável que suporte o tipo de dado que será armazenado.

Capítulo 5.3: A magia da manipulação de variáveis

Depois de declarar e atribuir valores às variáveis, é possível realizar operações e manipular os valores armazenados. Isso permite que os programas sejam mais dinâmicos e flexíveis.

Por exemplo, é possível realizar operações matemáticas com variáveis, como somar ou subtrair valores:

```

10
11 inteiro a = 10;
12 inteiro b = 5;
13 inteiro resultado = a + b; // resultado será 15
14

```

Também é possível usar as variáveis em condicionais e laços de repetição, permitindo que o programa realize diferentes ações dependendo dos valores armazenados nas variáveis:

```

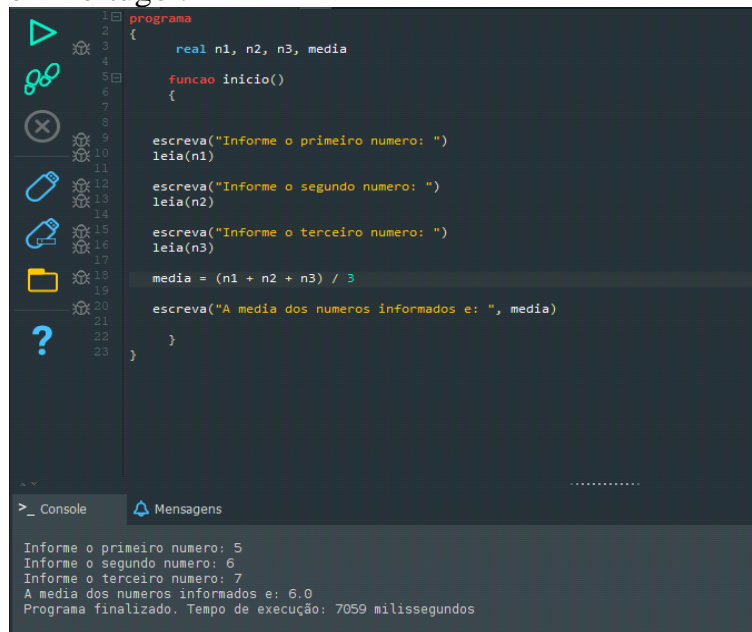
15
16 logico aprovado = verdadeiro;
17 se (aprovado) {
18     escreva("Parabéns, você foi aprovado!");
19 }

```

As variáveis também podem ser atualizadas ao longo do programa, permitindo que os valores armazenados mudem de acordo com a lógica do programa. Aqui estão alguns exemplos práticos de como utilizar variáveis em seus programas, para que você possa se inspirar e colocar em prática o que está aprendendo:

Exemplo 1: Cálculo da média

Suponha que você queira fazer um programa que calcule a média de três números informados pelo usuário. Nesse caso, você pode utilizar três variáveis (uma para cada número) e depois calcular a média utilizando essas variáveis. Veja como ficaria o código em Portugol:

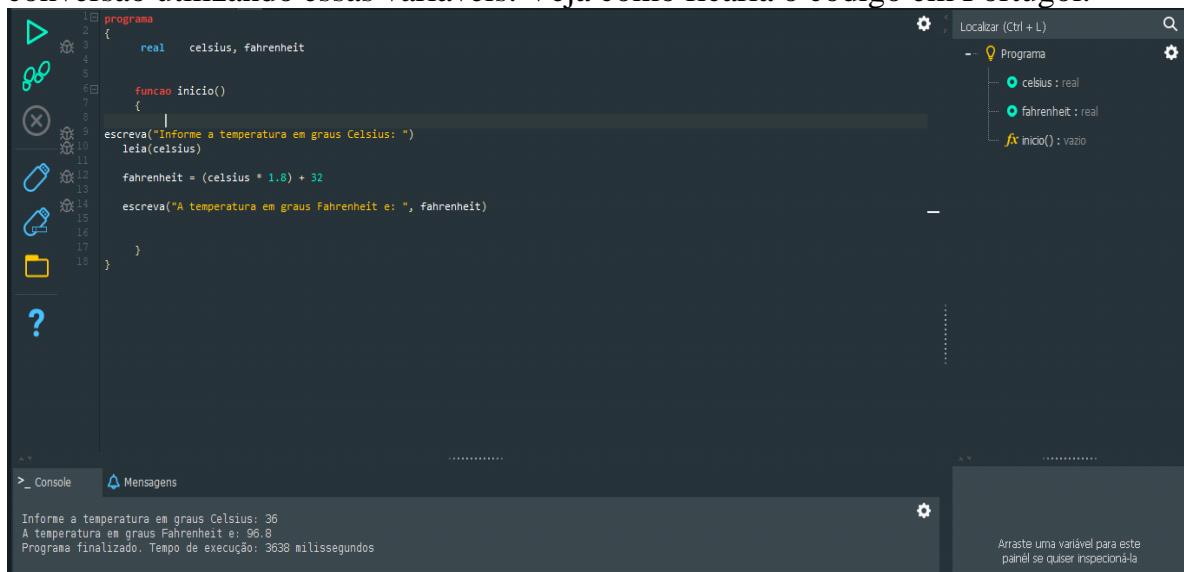


**Para se declarar a variável, não é obrigatório usar a palavra-chave "var".*

Nesse exemplo, as variáveis n1, n2 e n3 são utilizadas para armazenar os números informados pelo usuário, enquanto a variável média é utilizada para armazenar o resultado do cálculo da média.

Exemplo 2: Conversão de temperatura

Suponha que você queira fazer um programa que converta uma temperatura em graus Celsius para Fahrenheit. Nesse caso, você pode utilizar duas variáveis (uma para a temperatura em Celsius e outra para a temperatura em Fahrenheit) e depois realizar a conversão utilizando essas variáveis. Veja como ficaria o código em Portugol:



```
1 programa
2 {
3     real    celsius, fahrenheit
4
5     funcao inicio()
6     {
7         escreva("Informe a temperatura em graus Celsius: ")
8         leia(celsius)
9
10        fahrenheit = (celsius * 1.8) + 32
11
12        escreva("A temperatura em graus Fahrenheit e: ", fahrenheit)
13    }
14 }
15
```

Console Output:

```
Informe a temperatura em graus Celsius: 36
A temperatura em graus Fahrenheit e: 96.8
Programa finalizado. Tempo de execução: 3638 milissegundos
```

**Vou explicar algumas coisas do Portugol Studio. Você ta vendo la em cima a palavra "programa", então, em Portugol é uma representação de um programa de computador que será executado para realizar determinada tarefa. Ele é composto por um conjunto de instruções que serão interpretadas pelo computador para realizar a tarefa desejada. Já a função é um bloco de código que realiza uma tarefa específica e pode ser chamada a partir do programa principal ou de outras funções. A função pode receber parâmetros e retornar valores.*

No exemplo que você mencionou, o programa possui uma função principal chamada "início" que é responsável por executar o código quando o programa é iniciado. Dentro dessa função, há instruções para ler um valor em graus Celsius a partir do teclado, realizar o cálculo para converter essa temperatura para Fahrenheit e, por fim, exibir o resultado na tela. As variáveis "celsius" e "fahrenheit" são declaradas como reais para armazenar os valores de temperatura em graus Celsius e Fahrenheit, respectivamente.

Capítulo 5: A magia dos operadores:

No capítulo anterior, aprendemos como declarar variáveis em Portugol. Agora, vamos dar um mergulho mais profundo no mundo da programação e descobrir as "magias" por trás dos operadores aritméticos. Estes operadores nos permitem realizar operações matemáticas, como adição, subtração, multiplicação e divisão e módulo, comparando valores, verificar igualdades e tomar decisões.

Capítulo 6.1: Desvendando os operadores aritméticos

Desbravando os operadores aritméticos, é importante conhecer as ferramentas disponíveis para realizar operações matemáticas em programas. Entre as principais ferramentas estão os operadores aritméticos, que possibilitam a realização de diversas operações matemáticas de forma simples e eficiente.

+ (adição) - usado para adicionar valores

- (subtração) - usado para subtrair valores

* (multiplicação) - usado para multiplicar valores

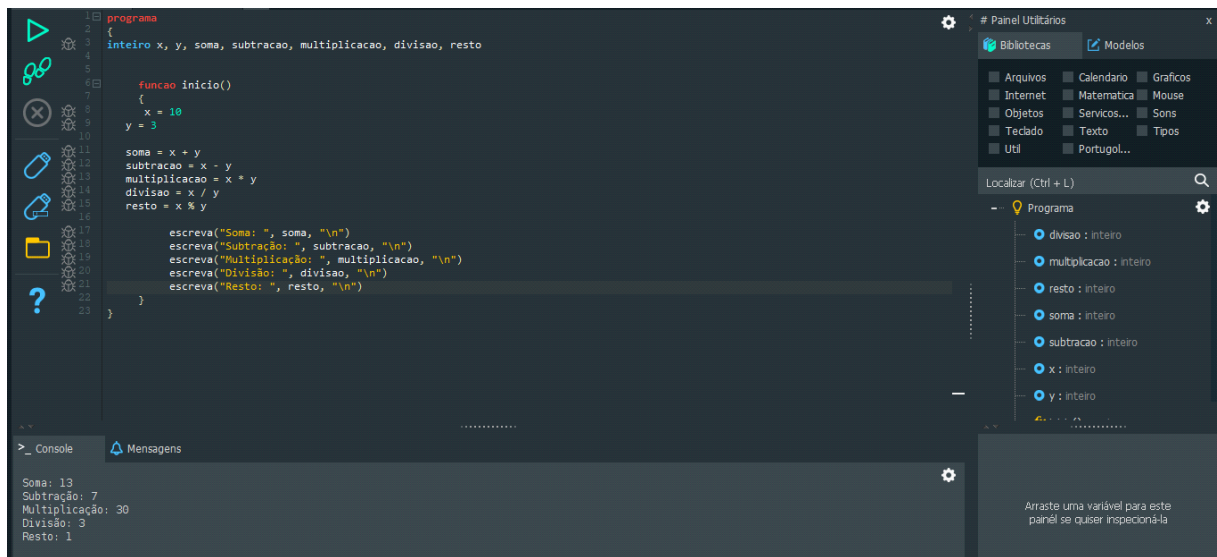
/ (divisão) - usado para dividir valores

% (módulo) - usado para retornar o resto da divisão de um número por outro

Por exemplo, imagine que você esteja criando um programa de cálculo de salário de funcionários. Utilizando os operadores aritméticos, seria possível realizar cálculos de adição para somar os salários dos funcionários, subtração para descontar impostos, multiplicação para calcular o valor de horas extras e divisão para obter a média salarial da equipe.

Em resumo, ao desbravar os operadores aritméticos, você amplia suas habilidades para lidar com operações matemáticas em programas, possibilitando a criação de soluções mais eficientes e automatizadas.

Vamos praticar:



Esse é um exemplo de programa em Portugol que utiliza operadores aritméticos para realizar operações matemáticas básicas em duas variáveis, no caso, x e y.

O programa começa declarando as variáveis inteiras x, y, soma, subtração, multiplicação, divisão e resto. Em seguida, dentro da função "início", as variáveis x e y recebem os valores 10 e 3, respectivamente.

A partir daí, o programa utiliza operadores aritméticos para realizar operações com as variáveis x e y. A soma é realizada utilizando o operador "+", a subtração é realizada utilizando o operador "-", a multiplicação é realizada utilizando o operador "*", e a divisão é realizada utilizando o operador "/". O operador "%" é utilizado para calcular o resto da divisão entre x e y.

Por fim, o programa utiliza o comando "escreva" para imprimir na tela o resultado de cada uma das operações realizadas.

Capítulo 6.2: Enfrentando os operadores lógicos

Enfrentando os operadores lógicos, é importante saber como utilizar essas ferramentas para realizar comparações entre valores em programas. Ao utilizar os operadores lógicos, é possível comparar valores e retornar um resultado lógico, que pode ser utilizado para tomar decisões no código.

== (igual a) - usado para verificar se dois valores são iguais

!= (diferente de) - usado para verificar se dois valores são diferentes

> (maior que) - usado para verificar se um valor é maior que outro

< (menor que) - usado para verificar se um valor é menor que outro

>= (maior ou igual a) - usado para verificar se um valor é maior ou igual a outro

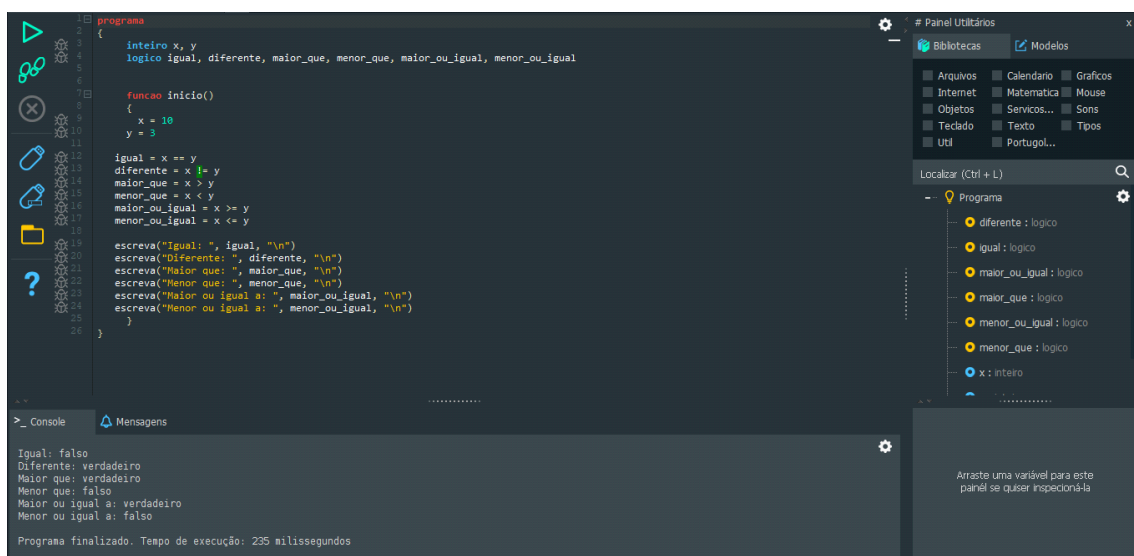
<= (menor ou igual a) - usado para verificar se um valor é menor ou igual a outro

Por exemplo, imagine que você esteja desenvolvendo um programa de login. Para verificar se o usuário inseriu as credenciais corretas, você pode utilizar o operador de igualdade (==) para comparar o nome de usuário e a senha que foram digitados com as informações armazenadas no banco de dados.

Outra situação em que os operadores lógicos são úteis é na validação de formulários. Ao criar um formulário em um site, é comum utilizar operadores lógicos para verificar se os campos foram preenchidos corretamente e se as informações fornecidas são válidas. Por exemplo, ao criar um campo para inserir um endereço de e-mail, você pode utilizar o operador de expressão regular para verificar se o endereço de e-mail inserido é válido.

Em resumo, enfrentar os operadores lógicos é fundamental para realizar comparações entre valores em programas e tomar decisões com base nos resultados obtidos. Saber como utilizar essas ferramentas é essencial para criar programas mais robustos e eficientes.

Vamos praticar:



```
1 programa
2 {
3     inteiro x, y
4     logico igual, diferente, maior_que, menor_que, maior_ou_igual, menor_ou_igual
5
6
7     funcao inicio()
8     {
9         x = 10
10        y = 3
11
12        igual = x == y
13        diferente = x != y
14        maior_que = x > y
15        menor_que = x < y
16        maior_ou_igual = x >= y
17        menor_ou_igual = x <= y
18
19        escreva("Igual: ", igual, "\n")
20        escreva("Diferente: ", diferente, "\n")
21        escreva("Maior que: ", maior_que, "\n")
22        escreva("Menor que: ", menor_que, "\n")
23        escreva("Maior ou igual a: ", maior_ou_igual, "\n")
24        escreva("Menor ou igual a: ", menor_ou_igual, "\n")
25    }
26 }
```

Console Output:

```
Igual: falso
Diferente: verdadeiro
Maior que: verdadeiro
Menor que: falso
Maior ou igual a: verdadeiro
Menor ou igual a: falso
Programa finalizado. Tempo de execução: 235 milissegundos
```

Esse é um exemplo de programa em Portugol que utiliza operadores relacionais para comparar valores de variáveis.

O programa começa declarando as variáveis inteiras x e y, e as variáveis lógicas igual, diferente, maior_que, menor_que, maior_ou_igual e menor_ou_igual.

Dentro da função "início", as variáveis x e y recebem os valores 10 e 3, respectivamente. A partir daí, o programa utiliza operadores relacionais para comparar os valores das variáveis.

O operador "==" é utilizado para verificar se x é igual a y. O operador "!=" é utilizado para verificar se x é diferente de y. O operador ">" é utilizado para verificar se x é maior que y. O operador "<" é utilizado para verificar se x é menor que y. O operador ">=" é utilizado para verificar se x é maior ou igual a y. O operador "<=" é utilizado para verificar se x é menor ou igual a y.

Por fim, o programa utiliza o comando "escreva" para imprimir na tela o resultado de cada uma das comparações realizadas, que serão valores lógicos "verdadeiro" ou "falso".

Capítulo 6.3: A saga dos operadores de atribuição

A saga dos operadores de atribuição é uma parte essencial da programação, já que esses operadores são utilizados para atribuir valores a variáveis em nossos programas. Os operadores de atribuição permitem que possamos armazenar e manipular dados em nossos programas de forma mais eficiente e organizada.

= (atribuição simples) - usado para atribuir um valor a uma variável

+= (atribuição de adição) - usado para adicionar um valor a uma variável existente

-= (atribuição de subtração) - usado para subtrair um valor de uma variável existente

*= (atribuição de multiplicação) - usado para multiplicar uma variável existente por um valor

/= (atribuição de divisão) - usado para dividir uma variável existente por um valor

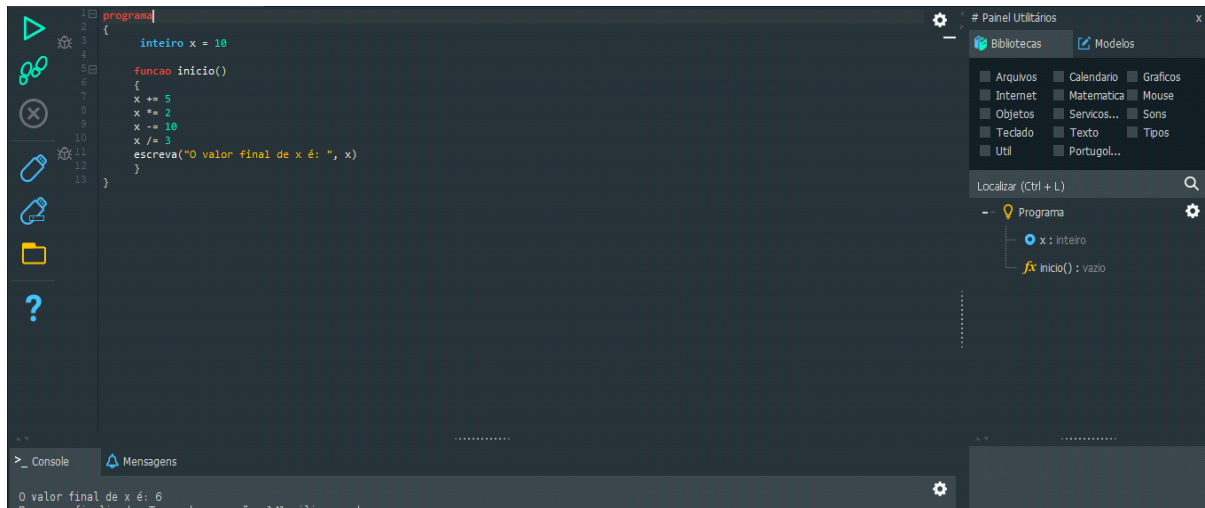
Por exemplo, imagine que você esteja criando um programa de cálculo de juros compostos. Utilizando o operador de atribuição simples (=), você pode atribuir o valor inicial da aplicação a uma variável, como por exemplo: "valor_inicial = 1000". A partir desse ponto, é possível utilizar os outros operadores de atribuição para atualizar o valor dessa variável com base nas taxas de juros e no tempo decorrido.

Outro exemplo de utilização dos operadores de atribuição é na concatenação de strings. Ao utilizar o operador de adição e atribuição (+=), é possível concatenar uma string existente com outra string, criando assim, uma nova string com os valores combinados.

** Uma string é uma sequência de caracteres, como letras, números, símbolos de pontuação e espaços, que é usada para representar texto em programas de computador. Strings são usadas para armazenar e manipular dados de texto em muitos tipos de programas, desde aplicativos de processamento de texto até sistemas de banco de dados.*

Em resumo, a saga dos operadores de atribuição é fundamental para armazenar e manipular dados em programas. Saber como utilizar essas ferramentas é essencial para criar programas mais eficientes e organizados.

Vamos praticar:



```
1 | programa
2 | {
3 |     inteiro x = 10
4 |
5 |     funcao inicio()
6 |     {
7 |         x += 5
8 |         x *= 2
9 |         x -= 10
10 |        x /= 3
11 |        escreva("O valor final de x é: ", x)
12 |    }
13 | }
```

Localizar (Ctrl + L)

- Programa
 - x : inteiro
 - inicio() : vazio

Console

O valor final de x é: 6

Neste exemplo, a variável x é inicializada com o valor 10. Em seguida, o operador de atribuição de adição é usado para adicionar 5 a x. Depois, o operador de atribuição de multiplicação é usado para multiplicar x por 2. Em seguida, o operador de atribuição de subtração é usado para subtrair 10 de x. Finalmente, o operador de atribuição de divisão é usado para dividir x por 3. O valor final de x será 6.

A linha escreva ("O valor final de x é: ", x) é usada para exibir o valor final da variável x na tela. O resultado esperado será: "O valor final de x é: 6".

Capítulo 7: A lógica das estruturas de controle

Neste capítulo, vamos explorar a lógica das estruturas de controle na programação, com exemplos práticos em Portugol. Vamos ver como essas estruturas podem nos ajudar a resolver problemas lógicos e executar tarefas de forma mais eficiente.

Capítulo 7.1: Desvendando o Labirinto estruturas de Controle de fluxo

Bem-vindo ao labirinto das estruturas de controle de fluxo! Aqui, você irá explorar os diversos caminhos que podem ser tomados para controlar o fluxo de um programa. Você irá desvendar o caminho das estruturas de repetição, também conhecidas como laços.

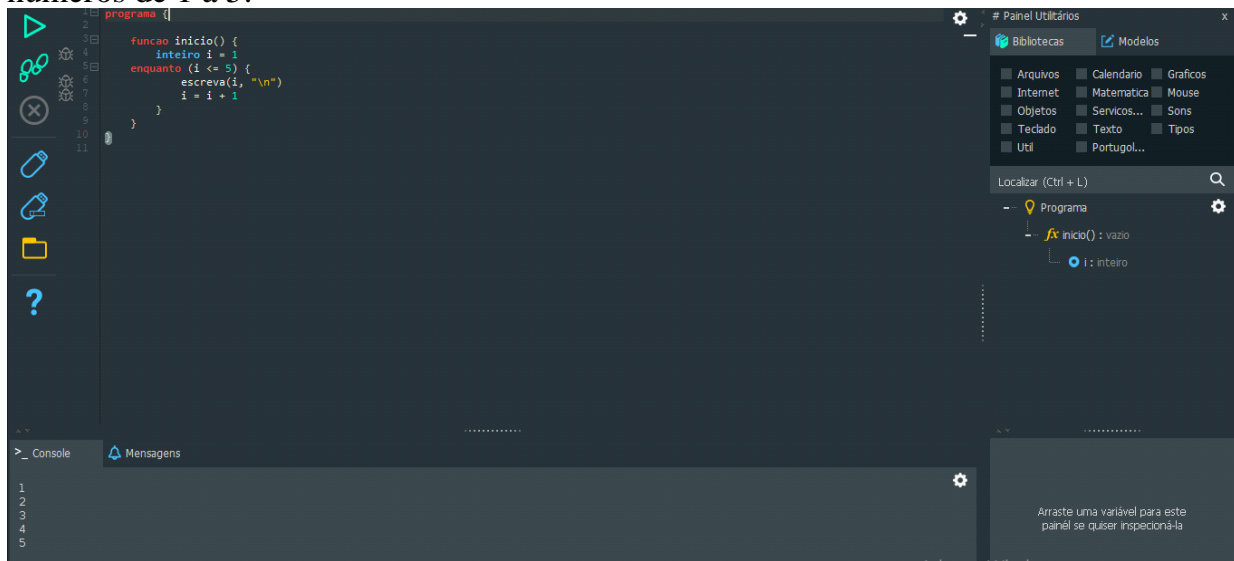
Ao navegar nesse labirinto, você irá descobrir como as estruturas de repetição podem ajudá-lo a executar uma parte do código várias vezes, sem precisar repeti-lo manualmente. Esses laços são muito úteis em situações em que é necessário executar uma determinada tarefa várias vezes, como na leitura de uma lista de itens ou na impressão de uma mensagem para muitos usuários.

As estruturas de repetição são um tipo de estrutura de controle de fluxo que permite modificar o fluxo do programa com base em condições específicas. Ao dominar essas estruturas, você estará mais apto a resolver os desafios do labirinto das estruturas de controle de fluxo. Pronto para se aventurar nesse caminho? Vamos lá!

Existem três tipos principais de estruturas de repetição: "for" (para), "while" (enquanto) e "do-while" (faça-enquanto).

Exemplo prático: Laço "for" (para): O laço "for" nos permitem repetir um bloco de código um número específico de vezes. No exemplo abaixo, o programa exibe os

números de 1 a 5:



Este é um código em Linguagem Algorítmica, que é uma linguagem de programação de alto nível utilizada para ensinar conceitos de programação.

O programa consiste em um bloco de código que define uma função "início", que é o ponto de partida do programa. Dentro dessa função, a primeira linha define uma variável inteira "i" e atribui o valor 1 a ela.

Em seguida, o programa entra em um laço de repetição "enquanto" que verifica se o valor da variável "i" é menor ou igual a 5. Se essa condição for verdadeira, o bloco de comandos dentro do laço é executado. Caso contrário, o laço é interrompido e o programa continua a execução.

Dentro do laço, o programa chama a função "escreva" passando o valor da variável "i" como argumento, seguido do caractere especial "\n", que representa uma quebra de linha. Em seguida, o valor da variável "i" é incrementado em 1 usando o operador "+=". Esse processo se repete até que o valor de "i" seja maior que 5, momento em que o laço é interrompido.

Em resumo, o código exibe na tela os números de 1 a 5 em linhas separadas. Cada número é exibido usando a função "escreva" e, em seguida, a variável "i" é incrementada para que o próximo número seja exibido.

Capítulo 7.2: Explorando a Terra do Código com Laços de Repetição

Aqui, onde os laços de repetição são fundamentais para explorar e iterar sobre diferentes coleções de dados! Nesta terra, você irá descobrir como as estruturas de repetição, ou laços de repetição, são usadas para executar um bloco de código várias vezes, tornando mais fácil a execução de tarefas repetitivas e a manipulação de grandes conjuntos de dados.

Ao explorar a Terra do Código, você descobrirá que os laços de repetição são comumente utilizados em muitas linguagens de programação para percorrer sequências de dados, como listas ou matrizes, ou para realizar tarefas que precisam ser executadas repetidamente, como imprimir uma mensagem várias vezes. Além disso, os laços de repetição também podem ser utilizados para controlar o fluxo do programa com base em condições específicas.

Com os laços de repetição, é possível explorar e manipular grandes conjuntos de dados com mais facilidade, permitindo a automação de tarefas repetitivas e a execução de tarefas complexas de forma mais eficiente. Portanto, se você deseja se aventurar na Terra do Código, é fundamental dominar os laços de repetição para navegar por esse mundo repleto de desafios e aventuras. Vamos começar?

Existem três tipos principais de laços de repetição: o "para" (ou "for"), o "enquanto" (ou "while") e o "faça-enquanto" (ou "do-while").

O laço "para" é usado quando sabemos quantas vezes queremos executar um bloco de código. Nós especificamos um valor inicial, um valor final e um incremento e o bloco de código é executado para cada valor entre o valor inicial e final.

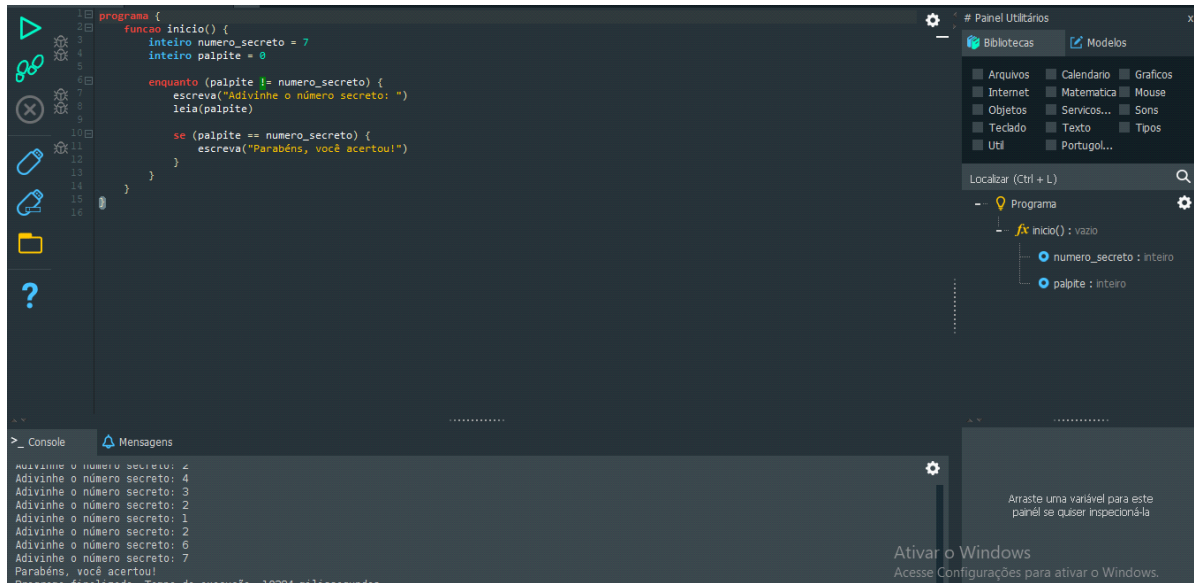
O laço "enquanto" é usado quando queremos executar um bloco de código enquanto uma determinada condição for verdadeira. O bloco de código é executado repetidamente até que a condição se torne falsa.

O laço "faça-enquanto" é semelhante ao "enquanto", mas a condição é verificada após a primeira execução do bloco de código. Isso significa que o bloco de código é sempre executado pelo menos uma vez.

Ao usar estruturas de repetição, é importante garantir que a condição de saída seja alcançada em algum momento, caso contrário, o programa entrará em um loop infinito. Também é importante considerar a eficiência do código ao escolher qual tipo de laço usar, especialmente se estivermos lidando com grandes quantidades de dados.

Exemplo: Laço "while" (enquanto)

O laço "while" nos permite repetir um bloco de código enquanto uma condição específica for verdadeira. No exemplo abaixo, o programa pede ao usuário para adivinhar um número e continua pedindo até que o usuário acerte:



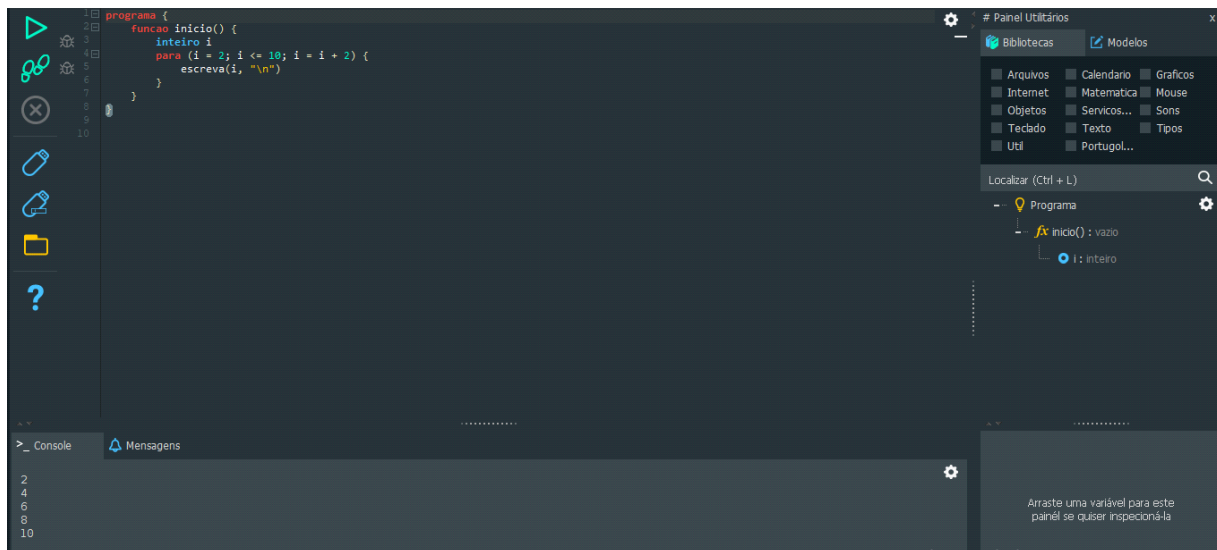
Este código é um jogo simples de adivinhação de números. A ideia é que o programa escolhe um número secreto (neste caso, o número 7) e pede que o usuário tente adivinhar qual é esse número.

O programa começa pedindo ao usuário que insira um palpite. Se o palpite for igual ao número secreto, o programa exibe a mensagem "Parabéns, você acertou!" e o jogo termina. Caso contrário, o programa solicita que o usuário tente novamente, até que ele acerte.

Para fazer isso, o programa utiliza um laço "enquanto". Enquanto o palpite do usuário não for igual ao número secreto, o programa continuará solicitando que o usuário insira um novo palpite. Quando o palpite correto for inserido, o programa sairá do laço "enquanto" e exibirá a mensagem de vitória.

Resumindo, este programa é um jogo simples que utiliza um laço "enquanto" para solicitar que o usuário adivinhe um número secreto. Quando o usuário acerta, o jogo termina.

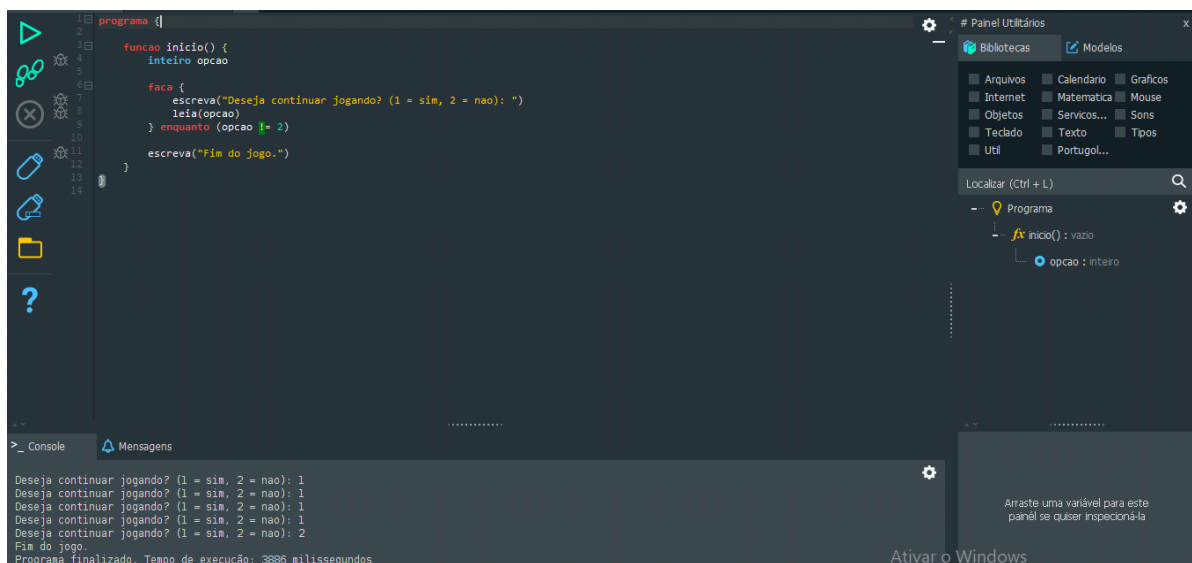
Exemplo: for(para)



Neste programa, a variável "i" é inicializada com o valor 2. A condição de execução do laço é que o valor de "i" seja menor ou igual a 10. E a cada iteração, a variável "i" é incrementada em 2, de forma a exibir somente números pares. A cada iteração do laço, o número é impresso na tela usando o comando "escreva".

Exemplo: do-while(faça-enquanto)

O comando "faça-enquanto" é uma estrutura de repetição que executa um bloco de código pelo menos uma vez e continua a repeti-lo enquanto uma determinada condição for verdadeira.



Neste exemplo, o bloco de código pergunta ao usuário se ele deseja continuar jogando, e só para de perguntar quando o usuário digitar 2 (para não continuar). O programa continuará a executar o loop até que a condição seja falsa e saia do loop.

Capítulo 7.3: Tomando decisões na Terra do Código com Estruturas

Já ia me esquecendo. Aqui também é importante saber que tomada de decisões é essencial para navegar por esse mundo cheio de desafios! Aqui, você irá descobrir como as estruturas condicionais são usadas para tomar decisões em seu programa com base em determinadas condições, tornando mais fácil a execução de um bloco de código apenas se uma condição específica for atendida.

Ao explorar a Terra do Código, você irá descobrir que as estruturas condicionais são fundamentais em muitas linguagens de programação para controlar o fluxo do programa com base em diferentes condições. Essas condições podem ser expressões booleanas, variáveis, expressões aritméticas, ou qualquer outra coisa que possa ser avaliada como verdadeira ou falsa.

Com as estruturas condicionais, é possível tomar decisões complexas em seu código e torná-lo mais eficiente e preciso. Você pode executar diferentes blocos de código dependendo de diferentes condições e tornar seu programa mais flexível e adaptável às necessidades do usuário.

Portanto, se você deseja navegar pela Terra do Código com segurança, é fundamental dominar as estruturas condicionais para tomar decisões precisas em seu programa. Está pronto para explorar essa terra repleta de desafios? Vamos começar a tomar decisões na Terra do Código com estruturas condicionais!

Existem três tipos principais de estruturas condicionais: "if" (se), "else if" (senão se) e "else" (senão).

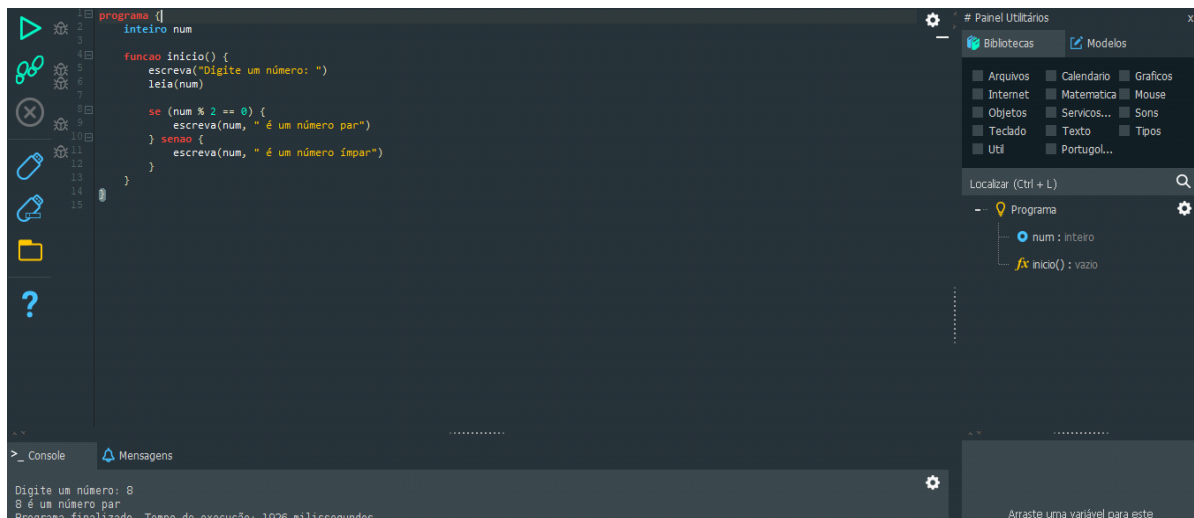
Exemplo prático: Estrutura condicional "if" (se)

A estrutura condicional "if" nos permite executar um bloco de código apenas se uma determinada condição for atendida. No exemplo abaixo, o programa verifica se o número digitado pelo usuário é maior que 10:

Exemplo: If-else (Se-então-senão)

O comando If-else nos permite executar um bloco de código somente se uma determinada condição for atendida. Se a condição não for atendida, o bloco de código do "else" será executado.

Veja o exemplo abaixo, onde o programa verifica se o número digitado pelo usuário é par ou ímpar:



```
1 programa {
2   inteiro num
3
4   funcao inicio() {
5     escreva("Digite um número: ")
6     leia(num)
7
8     se (num % 2 == 0) {
9       escreva(num, " é um número par")
10    } senao {
11      escreva(num, " é um número ímpar")
12    }
13  }
14 }
15 }
```

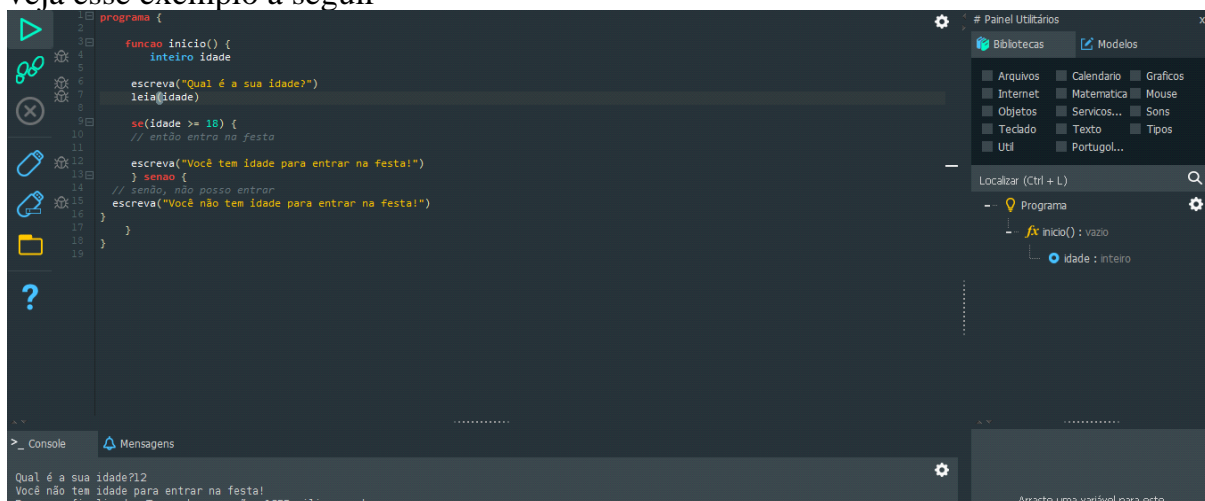
Console: Digite um número: 8
8 é um número par
Programa finalizado. Tempo de execução: 1026 milissegundos

Exemplo: If(se) e else (então)

Em programação, o if e o else são estruturas de controle de fluxo que permitem que um programa execute diferentes blocos de código com base em uma condição. Em outras palavras, se a condição for verdadeira, um bloco de código será executado, caso contrário, outro bloco de código será executado.

Em termos de lógica de programação, o if é uma estrutura de controle condicional que avalia uma expressão lógica (ou condição) e executa o bloco de código dentro do if somente se a condição for verdadeira.

veja esse exemplo a seguir



```
1 programa {
2   funcao inicio() {
3     inteiro idade
4
5     escreva("Qual é a sua idade?")
6     leia(idade)
7
8     se(idade >= 18) {
9       // então entra na festa
10      escreva("Você tem idade para entrar na festa!")
11    } senao {
12      // senão, não posso entrar
13      escreva("Você não tem idade para entrar na festa!")
14    }
15  }
16 }
17 }
18 }
19 }
```

Console: Qual é a sua idade?12
Você não tem idade para entrar na festa!
Programa finalizado. Tempo de execução: 1855 milissegundos

Nesse exemplo o **inteiro idade** declara uma variável chamada "idade" do tipo inteiro. Esta variável será usada para armazenar a idade digitada pelo usuário.

escreva ("Qual é a sua idade?") imprime a mensagem "Qual é a sua idade?" na tela.

leia (idade) lê um valor digitado pelo usuário e armazena-o na variável "idade".

se (idade >= 18) {...} é uma estrutura condicional. Ele verifica se a idade é maior ou igual a 18. Se for, o código dentro das chaves será executado. Caso contrário, o código dentro do bloco "senão" será executado.

escreva ("Você tem idade para entrar na festa!") imprime a mensagem "Você tem idade para entrar na festa!" na tela, caso a idade do usuário seja maior ou igual a 18.

senao {...} é um bloco de código opcional que só será executado se a condição do "se" não for atendida (ou seja, se a idade for menor que 18).

** O "então" é uma palavra que geralmente é usada em pseudocódigos ou em linguagens de programação estruturadas, como o Pascal. No Portugol, que é uma linguagem de programação mais simples e voltada para o ensino de programação, a palavra-chave "então" não é necessária para estruturas condicionais.*

Em vez disso, o Portugol usa a sintaxe de chaves para delimitar os blocos de código que devem ser executados se uma determinada condição for atendida. No exemplo que você mostrou, a palavra-chave "se" é usada para iniciar a condição, e as chaves são usadas para delimitar o bloco de código que deve ser executado se a condição for verdadeira.

Dessa forma, o Portugol torna a sintaxe de suas estruturas de controle de fluxo mais simples e intuitiva, o que facilita o aprendizado de programação para iniciantes.

Capítulo 8: Desafios à frente!

Esse capítulo, intitulado "Desafios à frente!", é um convite ao você para colocar em prática todo o conhecimento adquirido nos capítulos anteriores.

O objetivo deste capítulo é fazer com que o você aprimore suas habilidades de programação e se sinta mais confiante em enfrentar desafios maiores. Para isso, serão apresentados vários exercícios.

Ao final deste capítulo, você estará um nível a mais do que quando começou, antes de entrar nessa aventura.

- Crie um programa que solicite ao usuário dois números e uma operação matemática (+, -, * ou /). O programa deve realizar a operação selecionada e exibir o resultado na tela.

Dica: Utilize as funções de entrada de dados "leia" e as operações matemáticas correspondentes aos operadores (+, -, * e /).

- Crie um programa que solicite ao usuário um número inteiro positivo e exiba na tela um contador regressivo, do número digitado até 1. Ao final, o programa deve exibir a mensagem "Fim!" na tela.

Dica: Utilize a estrutura de repetição enquanto e a função "escreva" para exibir o contador regressivo. Lembre-se de decrementar a variável a cada iteração.

- Crie um programa que solicite ao usuário três notas e calcule a média ponderada das notas, considerando pesos 2, 3 e 5, respectivamente. O programa deve exibir a média calculada na tela.

Dica: Utilize as variáveis do tipo real para armazenar as notas e a média ponderada. Lembre-se de utilizar a fórmula de cálculo da média ponderada e os pesos correspondentes.

- Crie um programa que solicite ao usuário dois números inteiros e exiba na tela o resultado da soma, subtração, multiplicação e divisão desses números. Utilize variáveis para armazenar os valores digitados e os resultados das operações.

Dicas: Utilize o comando "leia" para solicitar os números ao usuário e armazená-los em variáveis.

Utilize os operadores aritméticos para realizar as operações e armazene o resultado em variáveis diferentes para cada operação.

Utilize o comando "escreva" para exibir os resultados das operações na tela.

- Crie um programa que solicite ao usuário a base e a altura de um retângulo e exiba na tela sua área e seu perímetro. Utilize variáveis para armazenar os valores digitados e os resultados das operações.

Dicas: Utilize o comando "leia" para solicitar a base e a altura do retângulo e armazená-las em variáveis.

Utilize os operadores aritméticos para calcular a área (base x altura) e o perímetro ($2 \times \text{base} + 2 \times \text{altura}$) do retângulo e armazene os resultados em variáveis diferentes.

Utilize o comando "escreva" para exibir os resultados na tela.

- Crie um programa que solicite ao usuário um número inteiro e verifique se ele é positivo, negativo ou zero. Exiba o resultado na tela.

Dicas: Utilize o comando "leia" para solicitar o número ao usuário e armazená-lo em uma variável.

Utilize o operador de comparação para verificar se o número é maior, menor ou igual a zero.

Utilize o comando "escreva" para exibir o resultado na tela.

- Crie um programa que solicite ao usuário o seu nome e sua idade e exiba na tela uma mensagem personalizada de acordo com a idade fornecida. Se a idade for menor que 18 anos, exiba a mensagem "Você é menor de idade". Caso contrário, exiba a mensagem "Você é maior de idade".

Dicas: Utilize o comando "leia" para solicitar o nome e a idade do usuário e armazená-los em variáveis.

Utilize o operador de comparação para verificar se a idade é menor ou maior que 18 anos.

Utilize o comando "escreva" para exibir a mensagem personalizada na tela.

- Crie um programa que solicite ao usuário um número inteiro e verifique se ele é par e se é divisível por 3. Exiba o resultado na tela.

Dicas: Utilize o comando "leia" para solicitar o número ao usuário e armazená-lo em uma variável.

Utilize o operador "resto da divisão" para verificar se o número é par e se é divisível por 3.

Utilize o comando "escreva" para exibir o resultado na tela.

Tente passar por esses desafios meu caro aventureiro!

Parte 3:

Descobrindo os segredos da Terra do Código

Capítulo 9: A jornada para aprender algoritmos

Vamos a Jornada para Aprender algoritmos, onde você irá descobrir o que é um algoritmo e como ele é fundamental para compreender qualquer linguagem de programação! Nesta jornada, você irá entender como a lógica de programação se aplica à criação de algoritmos, que são a base para a criação de programas eficientes e precisos.

Ao iniciar sua jornada, você irá descobrir que um algoritmo é uma sequência de passos lógicos que são seguidos para realizar uma tarefa específica. Essa tarefa pode variar de simples cálculos matemáticos até tarefas mais complexas, como a criação de um sistema de gestão de estoque. Os algoritmos são essenciais para a criação de programas de computador e, sem eles, a criação de qualquer software seria impossível.

Durante sua jornada, você irá aprender como criar algoritmos eficientes e precisos, utilizando técnicas de lógica de programação, como a estruturação de dados, a organização de processos e a definição de entradas e saídas. Você também irá descobrir como a lógica de programação se aplica a diferentes linguagens de programação, como Python, Java e C++, entre outras.

Portanto, se você deseja se tornar um programador de sucesso, é fundamental dominar os algoritmos e a lógica de programação. Com essas habilidades, você será capaz de criar programas precisos e eficientes para atender às necessidades de qualquer usuário. Então, está pronto para embarcar nessa jornada emocionante? Vamos começar a aprender algoritmos!

Os algoritmos são usados em muitas áreas, desde a ciência da computação até a matemática e a engenharia. Na computação, os algoritmos são particularmente importantes porque eles são usados para realizar cálculos, ordenar dados, buscar informações em bancos de dados, entre outras tarefas.

Por exemplo, um algoritmo simples poderia ser usado para ordenar uma lista de números em ordem crescente. O algoritmo poderia começar comparando o primeiro número da lista com o segundo, e se o primeiro número for maior que o segundo, eles seriam trocados de lugar. Em seguida, o algoritmo compararia o segundo número com o terceiro, e assim por diante, até que todos os números na lista estivessem em ordem crescente.

Para escrever um algoritmo eficaz, é importante que as instruções sejam claras, concisas e fáceis de seguir. Além disso, o algoritmo deve ser projetado para lidar com todas as situações possíveis que possam surgir, para garantir que ele produza resultados precisos e confiáveis.

Vou dar alguns exemplos de como os algoritmos podem ser usados em situações do dia a dia.

- Algoritmo de busca na internet: Quando você digita uma palavra ou frase em um mecanismo de busca, como o Google, o motor de busca usa um algoritmo para encontrar e classificar os resultados relevantes. O algoritmo avalia fatores como a relevância do conteúdo, a popularidade do site e a qualidade da experiência do usuário para apresentar os resultados mais úteis e precisos.
- Algoritmos de navegação GPS: Os sistemas de navegação GPS usam algoritmos para determinar a rota mais eficiente para chegar ao destino. Os algoritmos levam em consideração fatores como a distância, o tempo estimado de chegada, o tráfego e a velocidade média do veículo para apresentar a melhor rota.
- Algoritmos de recomendação de filmes e música: Os serviços de streaming de música e vídeo, como o Netflix e o Spotify, usam algoritmos para recomendar conteúdo com base no seu histórico de visualização ou audição. O algoritmo analisa as suas escolhas anteriores para apresentar sugestões personalizadas e relevantes.
- Algoritmos de classificação de e-mails: Os provedores de e-mail usam algoritmos para classificar os e-mails recebidos em categorias, como "Caixa de entrada", "Spam" e "Lixo eletrônico". O algoritmo avalia os atributos do e-mail, como o remetente, o assunto e o conteúdo, para determinar a categoria correta.
- Algoritmos de reconhecimento de voz e imagem: Os assistentes virtuais, como a Siri e o Google Assistant, usam algoritmos para reconhecer e processar comandos de voz. As câmeras digitais usam algoritmos para reconhecer faces e ajustar automaticamente as configurações de exposição e foco.

Esses são apenas alguns exemplos de como os algoritmos são usados no nosso cotidiano. Espero que isso ajude a entender como os algoritmos podem ser úteis e importantes em diversas áreas da nossa vida!

Exemplos práticos:

- Algoritmo para calcular a média de um aluno:

```

1 programa {
2   funcao inicio() {
3     real nota1, nota2, nota3, media
4     escreva("Digite a primeira nota: ")
5     leia(nota1)
6     escreva("Digite a segunda nota: ")
7     leia(nota2)
8     escreva("Digite a terceira nota: ")
9     leia(nota3)
10    media = (nota1 + nota2 + nota3) / 3
11    escreva("A média do aluno é: ", media)
12  }
13 }

```

Console Output:

```

Digite a primeira nota: 7
Digite a segunda nota: 8
Digite a terceira nota: 9
A média do aluno é: 8.0

```

Neste exemplo, o algoritmo pede ao usuário para digitar as três notas de um aluno e, em seguida, calcula a média das notas. Por fim, o algoritmo exibe a média na tela.

Enfim, os algoritmos podem ser criados em diversas linguagens de programação, como Python, Java, C++, JavaScript, entre outras. Além disso, existem diversas técnicas para analisar e otimizar algoritmos, de modo a torná-los mais eficientes e rápidos.

Aqui estão algumas recomendações de livros para se aprofundar mais sobre algoritmos:

- **"Introduction to Algorithms" (Introdução aos Algoritmos)** de Thomas Cormen, Charles Leiserson, Ronald Rivest e Clifford Stein: Este livro é frequentemente referido como o "CLRS" em homenagem aos autores e é um dos livros mais populares sobre algoritmos. Ele cobre uma ampla gama de tópicos, desde algoritmos básicos de ordenação e busca até algoritmos mais avançados de programação dinâmica e grafos. O livro também apresenta análises teóricas de desempenho de algoritmos e técnicas para avaliar o tempo de execução e o uso de memória.
- **"Algorithms" (Algoritmos)** de Robert Sedgewick e Kevin Wayne: Este livro abrange uma ampla gama de algoritmos e estruturas de dados, incluindo ordenação, busca, grafos e programação dinâmica. Ele apresenta muitos exemplos práticos e é muito bem escrito e organizado.
- **"The Algorithm Design Manual" (O Manual de Design de Algoritmos)** de Steven Skiena: Este livro é um excelente recurso para aprender sobre algoritmos e projetar seus próprios algoritmos. Ele aborda tópicos como ordenação, busca, programação dinâmica e grafos, mas também inclui capítulos sobre questões de implementação e design de algoritmos. O livro inclui muitos exercícios e exemplos práticos.

- **"Grokking Algorithms" (Entendendo Algoritmos)** de Aditya Bhargava: Este livro é uma excelente escolha para quem está começando a aprender sobre algoritmos. Ele apresenta conceitos fundamentais como ordenação, busca e grafos de uma maneira fácil de entender, e inclui muitos exemplos práticos em Python.

Capítulo 10: As estruturas de dados que habitam a Terra do Código

Na Terra do Código, as estruturas de dados são de extrema importância para os habitantes, pois são fundamentais para o armazenamento, processamento e manipulação de informações. Assim como no mundo real, existem diversas estruturas de dados na Terra do Código, cada uma com suas próprias características e usos. Os habitantes da Terra do Código precisam conhecer essas estruturas para poderem utilizar as mais adequadas para cada situação.

- **Arrays:** Arrays são estruturas de dados que armazenam uma coleção de elementos do mesmo tipo em uma única variável. Eles são indexados com base em sua posição na matriz, e podem ser usados para armazenar listas de valores. Em muitas linguagens de programação, os arrays têm um tamanho fixo, o que significa que o número de elementos que eles podem armazenar é definido no momento da criação da matriz.
- **Listas:** Listas são semelhantes aos arrays, mas diferem em alguns aspectos importantes. Enquanto os arrays têm um tamanho fixo, as listas podem crescer ou diminuir dinamicamente à medida que os elementos são adicionados ou removidos. Além disso, as listas podem conter elementos de diferentes tipos.
- **Pilhas:** Pilhas são estruturas de dados que seguem o princípio "último a entrar, primeiro a sair". Elementos são adicionados e removidos apenas na parte superior da pilha, o que torna as pilhas úteis para implementar algoritmos de pesquisa em profundidade.
- **Filas:** Filas são estruturas de dados que seguem o princípio "primeiro a entrar, primeiro a sair". Os elementos são adicionados ao final da fila e removidos do início da fila, o que as torna úteis para implementar algoritmos de busca em largura.
- **Árvores:** Árvores são estruturas de dados hierárquicas que consistem em nós conectados por arestas. Cada nó pode ter vários filhos, mas apenas um pai. As árvores são amplamente utilizadas em algoritmos de busca, como o algoritmo de busca em profundidade limitada.

- **Grafos:** Grafos são estruturas de dados que consistem em um conjunto de nós e um conjunto de arestas que conectam esses nós. Os grafos podem ser usados para representar muitas coisas diferentes, como redes sociais, sistemas de transporte e diagramas de fluxo.
- **matrizes:** Matrizes são estruturas de dados retangulares que armazenam valores em linhas e colunas. Elas são muito utilizadas em programação e em computação em geral, pois permitem o armazenamento e manipulação de dados em uma estrutura organizada e eficiente.

Em termos de implementação, as estruturas de dados podem ser criadas usando matrizes, ponteiros, listas encadeadas ou outras técnicas de programação. O importante é escolher a estrutura de dados certa para o problema em questão, levando em consideração os requisitos de desempenho e escalabilidade.

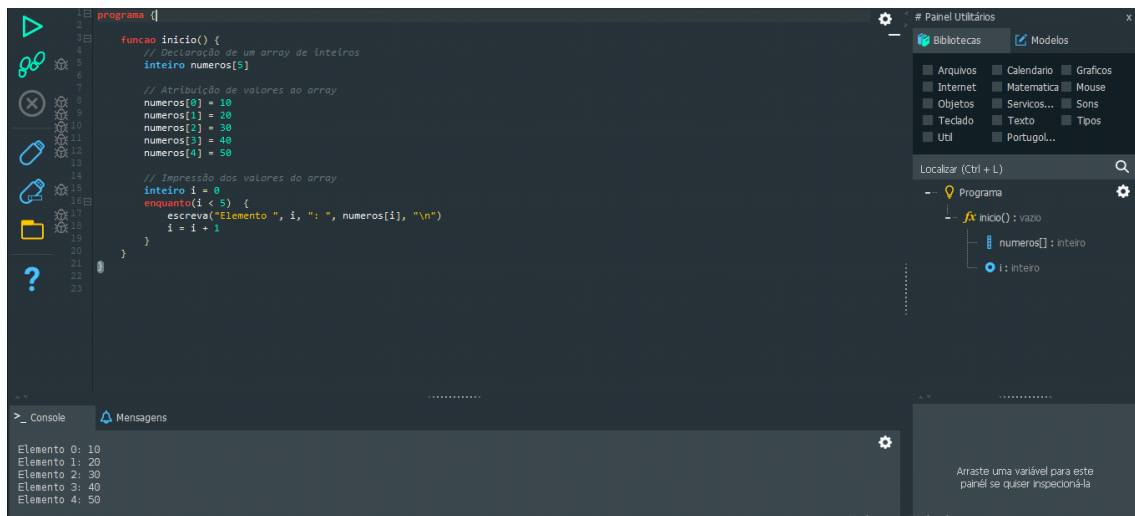
Capítulo 10.1: A tribo dos Arrays

A tribo dos arrays é uma tribo de elementos do mesmo tipo que vivem juntos em uma estrutura organizada. Cada membro dessa tribo é identificado por uma posição específica, que é numerada sequencialmente. Essas posições são como casas, onde cada elemento tem sua própria casa numerada.

Os membros da tribo dos arrays são muito eficientes em armazenar e manipular dados. Eles podem adicionar novos membros à tribo, remover membros existentes e acessar elementos em posições específicas com facilidade. Eles são especialmente bons em trabalhar com grandes quantidades de dados, pois organizam todos os elementos em um único local, o que facilita o acesso e a manipulação.

Em resumo, a tribo dos arrays é uma comunidade de elementos organizados que são capazes de trabalhar juntos de maneira eficiente e realizar operações específicas, como adicionar, remover e acessar elementos em posições específicas. Eles são muito úteis na programação para armazenar e manipular dados de forma eficiente.

Um exemplo de uso de arrays na programação é em um jogo de labirinto, em que o labirinto pode ser representado por uma matriz de números, em que cada número representa um tipo de parede ou caminho. Outro exemplo é em um programa de estoque de produtos, em que os produtos podem ser armazenados em um array, com seus respectivos preços e quantidades:



Nesse exemplo, um array de inteiros chamado "numeros" é declarado com tamanho 5. Em seguida, valores são atribuídos a cada posição do array usando o operador de índice [], que permite acessar elementos específicos do array.

Para imprimir os valores do array, é utilizado um laço de repetição enquanto, que percorre todas as posições do array de 0 a 4 (pois o array tem tamanho 5). Dentro do laço, cada elemento é impresso utilizando o operador de índice [] e a variável i para acessar a posição correta.

É importante lembrar que os arrays em muitas linguagens de programação, incluindo o Portugol, são indexados a partir de 0. Ou seja, o primeiro elemento do array tem índice 0, o segundo tem índice 1, e assim por diante.

Capítulo 10.2: A lista dos Listas

As listas são uma das estruturas de dados mais comuns em programação, permitindo armazenar uma sequência de elementos em uma única variável. Mas, dentro desse universo das listas, existe uma comunidade especial: a Lista das Listas, ou "O Povo das Posições e Índices".

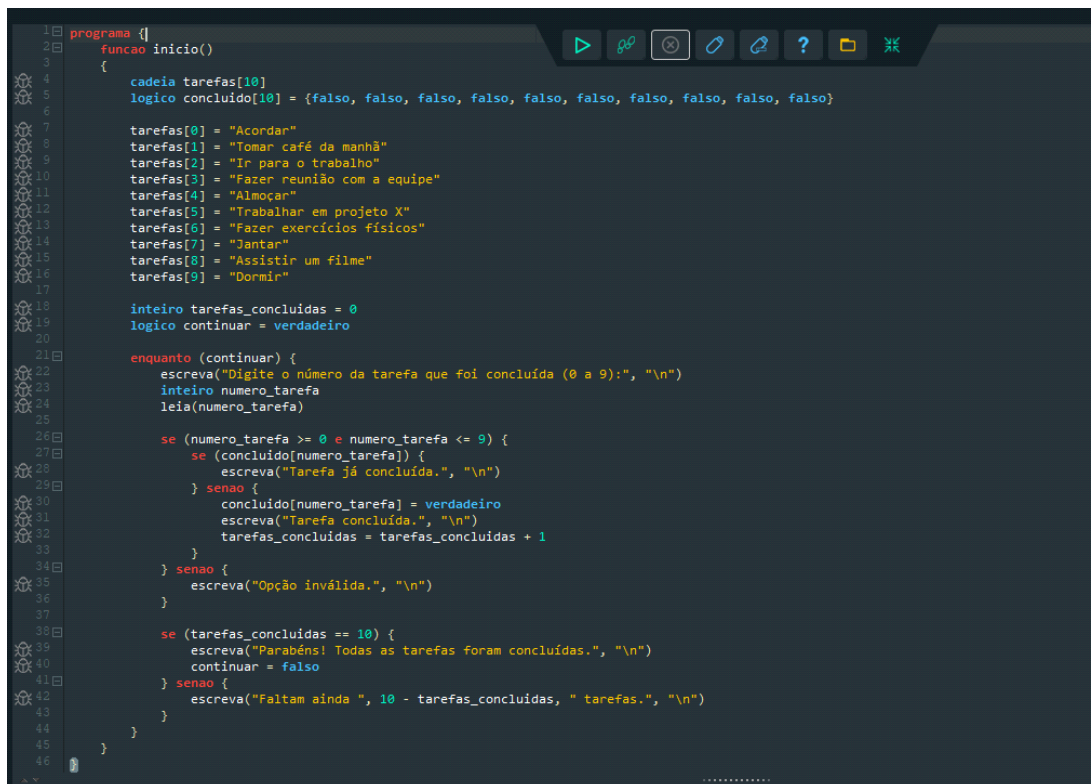
Assim como em uma cidade, cada elemento da Lista das Listas possui sua própria posição, ou índice, que define seu lugar na sequência. E assim como uma comunidade, a Lista das Listas é eficiente em realizar tarefas em grupo, como adicionar e remover elementos. Mas, assim como qualquer sociedade, também possui suas limitações.

Uma dessas limitações é a dificuldade em acessar elementos em posições específicas. Isso se deve ao fato de que, para acessar um elemento, é necessário percorrer a lista até chegar à posição desejada. Em uma lista com milhares de elementos, isso pode ser bastante ineficiente.

Apesar disso, a Lista das Listas é uma poderosa ferramenta na programação, e seu povo das posições e índices é essencial para armazenar conjuntos de elementos, como uma lista de nomes ou de tarefas a serem executadas. E, assim como em qualquer comunidade, é importante conhecer suas particularidades e limitações para utilizá-las da melhor forma possível.

Um exemplo de uso de listas na programação é em um programa de edição de texto, em que as alterações feitas pelo usuário podem ser armazenadas em uma lista de comandos, que pode ser facilmente desfeita ou refeita pelo usuário. Outro exemplo é em um programa de gerenciamento de contatos, em que os contatos podem ser armazenados em uma lista, que pode ser facilmente percorrida e manipulada.

Para entendermos melhor, vamos usar o portugol. aqui um exemplo para entendermos desse assunto:



```
1 programa {
2   funcao inicio()
3   {
4     cadeia tarefas[10]
5     logico concluido[10] = {falso, falso, falso, falso, falso, falso, falso, falso, falso, falso}
6
7     tarefas[0] = "Acordar"
8     tarefas[1] = "Tomar café da manhã"
9     tarefas[2] = "Ir para o trabalho"
10    tarefas[3] = "Fazer reunião com a equipe"
11    tarefas[4] = "Almoçar"
12    tarefas[5] = "Trabalhar em projeto X"
13    tarefas[6] = "Fazer exercícios físicos"
14    tarefas[7] = "Jantar"
15    tarefas[8] = "Assistir um filme"
16    tarefas[9] = "Dormir"
17
18    inteiro tarefas_concluidas = 0
19    logico continuar = verdadeiro
20
21    enquanto (continuar) {
22      escreva("Digite o número da tarefa que foi concluída (0 a 9):", "\n")
23      inteiro numero_tarefa
24      leia(numero_tarefa)
25
26      se (numero_tarefa >= 0 e numero_tarefa <= 9) {
27        se (concluido[numero_tarefa]) {
28          escreva("Tarefa já concluída.", "\n")
29        } senao {
30          concluido[numero_tarefa] = verdadeiro
31          escreva("Tarefa concluída.", "\n")
32          tarefas_concluidas = tarefas_concluidas + 1
33        }
34      } senao {
35        escreva("Opção inválida.", "\n")
36      }
37
38      se (tarefas_concluidas == 10) {
39        escreva("Parabéns! Todas as tarefas foram concluídas.", "\n")
40        continuar = falso
41      } senao {
42        escreva("Faltam ainda ", 10 - tarefas_concluidas, " tarefas.", "\n")
43      }
44    }
45  }
46 }
```


The screenshot shows a Scratch IDE with a program that uses a list to store tasks and a loop to mark them as completed. The code is as follows:

```
tarefas[4] = "Ir para o trabalho"
tarefas[3] = "Fazer reunião com a equipe"
tarefas[4] = "Almoçar"
tarefas[5] = "Trabalhar em projeto X"
tarefas[6] = "Fazer exercícios físicos"
tarefas[7] = "Jantar"
tarefas[8] = "Assistir um filme"
tarefas[9] = "Dormir"

inteiro tarefas_concluidas = 0
logico continuar = verdadeiro

enquanto (continuar) {
  escreva("Digite o número da tarefa que foi concluída (0 a 9):", "\n")
  inteiro numero_tarefa
  leia(numero_tarefa)
```

The console shows the following output:

```
Digite o número da tarefa que foi concluída (0 a 9):
4
Tarefa concluída.
Faltam ainda 5 tarefas.
Digite o número da tarefa que foi concluída (0 a 9):
5
Tarefa concluída.
Faltam ainda 4 tarefas.
Digite o número da tarefa que foi concluída (0 a 9):
6
Tarefa concluída.
Faltam ainda 3 tarefas.
Digite o número da tarefa que foi concluída (0 a 9):
7
Tarefa concluída.
Faltam ainda 2 tarefas.
Digite o número da tarefa que foi concluída (0 a 9):
8
Tarefa concluída.
Faltam ainda 1 tarefa.
Digite o número da tarefa que foi concluída (0 a 9):
9
Tarefa concluída.
Parabéns! Todas as tarefas foram concluídas.
```

The right panel shows the 'Monitor' tab with the following variables:

- tarefas[] : cadeia
- concluido[] : logico
- tarefas_concluidas : inteiro
- continuar : logico
- enquanto
- numero_tarefa : inteiro

Esse código é um programa que usa a estrutura de dados "lista" para armazenar uma série de tarefas a serem realizadas, e permite ao usuário marcar quando cada tarefa é concluída.

Na primeira parte do programa, são definidas duas listas: uma lista de tarefas, que armazena as descrições de cada tarefa, e uma lista de variáveis booleanas (concluido), que indica se cada tarefa foi concluída ou não. São definidos 10 elementos para cada lista.

Na segunda parte do programa, o usuário é solicitado a digitar o número da tarefa que foi concluída. Se o número estiver dentro do intervalo válido (de 0 a 9), o programa verifica se a tarefa correspondente já foi concluída anteriormente. Se já tiver sido concluída, o programa informa que a tarefa já foi concluída. Caso contrário, o programa marca a tarefa como concluída e atualiza a contagem de tarefas concluídas. Se todas as tarefas foram concluídas, o programa para e exibe uma mensagem de parabenização. Caso contrário, o programa informa quantas tarefas ainda precisam ser concluídas.

A lista é uma estrutura de dados que permite armazenar uma coleção de elementos em uma única variável. Nesse programa, a lista de tarefas e a lista de variáveis booleanas são usadas para armazenar as descrições das tarefas e se cada uma foi concluída ou não. Cada elemento da lista pode ser acessado usando um índice numérico, começando em 0 para o primeiro elemento e indo até n-1 para o último elemento, onde n é o tamanho da lista. Nesse programa, o tamanho da lista é definido como 10.

Listas encadeadas

```
1 programa {
2     funcao inicio() {
3         const inteiro MAX = 100
4         inteiro lista[MAX][2]
5         inteiro cabeca = -1
6         inteiro livre = 0
7
8         // Inicializa lista livre
9         para (inteiro i = 0; i < MAX-1; i++) {
10             lista[i][1] = i+1
11         }
12         lista[MAX-1][1] = -1
13
14         inteiro opcao = 0
15         enquanto (opcao != 4) {
16             escreva("Escolha uma opção:\n")
17             escreva("1 - Inserir elemento\n")
18             escreva("2 - Remover elemento\n")
19             escreva("3 - Imprimir lista\n")
20             escreva("4 - Sair\n")
21             leia(opcao)
22
23             se (opcao == 1) {
24                 inteiro valor
25                 escreva("Digite o valor a ser inserido: ")
26                 leia(valor)
27
28                 // Verifica se há espaço livre na lista
29                 se (livre == -1) {
30                     escreva("A lista está cheia.\n")
31                 } senao {
32                     // Inserir no início da lista
33                     lista[livre][0] = valor
34                     lista[livre][1] = cabeca
35                     cabeca = livre
36                     livre = lista[livre][1]
37                     escreva("Elemento inserido com sucesso.\n")
38                 }
39             } senao se (opcao == 2) {
40                 se (cabeca == -1) {
41                     escreva("A lista está vazia.\n")
42                 } senao {
43                     // Remove o primeiro elemento da lista
44                     inteiro removido = cabeca
45                     cabeca = lista[cabeca][1]
46                     lista[removido][1] = livre

```

```
20             escreva("4 - Sair\n")
21             leia(opcao)
22
23             se (opcao == 1) {
24                 inteiro valor
25                 escreva("Digite o valor a ser inserido: ")
26                 leia(valor)
27
28                 // Verifica se há espaço livre na lista
29                 se (livre == -1) {
30                     escreva("A lista está cheia.\n")
31                 } senao {
32                     // Inserir no início da lista
33                     lista[livre][0] = valor
34                     lista[livre][1] = cabeca
35                     cabeca = livre
36                     livre = lista[livre][1]
37                     escreva("Elemento inserido com sucesso.\n")
38                 }
39             } senao se (opcao == 2) {
40                 se (cabeca == -1) {
41                     escreva("A lista está vazia.\n")
42                 } senao {
43                     // Remove o primeiro elemento da lista
44                     inteiro removido = cabeca
45                     cabeca = lista[cabeca][1]
46                     lista[removido][1] = livre
47                     livre = removido
48                     escreva("Elemento ", lista[removido][0], " removido com sucesso.\n")
49                 }
50             } senao se (opcao == 3) {
51                 inteiro atual = cabeca
52                 enquanto (atual != -1) {
53                     escreva(lista[atual][0], " ")
54                     atual = lista[atual][1]
55                 }
56                 escreva("\n")
57             } senao se (opcao == 4) {
58                 escreva("Programa encerrado.\n")
59             } senao {
60                 escreva("Opção inválida.\n")
61             }
62         }
63     }
64 }
65 }
```

Este é um exemplo de implementação de lista encadeada em Portugol. Uma lista encadeada é uma estrutura de dados que consiste em uma série de elementos, onde cada elemento aponta para o próximo elemento da lista.

No programa, a lista é representada por um array bidimensional chamado lista, que possui uma coluna para armazenar o valor do elemento e outra coluna para armazenar o índice do próximo elemento. A lista começa vazia, com a variável cabeça apontando para -1 e a variável livre apontando para o primeiro índice disponível no array.

O programa permite ao usuário escolher uma das quatro opções disponíveis: inserir um elemento na lista, remover um elemento da lista, imprimir a lista ou sair do programa.

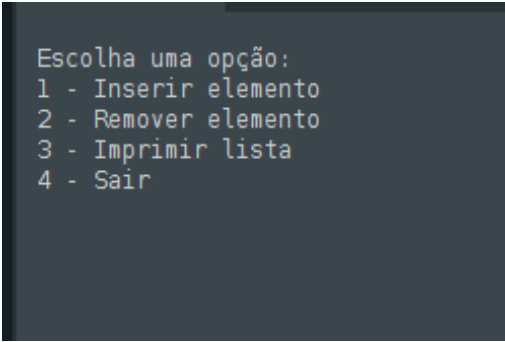
Para inserir um elemento na lista, o programa verifica se há espaço livre no array e, se houver, insere o novo elemento no início da lista, atualizando as variáveis cabeça e livre. Se não houver espaço livre, uma mensagem de erro é exibida.

Para remover um elemento da lista, o programa verifica se a lista está vazia e, se não estiver, remove o primeiro elemento da lista e atualiza as variáveis cabeça e livre. Se a lista estiver vazia, uma mensagem de erro é exibida.

Para imprimir a lista, o programa percorre os elementos da lista, começando pelo primeiro elemento (cabeça) e imprimindo o valor de cada elemento.

Ao escolher a opção de sair do programa, uma mensagem de encerramento é exibida.

Este exemplo ilustra o uso de uma lista encadeada como uma estrutura de dados dinâmica e flexível, que pode crescer ou diminuir em tamanho conforme necessário, sem exigir um tamanho fixo predefinido.



```
Escolha uma opção:  
1 - Inserir elemento  
2 - Remover elemento  
3 - Imprimir lista  
4 - Sair
```

Capítulo 10.3: O reino das Pilhas

Imagine um reino onde os elementos são organizados de forma muito peculiar: o último a entrar é o primeiro a sair. Esse é o reino das pilhas, uma estrutura de dados fundamental na programação. Em uma pilha, cada elemento adicionado é colocado no topo da pilha, e somente o elemento no topo pode ser removido. Pilhas são amplamente utilizadas na programação, desde o controle de histórico de operações até a resolução de expressões matemáticas complexas. Venha conosco explorar os segredos do reino das pilhas e descobrir como essa estrutura pode ser útil em seus projetos de programação.

Um exemplo de uso de pilhas na programação é em um programa de processamento de expressões matemáticas, em que os parênteses podem ser armazenados em uma pilha para garantir que eles sejam fechados na ordem correta. Outro exemplo é em um programa de edição de texto, em que as operações de desfazer e refazer podem ser implementadas com o uso de pilhas.

Agora vamos aprender ela usando o portugol. Mas para implementar uma pilha em Portugol, é necessário definir uma variável para representar a pilha e utilizar as operações básicas de inserção e remoção de elementos.

Vamos praticar!

Este código implementa uma estrutura de dados de pilha simples para armazenar informações sobre livros:

```
1 programa
2     const inteiro MAX = 3
3     inteiro topo = -1
4     cadeia pilha[MAX]
5
6     funcao empilha(){
7         se (topo == MAX-1) {
8             escreva("A pilha está cheia.\n")
9             retorne
10        }
11
12        cadeia livro
13        escreva("Digite o título e o autor do livro: ")
14        leia(livro)
15
16        topo++
17        pilha[topo] = livro
18        escreva("Livro adicionado: ", livro, "\n")
19    }
20
21    funcao desempilha(){
22        se (topo == -1) {
23            escreva("A pilha está vazia.\n")
24            retorne
25        }
26
27        escreva("Livro removido: ", pilha[topo], "\n")
28        topo--
29    }
30
31    funcao imprimir(){
32        escreva("\n----Pilha de livros----\n")
33        para (inteiro i = topo; i >= 0; i--) {
34            escreva(pilha[i], "\n")
35        }
36    }
37
38    funcao inicio(){
39        enquanto (verdadeiro) {
40            escreva("\nO que você gostaria de fazer?\n")
41            escreva("1. Adicionar um livro\n")
42            escreva("2. Remover um livro\n")
43            escreva("3. Imprimir a pilha de livros\n")
44            escreva("4. Sair\n")
45
46            inteiro opcao
```

```

18     escreva("Livro adicionado: ", livro, "\n")
19 }
20
21 funcao desempilha(){
22     se (topo == -1) {
23         escreva("A pilha está vazia.\n")
24         retorne
25     }
26
27     escreva("Livro removido: ", pilha[topo], "\n")
28     topo--
29
30
31 funcao imprimir(){
32     escreva("\n---Pilha de Livros---\n")
33     para (inteiro i = topo; i >= 0; i--) {
34         escreva(pilha[i], "\n")
35     }
36 }
37
38 funcao inicio(){
39     enquanto (verdadeiro) {
40         escreva("\n0 que você gostaria de fazer?\n")
41         escreva("1. Adicionar um livro\n")
42         escreva("2. Remover um livro\n")
43         escreva("3. Imprimir a pilha de livros\n")
44         escreva("4. Sair\n")
45
46         inteiro opcao
47         leia(opcao)
48
49         se (opcao == 1) {
50             empilha()
51         } senao se (opcao == 2) {
52             desempilha()
53         } senao se (opcao == 3) {
54             imprimir()
55         } senao se (opcao == 4) {
56             escreva("Até logo!\n")
57             pare
58         } senao {
59             escreva("Opção inválida.\n")
60         }
61     }
62 }
63 }

```

Vou explicar cada parte do código em detalhes:

Na primeira linha, é definido um valor constante chamado MAX, que representa o tamanho máximo da pilha.

Em seguida, duas variáveis são criadas: "topo", que representa a posição atual do topo da pilha (inicializado com -1, indicando que a pilha está vazia), e "pilha", que é um array de tamanho MAX que armazenará as informações sobre os livros.

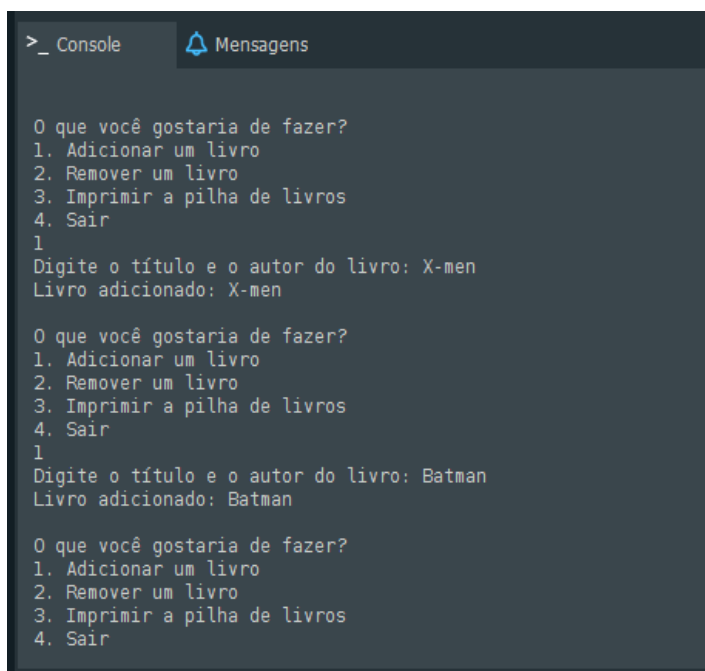
Depois, três funções são definidas: empilha, desempilha e imprimir.

A função empilha é responsável por adicionar um novo livro à pilha. Ele verifica se a pilha já está cheia (ou seja, se o topo atingiu o tamanho máximo - 1) e, se sim, emite uma mensagem de erro e sai da função. Caso contrário, ele solicita ao usuário que insira o título e o autor do livro, armazena as informações na próxima posição disponível na pilha (atualiza o topo e adiciona o livro ao topo da pilha), e emite uma mensagem confirmando que o livro foi adicionado com sucesso.

A função `desempilha` é responsável por remover o livro no topo da pilha e exibi-lo. Ele verifica se a pilha já está vazia (ou seja, se o topo está no índice -1), e, se sim, emite uma mensagem de erro e sai da função. Caso contrário, ele exibe o livro no topo da pilha, atualiza o topo para apontar para a próxima posição da pilha e emite uma mensagem confirmando que o livro foi removido com sucesso.

A função `imprimir` é responsável por exibir todos os livros na pilha, começando pelo topo e indo até o início. Ele começa exibindo um cabeçalho de "Pilha de Livros" e, em seguida, itera sobre o array da pilha do topo até o índice 0, exibindo cada livro na pilha em uma nova linha.

Por fim, a função `início` inicia um loop infinito que solicita ao usuário que selecione uma opção (adicionar um livro, remover um livro, imprimir a pilha de livros ou sair do programa), lê a entrada do usuário e chama a função apropriada com base na opção selecionada. Se a opção selecionada for inválida, uma mensagem de erro é exibida. O loop continua até que o usuário selecione a opção "sair".




```
> _ Console  Mensagens

O que você gostaria de fazer?
1. Adicionar um livro
2. Remover um livro
3. Imprimir a pilha de livros
4. Sair
1
Digite o título e o autor do livro: X-men
Livro adicionado: X-men

O que você gostaria de fazer?
1. Adicionar um livro
2. Remover um livro
3. Imprimir a pilha de livros
4. Sair
1
Digite o título e o autor do livro: Batman
Livro adicionado: Batman

O que você gostaria de fazer?
1. Adicionar um livro
2. Remover um livro
3. Imprimir a pilha de livros
4. Sair
```

>_ Console

 Mensagens

```
1
Digite o título e o autor do livro: You
Livro adicionado: You
```

```
0 que você gostaria de fazer?
1. Adicionar um livro
2. Remover um livro
3. Imprimir a pilha de livros
4. Sair
```


```
1
A pilha está cheia.
```

```
0 que você gostaria de fazer?
1. Adicionar um livro
2. Remover um livro
3. Imprimir a pilha de livros
4. Sair
```

```
1
A pilha está cheia.
```

```
0 que você gostaria de fazer?
1. Adicionar um livro
2. Remover um livro
3. Imprimir a pilha de livros
.
```

>_ Console

 Mensagens

```
0 que você gostaria de fazer?
1. Adicionar um livro
2. Remover um livro
3. Imprimir a pilha de livros
4. Sair
3
```

```
----Pilha de Livros----
You
Batman
X-men
```

```
0 que você gostaria de fazer?
1. Adicionar um livro
2. Remover um livro
3. Imprimir a pilha de livros
4. Sair
```

```
2
Livro removido: You
```

```
0 que você gostaria de fazer?
1. Adicionar um livro
2. Remover um livro
3. Imprimir a pilha de livros
```



```
>_ Console  Mensagens

0 que você gostaria de fazer?
1. Adicionar um livro
2. Remover um livro
3. Imprimir a pilha de livros
4. Sair
2
Livro removido: You

0 que você gostaria de fazer?
1. Adicionar um livro
2. Remover um livro
3. Imprimir a pilha de livros
4. Sair
3

----Pilha de Livros----
Batman
X-men

0 que você gostaria de fazer?
1. Adicionar um livro
2. Remover um livro
3. Imprimir a pilha de livros
4. Sair
```

Capítulo 10.4: A Jornada pela Fila Perfeita

Aqui estamos em busca da fila perfeita, pois as filas são uma estrutura de dados importante na programação, usadas para gerenciar recursos compartilhados e manter a ordem de processamento de elementos. Com elas, podemos controlar o acesso a recursos, garantindo que cada elemento seja processado na ordem correta. Ao longo dessa jornada, vamos explorar as principais características das filas e como utilizá-las para construir programas eficientes e confiáveis.

Um exemplo de uso de filas na programação é em um programa de gerenciamento de tarefas, em que as tarefas são adicionadas a uma fila para serem executadas na ordem em que foram recebidas. Outro exemplo é em um programa de gerenciamento de pedidos online, em que os pedidos são adicionados a uma fila para serem processados na ordem em que foram recebidos.

agora vamos praticar!

```
1 programa {
2     const inteiro TAMANHO_MAXIMO_FILA = 5 // define o tamanho m
3     inteiro filaInicio = 0 // define o índice inicial da fila
4     inteiro fim = 0 // define o índice final da fila
5     cadeia fila[TAMANHO_MAXIMO_FILA] // declara a fila como uma matriz de strings
6
7     funcao enfileira(cadeia nome_cliente) {
8         se ((fim + 1) % TAMANHO_MAXIMO_FILA == filaInicio) {
9             // verifica se a fila está cheia
10            escreva("A fila está cheia. Por favor, aguarde um pouco.\n")
11            retorne
12        }
13
14        fila[fim] = nome_cliente // adiciona o nome do cliente na última posição da fila
15        fim = (fim + 1) % TAMANHO_MAXIMO_FILA // atualiza o índice final da fila
16        escreva("Cliente ", nome_cliente, " adicionado à fila.\n")
17    }
18
19    funcao desenfileira() {
20        se (filaInicio == fim) {
21            // verifica se a fila está vazia
22            escreva("A fila está vazia.\n")
23            retorne
24        }
25
26        cadeia nome_cliente = fila[filaInicio] // obtém o nome do cliente que está na primeira posição da fila
27        filaInicio = (filaInicio + 1) % TAMANHO_MAXIMO_FILA // atualiza o índice inicial da fila
28        escreva("Cliente ", nome_cliente, " retirado da fila.\n")
29    }
30
31    funcao imprime_fila() {
32        escreva("\n--- Fila de clientes ---\n")
33        inteiro i = filaInicio
34        enquanto (i != fim) {
35            escreva(fila[i], "\n")
36            i = (i + 1) % TAMANHO_MAXIMO_FILA // atualiza o índice para imprimir o próximo nome da fila
37        }
38    }
39
40    funcao inicio() {
41        enquanto (verdadeiro) {
42            escreva("\nO que você gostaria de fazer?\n")
43            escreva("1. Adicionar um cliente à fila\n")
44            escreva("2. Retirar um cliente da fila\n")
45            escreva("3. Imprimir a fila de clientes\n")
46            escreva("4. Sair\n")
47
48            inteiro opcao
49            leia(opcao)
50
51            se (opcao == 1) {
52                cadeia nome_cliente
53                escreva("Digite o nome do cliente: ")
54                leia(nome_cliente)
55                enfileira(nome_cliente)
56            } senao se (opcao == 2) {
57                desenfileira()
58            } senao se (opcao == 3) {
59                imprime_fila()
60            } senao se (opcao == 4) {
61                escreva("Até logo!\n")
62                pare
63            } senao {
64                escreva("Opção inválida.\n")
65            }
66        }
67    }
68 }
69
```

Esse código é um exemplo de implementação de uma fila de atendimento de clientes em uma linguagem de programação imaginária. A ideia é simular o funcionamento de uma

fila de atendimento em que os clientes são adicionados e retirados conforme sua ordem de chegada.

O programa começa definindo o tamanho máximo da fila, que é de 5 clientes, e criando variáveis para indicar o índice inicial e final da fila. Além disso, é criada uma matriz de strings para representar a fila de clientes.

Em seguida, são implementadas três funções principais para manipular a fila:

A função `enfileira` recebe o nome de um cliente como entrada e adiciona esse nome ao final da fila, atualizando o índice final da fila. Antes de adicionar o cliente, a função verifica se a fila está cheia e, se estiver, exibe uma mensagem informando que não é possível adicionar mais clientes no momento.

A função `desenfileira` remove o primeiro cliente da fila, atualizando o índice inicial. Antes de remover o cliente, a função verifica se a fila está vazia e, se estiver, exibe uma mensagem informando que não há clientes na fila.

A função `imprime_fila` exibe na tela todos os nomes dos clientes na fila, na ordem em que foram adicionados. Essa função é útil para visualizar como está a fila em determinado momento.

Por fim, é implementada a função principal `início`, que consiste em um loop infinito que exibe um menu com as opções disponíveis para o usuário. A partir da escolha do usuário, a função chama uma das três funções de manipulação da fila ou encerra o programa, caso o usuário escolha a opção de sair.

No geral, esse código é uma boa ilustração de como as filas funcionam e como podem ser implementadas em uma linguagem de programação. É importante notar, porém, que a implementação pode variar de acordo com a linguagem e a plataforma utilizada, além de existirem outras formas de implementar filas com diferentes características e funcionalidades.

```
>_ Console  Mensagens

O que você gostaria de fazer?
1. Adicionar um cliente à fila
2. Retirar um cliente da fila
3. Imprimir a fila de clientes
4. Sair
1
Digite o nome do cliente: ana
Cliente ana adicionado à fila.

O que você gostaria de fazer?
1. Adicionar um cliente à fila
2. Retirar um cliente da fila
3. Imprimir a fila de clientes
4. Sair
1
Digite o nome do cliente: paulo
Cliente paulo adicionado à fila.

O que você gostaria de fazer?
1. Adicionar um cliente à fila
2. Retirar um cliente da fila
3. Imprimir a fila de clientes
4. Sair
```

```
>_ Console  Mensagens

1
Digite o nome do cliente: joana
Cliente joana adicionado à fila.

O que você gostaria de fazer?
1. Adicionar um cliente à fila
2. Retirar um cliente da fila
3. Imprimir a fila de clientes
4. Sair
1
Digite o nome do cliente: pedro
Cliente pedro adicionado à fila.

O que você gostaria de fazer?
1. Adicionar um cliente à fila
2. Retirar um cliente da fila
3. Imprimir a fila de clientes
4. Sair
1
Digite o nome do cliente: lucio
A fila está cheia. Por favor, aguarde um pouco.

O que você gostaria de fazer?
1. Adicionar um cliente à fila
2. Retirar um cliente da fila
```

```
> _ Console Mensagens

0 que você gostaria de fazer?
1. Adicionar um cliente à fila
2. Retirar um cliente da fila
3. Imprimir a fila de clientes
4. Sair
3

--- Fila de clientes ---
ana
paulo
joana
pedro

0 que você gostaria de fazer?
1. Adicionar um cliente à fila
2. Retirar um cliente da fila
3. Imprimir a fila de clientes
4. Sair
2
Cliente ana retirado da fila.

0 que você gostaria de fazer?
1. Adicionar um cliente à fila
2. Retirar um cliente da fila
3. Imprimir a fila de clientes
```

```
> _ Console Mensagens

2. Retirar um cliente da fila
3. Imprimir a fila de clientes
4. Sair
2
Cliente ana retirado da fila.

0 que você gostaria de fazer?
1. Adicionar um cliente à fila
2. Retirar um cliente da fila
3. Imprimir a fila de clientes
4. Sair
3

--- Fila de clientes ---
paulo
joana
pedro

0 que você gostaria de fazer?
1. Adicionar um cliente à fila
2. Retirar um cliente da fila
3. Imprimir a fila de clientes
4. Sair
```

É importante destacar que a fila foi definida com um tamanho máximo de 5 clientes, e que o programa possui um mecanismo para tratar o caso em que a fila está cheia.

Quando a fila atinge o seu tamanho máximo (5 clientes), o programa não permite que um novo cliente seja adicionado à fila e informa ao usuário que a fila está cheia, pedindo que aguarde um pouco. Isso é feito na função `enfileira()`, que utiliza a expressão `(fim + 1) % TAMANHO_MAXIMO_FILA == filaInicio` para verificar se a fila está cheia.

Dessa forma, se o usuário tentar adicionar um quinto cliente à fila, ele receberá uma mensagem informando que a fila está cheia e que é necessário esperar um pouco. O programa não adicionará o novo cliente à fila e o usuário precisará escolher outra opção.

Capítulo 10.5: A floresta das Árvores

Agora chegamos na floresta das árvores! E No coração da Floresta das Árvores, há um conhecimento antigo que tem sido passado de geração em geração entre os programadores. É a arte de usar árvores para representar hierarquias de dados.

Uma árvore é uma estrutura de dados que contém elementos organizados em uma hierarquia de pais e filhos. Cada elemento é chamado de nó e pode ter zero ou mais nós filhos. O nó no topo da hierarquia é chamado de raiz da árvore, e os nós sem filhos são chamados de folhas.

As árvores são muito úteis na programação para representar uma grande variedade de estruturas hierárquicas, como a estrutura de arquivos em um sistema operacional ou a organização de uma empresa. Além disso, elas são muito utilizadas em algoritmos de busca, como o algoritmo de busca binária, que utiliza a estrutura de árvore para localizar um determinado elemento em uma coleção ordenada de dados.

Ao caminhar pela Floresta das Árvores, podemos ver as mais diversas espécies de árvores. Algumas são mais largas e ramificadas, enquanto outras são mais altas e esguias. Cada árvore é única em sua estrutura e características.

Assim como na floresta, cada programador pode encontrar diferentes maneiras de utilizar árvores em seus projetos. Mas é importante lembrar que a chave para o sucesso é entender bem a estrutura e como utilizá-la de maneira eficiente.

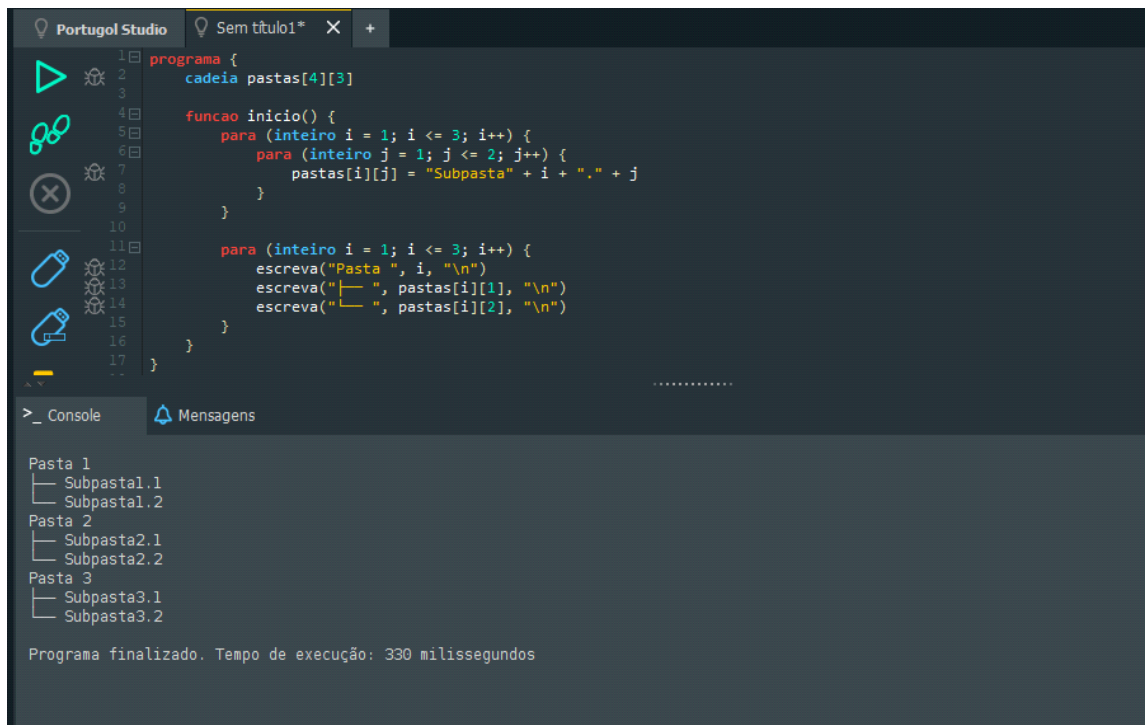
Com sua jornada pela Floresta das Árvores completa, você se tornou um mestre na arte de utilizar árvores em sua programação, capaz de criar estruturas complexas e eficientes para resolver qualquer problema que surgir em seu caminho.

Um exemplo de uso de árvores na programação é em um programa de organização de arquivos, em que os arquivos são organizados em uma estrutura de árvore, em que as pastas são os nós pais e os arquivos são os nós filhos. Outro exemplo é em um programa de busca em uma base de dados, em que a base de dados é representada por uma árvore, em que cada nó representa uma entrada na base de dados.

vou deixar um exemplo do que estou falando:

```
raiz
├─ pasta1
|   ├─ arquivo1
|   └─ arquivo2
├─ pasta2
|   ├─ arquivo3
|   └─ pasta3
|       └─ arquivo4
└─ arquivo5
```

Agora vamos praticar!



```
1 programa {
2     cadeia pastas[4][3]
3
4     funcao inicio() {
5         para (inteiro i = 1; i <= 3; i++) {
6             para (inteiro j = 1; j <= 2; j++) {
7                 pastas[i][j] = "Subpasta" + i + "." + j
8             }
9         }
10
11        para (inteiro i = 1; i <= 3; i++) {
12            escreva("Pasta ", i, "\n")
13            escreva("├── ", pastas[i][1], "\n")
14            escreva("└── ", pastas[i][2], "\n")
15        }
16    }
17 }
```

Pasta 1
├── Subpasta1.1
└── Subpasta1.2
Pasta 2
├── Subpasta2.1
└── Subpasta2.2
Pasta 3
├── Subpasta3.1
└── Subpasta3.2

Programa finalizado. Tempo de execução: 330 milissegundos

Então, nesse exemplo de código em Portugol, o objetivo é representar de maneira básica a estrutura de uma árvore de pastas, utilizando uma matriz para armazenar os nomes das pastas e subpastas.

Para isso, é criada uma matriz chamada "pastas" com dimensões 4x3 e são utilizados dois laços "para" aninhados para preencher essa matriz com valores do tipo "cadeia" (string).

A cada iteração dos laços "para", uma cadeia de caracteres é gerada e atribuída a um elemento da matriz "pastas", de acordo com os índices das variáveis "i" e "j". O padrão utilizado para gerar a cadeia de caracteres é "Subpasta" + i + "." + j, ou seja, a palavra "Subpasta" concatenada com o valor de "i", um ponto (".") e o valor de "j".

O primeiro laço "para" varia o valor da variável "i" de 1 a 3, enquanto o segundo laço "para" varia o valor da variável "j" de 1 a 2. Isso resulta em 6 elementos da matriz "pastas" preenchidos com valores do tipo "cadeia".

Em seguida, o programa utiliza outro laço "para" para percorrer os elementos preenchidos da matriz "pastas" e exibir os seus valores na tela, junto com outras informações sobre pastas.

É importante ressaltar que essa representação é apenas uma maneira básica de ilustrar a estrutura de uma árvore de pastas. A estrutura de dados árvore é bem mais complexa e pode ter diversas aplicações, como em sistemas de gerenciamento de bancos de dados, organização de arquivos e até mesmo na análise de algoritmos. Ela é composta por nós e ramificações, que podem ser organizados de diversas maneiras, dependendo da aplicação.

Capítulo 10.6: O labirinto dos Grafos

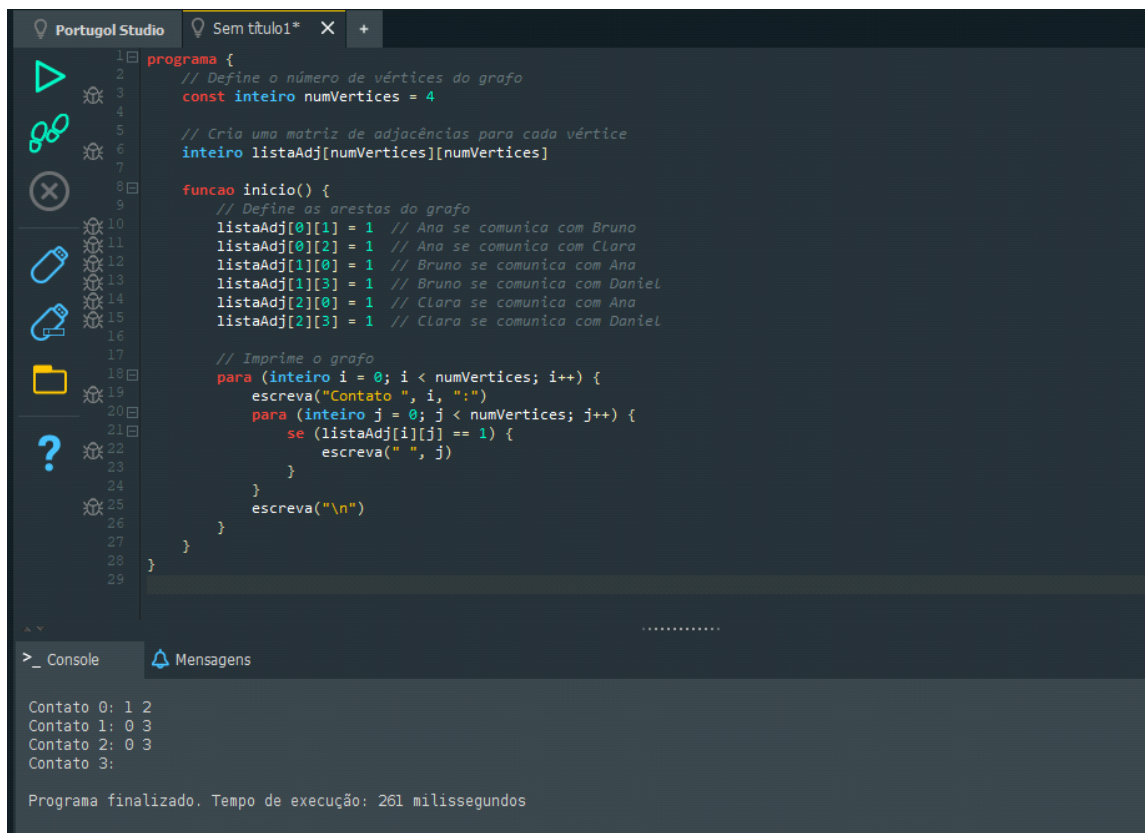
Chegamos a mais uma aventura, o "Labirinto dos Grafos", um dos desafios mais fascinantes que você encontrará aqui na terra do código. Nesse labirinto, as paredes são representadas pelas arestas, e os caminhos que você pode seguir são os vértices ou nós. Cada um desses "nós" está conectado a outros por meio de arestas, criando uma rede complexa de possibilidades.

Mas não se preocupe, com um pouco de conhecimento sobre grafos e algoritmos, você poderá navegar por esse labirinto e descobrir seus segredos. Os grafos são usados em muitos contextos da programação, como em redes de computadores, rotas de transporte e até mesmo em redes sociais. Através da representação desses sistemas complexos por meio de grafos, é possível visualizar e manipular informações de forma eficiente e precisa.

Então, prepare-se para mergulhar no "Labirinto dos Grafos" e descobrir como essa estrutura de dados pode ser usada para resolver problemas complexos e desafiadores na programação.

Um exemplo de uso de grafos na programação é em um programa de planejamento de rotas, em que as rotas entre diferentes locais são representadas por arestas em um grafo, e o caminho mais curto entre dois locais pode ser encontrado usando um algoritmo de busca em grafos. Outro exemplo é em um programa de recomendação de amigos em uma rede social, em que os amigos em comum entre dois usuários podem ser encontrados usando um algoritmo de busca em grafos.

Agora vamos praticar:



```
1 programa {
2     // Define o número de vértices do grafo
3     const inteiro numVertices = 4
4
5     // Cria uma matriz de adjacências para cada vértice
6     inteiro listaAdj[numVertices][numVertices]
7
8     funcao inicio() {
9         // Define as arestas do grafo
10        listaAdj[0][1] = 1 // Ana se comunica com Bruno
11        listaAdj[0][2] = 1 // Ana se comunica com Clara
12        listaAdj[1][0] = 1 // Bruno se comunica com Ana
13        listaAdj[1][3] = 1 // Bruno se comunica com Daniel
14        listaAdj[2][0] = 1 // Clara se comunica com Ana
15        listaAdj[2][3] = 1 // Clara se comunica com Daniel
16
17        // Imprime o grafo
18        para (inteiro i = 0; i < numVertices; i++) {
19            escreva("Contato ", i, ":")
20            para (inteiro j = 0; j < numVertices; j++) {
21                se (listaAdj[i][j] == 1) {
22                    escreva(" ", j)
23                }
24            }
25            escreva("\n")
26        }
27    }
28 }
29
```

Console

```
> Contato 0: 1 2
Contato 1: 0 3
Contato 2: 0 3
Contato 3:

Programa finalizado. Tempo de execução: 261 milissegundos
```

O código apresentado é um programa que representa um grafo através de uma matriz de adjacências. A matriz de adjacências é uma tabela que mostra as conexões entre cada par de vértices no grafo. O programa define as conexões entre 4 vértices, representando pessoas chamadas Ana, Bruno, Clara e Daniel.

O programa cria uma matriz de adjacências de tamanho 4x4, onde cada linha e coluna representa um vértice. As conexões entre os vértices são definidas pelos valores 1 na matriz de adjacências. Por exemplo, se há uma conexão entre Ana e Bruno, a matriz terá o valor 1 na posição (0,1).

O programa percorre a matriz de adjacências e imprime as conexões de cada vértice com os demais vértices. O programa usa dois loops for, um para percorrer as linhas da matriz (representando cada vértice) e outro para percorrer as colunas da matriz (representando as conexões com outros vértices). Se o valor na posição da matriz for 1, indica que há uma conexão entre os dois vértices correspondentes. O programa então imprime o número do vértice conectado.

O resultado da execução do programa é a lista de contatos para cada vértice, mostrando com quem cada pessoa está conectada. Por exemplo, o contato 0 (Ana) está conectado

com os contatos 1 (Bruno) e 2 (Clara), enquanto o contato 3 (Daniel) não está conectado com nenhum outro contato.

Então o que deu no resultado do código mostra as conexões entre os vértices do grafo. Cada linha do resultado representa um vértice do grafo, e os números na linha representam os vértices com os quais o vértice correspondente está conectado.

Por exemplo, a primeira linha "contato 0: 1 2" significa que o vértice 0 (representando a pessoa Ana) está conectado com os vértices 1 (Bruno) e 2 (Clara).

A segunda linha "contato 1: 0 3" significa que o vértice 1 (representando Bruno) está conectado com os vértices 0 (Ana) e 3 (Daniel).

A terceira linha "contato 2: 0 3" significa que o vértice 2 (representando Clara) está conectado com os vértices 0 (Ana) e 3 (Daniel).

A última linha "contato 3:" significa que o vértice 3 (representando Daniel) não está conectado com nenhum outro vértice do grafo.

Essas conexões são definidas na matriz de adjacência do grafo, onde um valor 1 na posição (i, j) indica que os vértices i e j estão conectados. O loop externo percorre cada linha da matriz de adjacência, e o loop interno percorre cada coluna da linha. Se o valor na posição (i, j) é 1, significa que os vértices i e j estão conectados, e o vértice j é adicionado à lista de conexões do vértice i. O resultado é impresso na tela para cada vértice.

Capítulo 10.7: O planalto das Matrizes

Agora estamos passando pelo planalto das matrizes, um lugar onde a organização é essencial para se obter sucesso. As matrizes são uma estrutura de dados bidimensional que permite armazenar informações em linhas e colunas. Elas são muito utilizadas na programação para armazenar dados tabulares, como os encontrados em planilhas eletrônicas, e em operações matemáticas, como a multiplicação de matrizes.

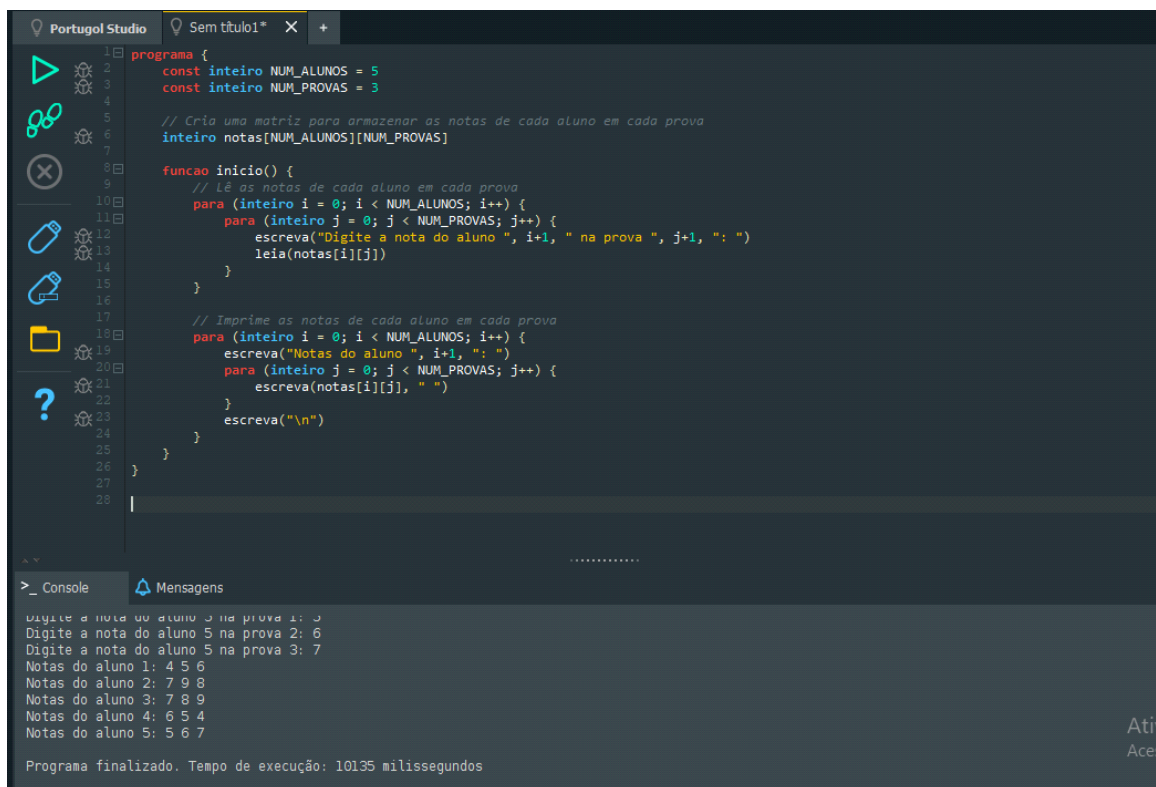
Neste vasto planalto, é possível encontrar muitas aplicações para as matrizes. Elas são usadas em sistemas de gerenciamento de estoque para armazenar informações sobre quantidades de produtos em diferentes filiais, em programas de edição de imagens para armazenar os pixels que compõem uma imagem e em jogos para representar tabuleiros e posições de peças.

Além disso, as matrizes são uma importante ferramenta para resolução de problemas matemáticos. Elas permitem a realização de operações como a soma e a multiplicação de matrizes, que são utilizadas em diversas áreas da matemática, como na álgebra linear.

Porém, o caminho no planalto das matrizes pode ser desafiador. É preciso ter cuidado para não confundir as linhas e colunas, e para garantir que os dados estejam organizados corretamente. Mas com a prática e a habilidade, é possível navegar pelas matrizes com destreza e tirar o máximo proveito dessa importante estrutura de dados.

Um exemplo de uso de matrizes na programação é em um programa de simulação de um jogo, em que os jogadores e as jogadas podem ser representados em uma matriz. Outro exemplo é em um programa de análise de dados, em que os dados podem ser organizados em uma matriz, para que possam ser facilmente manipulados e analisados.

Agora vamos praticar:



```
1 programa {
2     const inteiro NUM_ALUNOS = 5
3     const inteiro NUM_PROVAS = 3
4
5     // Cria uma matriz para armazenar as notas de cada aluno em cada prova
6     inteiro notas[NUM_ALUNOS][NUM_PROVAS]
7
8     funcao inicio() {
9         // Lê as notas de cada aluno em cada prova
10        para (inteiro i = 0; i < NUM_ALUNOS; i++) {
11            para (inteiro j = 0; j < NUM_PROVAS; j++) {
12                escreva("Digite a nota do aluno ", i+1, " na prova ", j+1, ": ")
13                leia(notas[i][j])
14            }
15        }
16
17        // Imprime as notas de cada aluno em cada prova
18        para (inteiro i = 0; i < NUM_ALUNOS; i++) {
19            escreva("Notas do aluno ", i+1, ": ")
20            para (inteiro j = 0; j < NUM_PROVAS; j++) {
21                escreva(notas[i][j], " ")
22            }
23            escreva("\n")
24        }
25    }
26 }
27
28
```

Console

```
Digite a nota do aluno 1 na prova 1: 5
Digite a nota do aluno 1 na prova 2: 6
Digite a nota do aluno 1 na prova 3: 7
Notas do aluno 1: 5 6 7
Notas do aluno 2: 7 9 8
Notas do aluno 3: 7 8 9
Notas do aluno 4: 6 5 4
Notas do aluno 5: 5 6 7

Programa finalizado. Tempo de execução: 10135 milissegundos
```

```
>_ Console  Mensagens

Digite a nota do aluno 1 na prova 1: 4
Digite a nota do aluno 1 na prova 2: 5
Digite a nota do aluno 1 na prova 3: 6
Digite a nota do aluno 2 na prova 1: 7
Digite a nota do aluno 2 na prova 2: 9
Digite a nota do aluno 2 na prova 3: 8
Digite a nota do aluno 3 na prova 1: 7
Digite a nota do aluno 3 na prova 2: 8
Digite a nota do aluno 3 na prova 3: 9
Digite a nota do aluno 4 na prova 1: 6
Digite a nota do aluno 4 na prova 2: 5
Digite a nota do aluno 4 na prova 3: 4
Digite a nota do aluno 5 na prova 1: 5
Digite a nota do aluno 5 na prova 2: 6
Digite a nota do aluno 5 na prova 3: 7
Notas do aluno 1: 4 5 6
Notas do aluno 2: 7 9 8
Notas do aluno 3: 7 8 9
Notas do aluno 4: 6 5 4
Notas do aluno 5: 5 6 7

Programa finalizado. Tempo de execução: 10135 milissegundos
```

Uma matriz é uma estrutura de dados que armazena valores em uma tabela de linhas e colunas. É semelhante a uma planilha de Excel ou uma tabela em um banco de dados. Cada valor na matriz pode ser acessado por sua posição, que é especificada por um número de linha e um número de coluna.

Nesse código apresentado cria uma matriz chamada "notas" com o tamanho "NUM_ALUNOS" x "NUM_PROVAS", que é definido pelas constantes "NUM_ALUNOS" e "NUM_PROVAS". Cada célula da matriz representa a nota de um aluno em uma prova.

O programa, então, usa loops "para" para percorrer cada aluno e cada prova, e pede ao usuário para digitar a nota do aluno na prova correspondente. Cada nota digitada é armazenada na célula correspondente da matriz "notas".

Finalmente, o programa usa outro loop "para" para percorrer cada aluno e imprimir suas notas em cada prova na tela. Ele faz isso acessando os valores armazenados na matriz "notas" e imprimindo-os em uma mensagem para o usuário.

Capítulo 11: Desvendando os mistérios da Terra do Código

Depois de tantas experiências e aprendizados na Terra do Código, chegou a hora de colocar todo esse conhecimento em prática e desvendar os mistérios que ainda existem neste mundo virtual.

Com determinação e curiosidade, você está pronto para encarar os desafios e explorar o que a Terra do Código tem a oferecer. A cada passo dado, novas descobertas serão feitas e novos obstáculos surgirão, mas com perseverança e dedicação, você será capaz de superá-los.

Prepare-se para mergulhar em aventuras emocionantes, que vão testar suas habilidades e expandir ainda mais o seu conhecimento. O caminho a seguir não será fácil, mas com coragem e sabedoria, você será capaz de desvendar todos os mistérios que a Terra do Código tem a revelar.

Exercícios:

- Valor da compra

Escreva um programa que calcule o valor de uma compra com desconto.

Dica: Leia o valor total da compra e o percentual de desconto. Em seguida, calcule o valor do desconto e subtraia-o do valor total da compra.

- Calculadora IMC

Escreva um programa que calcule o IMC (índice de massa corporal) de uma pessoa.

Dica: Leia o peso e a altura da pessoa. Em seguida, calcule o IMC usando a fórmula $IMC = \text{peso} / \text{altura}^2$.

- **Ler temperatura ambiente**

Escreva um programa que leia a temperatura ambiente e informe se está quente, frio ou agradável.

Dica: Defina uma faixa de temperaturas para cada categoria e use uma estrutura condicional para determinar em qual categoria a temperatura se encaixa.

- **Verificador de idade**

Escreva um programa que solicita ao usuário sua data de nascimento e em seguida calcula sua idade. O programa deve então verificar se o usuário é maior de idade ou não.

Dica: use variáveis para armazenar a data de nascimento e a idade, e uma estrutura condicional para verificar se o usuário é maior de idade.

- **Contador de caracteres**

Escreva um programa que solicita ao usuário uma frase e em seguida conta quantos caracteres a frase possui.

Dica: use uma variável para armazenar a frase e a função **comprimento** para contar os caracteres.

- **Calculadora de gorjeta**

Escreva um programa que solicita ao usuário o valor da conta em um restaurante e em seguida calcula a gorjeta, considerando que o usuário quer dar uma porcentagem específica.

Dica: use variáveis para armazenar os valores e a fórmula para calcular a gorjeta.

- **Verificador de palavras palíndromas**

Escreva um programa que solicita ao usuário uma palavra e verifica se ela é um palíndromo. Um palíndromo é uma palavra que pode ser lida da mesma forma da esquerda para a direita e vice-versa.

Dica: use variáveis para armazenar a palavra e uma estrutura de repetição para verificar se a palavra é um palíndromo.

- **Verificador de números primos**

Escreva um programa que solicita ao usuário um número e verifica se ele é primo ou não. Um número primo é um número que é divisível apenas por 1 e por ele mesmo.

Dica: use variáveis para armazenar o número e uma estrutura de repetição para verificar se ele é primo.

- **Gerador de senhas aleatórias**

Escreva um programa que gera uma senha aleatória para o usuário, com um número específico de caracteres.

Dica: use uma variável para armazenar a senha e uma função para gerar números aleatórios.

- **Calculadora de média**

Escreva um programa que solicita ao usuário três números e em seguida calcula a média aritmética entre eles.

Dica: use variáveis para armazenar os números e a fórmula para calcular a média.

- **Conversor de moedas**

Escreva um programa que solicita ao usuário o valor em reais e a cotação do dólar, e em seguida converte o valor em reais para dólares.

Dica: Declare as variáveis para armazenar o valor em reais e a cotação do dólar. Solicite ao usuário o valor em reais e armazene-o na variável correspondente.

Solicite ao usuário a cotação do dólar e armazene-a na variável correspondente.

Realize a conversão do valor em reais para dólares multiplicando-o pela cotação do dólar.

Imprima o valor convertido em dólares na tela.

Lembre-se de tratar possíveis erros de entrada do usuário, como valores não numéricos ou negativos. Também é importante formatar o valor de saída de forma clara e legível.

Parte 4:

A escolha

Capítulo 12: Agora escolha um caminho

Parabéns! Chegamos ao fim da viagem, mas ainda a muito o que explorar e agora é hora de escolher um caminho a seguir. Antes de escolher uma linguagem para começar a estudar, é importante lembrar que existem alguns conceitos básicos que precisam ser aprendidos antes de se aventurar na programação.

O livro abordou três conceitos fundamentais: algoritmos, estruturas de dados e lógica de programação. Algoritmos são passos sequenciais que resolvem um problema específico. Estruturas de dados referem-se à organização de dados em um computador para que possam ser acessados e manipulados com eficiência. Já a lógica de programação é a capacidade de pensar de forma lógica e sistêmica para resolver problemas.

Ao escolher uma linguagem para começar a estudar, é importante considerar seus objetivos e interesses pessoais. Se você está buscando uma linguagem para desenvolver aplicativos móveis, por exemplo, pode ser interessante começar com Java ou Swift. Se sua intenção é trabalhar com análise de dados, Python pode ser a melhor opção. Já se você está interessado em desenvolvimento web, pode ser interessante começar com HTML, CSS e JavaScript.

Lembre-se que não existe uma escolha certa ou errada. Cada linguagem tem suas vantagens e desvantagens, e o que pode ser ideal para uma pessoa, pode não ser o mesmo para outra.

Uma dica importante é não se limitar a uma única linguagem. Aprender várias linguagens pode expandir seus horizontes e torná-lo mais versátil no mercado de trabalho. Além disso, aprender novas linguagens pode ajudá-lo a compreender melhor os conceitos fundamentais da programação.

Porém, para iniciar, é fundamental focar em uma linguagem de programação e dedicar-se a aprender seus fundamentos. Aprender os conceitos básicos de programação, como variáveis, estruturas de controle de fluxo e funções, é essencial para dominar qualquer linguagem.

Por fim, gostaria de incentivá-lo a continuar estudando e praticando. A programação é uma habilidade que requer dedicação e prática constante. Seja persistente e não desanime diante dos desafios. Com o tempo e esforço, você conseguirá atingir seus objetivos.

Espero que este livro tenha sido útil e desejamos boa sorte em sua jornada de aprendizado de programação!

SOBRE A AUTORA:

Ana Caroline Vasconcellos é desenvolvedora Full Stack com 2 anos de experiência na área da programação. Graduanda em Análise e Desenvolvimento de Sistemas e tem se dedicado a projetos de desenvolvimento de software em diversos setores. Além de sua carreira profissional, ela também é uma entusiasta do voluntariado e dedica seu tempo livre a projetos sociais, onde aplica suas habilidades em programação para auxiliar na construção de soluções tecnológicas. Seu objetivo é ajudar a desenvolver soluções inovadoras e eficientes que impactam positivamente a sociedade e aprimoram a experiência do usuário.

Você pode acompanhar o trabalho de Ana Caroline em seu perfil do GitHub e conectá-la em suas redes sociais:

- **LinkedIn:** <https://www.linkedin.com/in/ana-caroline-vasconcellos/>
- **Instagram:** <https://www.instagram.com/anacaroline.vasconcellos/>
- **GitHub pessoal:** <https://www.github.com/JovemDevv/>
- **GitHub do programa:** <https://www.github.com/devprodigio/>